HITACHI
Inspire the Next

OpenTP1 Version 7
分散トランザクション処理機能

OpenTP1 プログラム作成リファレンス C 言語編

手引・文法書

3000-3-D54-90

前書き

■ 対象製品

マニュアル「OpenTP1 解説」を参照してください。

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

HITACHI，Cosminexus，OpenTP1，OSAS，uCosminexus，XMAP は，株式会社 日立製作所の商標または登録商標です。

Windows は，マイクロソフト 企業グループの商標です。

その他記載の会社名，製品名などは，それぞれの会社の商標もしくは登録商標です。

本書には，X/Open の許諾に基づき X/Open CAE Specification System Interfaces and Headers, Issue4,（C202 ISBN 1-872630-47-2）Copyright (C) July 1992，X/Open Company Limited の内容が含まれています；

なお，その一部は IEEE Std 1003.1-1990，(C) 1990 Institute of Electrical and Electronics Engineers, Inc.及び IEEE std 1003.2/D12，(C) 1992 Institute of Electrical and Electronics Engineers, Inc.を基にしています。

事前に著作権所有者の許諾を得ずに，本書の該当部分を複製，複写及び転記することは禁じられています。

本書には，X/Open の許諾に基づき X/Open Preliminary Specification Distributed Transaction Processing：The TxRPC Specification（P305 ISBN 1-85912-000-8）Copyright (C) July 1993, X/Open Company Limited の内容が含まれています；

事前に著作権所有者の許諾を得ずに，本書の該当部分を複製，複写及び転記することは禁じられています。

本書には，Open Software Foundation, Inc.が著作権を有する内容が含まれています。

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

■ 発行

2023 年 7 月　3000-3-D54-90

■ 著作権

All Rights Reserved. Copyright (C) 2006, 2023, Hitachi, Ltd.

変更内容

変更内容（3000-3-D54-90） uCosminexus TP1/Server Base 07-60，uCosminexus TP1/Server Base(64) 07-60

追加・変更内容	変更個所
追加，変更個所はない。	－

単なる誤字・脱字などはお断りなく訂正しました。

変更内容（3000-3-D54-80） uCosminexus TP1/Server Base 07-57，uCosminexus TP1/Server Base(64) 07-57，uCosminexus TP1/Server Base 07-56，uCosminexus TP1/Server Base(64) 07-56，uCosminexus TP1/Server Base 07-54，uCosminexus TP1/Server Base(64) 07-54，uCosminexus TP1/Server Base 07-53，uCosminexus TP1/Server Base(64) 07-53

追加・変更内容
マニュアル訂正の内容を反映した。

変更内容（3000-3-D54-72） uCosminexus TP1/Server Base 07-54，uCosminexus TP1/Server Base(64) 07-54，uCosminexus TP1/Server Base 07-53，uCosminexus TP1/Server Base(64) 07-53

追加・変更内容
マニュアル訂正の内容を反映した。
dc_dam_create 関数の blksize にセクタ長に関する説明を追加した。
dc_rpc_call 関数のリターン値「DCRPCER_TIMED_OUT」の意味を追加した。
dc_rpc_call_to 関数について，次に示す変更をした。 • リターン値「DCRPCER_INVALID_ARGS」の意味を追加 • リターン値「DCRPCER_SERVICE_NOT_UP」を追加 • リターン値「DCRPCER_TRNCHK_EXTEND」の意味を変更 • 注意事項の説明を削除

変更内容（3000-3-D54-71） uCosminexus TP1/Server Base 07-53，uCosminexus TP1/Server Base(64) 07-53

追加・変更内容
サービス関数の処理での注意事項に，サービス関数から使えない関数の説明を追加した。
dc_rap_disconnect 関数の注意事項に記載している UAP トレースのエラー要因コードについての説明内容を変更した。
dc_rpc_call 関数のリターン値「DCRPCER_PROTO(-302)」の意味を追加した。

追加・変更内容
DCRPC_DIRECT_SCHEDULE 関数の機能についての説明内容を変更した。

変更内容（3000-3-D54-70） uCosminexus TP1/Server Base 07-51，uCosminexus TP1/Server Base(64) 07-51，uCosminexus TP1/Message Control 07-51，uCosminexus TP1/Message Control(64) 07-51，uCosminexus TP1/NET/Library 07-51，uCosminexus TP1/NET/Library(64) 07-51

追加・変更内容
次の関数に設定する MCF 通信プロセス識別子またはアプリケーション起動プロセス識別子に関する注意事項を追加した。 • dc_mcf_adltap • dc_mcf_tactcn • dc_mcf_tactle • dc_mcf_tdctcn • dc_mcf_tdctle • dc_mcf_tdlqle • dc_mcf_tlscn • dc_mcf_tlsle • dc_mcf_tlsln • dc_mcf_tofln • dc_mcf_tonln
次の項目の時間監視の設定時間と所要時間の誤差に関する説明を追加した。 • dc_mcf_execap の active • dc_mcf_timer_set の timer

変更内容（3000-3-D54-60） uCosminexus TP1/Server Base 07-50，uCosminexus TP1/Server Base(64) 07-50，uCosminexus TP1/Message Control 07-50，uCosminexus TP1/Message Control(64) 07-50，uCosminexus TP1/NET/Library 07-50，uCosminexus TP1/NET/Library(64) 07-50

追加・変更内容
dc_adm_call_command 関数の機能と引数の説明を変更した。
次の項目の設定値に関する説明を追加した。 • dc_mcf_receive の resv01 • dc_mcf_timer_set の lename
初期状態での受け取り領域の説明を追加した。
アプリケーション属性定義との関連について説明を追加した。
dc_tam_read 関数の引数の説明を変更した。
次の関数のリターン値の説明を変更した。

追加・変更内容
• dc_tam_rewrite • dc_tam_write

変更内容（3000-3-D54-50） uCosminexus TP1/Server Base 07-06，uCosminexus TP1/Server Base(64) 07-06

追加・変更内容
DCUAPCONFPATH 環境変数にユーザサービスデフォルト定義ファイルが設定できる説明を追加した。
dcsvstart コマンドの-a オプションを使用して，SUP のメイン関数に第 1 引数を渡せる旨の説明を追加した。
次の関数のリターン値，および注意事項の説明を追加した。 • dc_dam_read • dc_dam_rewrite • dc_dam_write
dc_mcf_receive 関数で受信できるメッセージに，ユーザタイマ監視を設定したときに指定したメッセージを追加した。

uCosminexus TP1/Server Base 07-05，uCosminexus TP1/Server Base(64) 07-05，uCosminexus TP1/Message Control 07-05，uCosminexus TP1/Message Control(64) 07-05

追加・変更内容
一つのリソースマネジャを複数の制御単位に分け，接続するユーザ名称などを変更してリソースマネジャに接続できるようにした（リソースマネジャ接続先選択機能）。 これに伴い，次の関数を追加した。 • dc_trn_rm_select
コーディング上の注意を追加した。
dc_log_audit_print 関数のリターン値，および注意事項の説明を追加した。
dc_rpc_call 関数のリターン値の説明を追加した。
dc_rpc_open 関数が OpenTP1 の各機能の環境設定（初期化）をする処理について注意事項を追加した。
dc_rpc_set_service_prio 関数ではリモート API 機能を使用できない旨を追加した。

uCosminexus TP1/Message Control 07-00，uCosminexus TP1/Message Control(64) 07-00

追加・変更内容
リモート MCF サービスに関する記述を削除した。

変更内容（3000-3-D54-40） uCosminexus TP1/Server Base 07-04，uCosminexus TP1/Server Base(64) 07-04，uCosminexus TP1/Message Control 07-05，uCosminexus TP1/Message

Control(64) 07-05，uCosminexus TP1/NET/Library 07-05，uCosminexus TP1/NET/Library(64) 07-05

追加・変更内容
次の項目の設定値に関する説明を追加した。 • dc_dam_create の blksize，blknum，および pnum • dc_dam_iopen の pnum
非応答型の MHP からの問い合わせ応答をできるようにした。
dc_mcf_rollback 関数の action に設定する値の説明を追加した。
メッセージ送受信形態の UAP のコーディング例（MHP）を変更した。

変更内容（3000-3-D54-30） 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，uCosminexus TP1/NET/Library(64) 07-04

追加・変更内容
送受信できる一つのセグメントの最大長について説明を追加した。
グローバルドメインについての説明を追加した。
TP1/Message Control のバージョン 6 以前から移行する場合のインタフェースの変更一覧を追加した。

uCosminexus TP1/Message Control 07-02，uCosminexus TP1/NET/Library 07-03

追加・変更内容
アプリケーションに関するタイマ起動要求を，ライブラリ関数で削除できるようにした。 これに伴い，次の関数を追加した。 • dc_mcf_adltap
コネクションの状態表示，確立，および解放を，ライブラリ関数でできるようにした。 これに伴い，次の関数を追加した。 • dc_mcf_tactcn • dc_mcf_tdctcn • dc_mcf_tlscn
MCF 通信サービスまたはアプリケーション起動サービスの状態を，ライブラリ関数で表示できるようにした。 これに伴い，次の関数を追加した。 • dc_mcf_tlscom
論理端末の状態表示，閉塞，閉塞解除，および出力キューの削除を，ライブラリ関数でできるようにした。 これに伴い，次の関数を追加した。 • dc_mcf_tactle • dc_mcf_tdctle

追加・変更内容
• dc_mcf_tdlqle • dc_mcf_tlsle
コネクションの確立要求の受付状態を，ライブラリ関数で表示できるようにした。 これに伴い，次の関数を追加した。 • dc_mcf_tlsln
サーバ型コネクションの確立要求の受付開始・終了を，ライブラリ関数でできるようにした。 これに伴い，次の関数を追加した。 • dc_mcf_tofln • dc_mcf_tonln
MHP でサービス関数動的ローディング機能を使用できるようにした。

変更内容（3000-3-D54-20） uCosminexus TP1/Server Base 07-02，uCosminexus TP1/Message Control 07-01，uCosminexus TP1/NET/Library 07-01

追加・変更内容
監査ログを出力する機能を追加した。 これに伴い，dc_log_audit_print 関数を追加した。
サービス関数を動的にローディングできる機能を追加した。
システムジャーナルファイルを使用しないでシステムを運用する機能（ジャーナルファイルレス機能）を追加した。 これに伴い，関数の引数，リターン値，および注意事項を変更した。
リモート API 機能に関する説明を変更した。 これに伴い，リターン値を追加，変更した。

はじめに

このマニュアルは，OpenTP1 のアプリケーションプログラムで使える，専用のライブラリ関数の文法について説明しています。OpenTP1 のプログラムプロダクトを次に示します。

- 分散トランザクション処理機能　TP1/Server Base
- 分散アプリケーションサーバ　　TP1/LiNK

このマニュアルでは，アプリケーションプログラムの英略称を「ユーザが作成するアプリケーションプログラム」の意味で，UAP（User Application Program）と表記します。

本文中に記載されている製品のうち，このマニュアルの対象製品ではない製品については，OpenTP1 Version 7 対応製品の発行時期をご確認ください。

次に示す製品，および各製品に示したバージョン以降で，ソケット受信型サーバに関する機能はすべて廃止しました。そのため，ユーザサービス定義とユーザサービスデフォルト定義の receive_from オペランドで socket は使用できません。

- P-1M64-2141 uCosminexus TP1/Server Base：07-53-01 以降
- P-1M64-1121 uCosminexus TP1/Server Base(64)：07-53-01 以降
- P-1J64-2171 uCosminexus TP1/Server Base：07-51-02 以降
- P-1J64-1171 uCosminexus TP1/Server Base(64)：07-51-01 以降
- P-8164-2111 uCosminexus TP1/Server Base：07-57 以降
- P-8264-2111 uCosminexus TP1/Server Base(64)：07-57 以降
- P-2464-2294 uCosminexus TP1/Server Base：07-60 以降
- P-2964-2234 uCosminexus TP1/Server Base(64)：07-60 以降

なお，該当する機能を使用した場合の動作は保証できないため，ご注意ください。

■ 対象読者

TP1/Server Base，または TP1/LiNK で使うアプリケーションプログラムを作成するプログラマの方々を対象としています。

オペレーティングシステム，オンラインシステム，使うマシンの操作，およびコーディングに使う C 言語（ANSI C，C++，または K&R 版 C）の文法の知識があることを前提としています。

このマニュアルは，マニュアル「OpenTP1 プログラム作成の手引」の知識があることを前提としていますので，あらかじめお読みいただくことをお勧めします。

■ 文法の記号

このマニュアルで使用する各種記号を説明します。

(1)文法記述記号

指定する値の説明で使用する記号の一覧を示します。

文法記述記号	意味
{　} 波括弧	この記号で囲まれている複数の項目のうちから一つを選択することを示します。 (例) {DCMCFESI ｜ DCMCFEMI} 　DCMCFESI と DCMCFEMI の二つの項目のうち，どちらかを指定することを示します。
[　] きっ甲	この記号で囲まれている項目は，省略できることを示します。 (例) ["エントリポイント名"…] 　"エントリポイント名"…は省略できることを示します。
｜ ストローク	一つの引数に複数の項目を指定するときに，項目同士の区切りを示します。この記号を項目との間に入れて指定してください。 (例) {DCMCFESI ｜ DCMCFEMI} [｜{DCMCFSEQ ｜ DCMCFNSEQ}] 　DCMCFESI と DCMCFSEQ を指定するときは，「DCMCFESI ｜ DCMCFSEQ」と指定します。
＿ 下線	きっ甲 [ ] で囲まれた複数項目のうちで，きっ甲内のすべてを省略したときに，OpenTP1 で仮定する標準値を示します。 (例) [{DCMCFSEQ ｜ DCMCFNSEQ}] 　DCMCFSEQ，DCMCFNSEQ の両方とも省略した場合，DCMCFNSEQ が仮定されます。
… 点線	記述が省略されていることを示します。この記号の直前に示された項目を繰り返し複数個指定できます。 (例)　entry "エントリポイント名" ["エントリポイント名"…]； 　"エントリポイント名"を続けて指定できることを示します。

(2)属性表示記号

コマンドに指定する値の範囲の説明で使用する記号の一覧を示します。

属性表示記号	意味
～	この記号の前に示された項目が，記号 ～ に続く〈　〉で示す規則に従って，値を指定することを示します。
〈　〉	項目を記述するときに従う構文要素を示します。

(3)構文要素

説明で使用する構文要素の一覧を示します。すべて**半角文字**で指定します。

構文要素	指定できる値
数字	0,1,2,3,4,5,6,7,8,9
英字	A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z, a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z，アンダスコア（ _ ）
英数字	数字，英字
記号	! # $ % & ’ ( ) * + - . / : ; < = > ? @ [＼ ¥] ^ _ ` { | } ~
パス名	/（スラント），英数字，および.（ピリオド）の並び パス名は使用している OS に依存します。

(4)設定する値の条件

設定する値の条件を示します。

設定する値	指定できる値の条件
サービスグループ名	31 バイトのアスキー文字列で設定します。ただしヌル文字，空白，ピリオド，および@（アットマーク）は除きます。 引数に設定するときはヌル文字で終了させてください。このヌル文字は文字列の長さには数えません。
サービス名	31 バイトのアスキー文字列で設定します。ただしヌル文字，空白は除きます。 引数に設定するときはヌル文字で終了させてください。このヌル文字は文字列の長さには数えません。
物理ファイル名	スペシャルファイル名に 14 バイト以内の名称を続けたパス名で設定します。このパス名は 63 文字以内で設定してください。
論理ファイル名	1～8 バイト以内の，先頭が英字の英数字の名称で設定します。

■ X/Open 発行のドキュメントの内容から引用した記述について

X/Open 発行の「X/Open CAE Specification Distributed Transaction Processing : The XATMI Specification」の内容から引用した部分

このマニュアルの記述のうち，次に示す部分は，上記ドキュメントの「Chapter 5　C Reference Manual Pages」の記述を，日本語訳したものです。

- 4 章　X/Open に準拠したアプリケーションプログラミングインタフェース

XATMI インタフェースのアプリケーションプログラミングインタフェース（tp～）

X/Open 発行の「X/Open CAE Specification Distributed Transaction Processing : The TX（Transaction Demarcation） Specification」の内容から引用した部分

このマニュアルの記述のうち，次に示す部分は，上記ドキュメントの「Chapter 5　C Reference Manual Pages」の記述を，日本語訳したものです。

- 4 章　X/Open に準拠したアプリケーションプログラミングインタフェース

TX インタフェースのアプリケーションプログラミングインタフェース（tx_～）

X/Open 発行の「X/Open Preliminary Specification Distributed Transaction Processing : The TxRPC Specification」の内容から引用した部分

このマニュアルの記述のうち，次に示す部分は，上記ドキュメントの「Chapter 1　Introduction」「Chapter 2　Model and Definitions」「Chapter 3　Interface Overview」の記述内容の一部を，日本語訳して引用したものです。

- 6 章　X/Open に準拠したアプリケーション間通信（TxRPC）

■ KB（キロバイト）などの単位表記について

1KB（キロバイト），1MB（メガバイト），1GB（ギガバイト），1TB（テラバイト）はそれぞれ 1,024 バイト，$1{,}024^2$ バイト，$1{,}024^3$ バイト，$1{,}024^4$ バイトです。

■ その他の前提条件

このマニュアルをお読みになる際のその他の前提情報については，マニュアル「OpenTP1 解説」を参照してください。

目次

1 アプリケーションプログラムの作成

この章では，OpenTP1 のアプリケーションプログラムを C 言語でコーディングする場合の概要について説明します。

1.1　アプリケーションプログラムのコーディング

1.1.1　アプリケーションプログラムと関数の対応

OpenTP1 のライブラリ関数と機能の対応を次の表に示します。

表 1-1　OpenTP1 のライブラリ関数と機能の対応

機能	OpenTP1 のライブラリ関数名と機能	
システム運用の管理	dc_adm_call_command	運用コマンドの実行
	dc_adm_complete	ユーザサーバの開始処理完了の報告
	dc_adm_status	ユーザサーバの状態の報告
マルチノード機能	dc_adm_get_nd_status	指定した OpenTP1 ノードのステータスの取得
	dc_adm_get_nd_status_begin	OpenTP1 ノードのステータス取得の開始
	dc_adm_get_nd_status_done	OpenTP1 ノードのステータス取得の終了
	dc_adm_get_nd_status_next	OpenTP1 ノードのステータスの取得
	dc_adm_get_nodeconf_begin	ノード識別子の取得の開始
	dc_adm_get_nodeconf_done	ノード識別子の取得の終了
	dc_adm_get_nodeconf_next	ノード識別子の取得
	dc_adm_get_node_id	自ノードのノード識別子の取得
	dc_adm_get_sv_status	指定したユーザサーバのステータスの取得
	dc_adm_get_sv_status_begin	ユーザサーバのステータス取得の開始
	dc_adm_get_sv_status_done	ユーザサーバのステータス取得の終了
	dc_adm_get_sv_status_next	ユーザサーバのステータスの取得
DAM ファイルサービス	dc_dam_bseek	物理ファイルのブロックの検索
	dc_dam_close	論理ファイルのクローズ
	dc_dam_create	物理ファイルの割り当て
	dc_dam_dget	物理ファイルからブロックの直接入力
	dc_dam_dput	物理ファイルへブロックの直接出力
	dc_dam_end	回復対象外 DAM ファイル使用の終了
	dc_dam_get	物理ファイルからブロックの入力
	dc_dam_hold	論理ファイルの閉塞
	dc_dam_iclose	物理ファイルのクローズ

機能	OpenTP1 のライブラリ関数名と機能	
DAM ファイルサービス	dc_dam_iopen	物理ファイルのオープン
	dc_dam_open	論理ファイルのオープン
	dc_dam_put	物理ファイルへブロックの出力
	dc_dam_read	論理ファイルからブロックの入力
	dc_dam_release	論理ファイルの閉塞の解除
	dc_dam_rewrite	論理ファイルのブロックの更新
	dc_dam_start	回復対象外 DAM ファイル使用の開始
	dc_dam_status	論理ファイルの状態の参照
	dc_dam_write	論理ファイルへブロックの出力
IST サービス	dc_ist_close	IST テーブルのクローズ
	dc_ist_open	IST テーブルのオープン
	dc_ist_read	IST テーブルからレコードの入力
	dc_ist_write	IST テーブルへレコードの出力
ユーザジャーナルの取得	dc_jnl_ujput	ユーザジャーナルの取得
資源の排他制御	dc_lck_get	資源の排他
	dc_lck_release_all	全資源の排他の解除
	dc_lck_release_byname	資源名称を指定した排他の解除
監査ログの出力	dc_log_audit_print	監査ログの出力
メッセージログの出力	dc_logprint	メッセージログの出力
メッセージ送受信	dc_mcf_adltap	アプリケーションに関するタイマ起動要求の削除
	dc_mcf_ap_info	アプリケーション情報通知
	dc_mcf_ap_info_uoc	UOC へのアプリケーション情報通知
	dc_mcf_close	MCF 環境のクローズ
	dc_mcf_commit	MHP のコミット
	dc_mcf_contend	継続問い合わせ応答の終了
	dc_mcf_execap	アプリケーションプログラムの起動
	dc_mcf_mainloop	MHP のサービス開始
	dc_mcf_open	MCF 環境のオープン
	dc_mcf_receive	メッセージの受信
	dc_mcf_recvsync	同期型のメッセージの受信

機能	OpenTP1 のライブラリ関数名と機能	
メッセージ送受信	dc_mcf_reply	応答メッセージの送信
	dc_mcf_resend	メッセージの再送
	dc_mcf_rollback	MHP のロールバック
	dc_mcf_send	メッセージの送信
	dc_mcf_sendrecv	同期型のメッセージの送受信
	dc_mcf_sendsync	同期型のメッセージの送信
	dc_mcf_tactcn	コネクションの確立
	dc_mcf_tactle	論理端末の閉塞解除
	dc_mcf_tdctcn	コネクションの解放
	dc_mcf_tdctle	論理端末の閉塞
	dc_mcf_tdlqle	論理端末の出力キュー削除
	dc_mcf_tempget	一時記憶データの受け取り
	dc_mcf_tempput	一時記憶データの更新
	dc_mcf_timer_set	ユーザタイマ監視の設定
	dc_mcf_timer_cancel	ユーザタイマ監視の取り消し
	dc_mcf_tlscn	コネクションの状態取得
	dc_mcf_tlscom	MCF 通信サービスの状態取得
	dc_mcf_tlsle	論理端末の状態取得
	dc_mcf_tlsln	サーバ型コネクションの確立要求の受付状態取得
	dc_mcf_tofln	サーバ型コネクションの確立要求の受付終了
	dc_mcf_tonln	サーバ型コネクションの確立要求の受付開始
性能検証用トレース	dc_prf_get_trace_num	性能検証用トレース取得通番の通知
	dc_prf_utrace_put	ユーザ固有の性能検証用トレースの取得
リモート API 機能	dc_rap_connect	rap リスナーとのコネクションの確立
	dc_rap_disconnect	rap リスナーとのコネクションの解放
リモートプロシジャコール	dc_rpc_call	遠隔サービスの要求
	dc_rpc_call_to	通信先を指定した遠隔サービスの呼び出し
	dc_rpc_close	アプリケーションプログラムの終了
	dc_rpc_cltsend	CUP への一方通知
	dc_rpc_discard_further_replies	処理結果の受信の拒否

機能	OpenTP1 のライブラリ関数名と機能	
リモートプロシジャコール	dc_rpc_discard_specific_reply	特定の処理結果の受信の拒否
	dc_rpc_get_callers_address	クライアント UAP のノードアドレスの取得
	dc_rpc_get_error_descriptor	エラーが発生した非同期応答型 RPC 要求の記述子の取得
	dc_rpc_get_gateway_address	ゲートウェイのノードアドレスの取得
	dc_rpc_get_service_prio	サービス要求のスケジュールプライオリティの参照
	dc_rpc_get_watch_time	サービスの応答待ち時間の参照
	dc_rpc_mainloop	SPP のサービス開始
	dc_rpc_open	アプリケーションプログラムの開始
	dc_rpc_poll_any_replies	処理結果の非同期受信
	dc_rpc_service_retry	サービス関数のリトライ
	dc_rpc_set_service_prio	サービス要求のスケジュールプライオリティの設定
	dc_rpc_set_watch_time	サービスの応答待ち時間の更新
リアルタイム統計情報サービス	dc_rts_utrace_put	任意区間でのリアルタイム統計情報の取得
TAM ファイルサービス	dc_tam_close	TAM テーブルのクローズ
	dc_tam_delete	TAM テーブルのレコードの削除
	dc_tam_get_inf	TAM テーブルの状態の取得
	dc_tam_open	TAM テーブルのオープン
	dc_tam_read	TAM テーブルのレコードの入力
	dc_tam_read_cancel	TAM テーブルのレコードの入力取り消し
	dc_tam_rewrite	TAM テーブルのレコード入力を前提の更新
	dc_tam_status	TAM テーブルの情報の取得
	dc_tam_write	TAM テーブルのレコードの更新/追加
トランザクション制御	dc_trn_begin	トランザクションの開始
	dc_trn_chained_commit	連鎖モードのコミット
	dc_trn_chained_rollback	連鎖モードのロールバック
	dc_trn_info	現在のトランザクションに関する情報の報告
	dc_trn_unchained_commit	非連鎖モードのコミット
	dc_trn_unchained_rollback	非連鎖モードのロールバック
	dc_trn_rm_select	リソースマネジャ接続先選択

機能	OpenTP1 のライブラリ関数名と機能	
オンラインテスタの管理	dc_uto_test_status	ユーザサーバのテスト状態の報告

(1) SUP で使える機能と関数

SUP で使える機能と関数を次の表に示します。

表 1-2 SUP で使える機能と関数

SUP で使える機能		OpenTP1 の関数	SUP が稼働している条件	
			トランザクションの処理の範囲でない	トランザクションの処理の範囲
システム運用の管理	運用コマンドの実行	dc_adm_call_command	○	○
	ユーザサーバの開始処理完了の報告	dc_adm_complete	○	−
	ユーザサーバの状態の報告	dc_adm_status	○	○
マルチノード機能	指定した OpenTP1 ノードのステータスの取得	dc_adm_get_nd_status	○	○
	OpenTP1 ノードのステータス取得の開始	dc_adm_get_nd_status_begin	○	○
	OpenTP1 ノードのステータス取得の終了	dc_adm_get_nd_status_done	○	○
	OpenTP1 ノードのステータスの取得	dc_adm_get_nd_status_next	○	○
	ノード識別子の取得の開始	dc_adm_get_nodeconf_begin	○	○
	ノード識別子の取得の終了	dc_adm_get_nodeconf_done	○	○
	ノード識別子の取得	dc_adm_get_nodeconf_next	○	○
	自ノードのノード識別子の取得	dc_adm_get_node_id	○	○
	指定したユーザサーバのステータスの取得	dc_adm_get_sv_status	○	○
	ユーザサーバのステータス取得の開始	dc_adm_get_sv_status_begin	○	○
	ユーザサーバのステータス取得の終了	dc_adm_get_sv_status_done	○	○
	ユーザサーバのステータスの取得	dc_adm_get_sv_status_next	○	○

SUP で使える機能		OpenTP1 の関数	SUP が稼働している条件	
			トランザクションの処理の範囲でない	トランザクションの処理の範囲
DAM ファイルサービス	論理ファイルのクローズ	dc_dam_close	○	○
	回復対象外 DAM ファイル使用の終了	dc_dam_end	○	○
	論理ファイルの閉塞	dc_dam_hold	○	○
	論理ファイルのオープン	dc_dam_open	○	○
	論理ファイルからブロックの入力	dc_dam_read	○	○
	論理ファイルの閉塞の解除	dc_dam_release	○	○
	論理ファイルのブロックの更新	dc_dam_rewrite	(○)	○
	回復対象外 DAM ファイル使用の開始	dc_dam_start	○	○
	論理ファイルの状態の参照	dc_dam_status	○	○
	論理ファイルへブロックの出力	dc_dam_write	(○)	○
IST サービス	IST テーブルのクローズ	dc_ist_close	○	○
	IST テーブルのオープン	dc_ist_open	○	○
	IST テーブルからレコードの入力	dc_ist_read	○	○
	IST テーブルへレコードの出力	dc_ist_write	○	○
ユーザジャーナルの取得	ユーザジャーナルの取得	dc_jnl_ujput	○	○
資源の排他制御	資源の排他	dc_lck_get	−	○
	全資源の排他の解除	dc_lck_release_all	−	○
	資源名称を指定した排他の解除	dc_lck_release_byname	−	○
監査ログの出力	監査ログの出力	dc_log_audit_print	○	○
メッセージログの出力	メッセージログの出力	dc_logprint	○	○
性能検証用トレース	性能検証用トレース取得通番の通知	dc_prf_get_trace_num	○	○

SUP で使える機能		OpenTP1 の関数	SUP が稼働している条件	
			トランザクションの処理の範囲でない	トランザクションの処理の範囲
性能検証用トレース	ユーザ固有の性能検証用トレースの取得	dc_prf_utrace_put	○	○
リモート API 機能	rap リスナーとのコネクションの確立	dc_rap_connect	○	−
	rap リスナーとのコネクションの解放	dc_rap_disconnect	○	−
リモートプロシジャコール	遠隔サービスの要求	dc_rpc_call	○	○
	通信先を指定した遠隔サービスの呼び出し	dc_rpc_call_to	○	○
	アプリケーションプログラムの終了	dc_rpc_close	○	−
	処理結果の受信の拒否	dc_rpc_discard_further_replies	○	○
	特定の処理結果の受信の拒否	dc_rpc_discard_specific_reply	○	○
	エラーが発生した非同期応答型 RPC 要求の記述子の取得	dc_rpc_get_error_descriptor	○	○
	サービス要求のスケジュールプライオリティの参照	dc_rpc_get_service_prio	○	○
	サービス要求の応答待ち時間の参照	dc_rpc_get_watch_time	○	○
	アプリケーションプログラムの開始	dc_rpc_open	○	−
	処理結果の非同期受信	dc_rpc_poll_any_replies	○	○
	サービス要求のスケジュールプライオリティの設定	dc_rpc_set_service_prio	○	○
	サービス要求の応答待ち時間の更新	dc_rpc_set_watch_time	○	○
リアルタイム統計情報サービス	任意区間でのリアルタイム統計情報の取得	dc_rts_utrace_put	○	○
TAM ファイルサービス	TAM テーブルのクローズ	dc_tam_close	○	○
	TAM テーブルのレコードの削除	dc_tam_delete	−	○
	TAM テーブルの状態の取得	dc_tam_get_inf	○	○
	TAM テーブルのオープン	dc_tam_open	○	○

SUP で使える機能		OpenTP1 の関数	SUP が稼働している条件	
			トランザクションの処理の範囲でない	トランザクションの処理の範囲
TAM ファイルサービス	TAM テーブルのレコードの入力	dc_tam_read	−	○
	TAM テーブルのレコードの入力取り消し	dc_tam_read_cancel	−	○
	TAM テーブルのレコード入力を前提の更新	dc_tam_rewrite	−	○
	TAM テーブルの情報の取得	dc_tam_status	○	○
	TAM テーブルのレコードの更新／追加	dc_tam_write	−	○
トランザクション制御	トランザクションの開始	dc_trn_begin	○	−
	連鎖モードのコミット	dc_trn_chained_commit	−	○
	連鎖モードのロールバック	dc_trn_chained_rollback	−	○
	現在のトランザクションに関する情報の出力	dc_trn_info	○	○
	非連鎖モードのコミット	dc_trn_unchained_commit	−	○
	非連鎖モードのロールバック	dc_trn_unchained_rollback	−	○
	リソースマネジャ接続先選択	dc_trn_rm_select	○	−
オンラインテスタの管理	ユーザサーバのテスト状態の報告	dc_uto_test_status	○	○

（凡例）

○：該当する条件で使えます。

(○)：回復対象外の DAM ファイルにアクセスするときだけ，使えます。

－ ：該当する条件では使えません。

(2) SPP で使える機能と関数

SPP で使える機能と関数を次の表に示します。

表 1-3　SPP で使える機能と関数

SPP で使える機能		OpenTP1 の関数	SPP が稼働している条件		
			トランザクションの処理の範囲でない	トランザクションの範囲	
				ルート	ルート以外
システム運用の管理	運用コマンドの実行	dc_adm_call_command	○	○	○
	ユーザサーバの状態の報告	dc_adm_status	○	○	○
マルチノード機能	指定した OpenTP1 ノードのステータスの取得	dc_adm_get_nd_status	○	○	○
	OpenTP1 ノードのステータス取得の開始	dc_adm_get_nd_status_begin	○	○	○
	OpenTP1 ノードのステータス取得の終了	dc_adm_get_nd_status_done	○	○	○
	OpenTP1 ノードのステータスの取得	dc_adm_get_nd_status_next	○	○	○
	ノード識別子の取得の開始	dc_adm_get_nodeconf_begin	○	○	○
	ノード識別子の取得の終了	dc_adm_get_nodeconf_done	○	○	○
	ノード識別子の取得	dc_adm_get_nodeconf_next	○	○	○
	自ノードのノード識別子の取得	dc_adm_get_node_id	○	○	○
	指定したユーザサーバのステータスの取得	dc_adm_get_sv_status	○	○	○
	ユーザサーバのステータス取得の開始	dc_adm_get_sv_status_begin	○	○	○
	ユーザサーバのステータス取得の終了	dc_adm_get_sv_status_done	○	○	○
	ユーザサーバのステータスの取得	dc_adm_get_sv_status_next	○	○	○
DAM ファイルサービス	論理ファイルのクローズ	dc_dam_close	○	○	○
	回復対象外 DAM ファイル使用の終了	dc_dam_end	○	○	○
	論理ファイルの閉塞	dc_dam_hold	○	○	○
	論理ファイルのオープン	dc_dam_open	○	○	○
	論理ファイルからブロックの入力	dc_dam_read	○	○	○
	論理ファイルの閉塞の解除	dc_dam_release	○	○	○

SPP で使える機能		OpenTP1 の関数	SPP が稼働している条件		
			トランザクションの処理の範囲でない	トランザクションの範囲	
				ルート	ルート以外
DAM ファイルサービス	論理ファイルのブロックの更新	dc_dam_rewrite	(○)	○	○
	回復対象外 DAM ファイル使用の開始	dc_dam_start	○	○	○
	論理ファイルの状態の参照	dc_dam_status	○	○	○
	論理ファイルへブロックの出力	dc_dam_write	(○)	○	○
IST サービス	IST テーブルのクローズ	dc_ist_close	○	○	○
	IST テーブルのオープン	dc_ist_open	○	○	○
	IST テーブルからレコードの入力	dc_ist_read	○	○	○
	IST テーブルへレコードの出力	dc_ist_write	○	○	○
ユーザジャーナルの取得	ユーザジャーナルの取得	dc_jnl_ujput	○	○	○
資源の排他制御	資源の排他	dc_lck_get	−	○	○
	全資源の排他の解除	dc_lck_release_all	−	○	○
	資源名称を指定した排他の解除	dc_lck_release_byname	−	○	○
監査ログの出力	監査ログの出力	dc_log_audit_print	○	○	○
メッセージログの出力	メッセージログの出力	dc_logprint	○	○	○
メッセージ送受信	アプリケーションに関するタイマ起動要求の削除	dc_mcf_adltap	○	○	○
	MCF 環境のクローズ	dc_mcf_close	$○_M$	−	−
	アプリケーションプログラムの起動	dc_mcf_execap	−	○	○
	MCF 環境のオープン	dc_mcf_open	$○_M$	−	−
	同期型のメッセージの受信	dc_mcf_recvsync	○	○	○
	メッセージの再送	dc_mcf_resend	−	○	○
	メッセージの送信	dc_mcf_send	−	○	○

SPP で使える機能		OpenTP1 の関数	SPP が稼働している条件		
			トランザクションの処理の範囲でない	トランザクションの範囲	
				ルート	ルート以外
メッセージ送受信	同期型のメッセージの送受信	dc_mcf_sendrecv	○	○	○
	同期型のメッセージの送信	dc_mcf_sendsync	○	○	○
	コネクションの確立	dc_mcf_tactcn	○	○	○
	論理端末の閉塞解除	dc_mcf_tactle	○	○	○
	コネクションの解放	dc_mcf_tdctcn	○	○	○
	論理端末の閉塞	dc_mcf_tdctle	○	○	○
	論理端末の出力キュー削除	dc_mcf_tdlqle	○	○	○
	ユーザタイマ監視の設定	dc_mcf_timer_set	○	○	○
	ユーザタイマ監視の取り消し	dc_mcf_timer_cancel	○	○	○
	コネクションの状態取得	dc_mcf_tlscn	○	○	○
	MCF 通信サービスの状態取得	dc_mcf_tlscom	○	○	○
	論理端末の状態取得	dc_mcf_tlsle	○	○	○
	サーバ型コネクションの確立要求の受付状態取得	dc_mcf_tlsln	○	○	○
	サーバ型コネクションの確立要求の受付終了	dc_mcf_tofln	○	○	○
	サーバ型コネクションの確立要求の受付開始	dc_mcf_tonln	○	○	○
性能検証用トレース	性能検証用トレース取得通番の通知	dc_prf_get_trace_num	○	○	○
	ユーザ固有の性能検証用トレースの取得	dc_prf_utrace_put	○	○	○
リモート API 機能	rap リスナーとのコネクションの確立	dc_rap_connect	○	−	−
	rap リスナーとのコネクションの解放	dc_rap_disconnect	○	−	−
リモートプロシジャコール	遠隔サービスの要求	dc_rpc_call	○	○	○
	通信先を指定した遠隔サービスの呼び出し	dc_rpc_call_to	○	○	○
	アプリケーションプログラムの終了	dc_rpc_close	○M	−	−

SPP で使える機能		OpenTP1 の関数	SPP が稼働している条件		
			トランザクションの処理の範囲でない	トランザクションの範囲	
				ルート	ルート以外
リモートプロシジャコール	CUP への一方通知	dc_rpc_cltsend	○	○	○
	処理結果の受信の拒否	dc_rpc_discard_further_replies	○	○	○
	特定の処理結果の受信の拒否	dc_rpc_discard_specific_reply	○	○	○
	クライアント UAP のノードアドレスの取得	dc_rpc_get_callers_address	○	○	○
	エラーが発生した非同期応答型 RPC 要求の記述子の取得	dc_rpc_get_error_descriptor	○	○	○
	ゲートウェイのノードアドレスの取得	dc_rpc_get_gateway_address	○	○	○
	サービス要求のスケジュールプライオリティの参照	dc_rpc_get_service_prio	○	○	○
	サービス要求の応答待ち時間の参照	dc_rpc_get_watch_time	○	○	○
	SPP のサービス開始	dc_rpc_mainloop	○M	−	−
	アプリケーションプログラムの開始	dc_rpc_open	○M	−	−
	処理結果の非同期受信	dc_rpc_poll_any_replies	○	○	○
	サービス関数のリトライ	dc_rpc_service_retry	○	−	−
	サービス要求のスケジュールプライオリティの設定	dc_rpc_set_service_prio	○	○	○
	サービス要求の応答待ち時間の更新	dc_rpc_set_watch_time	○	○	○
リアルタイム統計情報サービス	任意区間でのリアルタイム統計情報の取得	dc_rts_utrace_put	○	○	○
TAM ファイルサービス	TAM テーブルのクローズ	dc_tam_close	○	○	○
	TAM テーブルのレコードの削除	dc_tam_delete	−	○	○
	TAM テーブルの状態の取得	dc_tam_get_inf	○	○	○
	TAM テーブルのオープン	dc_tam_open	○	○	○
	TAM テーブルのレコードの入力	dc_tam_read	−	○	○

SPP で使える機能		OpenTP1 の関数	SPP が稼働している条件		
			トランザクションの処理の範囲でない	トランザクションの範囲	
				ルート	ルート以外
TAM ファイルサービス	TAM テーブルのレコードの入力取り消し	dc_tam_read_cancel	−	○	○
	TAM テーブルのレコード入力を前提の更新	dc_tam_rewrite	−	○	○
	TAM テーブルの情報の取得	dc_tam_status	○	○	○
	TAM テーブルのレコードの更新／追加	dc_tam_write	−	○	○
トランザクション制御	トランザクションの開始	dc_trn_begin	○	−	−
	連鎖モードのコミット	dc_trn_chained_commit	−	○	−
	連鎖モードのロールバック	dc_trn_chained_rollback	−	○	−
	現在のトランザクションに関する情報の報告	dc_trn_info	○	○	○
	非連鎖モードのコミット	dc_trn_unchained_commit	−	○	−
	非連鎖モードのロールバック	dc_trn_unchained_rollback	−	○	○
	リソースマネジャ接続先選択	dc_trn_rm_select	○	−	−
オンラインテスタの管理	ユーザサーバのテスト状態の報告	dc_uto_test_status	○	○	○

（凡例）

○：該当する条件で使えます。

(○)：回復対象外の DAM ファイルにアクセスするときだけ，使えます。

$○_M$：メイン関数からだけ，使えます。

−：該当する条件では使えません。

注

「ルート」とは，ルートトランザクションブランチ，「ルート以外」とは，ルートトランザクションブランチ以外のトランザクションブランチのことです。

(3) MHP で使える機能と関数

MHP で使える機能と関数を次の表に示します。

表 1-4　MHP で使える機能と関数

MHP で使える機能		OpenTP1 の関数	MHP が稼働している条件	
			トランザクションの処理の範囲でない	トランザクションの処理の範囲
システム運用の管理	運用コマンドの実行	dc_adm_call_command	○	○
	ユーザサーバの状態の報告	dc_adm_status	○	○
マルチノード機能	指定した OpenTP1 ノードのステータスの取得	dc_adm_get_nd_status	○	○
	OpenTP1 ノードのステータス取得の開始	dc_adm_get_nd_status_begin	○	○
	OpenTP1 ノードのステータス取得の終了	dc_adm_get_nd_status_done	○	○
	OpenTP1 ノードのステータスの取得	dc_adm_get_nd_status_next	○	○
	ノード識別子の取得の開始	dc_adm_get_nodeconf_begin	○	○
	ノード識別子の取得の終了	dc_adm_get_nodeconf_done	○	○
	ノード識別子の取得	dc_adm_get_nodeconf_next	○	○
	自ノードのノード識別子の取得	dc_adm_get_node_id	○	○
	指定したユーザサーバのステータスの取得	dc_adm_get_sv_status	○	○
	ユーザサーバのステータス取得の開始	dc_adm_get_sv_status_begin	○	○
	ユーザサーバのステータス取得の終了	dc_adm_get_sv_status_done	○	○
	ユーザサーバのステータスの取得	dc_adm_get_sv_status_next	○	○
DAM ファイルサービス	論理ファイルのクローズ	dc_dam_close	○	○
	回復対象外 DAM ファイル使用の終了	dc_dam_end	○	○
	論理ファイルの閉塞	dc_dam_hold	○	○
	論理ファイルのオープン	dc_dam_open	○	○
	論理ファイルからブロックの入力	dc_dam_read	○	○
	論理ファイルの閉塞の解除	dc_dam_release	○	○

MHP で使える機能		OpenTP1 の関数	MHP が稼働している条件	
			トランザクションの処理の範囲でない	トランザクションの処理の範囲
DAM ファイルサービス	論理ファイルのブロックの更新	dc_dam_rewrite	(○)	○
	回復対象外 DAM ファイル使用の開始	dc_dam_start	○	○
	論理ファイルの状態の参照	dc_dam_status	○	○
	論理ファイルへブロックの出力	dc_dam_write	(○)	○
IST サービス	IST テーブルのクローズ	dc_ist_close	○	○
	IST テーブルのオープン	dc_ist_open	○	○
	IST テーブルからレコードの入力	dc_ist_read	○	○
	IST テーブルへレコードの出力	dc_ist_write	○	○
ユーザジャーナルの取得	ユーザジャーナルの取得	dc_jnl_ujput	○	○
資源の排他制御	資源の排他	dc_lck_get	−	○
	全資源の排他の解除	dc_lck_release_all	−	○
	資源名称を指定した排他の解除	dc_lck_release_byname	−	○
監査ログの出力	監査ログの出力	dc_log_audit_print	○	○
メッセージログの出力	メッセージログの出力	dc_logprint	○	○
メッセージ送受信	アプリケーションに関するタイマ起動要求の削除	dc_mcf_adltap	○	○
	アプリケーション情報通知	dc_mcf_ap_info	$○_{NO}$	○
	MCF 環境のクローズ	dc_mcf_close	$○_{M}$	$○_{M}$
	MHP のコミット	dc_mcf_commit	−	○
	継続問い合わせ応答の終了	dc_mcf_contend	$○_{NO}$	○
	アプリケーションプログラムの起動	dc_mcf_execap	$○_{NO}$	○
	MHP のサービス開始	dc_mcf_mainloop	$○_{M}$	−
	MCF 環境のオープン	dc_mcf_open	$○_{M}$	$○_{M}$

MHP で使える機能		OpenTP1 の関数	MHP が稼働している条件	
			トランザクションの処理の範囲でない	トランザクションの処理の範囲
メッセージ送受信	メッセージの受信	dc_mcf_receive	○NO	○
	同期型のメッセージの受信	dc_mcf_recvsync	○	○
	応答メッセージの送信	dc_mcf_reply	○NO	○
	メッセージの再送	dc_mcf_resend	−	○
	MHP のロールバック	dc_mcf_rollback	−	○
	メッセージの送信	dc_mcf_send	○NO	○
	同期型のメッセージの送受信	dc_mcf_sendrecv	○	○
	同期型のメッセージの送信	dc_mcf_sendsync	○	○
	コネクションの確立	dc_mcf_tactcn	○	○
	論理端末の閉塞解除	dc_mcf_tactle	○	○
	コネクションの解放	dc_mcf_tdctcn	○	○
	論理端末の閉塞	dc_mcf_tdctle	○	○
	論理端末の出力キュー削除	dc_mcf_tdlqle	○	○
	一時記憶データの受け取り	dc_mcf_tempget	○NO	○
	一時記憶データの更新	dc_mcf_tempput	○NO	○
	ユーザタイマ監視の設定	dc_mcf_timer_set	○	○
	ユーザタイマ監視の取り消し	dc_mcf_timer_cancel	○	○
	コネクションの状態取得	dc_mcf_tlscn	○	○
	MCF 通信サービスの状態取得	dc_mcf_tlscom	○	○
	論理端末の状態取得	dc_mcf_tlsle	○	○
	サーバ型コネクションの確立要求の受付状態取得	dc_mcf_tlsln	○	○
	サーバ型コネクションの確立要求の受付終了	dc_mcf_tofln	○	○
	サーバ型コネクションの確立要求の受付開始	dc_mcf_tonln	○	○
性能検証用トレース	性能検証用トレース取得通番の通知	dc_prf_get_trace_num	○	○

MHP で使える機能		OpenTP1 の関数	MHP が稼働している条件	
			トランザクションの処理の範囲でない	トランザクションの処理の範囲
性能検証用トレース	ユーザ固有の性能検証用トレースの取得	dc_prf_utrace_put	○	○
リモート API 機能	rap リスナーとのコネクションの確立	dc_rap_connect	○	−
	rap リスナーとのコネクションの解放	dc_rap_disconnect	○	−
リモートプロシジャコール	遠隔サービスの要求	dc_rpc_call	○	○
	通信先を指定した遠隔サービスの呼び出し	dc_rpc_call_to	○	○
	アプリケーションプログラムの終了	dc_rpc_close	○M	−
	CUP への一方通知	dc_rpc_cltsend	○	○
	処理結果の受信の拒否	dc_rpc_discard_further_replies	○	○
	特定の処理結果の受信の拒否	dc_rpc_discard_specific_reply	○	○
	エラーが発生した非同期応答型 RPC 要求の記述子の取得	dc_rpc_get_error_descriptor	○	○
	サービス要求のスケジュールプライオリティの参照	dc_rpc_get_service_prio	○	○
	サービス要求の応答待ち時間の参照	dc_rpc_get_watch_time	○	○
	アプリケーションプログラムの開始	dc_rpc_open	○M	−
	処理結果の非同期受信	dc_rpc_poll_any_replies	○	○
	サービス要求のスケジュールプライオリティの設定	dc_rpc_set_service_prio	○	○
	サービス要求の応答待ち時間の更新	dc_rpc_set_watch_time	○	○
リアルタイム統計情報サービス	任意区間でのリアルタイム統計情報の取得	dc_rts_utrace_put	○	○
TAM ファイルサービス	TAM テーブルのクローズ	dc_tam_close	○	○
	TAM テーブルのレコードの削除	dc_tam_delete	−	○
	TAM テーブルの状態の取得	dc_tam_get_inf	○	○

MHP で使える機能		OpenTP1 の関数	MHP が稼働している条件	
			トランザクションの処理の範囲でない	トランザクションの処理の範囲
TAM ファイルサービス	TAM テーブルのオープン	dc_tam_open	○	○
	TAM テーブルのレコードの入力	dc_tam_read	−	○
	TAM テーブルのレコードの入力取り消し	dc_tam_read_cancel	−	○
	TAM テーブルのレコード入力を前提の更新	dc_tam_rewrite	−	○
	TAM テーブルの情報の取得	dc_tam_status	○	○
	TAM テーブルのレコードの更新/追加	dc_tam_write	−	○
トランザクション制御	トランザクションの開始	dc_trn_begin	$○_{M}$	−
	現在のトランザクションに関する情報の出力	dc_trn_info	○	○
	非連鎖モードのコミット	dc_trn_unchained_commit	−	$○_{M}$
	非連鎖モードのロールバック	dc_trn_unchained_rollback	−	$○_{M}$
	リソースマネジャ接続先選択	dc_trn_rm_select	−	−
オンラインテスタの管理	ユーザサーバのテスト状態の報告	dc_uto_test_status	○	○

(凡例)

○：該当する条件で使えます。

$○_{M}$：メイン関数からだけ，使えます。

$○_{NO}$：非トランザクション属性の MHP の，サービス関数の範囲からだけ使えます。

(○)：回復対象外の DAM ファイルにアクセスするときだけ，使えます。

−：該当する条件では使えません。

注

「トランザクションの処理の範囲でない」とは，非トランザクション属性の MHP，または MHP のメイン関数の範囲を示します。

(4) オフラインの業務をする UAP で使える機能と関数

オフラインの業務をする UAP で使える機能と関数を次の表に示します。

表 1-5 オフラインの業務をする UAP で使える機能と関数

オフラインの業務をする UAP で使える機能		OpenTP1 の関数
DAM ファイルサービス	物理ファイルのブロックの検索	dc_dam_bseek

オフラインの業務をする UAP で使える機能		OpenTP1 の関数
DAM ファイルサービス	物理ファイルの割り当て	dc_dam_create
	物理ファイルからブロックの直接入力	dc_dam_dget
	物理ファイルへブロックの直接出力	dc_dam_dput
	物理ファイルからブロックの入力	dc_dam_get
	物理ファイルのクローズ	dc_dam_iclose
	物理ファイルのオープン	dc_dam_iopen
	物理ファイルへブロックの出力	dc_dam_put
性能検証用トレース	性能検証用トレース取得通番の通知	dc_prf_get_trace_num
	ユーザ固有の性能検証用トレースの取得	dc_prf_utrace_put

1.1.2 コーディング規約

(1) コーディング上の注意

OpenTP1 の UAP のコーディングには，**C 言語**，および **C++言語**を使えます。C 言語の場合，**ANSI C** の形式 または ANSI 準拠前の **K&R の形式**のどちらかに従ってコーディングします。C++言語の場合，**C++言語の仕様**でコーディングします。一部の制限はありますが，OS で標準に提供するライブラリにある関数は，OpenTP1 のライブラリにある関数とあわせて使えます。

さらに，システムコールや任意のプログラムのライブラリも使えますが，コーディング時には，UAP の移植性を高めるため，OS で標準に提供する関数やシステムコールを使うことをお勧めします。

システムコールや任意のプログラムのライブラリを使う UAP を作成するときは，次のことに注意してください。

1. UAP でシグナルを使うときは，SIGILL や SIGBUS などシグナルのデフォルト動作がコアファイルを作成するタイプのシグナルハンドラを登録しないでください。これらのシグナルハンドラを登録すると，プログラムが異常終了してもコアファイルが作成されないため，トラブルシュートができない場合があります。
2. UAP でシグナルを使うときは，シグナルハンドラからは OpenTP1 のライブラリにある関数は呼び出さないでください。
3. 次のシステムコールは使わないでください。
   - chdir（カレントワーキングディレクトリを変更）
4. dc_rpc_open 関数を呼び出したあとには，次のシステムコールは使わないでください。
   - fork（新プロセスの生成）
   - exec（ファイルの実行）

- system（シェルコマンドの発行）

5. C 言語のライブラリ中の，関数間にわたるジャンプ関数（setjmp，longjmp）は使わないでください。
6. ほかのプログラムのライブラリを使うときは，Xlib や OSF/Motif などの，イベント駆動型のディスパッチング制御をする関数は使わないでください。
7. dc_rpc_open 関数発行後，UAP プロセスでオープンするファイルディスクリプタの最大数を変更しないでください。変更した場合の動作は保証できません。

OS が HP-UX の場合，リンケージ時のバインドモードには必ず"immediate"を指定してください。"immediate"以外のバインドモードで作成した実行形式ファイルを OpenTP1 の UAP として使った場合，システムの動作は保証しません。作成した UAP のバインドモードが"immediate"かどうかは，OS の chatr コマンドで確認してください。

(2) 名称の付け方の注意

ユーザがコーディングする変数名や定義名などには，先頭に何文字かのプレフィックスを付加することをお勧めします。OS や OpenTP1 などと名称が重複した場合の動作は保証しません。

(a) サービス関数名

サービス関数の名称は，20 文字以内の長さで，先頭が英字で始まる英数字を付けます。サービス関数名には，次の名称は使わないでください。

- "dc"で始まる名称
- "CBLDC"で始まる名称
- "tx"，または"TX"で始まる名称
- "tp"，または"TP"で始まる名称

(b) 外部変数名

外部変数名には，次の名称は使わないでください。ただし，このマニュアルの記述に従って使う場合を除きます。

- "dc"で始まる名称
- "CBLDC"で始まる名称
- "tx"，または"TX"で始まる名称
- "tp"，または"TP"で始まる名称

(c) 定数名

＃define 文で定義する定数名には，次の名称は使わないでください。ただし，このマニュアルの記述に従って使う場合を除きます。

- "DC"で始まる名称

- "CBLDC"で始まる名称
- "TX"で始まる名称
- "TP"で始まる名称

(3) 終了のしかたについて

C 言語で作成した UAP のプロセスで，一度でも COBOL 言語のプログラムを実行した場合は，CBLEND 関数を使って exit してください。CBLEND 関数を使わないで終了すると，COBOL 言語のカウント情報などが出力されません。CBLEND 関数については，該当する COBOL 言語のマニュアルを参照してください。

(4) Windows を使う場合

OpenTP1（TP1/LiNK）を Windows で使う場合に，UAP をコンパイル，リンケージするときは，Windows で使う C コンパイラの仕様に従ってください。

(5) TP1/Message Control を使う場合

C 言語で作成した 32 ビットアーキテクチャの Version 6 で使用していたユーザアプリケーションプログラムおよびユーザオウンコーディングのソースファイルを 32 ビットアーキテクチャの Version 7 で使用する場合，および，C 言語で作成した 64 ビットアーキテクチャの Version 6 で使用していたユーザアプリケーションプログラムおよびユーザオウンコーディングのソースファイルを 64 ビットアーキテクチャの Version 7 で使用する場合，ユーザアプリケーションプログラムおよびユーザオウンコーディングのソースファイルをそのまま使用できます。

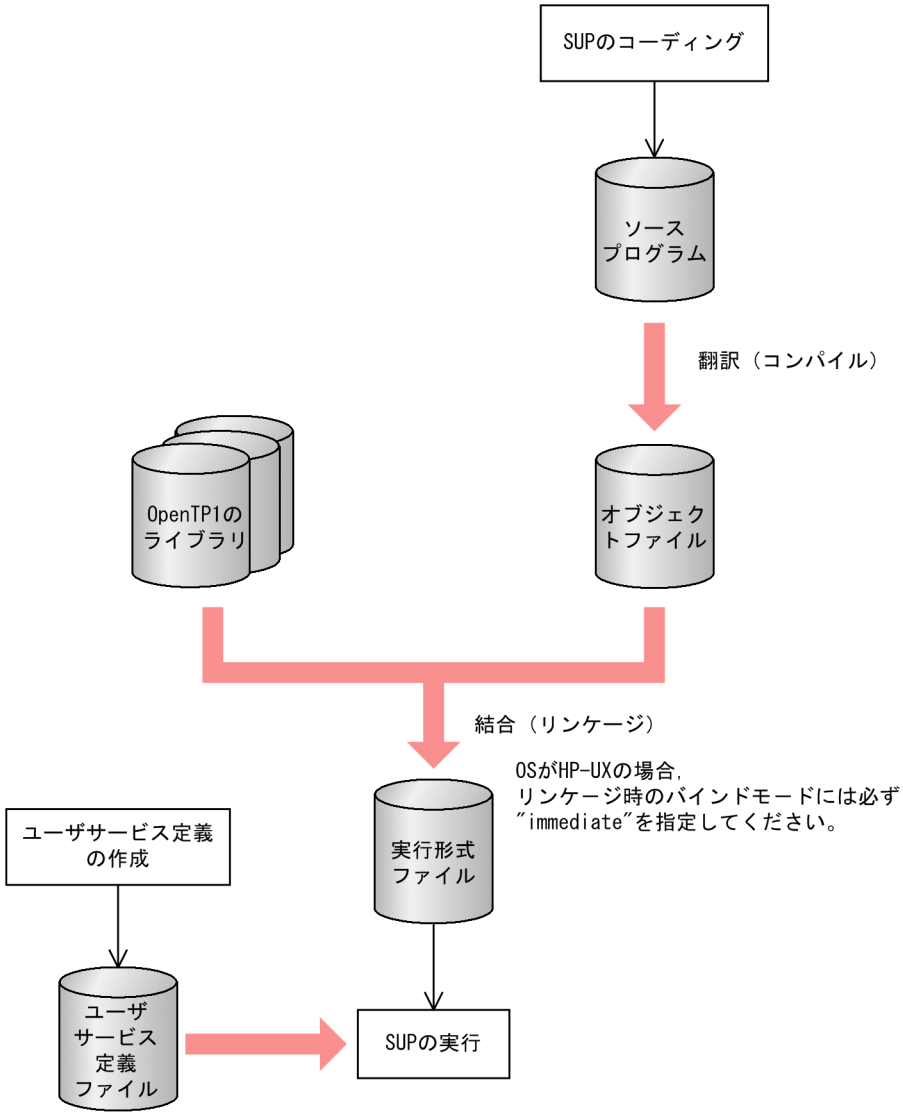
1.2　アプリケーションプログラムの作成（TCP/IP 通信）

1.2.1　アプリケーションプログラムの作成手順

(1)　SUP の作成手順

SUP の作成手順を次の図に示します。

図 1-1　SUP の作成手順



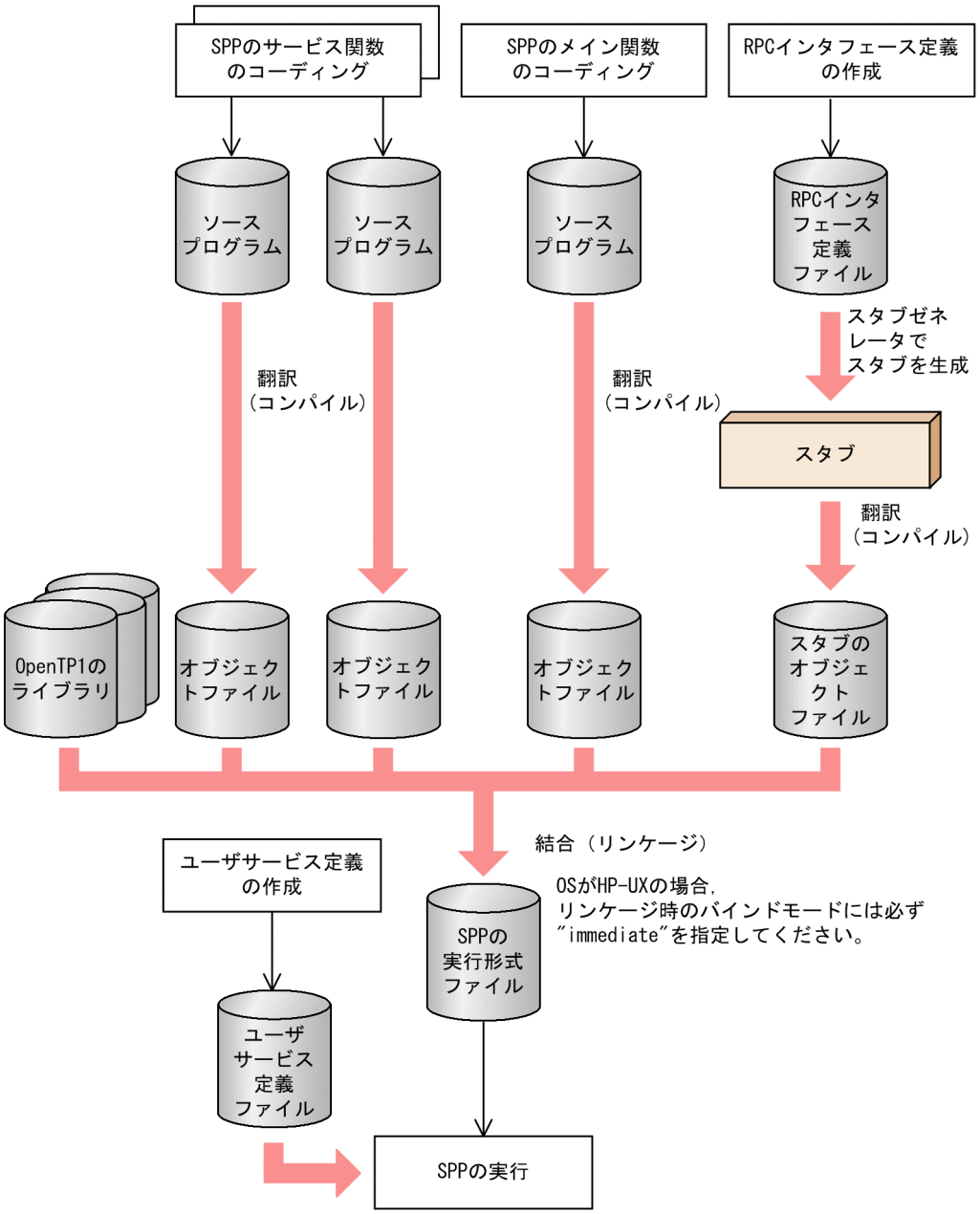
(2)　SPP の作成手順

SPP の作成手順は，「スタブを使用する SPP」か「サービス関数動的ローディング機能を使用する SPP」かで異なります。

(a)　SPP の作成手順（スタブ使用時）

スタブを使用する SPP の作成手順を，次の図に示します。

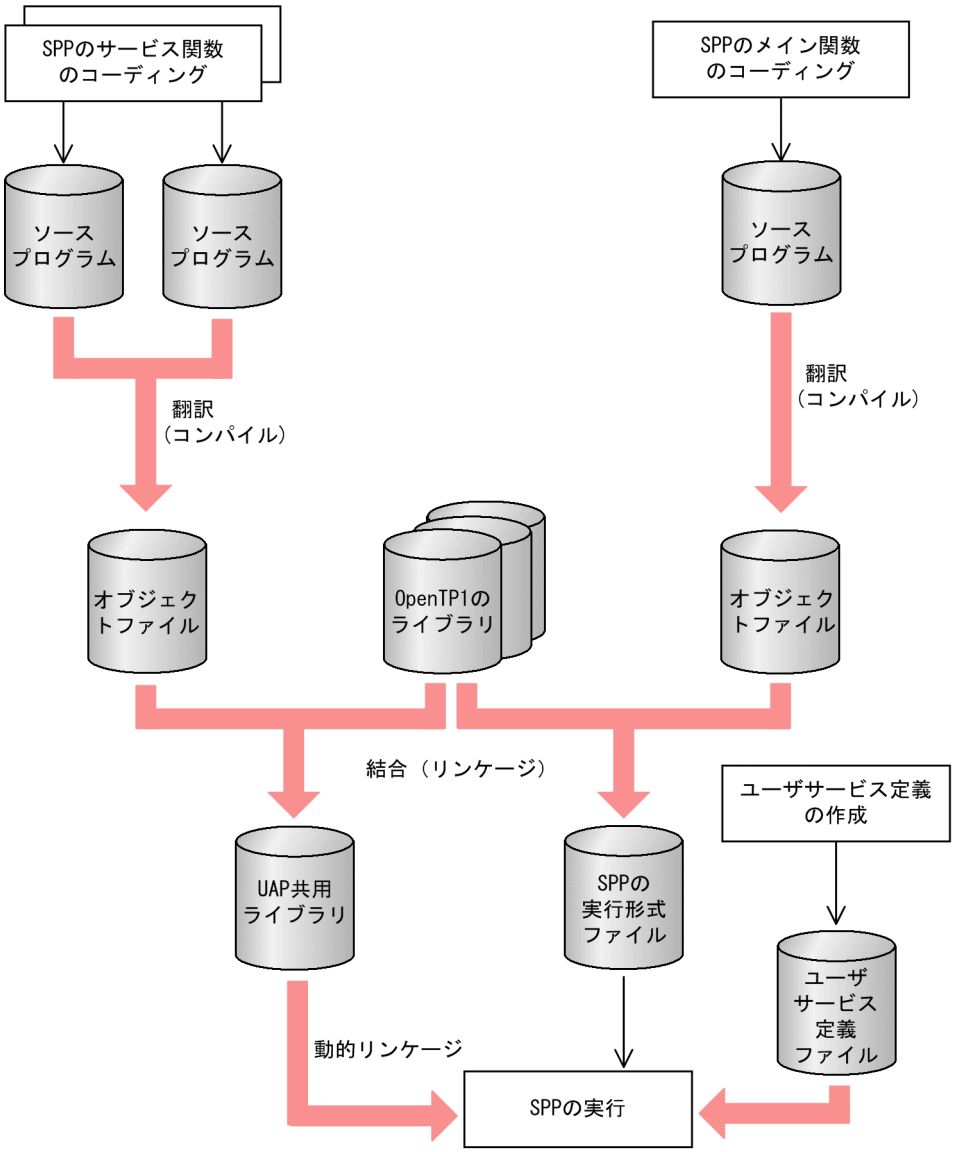
図 1-2　SPP の作成手順（スタブ使用時）



(b)　SPP の作成手順（サービス関数動的ローディング機能使用時）

サービス関数動的ローディング機能を使用する SPP の作成手順を，次の図に示します。

図 1-3　SPP の作成手順（サービス関数動的ローディング機能使用時）



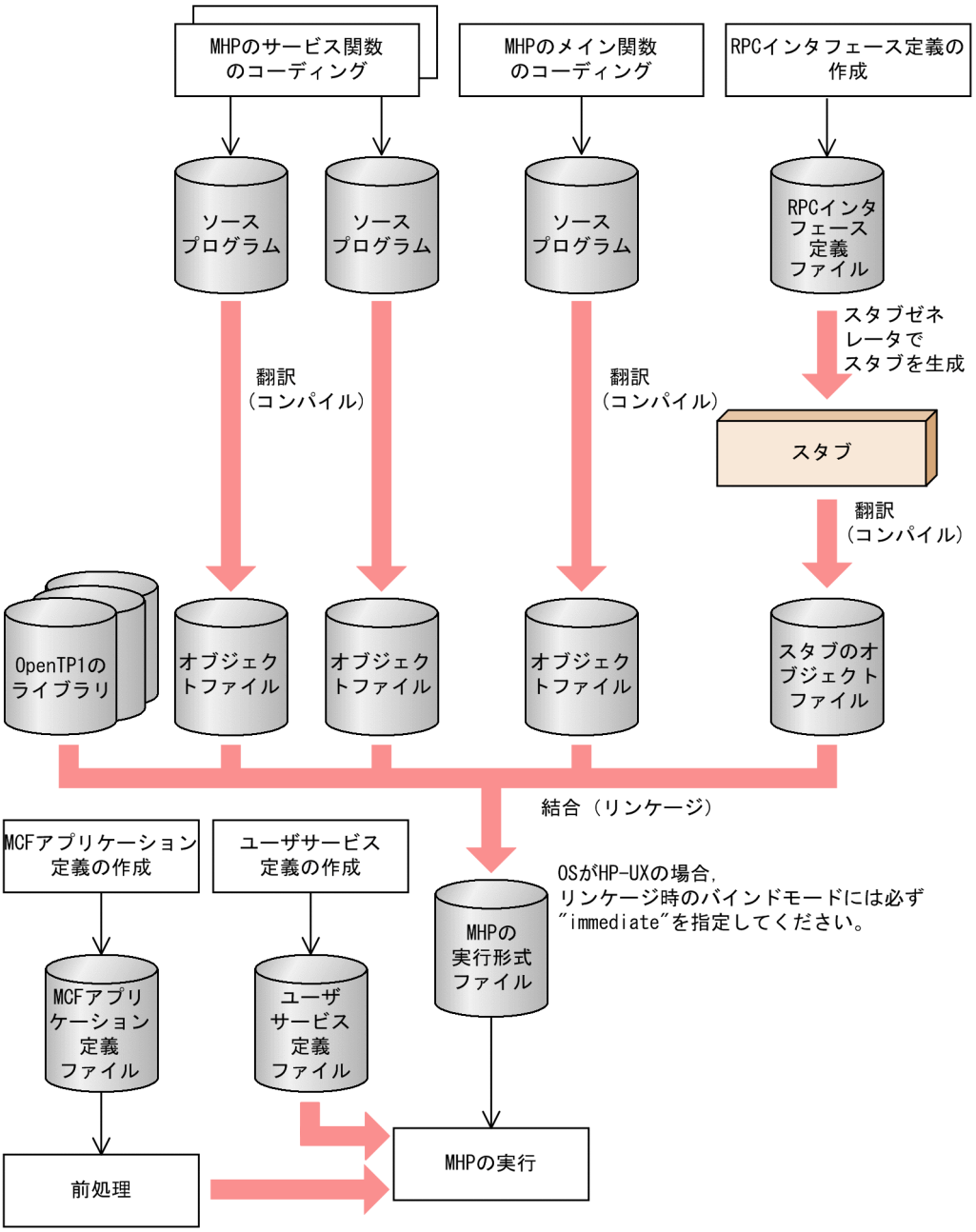
(3)　MHP の作成手順

MHP の作成手順は，「スタブを使用する MHP」か「サービス関数動的ローディング機能を使用する MHP」かで異なります。

(a)　MHP の作成手順（スタブ使用時）

スタブを使用する MHP の作成手順を，次の図に示します。

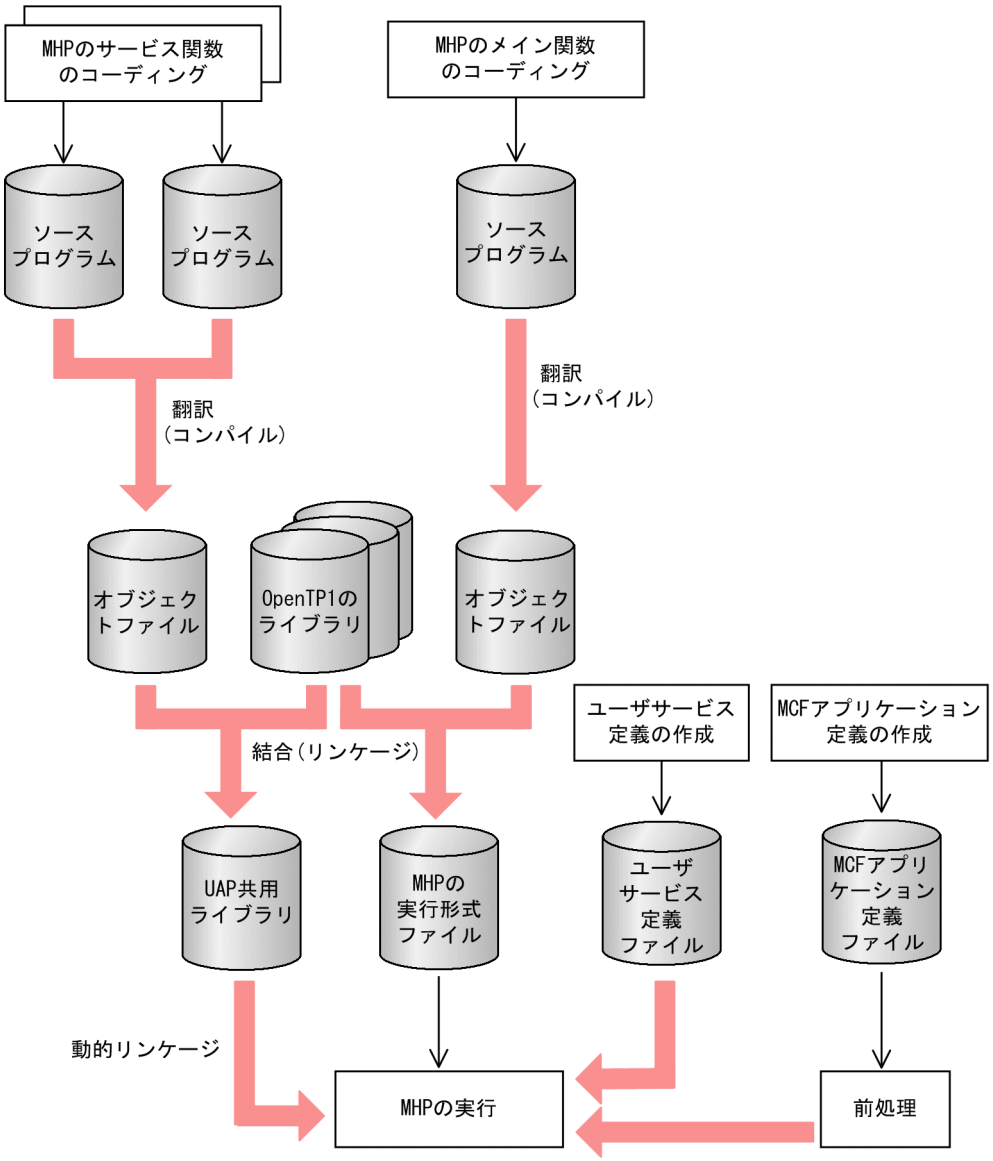
図 1-4　MHP の作成手順（スタブ使用時）



(b)　MHP の作成手順（サービス関数動的ローディング機能使用時）

サービス関数動的ローディング機能を使用する MHP の作成手順を，次の図に示します。

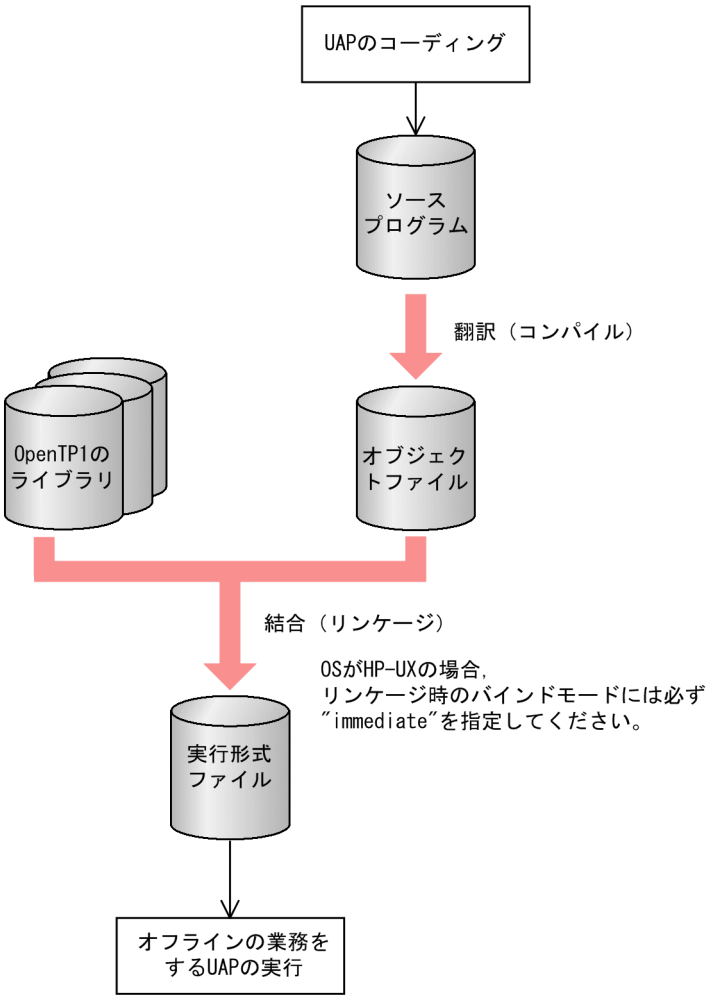
図 1-5　MHP の作成手順（サービス関数動的ローディング機能使用時）



(4)　オフラインの業務をする UAP の作成手順

オフラインの業務をする UAP の作成手順を次の図に示します。

図 1-6　オフラインの業務をする UAP の作成手順



1.2.2　スタブの作成方法

OpenTP1 で使う UAP には，UAP 間のサービス要求を実現するためのライブラリが必要となります。このライブラリを**スタブ**といいます。

ここでは，OpenTP1 のリモートプロシジャコール（dc_rpc_call 関数）を使う UAP（**SUP**，**SPP**）と，**MHP** のスタブについて説明します。XATMI インタフェースを使った通信をする場合のスタブの作成方法については，「1.3　XATMI インタフェースを使うアプリケーションプログラムの作成（TCP/IP 通信，OSI TP 通信）」を参照してください。

(1)　スタブが必要となるアプリケーションプログラム

OpenTP1 で使う UAP のうち，**サービス関数を持つ UAP**（SPP，MHP）には，スタブが必要です。ただし，すべてのサービス関数を **UAP 共用ライブラリ化**してサービス関数動的ローディング機能を使う場合は，スタブは不要です。UAP 共用ライブラリ化とは，UAP のソースファイルを翻訳（コンパイル）して作成した UAP オブジェクトファイルを結合（リンケージ）して，共用ライブラリとしてまとめることです。

また，SUP とオフラインの業務をする UAP は，サービス関数がないので，作成する必要はありません。

(2)　スタブの作成手順

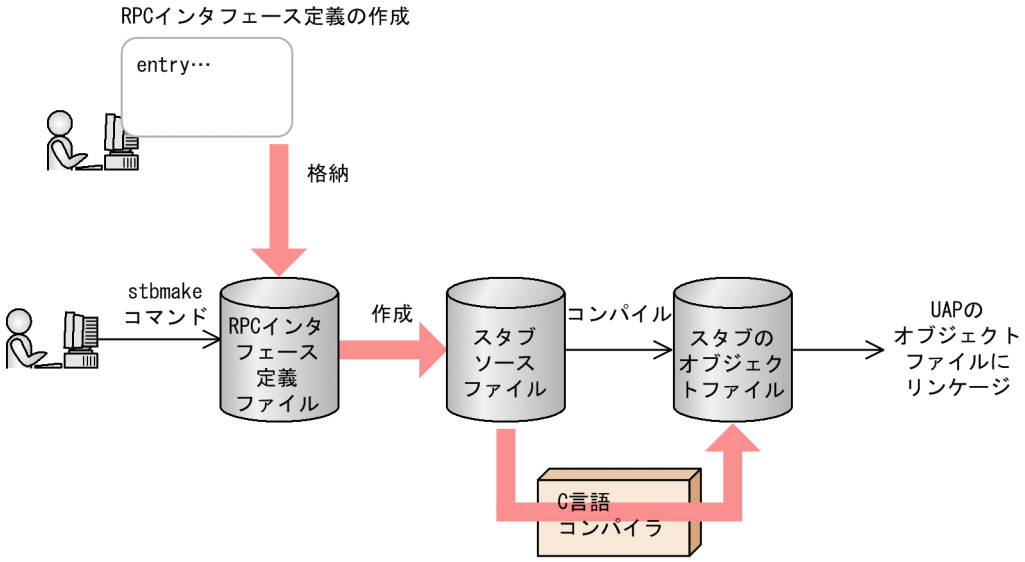
スタブを作成するときは，まず，UAP のサービス関数を定義したファイル（**RPC インタフェース定義ファイル**）を作成します。そのファイルを引数にして **stbmake コマンド**を実行します。

stbmake コマンドを実行すると，**スタブのソースファイル**（C 言語のソースファイル）が作成されます。このファイルを C 言語のコンパイラで翻訳して，UAP のオブジェクトファイルに結合させます。

スタブの内容を変更するときは，UAP を作成する一連の作業をやり直します。RPC インタフェース定義ファイルの内容を変更して，スタブを作り直してから，コンパイルし直した UAP のオブジェクトファイルに結合させてください。

スタブの作成手順を次の図に示します。

図 1-7　スタブの作成手順



(3)　RPC インタフェース定義ファイルの作成

スタブを作成するには，SPP および MHP のサービスのエントリポイント（入り口点）を定義したファイルを作成します。定義内容を **RPC インタフェース定義**といい，定義を格納するファイルを **RPC インタフェース定義**ファイルといいます。

RPC インタフェース定義ファイルは，SPP または MHP の一つの実行形式ファイルごとに作成します。

(a)　RPC インタフェース定義の形式

RPC インタフェース定義は，次のように記述します。

形式

```
entry ”エントリポイント名”〔”エントリポイント名”…〕;
```

機能

SPP および MHP のサービス関数の**エントリポイント名（入り口点）**を指定します。エントリポイント名として，**C 言語の関数名**を指定します。

エントリポイントは，20 文字以内としてください。

エントリポイント名とサービス名の対応は，ユーザサービス定義で指定している名称と合わせてください。

RPC インタフェース定義に注釈文を記述するときは，"/*"で始めて，"*/"で終わらせてください。注釈文のネストはできません。また，注釈文はキーワードや識別子など，文字列の中には記述できません。

entry の文は，一つのファイルに複数行にわたって定義できます。RPC インタフェース定義の作成例を次に示します。

使用例

エントリポイント名が「sv01」と「sv02」の関数がある UAP の RPC インタフェースの指定（次に示すどちらかの形式で指定します）

```
形式1
 entry ”sv01” ;
 entry ”sv02” ;

形式2
 entry ”sv01” ”sv02” ;
```

(4) RPC インタフェース定義ファイルの名称

ファイル名には，RPC インタフェース定義ファイルを示すサフィックス".def"を必ず付けてください。RPC インタフェース定義ファイルを格納するディレクトリは，stbmake コマンドが探せるパスであれば，特に制限はありません。

RPC インタフェース定義ファイルのファイル名の長さは，最大 255 文字です。ただし，OS の制限で 255 文字まで指定できないことがあります。

stbmake コマンドを実行したあと，スタブのソースファイルは RPC インタフェース定義ファイルとは別の名称で作成されます。そのため，OpenTP1 の稼働中には RPC インタフェース定義ファイルは使えません。

1.2.3 スタブのソースファイルの作成

スタブのソースファイルを作成するときは，RPC インタフェース定義ファイル名を引数に設定した，**stbmake コマンド**を実行します。

(1) stbmake コマンドで作成されるファイル

stbmake コマンドを実行すると，次のファイルが作成されます（**xxxxx** は，RPC インタフェース定義ファイルのサフィックス".def"を除いた名称です）。

- スタブのソースファイル（ファイル名：xxxxx_sstb.c）

  ソースファイルのディレクトリ名，およびファイル名は，コマンドのオプションで変更できます。

  ソースファイルのファイル名の長さは，最大 255 文字です。ただし，OS の制限で 255 文字まで指定できないことがあります。スタブのソースファイルは，C 言語のコンパイラで翻訳して，UAP のオブジェクトファイルに結合させます。

1.2.4 stbmake（スタブのソースファイルの作成）

(1) 形式

```
stbmake 〔 -s 〔 スタブソースファイル名 〕 〕 定義ファイル名
```

(2) 機能

RPC インタフェース定義ファイルから，スタブのソースファイルを作成します。

OpenTP1 のリモートプロシジャコールと XATMI インタフェースの両方を使う UAP を作成する場合は，「付録 A OpenTP1 のリモートプロシジャコールと XATMI インタフェースの関数を併用する場合」の stbmake コマンドについての記述を参照してください。

(3) オプション

- -s スタブソースファイル名 ～〈パス名〉

  作成するスタブのソースファイル名を，パス名で指定します。

  フラグ引数を省略した場合は，スタブのソースファイル名は，RPC インタフェース定義ファイルのサフィックス".def"が"_sstb.c"に置き換わった名称でカレントディレクトリに作成されます。

  指定したソースファイル名がすでにある場合は，上書きされて元のファイルの内容はなくなります。

(4) コマンド引数

- 定義ファイル名 ～〈パス名〉

  RPC インタフェース定義ファイルの名称を，パス名で指定します。

(5) 注意事項

stbmake コマンドで入出力できるファイル名の長さは，最大 255 文字です。ただし，OS の制限で 255 文字まで指定できないことがあります。

(6) 使用例

stbmake コマンドの使用例を次に示します。

（例）

カレントディレクトリの RPC インタフェース定義ファイル"test.def"から，スタブのソースファイルを作成する場合

形式 1

```
stbmake  test.def
```

カレントディレクトリの RPC インタフェース定義ファイル"test.def"から，スタブのソースファイル"test_sstb.c"が作成されます。

形式 2

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

カレントディレクトリの下にディレクトリ"stub"が作成されて，その下にスタブのソースファイル"test.c"が作成されます。

1.2.5　アプリケーションプログラムの翻訳と結合

UAP の翻訳と結合方法については，使用する OS のリファレンスマニュアルを参照してください。

- **UAP 作成時の注意**

  UAP を作成するときは，OpenTP1 のバージョンに気を付けてください。システムサービスによっては，古いバージョンの UAP からの関数呼び出しを受け付けないことがあります。旧バージョンで作成した UAP を使用する場合，現在使用しているバージョンの OpenTP1 で，コンパイル／リンケージし直すことをお勧めします。

(1)　翻訳（コンパイル）

C 言語でコーディングした UAP のオブジェクトファイルを作成するには，ソースプログラムを C コンパイラで翻訳します。スタブのソースプログラムを翻訳するときも同様です。

(2)　結合（リンケージ）

ここでの説明中の※1～※3 の意味を次に示します。

注※1

リソースマネジャに XA インタフェースアクセスするトランザクションを実行する場合に必要です（OpenTP1 で提供するリソースマネジャは，すべて XA インタフェースでアクセスします）。トランザクション制御用オブジェクトファイルは，OpenTP1 のコマンド（trnmkobj コマンド）で作成します。trnmkobj コマンドについてはマニュアル「OpenTP1 運用と操作」を参照してください。

注※2

リソースマネジャにアクセスする場合に必要です。OpenTP1 で提供するリソースマネジャのオブジェクトファイルを結合するときは，リンケージのコマンドに次の引数を指定します。

メッセージ送受信機能を使う場合：-lmcf，-lmnet

DAM アクセス機能を使う場合：-ldam

TAM アクセス機能を使う場合：-ltam

ISAM 機能を使う場合：-lismb，-lisam，-lrsort

メッセージキューイング機能を使う場合：-lmqa

他社のリソースマネジャのオブジェクトファイルの結合方法については，使用する他社リソースマネジャのリファレンスマニュアルを参照してください。

注※3

dc_uto_test_status 関数（ユーザサーバのテスト状態の報告）を使う場合に必要です。オンラインテスタのオブジェクトファイルを結合するときは，リンケージのコマンドに次の引数を指定します。

ユーザサーバのテスト状態を報告する場合：-luto

(a) SPP，MHP に結合させるファイル

SPP または MHP の実行形式ファイルは，次に示すファイルを結合させて作成します。

- UAP のオブジェクトファイル（メイン関数とサービス関数）
- スタブのオブジェクトファイル
- トランザクション制御用オブジェクトファイル※1
- リソースマネジャで提供するオブジェクトファイル※2
- オンラインテスタで提供するオブジェクトファイル※3
- OpenTP1 のライブラリ

(b) SUP に結合させるファイル

SUP の実行形式ファイルは，次に示すファイルを結合させて作成します。

- UAP のオブジェクトファイル（メイン関数）
- トランザクション制御用オブジェクトファイル※1
- リソースマネジャで提供するオブジェクトファイル※2
- オンラインテスタで提供するオブジェクトファイル※3
- OpenTP1 のライブラリ

(c) オフラインの業務をする UAP に結合させるファイル

オフラインの業務をする UAP の実行形式ファイルは，次に示すファイルを結合させて作成します。

- UAP のオブジェクトファイル（メイン関数）
- OpenTP1 のライブラリ

(d) サービス関数動的ローディング機能を使用する SPP，MHP に結合させるファイル

サービス関数動的ローディング機能を使用する SPP，MHP の実行形式ファイルは，次に示すファイルを結合させて作成します。

- UAP のオブジェクトファイル（メイン関数）
- OpenTP1 のライブラリ
- トランザクション制御用オブジェクトファイル※1
- リソースマネジャで提供するオブジェクトファイル※2
- オンラインテスタで提供するオブジェクトファイル※3

また，サービス関数動的ローディング機能とスタブでのサービス検索を併用する場合は，次のファイルが必要です。

- UAP のオブジェクトファイル（サービス関数）
- スタブのオブジェクトファイル

(3) 注意事項

OS が HP-UX の場合，リンケージ時のバインドモードには必ず"immediate"を指定してください。"immediate"以外のバインドモードで作成した実行形式ファイルを OpenTP1 の UAP として使った場合，システムの動作は保証しません。作成した UAP のバインドモードが"immediate"かどうかは，OS の chatr コマンドで確認してください。

1.3　XATMI インタフェースを使うアプリケーションプログラムの作成（TCP/IP 通信，OSI TP 通信）

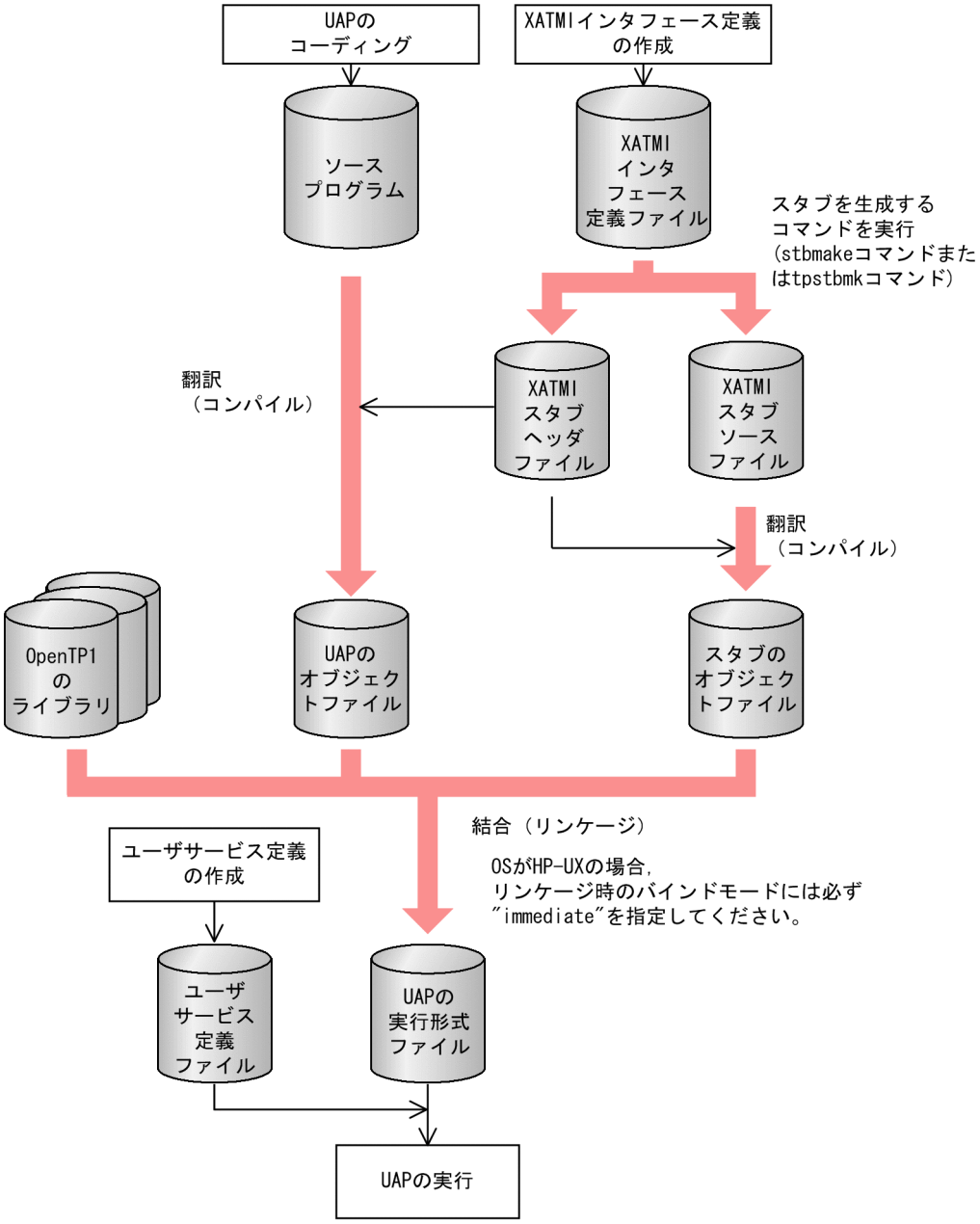
通信プロトコルに TCP/IP および OSI TP を使う場合の，XATMI インタフェースを使う UAP の作成方法について説明します。

OpenTP1 の RPC を使う UAP の作成方法とは，スタブの作成方法（stbmake コマンド，tpstbmk コマンドの実行形式）と UAP にリンケージするファイルが異なります。そのほかの手順は，OpenTP1 の UAP と同じです。UAP の作成手順については，「1.1　アプリケーションプログラムのコーディング」および「1.4　アプリケーションプログラムの実行」を参照してください。

1.3.1　アプリケーションプログラムの作成手順

XATMI インタフェースを使った UAP の作成手順を次の図に示します。

図 1-8　アプリケーションプログラムの作成手順（XATMI インタフェース TCP/IP 通信，OSI TP 通信）



1.3.2　XATMI インタフェース用スタブの作成方法

XATMI インタフェース用のスタブの作成方法について説明します。XATMI インタフェースの通信をする UAP の場合は，**クライアント UAP とサーバ UAP の両方に**，スタブが必要です。

スタブを作成するときは，XATMI インタフェース定義を格納したファイル（**XATMI インタフェース定義ファイル**）を作成して，スタブを生成するコマンドを実行します。スタブを生成するコマンドを，次に示します。

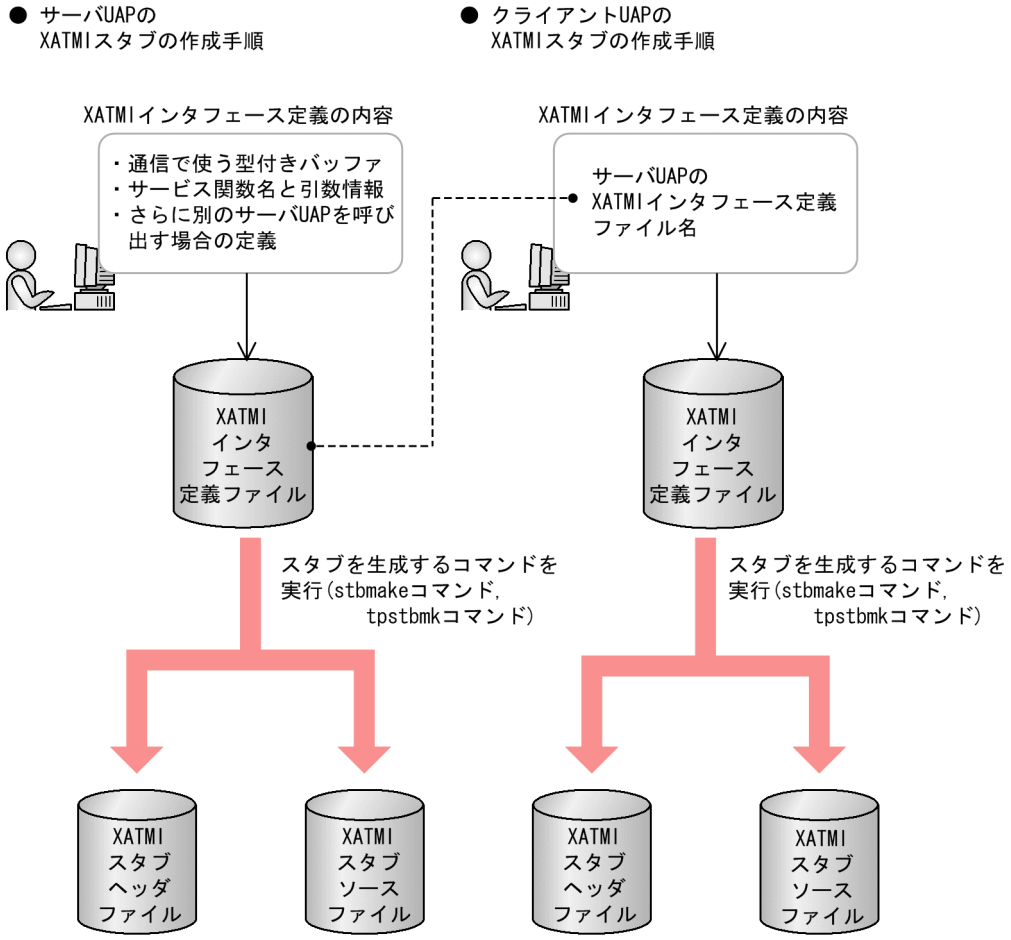
- TCP/IP 通信をする UAP の場合：**stbmake コマンド**

- OSI TP 通信をする UAP の場合：tpstbmk コマンド

作成したスタブのソースファイルは，C 言語のコンパイラで翻訳して，UAP のオブジェクトファイルに結合させます。

XATMI インタフェース用スタブの作成手順を次の図に示します。

図 1-9　XATMI インタフェース用スタブの作成手順（TCP/IP 通信，OSI TP 通信）



(1)　XATMI インタフェース定義（クライアント UAP 用）

クライアント UAP（SUP，または SPP）用の XATMI インタフェース定義の形式について説明します。

形式

```
called_servers = { ”サーバの定義ファイル名”〔,”サーバの定義ファイル名”〕…};
```

機能

サーバ UAP の XATMI インタフェース定義ファイル名を，すべて指定します。サーバ UAP の XATMI インタフェース定義ファイル名を指定することで，サーバ UAP で定義されている型付きバッファは，クライアント UAP でも使えるようになります。

パラメタ

- サーバの定義ファイル名

  サーバ UAP の XATMI インタフェース定義ファイルのファイル名を指定します。定義ファイル名は，サフィックスが".def"のファイルです。

  一つの called_servers 文の括弧 { } の中に，複数の定義ファイル名を指定できます。また，一つの XATMI インタフェース定義ファイルに複数の called_servers 文を記述することもできます。

指定例

サーバ UAP1 とサーバ UAP2 と XATMI インタフェースの通信をするクライアント UAP の定義（サーバ UAP1 の定義ファイル名を serv1.def，サーバ UAP2 の定義ファイル名を serv2.def とします）。

形式 1

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

形式 2

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

(2) XATMI インタフェース定義（サーバ UAP 用）

サーバ UAP の XATMI インタフェース定義に指定する項目を次に示します。指定は順不同です。

- 通信で使う型付きバッファの定義
- サービス関数名と引数情報の定義
- called_servers 文（サーバ UAP から，さらに別のサーバ UAP を呼び出す場合）

(a) 通信で使う型付きバッファの定義

形式

```
タイプ名　サブタイプ名{
            データ型　データ名；
          〔データ型　データ名；〕
                      :
                      :
           }；
```

機能

サーバ UAP で使う型付きバッファのタイプ名，サブタイプ名，および構造体を定義します。サーバ UAP から ほかのサーバ UAP プロセスのサービスを呼び出す場合，呼び出すプロセスで使える型付きバッファはすべて自プロセスでも使えます。そのため，ここでは自プロセス内のサービス関数が入出力として使う型付きバッファだけを定義します。

X_OCTET は定義しなくても常に認識されます。X_OCTET を定義した場合，スタブを生成するコマンド（stbmake コマンドまたは tpstbmk コマンド）の実行時にエラーとなります。

パラメタ

- タイプ名

  サーバ UAP で使う型付きバッファの，タイプ名を指定します。

- サブタイプ名

  サーバ UAP で使う型付きバッファの，サブタイプ名を指定します。

- データ型

  サーバ UAP で使う型付きバッファの構造体にあるデータの，データ型を指定します。

- データ名

  サーバ UAP で使う型付きバッファの構造体にあるデータの，データ名を指定します。

タイプで使えるデータ型の一覧

タイプで使えるデータ型の一覧を次の表に示します。識別子とは XATMI インタフェース定義に記述するデータ型を示し，C 言語のデータ型とは実際にスタブに定義される型付きバッファを示します。OpenTP1 以外のシステムと通信するためにデータ型を変換する場合は，変換する識別子を XATMI インタフェース定義に指定します。

OpenTP1 では，int を 4 バイトとしているので，定義ファイル内ではそれを明示するため「int4」と記述します。

表 1-6　タイプで使えるデータ型の一覧

タイプ	識別子	C 言語のデータ型	通信プロトコル		備考
			TCP/IP	OSI TP	
X_OCTET	―※1	―※1	○	○	なし
X_COMMON	short a	short a	○	○	なし
	short a[n]	short a[n]	○	○	なし
	long a	DCLONG a	○	○	なし
	long a[n]	DCLONG a[n]	○	○	なし
	char a※2	char a	○	○	無変換配列
	octet a	char a	○	○	無変換配列
	tchar a	char a	－	○	変換配列
	char a[n]※2	char a[n]	○	○	無変換配列
	octet a[n]	char a[n]	○	○	無変換配列
	tchar a[n]	char a[n]	－	○	変換配列
X_C_TYPE	short a	short a	○	×	なし
	short a[n]	short a[n]	○	×	なし
	long a	DCLONG a	○	×	なし

タイプ	識別子	C 言語のデータ型	通信プロトコル		備考
			TCP/IP	OSI TP	
X_C_TYPE	long a[n]	DCLONG a[n]	○	×	なし
	int4 a	DCLONG a	○	×	なし
	int4 a[n]	DCLONG a[n]	○	×	なし
	char a※2	char a	○	×	なし
	octet a	char a	○	×	なし
	tchar a	char a	○	×	なし
	char a[n]※2	char a[n]	○	×	なし
	octet a[n]	char a[n]	○	×	なし
	tchar a[n]	char a[n]	○	×	なし
	float a	float a	○	×	なし
	float a[n]	float a[n]	○	×	なし
	double a	double a	○	×	なし
	double a[n]	double a[n]	○	×	なし
	octet a[n][n]	char a[n][n]	○	×	なし
	tchar a[n][n]	char a[n][n]	○	×	なし
	str a[n]	char a[n]	○	×	なし
	str a[n][n]	char a[n][n]	○	×	なし
	tstr a[n]	char a[n]	○	×	なし
	tstr a[n][n]	char a[n][n]	○	×	なし

（凡例）

○：該当する通信プロトコルで使えます。

×：該当する通信プロトコルでは使えません。

－：変換の識別子でも，無変換としてそのまま処理されます。

注※1

X_OCTET は，定義しなくても自動的に認識されます。XATMI インタフェース定義に X_OCTET を指定した場合は，スタブを生成するコマンドを実行したときにエラーになります。

注※2

この識別子も使えますが，新規で作成する場合は次に示す識別子を使うことをお勧めします。

X_COMMON の場合：octet または tchar

X_C_TYPE の場合：str または tstr

指定例

```
X_C_TYPE subtype1 {
                char  name[8];
```

```
int4  data[10];
int4  flags;
};
```

(b) サービス関数名と引数情報の定義

形式

```
service　サービス関数名{(タイプ名 〔サブタイプ名〕) | (ALL) | ( 〔void〕 )};
```

機能

サーバ UAP のサービス関数の関数名と，引数として渡される型付きバッファのタイプ名とサブタイプ名を指定します。引数とは，サービス関数の実際の引数である svc_info 構造体の data メンバのことです。

X_OCTET の場合はサブタイプ名がないので，タイプ名だけを指定します。また，サービス関数内で svc_info 構造体の data メンバを一度も参照しない処理の場合は，サービス関数名のあとに何も指定しないか，void を指定します。

tpcall( )，tpacall( )，tpconnect( )は型付きバッファを送信しないで，サービス関数を呼び出せます。ただし，サービス関数で svc_info 構造体のメンバが指すデータを明示的に参照したくない場合は，引数に何も設定しないか，void を設定します。このように指定したサービス関数を呼び出すには，クライアント側の tpcall( )，tpacall( )，tpconnect( )が送信する型付きバッファへのポインタに NULL を設定してください。ただし，X_OCTET の場合は，ポインタが NULL でなくても，送信データの長さが 0 の場合でも，サービスを要求できます。

引数として受け取る型付きバッファを限定しない指定をする場合は，引数に ALL を指定します。ALL を引数に指定して定義したサービス関数は，自プロセスで認識できる型付きバッファであれば，どの型でも受信できます。

パラメタ

- **サービス関数名**

  サーバ UAP にある，関数名を指定します。

- **タイプ名**

  関数のデータ領域に指定した，タイプ名を指定します。

- **サブタイプ名**

  関数のデータ領域に指定した，サブタイプ名を指定します。

指定例

例 1

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

例 2（引数のタイプ名が X_OCTET の場合）

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

例 3（引数を受信しないサービス関数の場合）

```
service svc_func3(void);  または  service svc_func3();
```

例 4（引数を限定しないサービス関数の場合）

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

(c)　サーバ UAP から，さらに別のサーバ UAP を呼び出す場合

クライアント UAP の XATMI インタフェース定義（called_servers 文）を指定します。

(3)　XATMI インタフェース定義ファイルの名称

ファイル名には，XATMI インタフェース定義ファイルを示すサフィックス".def"を必ず付けてください。XATMI インタフェース定義ファイルを格納するディレクトリは，スタブを生成するコマンド（stbmake コマンドまたは tpstbmk コマンド）が探せるパスであれば，特に制限はありません。

XATMI インタフェース定義ファイルのファイル名の長さは，最大 255 文字です。ただし，OS の制限で 255 文字まで指定できないことがあります。

スタブを生成するコマンド（stbmake コマンドまたは tpstbmk コマンド）を実行したあと，スタブのソースファイルは XATMI インタフェース定義ファイルとは別の名称で作成されます。そのため，OpenTP1 の稼働中には XATMI インタフェース定義ファイルは使われません。

(4)　定義ファイルのインクルード

異なるプロセスで同じ型付きバッファを使うときは，共通の型付きバッファの定義ファイルを一つ作成して，それを各プロセスの定義ファイルにインクルードできます。

インクルードする文は，C 言語と同じ書式です。次のように記述します。

```
  #include <ファイル名>  または  #include "ファイル名"
```

インクルードファイルは，スタブを生成するコマンド（stbmake コマンドまたは tpstbmk コマンド）の-i オプションで指定したサーチパスから読み込まれます。サーチパス内に該当するファイルがない場合は，最後にカレントディレクトリを探します。

インクルードするファイルの名称は任意です（サフィックスが .h でなくてもかまいません）。ただし，そのファイルを XATMI インタフェース定義ファイルとして直接 スタブを生成するコマンド（stbmake コマンドまたは tpstbmk コマンド）に指定する場合は，その定義の名称規則に従ってください。

インクルードするファイルの内容は，XATMI インタフェース定義ファイルと同じです。ただし，名称が重複することもありますので，自プロセス内のサービス関数の定義は含めないことをお勧めします。

(5) 名称の付け方の注意

1. サービス関数名，サブタイプ名は，OpenTP1 で規定する条件に従ってください。
   - "dc"，"DC"，"CBLDC"，"tx"，"TX"，"tp"，"TP"で始まる名称は使えません。
   - サービス関数名は，20 文字以内で指定してください。
   - サブタイプ名の最大長は 32 文字です。そのうち 16 文字までが有効になります。重複があるかどうかは，16 文字までの範囲でチェックしています。
   - 型付きバッファの構造体の中で使うデータのデータ名の最大長は，32 文字です。
2. 同じプロセス内では，サービス関数名が重複しないようにしてください。
3. 同じプロセス内でサブタイプ名が重複した場合は，そのタイプおよび構造が一致するときは許可されます。不一致のときは，スタブを生成するコマンド（stbmake コマンドまたは tpstbmk コマンド）がエラーリターンします。
4. 異なるプロセス内では，サービス関数名やサブタイプ名は同じものを許可します。ただし，サーバとして異なるプロセスにあっても，一つのクライアントから呼ばれる場合，クライアント側からは同じプロセスと見なされます。

1.3.3 XATMI インタフェース用スタブのソースファイルの作成

作成した XATMI インタフェース定義ファイルから，**XATMI 用スタブ**を作成します。

スタブを作成するときは，XATMI インタフェース定義を格納したファイル（**XATMI インタフェース定義ファイル**）を作成して，スタブを生成するコマンドを実行します。スタブを生成するコマンドを，次に示します。

- TCP/IP 通信をする UAP の場合：**stbmake コマンド**
- OSI TP 通信をする UAP の場合：**tpstbmk コマンド**

クライアント UAP とサーバ UAP のそれぞれに，次に示す方法でスタブを作成してください。

(1) stbmake コマンドまたは tpstbmk コマンドで作成されるファイル

コマンドを実行すると，次の三つのファイルが作成されます（XXXXX は，XATMI インタフェース定義ファイルのサフィックス".def"を除いた名称です）。

- **XATMI スタブソースファイル（デフォルトのファイル名:XXXXX_stbx.c）**
- **XATMI スタブヘッダファイル（デフォルトのファイル名:XXXXX_stbx.h）**
- **XATMI スタブコピーファイル（サブタイプ名に".cbl"が付いた名称）**

ファイル名の長さは，最大 255 文字です。ただし，OS の制限で 255 文字まで指定できないことがあります。

ファイルを作成するディレクトリ，およびファイル名は，コマンドのオプションで変更できます。

(a) XATMI スタブソースファイル

作成された XATMI スタブソースファイルは，C 言語のコンパイラで翻訳して，UAP のオブジェクトファイルに結合させます。

(b) XATMI スタブヘッダファイル

作成された XATMI スタブヘッダファイルは，UAP のソースファイルおよび XATMI スタブソースファイルにインクルードされます。

(c) XATMI スタブコピーファイル

C 言語で作成した UAP では使いません。COBOL 言語で作成した UAP で使います。

1.3.4 stbmake（XATMI インタフェース用スタブの作成 TCP/IP 通信）

(1) 形式

```
stbmake 〔-x〕〔-b〕〔-S スタブソースファイル名〕
        〔-H スタブヘッダファイル名〕
        〔-i インクルードファイルのサーチパス名〕
        〔-m サーバの定義ファイルのサーチパス名〕〔-p〕 定義ファイル名
```

(2) 機能

XATMI インタフェースの通信を TCP/IP 通信で使う場合に必要な，XATMI 用スタブのソースファイルを作成します。stbmake コマンドは，XATMI インタフェース定義ファイルを基に，次に示すファイルを出力します。

- XATMI スタブソースファイル
- XATMI スタブヘッダファイル（C 言語で作成した UAP で使います）
- XATMI スタブコピーファイル（COBOL 言語で作成した UAP で使います）

XATMI インタフェースと OpenTP1 のリモートプロシジャコールの両方を使う UAP を作成する場合は，「付録 A OpenTP1 のリモートプロシジャコールと XATMI インタフェースの関数を併用する場合」の stbmake コマンドについての説明を参照してください。

(3) オプション

- -x

  XATMI インタフェースを使う UAP のスタブを作成することを示します。-x オプションは，省略できます。

- -b

  C 言語の UAP で使う XATMI 用スタブを作成する場合には，-b オプションは省略してください。COBOL 言語で作成した UAP で使う，XATMI スタブコピーファイルを作成するときに，-b オプションを指定します。

- -S スタブソースファイル名 ～〈パス名〉

  作成する XATMI スタブソースファイルのファイル名を指定します。ファイル名には，相対パス名，絶対パス名を使えます。

  -S オプションを省略すると，カレントディレクトリに XXXXX_stbx.c という名称で XATMI スタブソースファイルが作成されます。

- -H スタブヘッダファイル名　～〈パス名〉

  作成する XATMI スタブヘッダファイルのファイル名を指定します。ファイル名には，相対パス名，絶対パス名を使えます。

  -H オプションを省略すると，カレントディレクトリに XXXXX_stbx.h という名称で XATMI スタブヘッダファイルが作成されます。

- -i インクルードファイルのサーチパス名　～〈パス名〉

  XATMI インタフェース定義ファイルの＃include 文に指定したインクルードファイル名を，サーチパスで指定します。-i オプションで指定したディレクトリから，インクルードファイルを探します。

  -i オプションを省略した場合は，コマンドを実行したカレントディレクトリから探します。

  -i オプションを指定できるのは，1 回だけです。複数のサーチパスを指定したい場合は，複数のパスをコロン（:）で区切って指定します。複数のサーチパスを指定した場合は，-i オプションの引数に記述した順番で，パスが検索されます。

  サーチパスを指定するときは，英数字，アンダスコア（_），スラント（/），およびピリオド（.）を使ってください。

- -m サーバの定義ファイルのサーチパス名　～〈パス名〉

  XATMI インタフェース定義ファイルの called_servers 文に指定したサーバの定義ファイル名を，サーチパスで指定します。-m オプションで指定したディレクトリから，インクルードファイルを探します。

  -m オプションを省略した場合は，コマンドを実行したカレントディレクトリから探します。

  サーチパスを指定するときは，英数字，アンダスコア（_），スラント（/），およびピリオド（.）を使ってください。

  -m オプションを指定できるのは，1 回だけです。複数のサーチパスを指定したい場合は，複数のパスをコロン（:）で区切って指定します。複数のサーチパスを指定した場合は，-m オプションの引数に記述した順番で，パスが検索されます。

- -p

  型付きレコードのメモリ内での配置状態を，標準出力に出力する場合に指定します。オンラインテスタを使う場合で，XATMI で使う構造体の各メンバが，メモリ内でどのように配置されているかを知りたいときに，-p オプションを指定します。

-p オプションを指定した場合は，stbmake コマンドはファイルを作成しません。そのため，-S オプションおよび-H オプションに出力ファイルを指定しても無視されます。-m オプションおよび-i オプションはファイルを検索するため，必要に応じて指定しておいてください。

(4) コマンド引数

- **定義ファイル名 ～〈パス名〉**

  XATMI インタフェース定義ファイル名を指定します。このファイル名は，サフィックスが".def"であることが前提です。

(5) 注意事項

- stbmake コマンドのオプションを指定できるのは，すべて 1 回ずつだけです。複数回指定した場合は，最後に指定した値が有効になります。
- stbmake コマンドで入出力できるファイル名の長さは，最大 255 文字です。ただし，OS の制限で 255 文字まで指定できないことがあります。

1.3.5 tpstbmk（XATMI インタフェース用スタブの作成 OSI TP 通信）

(1) 形式

```
tpstbmk 〔-b〕〔-S スタブソースファイル名〕〔-H スタブヘッダファイル名〕
        〔-i インクルードファイルのサーチパス名〕
        〔-m サーバの定義ファイルのサーチパス名〕 定義ファイル名
```

(2) 機能

XATMI インタフェースの通信を OSI TP 通信で使う場合に必要な，XATMI 用スタブのソースファイルを作成します。tpstbmk コマンドは，XATMI インタフェース定義ファイルを基に，次に示すファイルを出力します。

- XATMI スタブソースファイル
- XATMI スタブヘッダファイル（C 言語で作成した UAP で使います）
- XATMI スタブコピーファイル（COBOL 言語で作成した UAP で使います）

XATMI インタフェースと OpenTP1 のリモートプロシジャコールの両方を使う UAP を作成する場合は，「付録 A OpenTP1 のリモートプロシジャコールと XATMI インタフェースの関数を併用する場合」の tpstbmk コマンドについての説明を参照してください。

(3) オプション

- -b

C 言語の UAP で使う XATMI 用スタブを作成する場合には，-b オプションは省略してください。COBOL 言語で作成した UAP で使う，XATMI スタブコピーファイルを作成するときに，-b オプションを指定します。

- **-S スタブソースファイル名　～〈パス名〉**

  作成する XATMI スタブソースファイルのファイル名を指定します。ファイル名には，相対パス名，絶対パス名を使えます。

  -S オプションを省略すると，カレントディレクトリに XXXXX_stbx.c という名称で XATMI スタブソースファイルが作成されます。

- **-H スタブヘッダファイル名　～〈パス名〉**

  作成する XATMI スタブヘッダファイルのファイル名を指定します。ファイル名には，相対パス名，絶対パス名を使えます。

  -H オプションを省略すると，カレントディレクトリに XXXXX_stbx.h という名称で XATMI スタブヘッダファイルが作成されます。

- **-i インクルードファイルのサーチパス名　～〈パス名〉**

  XATMI インタフェース定義ファイルの＃include 文に指定したインクルードファイル名を，サーチパスで指定します。-i オプションで指定したディレクトリから，インクルードファイルを探します。

  -i オプションを省略した場合は，コマンドを実行したカレントディレクトリから探します。

  -i オプションを指定できるのは，1 回だけです。複数のサーチパスを指定したい場合は，複数のパスをコロン（:）で区切って指定します。複数のサーチパスを指定した場合は，-i オプションの引数に記述した順番で，パスが検索されます。

  サーチパスを指定するときは，英数字，アンダスコア（_），スラント（/），およびピリオド（.）を使ってください。

- **-m サーバの定義ファイルのサーチパス名　～〈パス名〉**

  XATMI インタフェース定義ファイルの called_servers 文に指定したサーバの定義ファイル名を，サーチパスで指定します。-m オプションで指定したディレクトリから，インクルードファイルを探します。

  -m オプションを省略した場合は，コマンドを実行したカレントディレクトリから探します。

  サーチパスを指定するときは，英数字，アンダスコア（_），スラント（/），およびピリオド（.）を使ってください。

  -m オプションを指定できるのは，1 回だけです。複数のサーチパスを指定したい場合は，複数のパスをコロン（:）で区切って指定します。複数のサーチパスを指定した場合は，-m オプションの引数に記述した順番で，パスが検索されます。

(4)　コマンド引数

- **定義ファイル名　～〈パス名〉**

  XATMI インタフェース定義ファイル名を指定します。このファイル名は，サフィックスが".def"であることが前提です。

(5) 注意事項

- tpstbmk コマンドのオプションを指定できるのは，すべて 1 回ずつだけです。複数回指定した場合は，最後に指定した値が有効になります。
- tpstbmk コマンドで入出力できるファイル名の長さは，最大 255 文字です。ただし，OS の制限で 255 文字まで指定できないことがあります。

1.4　アプリケーションプログラムの実行

UAP の開始方法と終了方法，および実行環境について説明します。

1.4.1　アプリケーションプログラムの開始と終了

(1)　SUP の開始と終了

(a)　開 始

SUP は，次の場合に開始します。

- ユーザサービス構成定義に SUP のサーバ名を指定した場合は，OpenTP1 を開始したとき
- ユーザサービス構成定義に SUP のサーバ名を指定していない場合は，dcsvstart コマンドを実行したとき

SUP から SPP へサービスを要求する場合は，サービスがある SPP が開始していることが前提です。SUP からサービスを要求する先の SPP は，SUP よりも先に開始させておいてください。

(b)　終 了

開始した SUP を OpenTP1 から正常に終了させることはできません。OpenTP1 を正常終了させるコマンドを実行しても，システム内のすべての SUP が終了するまで，OpenTP1 は終了しません。SUP をコーディングするときに，SUP 自身で終了するように作成しておいてください。SUP の処理がうまくいかなかったため 異常終了させたい場合は，exit()または abort()を使って，SUP 自身で終了するように作成してください。

SUP は，dcsvstop コマンドで正常終了させることはできません。ただし，SUP を強制停止させたい場合に限り，dcsvstop -f コマンドで終了できます。

SUP のプロセスを，kill コマンドで終了させないでください。

(2)　SPP，MHP の開始と終了

(a)　開 始

SPP，MHP は，ユーザサーバ（サービスグループ）単位で開始します。SPP，MHP は，次に示す場合に開始します。

- ユーザサービス構成定義に SPP，MHP のサーバ名を指定した場合は，OpenTP1 を開始したとき
- ユーザサービス構成定義に SPP，MHP のサーバ名を指定していない場合は，dcsvstart コマンドを実行したとき

マルチサーバ機能を使っている場合，常駐プロセスとして指定した数だけ，ユーザサーバプロセスが確保されます。サービス要求が増えると，非常駐プロセスも開始します。

(b) 終了

SPP または MHP が終了するのは次の場合です。

- 次に示す OpenTP1 の終了コマンドを実行したために，終了処理に入ったとき
  dcstop コマンド（正常終了）
  dcstop -n（強制正常終了）
  dcstop -a（計画停止 A）
  dcstop -b（計画停止 B）
  dcstop -f（強制停止）
- オンラインで稼働中のプロセスを，次に示すサーバの終了コマンドを実行したために，終了処理に入ったとき
  dcsvstop コマンド（正常終了）
  dcsvstop -f（強制停止）
- オンライン稼働中のプロセスが，ユーザサービス定義の最大プロセス数の指定を超えたために，OpenTP1 からプロセスを終了させられるとき
- SPP または MHP を非常駐プロセスで実行しているときには，サービスの処理を終了したとき
- SPP または MHP をマルチサーバで負荷分散しているときは，該当するサービスグループへのサービス要求が減少したとき

SPP，または MHP のプロセスを，kill コマンドで終了させないでください。

(3) オフラインの業務をする UAP の開始と終了

開始方法は任意です。終了もシェルからプロセスを終了させます。オフラインの業務をする UAP はユーザの責任で開始／終了を管理してください。

1.4.2 OpenTP1 で開始したアプリケーションプログラムの動作環境

- SUP，SPP，MHP の標準入力（stdin），標準出力（stdout），および標準エラー出力（stderr）は，OpenTP1 によってリダイレクトされます。
- UAP を開始すると，「$DCDIR/tmp/home/**ユーザサーバ名.XX**」（XX は通し番号を示します）というディレクトリが作成されます。このディレクトリをカレントワーキングディレクトリとして，UAP が稼働します。
  このディレクトリはシステム共通定義の prc_current_work_path オペランドを設定することによって変更できます。
- ユーザ ID（UID）とグループ ID（GID）は，ユーザサーバの環境設定時に指定した値になります。

- ルートディレクトリは「／」のままです。
- 次に示すファイル記述子が，UAP 実行時にはオープンされています。
  ファイル記述子 0：標準入力ファイル記述子
  ファイル記述子 1：標準出力ファイル記述子
  ファイル記述子 2：標準エラー出力ファイル記述子
- umask は 000 です。
- 制御端末はありません。
- OpenTP1 では，UAP プロセスを生成するときに自動的に UAP のシグナルを設定します。OpenTP1 で設定する UAP のシグナルの一覧を次の表に示します。

表 1-7　OpenTP1 で設定する UAP のシグナル

シグナル名	UAP プロセスの生成時の設定	動作
SIGHUP	SIG_DFL（デフォルト）	exit
SIGINT	SIG_IGN（無視）	ignore
SIGQUIT	SIG_DFL（デフォルト）	core
SIGILL	SIG_DFL（デフォルト）	core
SIGTRAP	SIG_IGN（無視）	ignore
SIGIOT※	SIG_DFL（デフォルト）	core
SIGABRT※	SIG_DFL（デフォルト）	core
SIGEMT	SIG_DFL（デフォルト）	core
SIGFEP	SIG_DFL（デフォルト）	core
SIGKILL	－	exit
SIGBUS	SIG_DFL（デフォルト）	core
SIGSEGV	SIG_DFL（デフォルト）	core
SIGSYS	SIG_DFL（デフォルト）	core
SIGPIPE※	SIG_IGN（無視）	ignore
SIGALRM	SIG_IGN（無視）	ignore
SIGTERM	SIG_DFL（デフォルト）	exit
SIGUSR1	SIG_IGN（無視）	ignore
SIGUSR2	SIG_IGN（無視）	ignore
SIGCLD	SIG_DFL（デフォルト）	ignore

（凡例）
－：該当しません。

注

UAP でシグナルの動作を設定する場合，設定したシグナルハンドラ内で exit()や abort()などを呼び出してプロセスを停止させないでください。シグナルハンドラ内でプロセスを停止させた場合，シグナルに割り込まれたときに，OpenTP1 のクリティカルな処理を実行していると OpenTP1 システムがダウンします。また，シグナルハンドラ内で外部変数 errno の値を書き換えないでください。

注※

再設定できないシグナルを示します。UAP を作成する際には，プログラム内でこれらのシグナルの動作を設定し直さないでください。

1.4.3 アプリケーションプログラムの環境変数

UAP の環境変数は，ユーザサーバの環境を設定するときに，ユーザサーバごとに設定できます。ただし，次に示す環境変数は，OpenTP1 で設定されます。

- DCDIR：OpenTP1 ホームディレクトリ
- DCCONFPATH：OpenTP1 のシステム定義ファイルを格納するディレクトリ
- DCSVNAME：ユーザサーバ名
- DCSVGNAME：サービスグループ名（SPP，MHP の場合だけ参照できます）
- DCUAPCONFPATH：OpenTP1 のユーザサービス定義ファイルまたはユーザサービスデフォルト定義ファイルを格納するディレクトリ（DCCONFPATH とは別のディレクトリに格納したい場合に設定します）

上記のほかにも，"DC"で始まる環境変数は，OpenTP1 が使います。これらの環境変数は参照できますが，変更しないでください。変更した場合，システムの動作は保証しません。

OpenTP1 配下で動作する SUP，SPP，MHP には，telnet などで OpenTP1 管理者でログインした際に設定される環境変数は引き継がれません。これらの環境変数は，ユーザサービス定義で設定し直してください。

2 OpenTP1 のライブラリ関数の文法

この章では，OpenTP1 のライブラリ関数の文法について説明します。

関数の説明形式

OpenTP1 のライブラリ関数の文法を，次の形式で説明します。

形式

OpenTP1 のライブラリ関数の形式と，引数のデータ型を示します。

ANSI C の形式，C++言語でコーディングする場合は「ANSI C，C++の形式」に，ANSI 準拠前の K&R の形式でコーディングする場合は「K&R 版 C の形式」に従ってください。

引数に値を設定するときは，ここで示すデータ型に従ってください。特に断りがないかぎり，引数には固有の名称を任意で付けられます。

機能

ここで説明する関数の機能について説明します。

UAP で値を設定する引数

関数の実行時に，値を指定しておく引数を示します。各引数の説明に従って，値を設定してください。引数に値を設定する場合が限られているときは，その引数の説明に値を設定する場合を【　】で示します。

OpenTP1 から値が返される引数

関数を実行したあとに，OpenTP1 から値が返される引数です。関数の実行後にこの引数の内容を参照してください。引数に OpenTP1 から値が返される場合が限られているときは，その引数の説明に値が返される場合を【　】で示します。

クライアント UAP から値が渡される引数

サービス関数の場合で，クライアント UAP から値が渡される引数です。この引数の内容を参照して，サービス関数の処理をしてください。

サーバ UAP から値が返される引数

同期応答型 RPC，非同期応答型 RPC の場合に，サービス関数から値が返される引数です。dc_rpc_call 関数，dc_rpc_poll_any_replies 関数を呼び出した UAP では，ここに示す引数の値を参照できます。

リターン値

関数を実行したときに戻ってくる値を，表形式で説明します。このリターン値によって，関数が正常に実行されたかどうかがわかります。エラーが起こったときは，エラーの内容を示します。

UAP を作成するときは，互換性を保つために，必ずここで示す定数名でリターン値を使ってください。リターン値の定数名は，ヘッダファイルで定義されています。リターンされる値の情報が必要なときは，ヘッダファイルの定義を参照してください。

指定例

指定例が必要な関数には記述します。

注意事項

関数を使うときの注意について記述します。

メイン関数とサービス関数の作成

OpenTP1 の UAP のメイン関数とサービス関数の文法について説明します。SPP または MHP はメイン関数とサービス関数を，SUP はメイン関数だけを作成します。

- **メイン関数の作成（SUP，SPP，MHP）**
- **サービス関数の作成（SPP）**
- **サービス関数の作成（MHP）**
  サービス関数の作成方法は，使用するオープンシステムの仕様に従ってください。

OpenTP1 の UAP のうち，**TP1/LiNK** で使えるのは **SUP**，**SPP**，または **MHP** です。ただし，TP1/LiNK で MHP を作成するときは，TP1/Messaging が必要です。

メイン関数の作成（SUP，SPP，MHP）

形式

main という関数名を持ちます。その他は，コーディングに使う C 言語の仕様に従って作成します。OpenTP1 では特に制限はありません。メイン関数は，この節の記述に従って任意に作成できます。

機能

UAP のプロセスが開始して，OS から最初に呼ばれる関数です。

SUP のメイン関数の場合

SUP のメイン関数で，必ず呼び出す OpenTP1 の関数を次に示します。

1. dc_rpc_open（アプリケーションプログラムの開始）
2. dc_adm_complete（ユーザサーバの開始処理完了の報告）
3. dc_rpc_close（アプリケーションプログラムの終了　業務終了後）

上記の OpenTP1 の関数以外にも，業務に必要な UAP プロセスの初期化や，終了処理，dc_rpc_call 関数なども使えます。

SPP のメイン関数の場合

SPP から提供するサービスとして作成したサービス関数を，一つの実行形式ファイルにまとめます。一つのメイン関数と複数のサービス関数から構成される実行形式ファイルが，一つのサービスグループに対応します。

SPP のメイン関数で，必ず呼び出す OpenTP1 の関数を次に示します。SPP のサービスで，MCF の関数を使う場合は，dc_mcf_open 関数，dc_mcf_close 関数を呼び出してください。

1. dc_rpc_open（アプリケーションプログラムの開始）
2. dc_rpc_mainloop（SPP のサービス開始）
3. dc_rpc_close（アプリケーションプログラムの終了　業務終了後）

初期化の処理をしてから，このメイン関数自体は dc_rpc_mainloop 関数で止まっています。その間にサービス関数で要求された処理をします。上記の OpenTP1 の関数以外にも，業務に必要な SPP プロセスの初期化や，終了処理，dc_rpc_call 関数などもメイン関数で使えます。

MHP のメイン関数の場合

メッセージを処理するアプリケーションとして作成したサービス関数を，一つの実行形式ファイルにまとめます。一つのメイン関数と複数のサービス関数から構成される実行形式ファイルが，一つのサービスグループに対応します。サービスグループ名はドメイン内（全ネットワーク内）で一意になる名称にしてください。

MHP のメイン関数で，必ず呼び出す OpenTP1 の関数を次に示します。

1. dc_rpc_open（アプリケーションプログラムの開始）
2. dc_mcf_open（MCF 環境のオープン）
3. dc_mcf_mainloop（MHP のサービス開始）
4. dc_mcf_close（MCF 環境のクローズ）
5. dc_rpc_close（アプリケーションプログラムの終了　業務終了後）

アプリケーション名に該当するサービス関数がある MHP が開始され，初期化の処理をしたあと，このメイン関数自体は dc_mcf_mainloop 関数で止まっています。その間にサービス関数で要求された処理をします。上記の OpenTP1 の関数以外にも，業務に必要な MHP プロセスの初期化や，終了処理，dc_rpc_call 関数などもメイン関数で使えます。

引数

dcsvstart コマンドの-a オプションを使用して，SUP のメイン関数に第 1 引数を渡すことができます。dcsvstart コマンドの詳細については，マニュアル「OpenTP1 運用と操作」を参照してください。

サービス関数の作成（SPP）

形式

ANSI C，C++ の形式

```
void  関数名(char *in,DCULONG *in_len,char *out,DCULONG *out_len)
{
サービスの処理
}
```

K&R 版 C の形式

```
void 関数名(in,in_len,out,out_len)
char      *in;
DCULONG   *in_len;
char      *out;
DCULONG   *out_len;
{
サービスの処理
}
```

機能

サービスを実行して結果を返す SPP のサービス関数です。クライアント UAP の dc_rpc_call 関数で呼ばれます。サービス関数は，上記の形式で任意に作成してください。

サービス関数名は，サービス関数のエントリポイント（入り口点）の名称に対応します。この対応づけは，UAP の実行環境を設定するときに指定します。UAP の実行環境を設定する方法を次に示します。

- TP1/Server Base の場合
  ユーザサービス定義で指定します。
- TP1/LiNK の場合
  UAP の環境を設定するコマンドを実行して，対話形式で指定します。

引数の指定

サービス関数には，次の値が引数として渡されます。これはクライアント UAP が dc_rpc_call 関数で指定した値です。

- 入力パラメタ（in）
- 入力パラメタ長（in_len）
- 応答の長さ（out_len）

入力パラメタと入力パラメタの長さはクライアント UAP で設定した値が，文字コードや数字の表現形式などを変換しないでサービス関数にそのまま渡されます。応答の長さには，クライアント UAP で設定した長さが渡されます。

サービス関数では，次の値を引数に設定します。

- サービス関数の応答（out）
- サービス関数の応答の長さ（out_len）
  サービス関数は，out に応答を設定して，その長さを out_len に設定してからリターンしてください。

サービスのクライアント UAP への応答は，サービス関数がトランザクションとして実行したかどうか，またはコミットしたかロールバックしたかには無関係に送信されます。必要であればサービス関数でクライアント UAP にエラーを知らせる応答を作成してください。

クライアント UAP から値が渡される引数

●in

クライアント UAP で設定した，入力パラメタが渡されます。

●in_len

クライアント UAP で設定した，入力パラメタの長さが渡されます。

●out_len

クライアント UAP で設定した，応答の長さが渡されます。

UAP で値を設定する引数

●out

サービス関数からの応答を設定します。サービス関数の処理では，out にサービス関数での処理結果を設定してからリターンしてください。

●out_len

サービス関数からの，実際の応答の長さを設定します。クライアント UAP から渡された out_len の値以下の数値を設定してください。

サービス関数の処理での注意

1. 非応答型 RPC（flags に DCRPC_NOREPLY を指定）の dc_rpc_call 関数（flags に EERPC_NOREPLY を指定）で呼び出したサービス関数では，out と out_len は参照できません。
2. サービス関数を C 言語で記述した場合は，static 変数には前回のサービス要求時の値が残っているので，必要であれば初期化してから使用してください。
3. サービス関数から使えない関数を次に示します。使った場合の UAP の動作は保証しません。
   - dc_rpc_open()
   - dc_rpc_close
   - dc_rpc_mainloop()

- 子プロセスを生成する関数（fork()，system()など）
- プロセスが終了する関数（exit()など）

4. SPP のサービス関数からメッセージ送受信の関数（dc_mcf_～）を呼び出す場合は，メイン関数で dc_mcf_open 関数と dc_mcf_close 関数を呼び出しておいてください。
5. SPP のサービス関数からは，dc_mcf_receive 関数は呼び出せません。
6. in に渡された入力パラメタに対して，in_len に渡された入力パラメタ長の領域を超える操作や参照などはしないでください。操作や参照などをした場合の動作は保証しません。プロセスが異常終了する場合があります。

トランザクションとサービス関数の関係

サービス関数の UAP がトランザクション属性を指定していて，クライアント UAP がトランザクションとして実行している場合，サービス関数はサービスを要求された時点でトランザクションブランチとして実行されます。この場合は，サービス関数で dc_trn_begin 関数を使わないでください。

グローバルトランザクションのサービス全体は，コミットするかロールバックとなるかのどちらかが保証されています。トランザクションブランチとして実行されるサービス関数がリターン（return）することで，そのトランザクションブランチの正常終了を要求したと見なされます。

UAP にトランザクション属性を指定していても，クライアント UAP がトランザクションとして実行されてない場合は，サービス関数はトランザクションとして実行されません。このサービス関数の処理をトランザクションとしたい場合は，サービス関数から任意の時点で dc_trn_begin 関数と dc_trn_unchained_commit 関数を使って，トランザクションの開始と同期点取得をします。

UAP にトランザクション属性なしを指定したときは，サービス関数で dc_trn_begin 関数を呼び出していても，サービス関数はトランザクションとして実行できません。

リターン値

リターン値はありません。return()で指定した値はクライアント UAP には返りません。OpenTP1 でもリターン値を参照しません。リターン値として-1 を指定しても，ロールバックを要求したことにはなりません。

サービス関数の作成（MHP）

形式

ANSI C，C++の形式

```
void 関数名(void)
{
サービスの処理
}
```

K&R 版 C の形式

```
void 関数名()
{
サービスの処理
}
```

機能

サービスを実行して結果を返す MHP のサービス関数です。MCF でメッセージを受信すると，アプリケーション名に該当するサービス関数がある MHP が起動されます。

MHP のサービス関数は，上記の形式で任意に作成してください。サービス関数名はサービス関数のエントリポイント（入り口点）の名称に対応します。この対応づけはサービス関数を実行するプロセスのユーザサービス定義で指定します。

サービス名とアプリケーション名の対応づけは，MCF アプリケーション定義で指定します。

引数

なし。

サービス関数の処理での注意

1. サービス関数から使えない関数を次に示します。使った場合の UAP の動作の保証はしません。
   - dc_rpc_open()
   - dc_rpc_close()
   - dc_mcf_open()
   - dc_mcf_close()
   - dc_rpc_mainloop()
   - dc_mcf_mainloop()
   - 子プロセスを生成する関数（fork()，system()など）
   - プロセスが終了する関数（exit()など）

2. MHP のサービス関数に対して，ほかの UAP から dc_rpc_call 関数でサービスを要求できません。

リターン値

リターン値はありません。リターン値として-1 を指定しても，ロールバックを要求したことにはなりません。

システム運用の管理（dc_adm_～）

UAP から OpenTP1 システムの各種機能を使う関数について説明します。システム運用の管理の関数を次に示します。

- dc_adm_call_command － 運用コマンドの実行
- dc_adm_complete － ユーザサーバの開始処理完了の報告
- dc_adm_status － ユーザサーバの状態の報告

システム運用の管理の関数（dc_adm_～）は，TP1/Server Base と TP1/LiNK のどちらの UAP でも使えます。

dc_adm_call_command

名称

運用コマンドの実行

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

オンライン中に com をコマンド入力したときと同様に，UAP から com を sh(1)に渡します。この時のプロセスは，シェルが処理を完了するまで待ち，シェルの exit のステータスを戻します。コマンドの処理が終了すると，標準出力情報と標準エラー出力情報が返ります。標準出力または標準エラー出力メッセージのデータを，完全に取得できなかった場合，取得できた分のデータを引数に返して，エラーリターンします。

コマンドを実行する UAP を使う OpenTP1 には，コマンドを格納しているディレクトリをサーチパスに追加してください。なお，サーチパスは次に示すどれかの方法で追加してください。

- プロセスサービス定義の prcsvpath オペランドにコマンドのパス名を指定
- prcpath コマンドでサーチパスを追加
- ユーザサービス定義に環境変数を putenv PATH と指定

UAP で値を設定する引数

●com

実行するコマンドの文字列を設定します。文字列の最後にはヌル文字を設定してください。com に指定した文字列は，そのまま，OS の sh(1)に渡します。指定できる文字列長の詳細については，各 OS のマニュ

アル，およびシェルのマニュアルを参照してください。なお，ご使用の OS が Windows の場合は，指定できる com の文字列長は，500 バイトまでになります。

●outsiz

コマンドの実行結果として，標準出力に出力する内容（outmsg に返される値）の受け取る大きさをバイト数で設定します。事前に outmsg で指すアドレスから，outsiz に設定するバイト数分の領域を確保しておいてください。この引数に指定するバイト数は，UAP から実行するコマンドに応じて決めてください。

処理終了後は，コマンドの実行結果として標準出力に出力した実際の長さが返されます。

●errsiz

コマンドの実行結果として標準エラー出力に出力する内容（errmsg に返される値）を受け取る大きさをバイト数で設定します。事前に errmsg が指すアドレスから errsiz に設定するバイト数分の領域を確保しておいてください。この引数に指定するバイト数は，UAP で実行するコマンドに応じて決めてください。

処理終了後は，コマンドの実行結果として標準エラー出力に出力した実際の長さが返されます。

●flags

実行するコマンド内で，標準出力または標準エラー出力メッセージを出力する際の出力動作を設定します。

DCADM_DELAY…実行するコマンドから出力する標準出力または標準エラー出力メッセージのデータを同期（ブロッキングモード）で出力します。出力メッセージが出力できない場合は，実行コマンドが終了できないで，dc_adm_call_command 関数がリターンしなくなります。

DCNOFLAGS…実行するコマンドから出力する標準出力または標準エラー出力メッセージのデータを非同期（ノンブロッキングモード）で出力します。出力メッセージが出力できた分だけ出力し，実行コマンドは処理を続行できます。dc_adm_call_command 関数は，取得できた分のデータを引数に返します。

OpenTP1 から値が返される引数

●stat

指定したコマンドが，正常終了したか異常終了したか（シェルの終了コード）＊が返されます。

＊：waitpid(2)で指定したフォーマットで，sh(1)の終了ステータス

●outmsg

コマンドの実行結果として，標準出力に出力した文字列が，最大（outsiz-1）バイト返されます。（outsiz-1）バイトを超える文字は切り捨てられます。また，パイプの容量を超えた分も切り捨てられます。（outsiz-1）バイトを超えなかった場合は，その文字列の長さだけ返されます。格納される文字列の最後にはヌル文字が付けられます。

●outsiz

コマンドの実行結果として，標準出力に出力した文字列の長さが返されます。

●errmsg

コマンドの実行結果として，標準エラー出力に出力した文字列が最大（errsiz-1）バイト返されます。（errsiz-1）バイトを超える文字は切り捨てられます。また，パイプの容量を超えた分も切り捨てられます。（errsiz-1）バイトを超えなかった場合は，その文字列の長さだけ返されます。格納される文字列の最後にはヌル文字が付けられます。

●errsiz

コマンドの実行結果として，標準エラー出力に出力した文字列の長さが返されます。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	シェルの終了コードは 0（コマンドの実行が正常終了）です。標準出力および標準エラー出力の領域に文字列を格納しました。
DCADMER_PARAM	-1852	引数に設定した値が間違っています。
DCADMER_STATNOTZERO	-1855	シェルの終了コードは 0 以外（コマンドの実行が異常終了）です。標準出力および標準エラー出力のデータを領域に格納しました。
DCADMER_MEMORY_OUT	-1856	標準出力のデータが，領域に入り切りませんでした。
DCADMER_MEMORY_ERR	-1857	標準エラー出力のデータが，領域に入り切りませんでした。
DCADMER_MEMORY_OUTERR	-1858	標準出力のデータと標準エラー出力のデータの両方が，領域に入り切りませんでした。
DCADMER_SYSTEMCALL	-1859	システムコール（close，pipe，dup，または read）の呼び出しに失敗しました。

注意事項

サーチパスに指定したディレクトリ間で，コマンド名が重複しないように注意してください。コマンド名が重複している場合，正しいコマンドが起動されないで別のコマンドが起動されます。また，コマンド名は，OpenTP1 が提供するコマンド群（$DCDIR/bin の下）のコマンド名とも重複しないようにしてください。

dc_adm_complete

名称

ユーザサーバの開始処理完了の報告

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

SUP の開始処理が終了したことを，OpenTP1 に報告します。dc_adm_complete 関数が正常に終了したことで，SUP の起動は完了します。

SPP と MHP では，dc_rpc_mainloop または dc_mcf_mainloop 関数が正常に終了したことで開始処理の完了と見なすので，dc_adm_complete 関数を呼び出す必要はありません。

オフラインの業務をする UAP からは dc_adm_complete 関数は呼び出せません。

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCADMER_COMM	-1851	プロセス間通信でエラーが起こりました。
DCADMER_PARAM	-1852	引数の値が正しくありません。
DCADMER_STS_IO	-1853	ステータス情報の入出力エラーが起こりました。
DCADMER_PROTO	-1854	ユーザサーバが正常開始中，または再開始中ではありません。

リターン値	リターン値（数値）	意味
DCADMER_PROTO	-1854	dc_rpc_open 関数を呼び出していません。

dc_adm_status

名称

ユーザサーバの状態の報告

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

この関数を呼び出したユーザサーバの状態を報告します。ユーザサーバの状態はリターン値で報告されます。

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

リターン値

リターン値が正の値のとき（ユーザサーバの状態を示します）

リターン値	リターン値（数値）	意味
DCADM_STAT_START_NORMAL	2	ユーザサーバは正常開始中です。
DCADM_STAT_START_RECOVER	3	ユーザサーバは再開始中です。
DCADM_STAT_ONLINE	4	ユーザサーバはオンライン中です。
DCADM_STAT_STOP	5	ユーザサーバは終了中です。

リターン値が負の値のとき（エラーが起こったかどうかを示します）

リターン値	リターン値（数値）	意味
DCADMER_COMM	-1851	プロセス間の通信エラーが起こりました。
DCADMER_PARAM	-1852	引数に設定した値が間違っています。
DCADMER_STS_IO	-1853	ステータス情報の入出力エラーが起こりました。
DCADMER_PROTO	-1854	dc_adm_status 関数をオフラインの業務をする UAP から呼び出しています。オフラインの業務をする UAP では dc_adm_status 関数を使えません。
		dc_rpc_open 関数を呼び出していません。

マルチノード機能（dc_adm_get_～）

マルチノード機能（TP1/Multi）を使った形態の UAP で使う関数について説明します。マルチノード機能の関数を次に示します。

- dc_adm_get_nd_status － 指定した OpenTP1 ノードのステータスの取得
- dc_adm_get_nd_status_begin － OpenTP1 ノードのステータス取得の開始
- dc_adm_get_nd_status_done － OpenTP1 ノードのステータス取得の終了
- dc_adm_get_nd_status_next － OpenTP1 ノードのステータスの取得
- dc_adm_get_nodeconf_begin － ノード識別子の取得の開始
- dc_adm_get_nodeconf_done － ノード識別子の取得の終了
- dc_adm_get_nodeconf_next － ノード識別子の取得
- dc_adm_get_node_id － 自ノードのノード識別子の取得
- dc_adm_get_sv_status － 指定したユーザサーバのステータスの取得
- dc_adm_get_sv_status_begin － ユーザサーバのステータス取得の開始
- dc_adm_get_sv_status_done － ユーザサーバのステータス取得の終了
- dc_adm_get_sv_status_next － ユーザサーバのステータスの取得

マルチノード機能の関数（dc_adm_get_～）は，TP1/Server Base の UAP でだけ使えます。TP1/LiNK の UAP では，マルチノード機能の関数は使えません。

dc_adm_get_nd_status

名称

指定した OpenTP1 ノードのステータスの取得

形式

ANSI C C++の形式

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

K&R 版 C の形式

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

機能

指定した OpenTP1 ノードのステータスを取得します。

系切り替え構成の OpenTP1 ノードを指定して dc_adm_get_nd_status 関数を呼び出した場合は，実行系の状態を取得します。

UAP で値を設定する引数

●node_id

ノード識別子へのポインタを設定します。ノード識別子の後ろにはヌル文字を付けます。

ノード識別子の長さは，DCADM_NODE_ID_LEN で定義されている長さにしてください。これ以外の長さの名称を設定した場合は，エラーリターンします。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値が正の値のとき（OpenTP1 ノードの状態を示します）

リターン値	リターン値（数値）	意味
DCADM_STAT_START_NORMAL	2	OpenTP1 ノードは正常開始中です。

リターン値	リターン値（数値）	意味
DCADM_STAT_START_RECOVER	3	OpenTP1 ノードは再開始（リラン）中です。
DCADM_STAT_ONLINE	4	OpenTP1 ノードはオンライン中です。
DCADM_STAT_STOP	5	OpenTP1 ノードは正常終了処理中です。
DCADM_STAT_STOPA	6	OpenTP1 ノードは計画停止 A で終了処理中です。
DCADM_STAT_STOPB	7	OpenTP1 ノードは計画停止 B で終了処理中です。
DCADM_STAT_TERM	8	OpenTP1 ノードが停止中です。または異常終了中です。
DCADM_STAT_NOT_UP	9	指定した OpenTP1 ノードとは，次に示す理由で通信できません。 • OpenTP1 ノードの OpenTP1 を，dcsetup コマンドで登録するか，または登録し直す必要があります。 • マルチノード物理定義に指定した値が間違っています（OpenTP1 ノードを登録していないか，指定したホスト名，またはポート番号が間違っています）。 • 通信障害が起こりました（OpenTP1 ノードのマシンの電源を入れていない，またはネットワーク障害が起こりました）。
DCADM_STAT_SWAP	10	系切り替えが起こっています。

リターン値が負の値のとき（エラーが起こったかどうかを示します）

リターン値	リターン値（数値）	意味
DCADMER_COMM	-1851	プロセス間の通信エラーが起こりました。
DCADMER_PARAM	-1852	引数に設定した値が間違っています。
DCADMER_PROTO	-1854	dc_rpc_open 関数を呼び出していません。
DCADMER_MEMORY	-1861	メモリが不足しました。
DCADMER_DEF	-1862	マルチノード構成定義に指定した値が間違っています。または，マルチノード物理定義に指定した値が間違っています。
DCADMER_MULTI_DEF	-1864	システム共通定義の multi_node_option オペランドに N を指定しています。
		システムに TP1/Multi が組み込まれていません。
		システムに正しいバージョンの TP1/Multi が組み込まれていません。
DCADMER_REMOTE	-1866	指定した OpenTP1 ノードでは，次に示す理由でマルチノード機能は使えません。

リターン値	リターン値（数値）	意味
DCADMER_REMOTE	-1866	• システム共通定義の multi_node_option オペランドに N を指定しています。 • システムに TP1/Multi が組み込まれていません。 • システムに正しいバージョンの TP1/Multi が組み込まれていません。 • メモリが不足しました。
DCADMER_NODE_NOT_EXIST	-1867	node_id に設定したノード識別子に該当する OpenTP1 ノードはありません。

dc_adm_get_nd_status_begin

名称

OpenTP1 ノードのステータス取得の開始

形式

ANSI C C++の形式

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

K&R 版 C の形式

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

機能

OpenTP1 ノードの状態の取得を開始します。この関数が正常に終了すると，状態を取得する対象となる OpenTP1 ノードの個数がリターンされます。

UAP で値を設定する引数

●sub_area

マルチノードサブエリア識別子，または文字列 '*' へのポインタを設定します。マルチノードサブエリア識別子の後ろには，ヌル文字を付けます。文字列 '*' へのポインタを設定した場合は，マルチノードエリアを構成する，すべての OpenTP1 ノードの状態を取得できます。マルチノードサブエリア識別子の長さは，DCADM_SUB_AREA_NAME_SIZE で定義されている最大長以下にしてください。この最大長を超えた名称を設定した場合は，エラーリターンします。

●entry_count

OpenTP1 ノードの個数をリターンする領域へのポインタを設定します。ここに設定した領域へ，sub_area に設定したマルチノードサブエリアにある OpenTP1 ノードの個数をリターンします。sub_area に文字列 '*' へのポインタを設定した場合は，マルチノードエリアを構成する，すべての OpenTP1 ノードの個数がリターンされます。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。指定したマルチノードサブエリアに通信障害の OpenTP1 があっても，DC_OK がリターンされます。
DCADMER_PARAM	-1852	引数に設定した値が間違っています。
DCADMER_PROTO	-1854	dc_adm_get_nd_status_begin 関数は，すでに呼び出しています。
		dc_rpc_open 関数を呼び出していません。
DCADMER_SUBAREA_NOT_EXIST	-1860	sub_area に設定した名称に該当するマルチノードサブエリアはありません。
DCADMER_MEMORY	-1861	メモリが不足しました。
DCADMER_DEF	-1862	マルチノード構成定義に指定した値が間違っています。または，マルチノード物理定義に指定した値が間違っています。
DCADMER_MULTI_DEF	-1864	システム共通定義の multi_node_option オペランドに N を指定しています。
		システムに TP1/Multi が組み込まれていません。
		システムに正しいバージョンの TP1/Multi が組み込まれていません。

dc_adm_get_nd_status_done

名称

OpenTP1 ノードのステータス取得の終了

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

OpenTP1 ノードの状態の取得を終了します。この関数は，dc_adm_get_nd_status_begin 関数のリターン値が DC_OK の場合に，呼び出してください。

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCADMER_PARAM	-1852	引数に設定した値が間違っています。
DCADMER_PROTO	-1854	dc_adm_get_nd_status_begin 関数を呼び出していません。
		dc_rpc_open 関数を呼び出していません。
DCADMER_MULTI_DEF	-1864	システム共通定義の multi_node_option オペランドに N を指定しています。
		システムに TP1/Multi が組み込まれていません。
		システムに正しいバージョンの TP1/Multi が組み込まれていません。

dc_adm_get_nd_status_next

名称

OpenTP1 ノードのステータスの取得

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

この関数を呼び出したユーザサーバがあるマルチノードエリアの OpenTP1 ノード，または指定したマルチノードサブエリアにある OpenTP1 ノードの状態を，1 件取得します。

系切り替え構成の OpenTP1 ノードを指定して dc_adm_get_nd_status_next 関数を呼び出した場合は，実行系の状態を取得します。

この関数で取得する OpenTP1 ノードの状態は，dc_adm_get_nd_status_begin 関数を呼び出した時点のものです。

UAP で値を設定する引数

●node_id

OpenTP1 ノードのノード識別子を受け取る領域へのポインタを設定します。ノード識別子の最後にはヌル文字が設定されます。この領域は，DCADM_NODE_ID_SIZE で定義している長さだけ必要です。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値が正の値のとき（OpenTP1 ノードの状態を示します）

リターン値	リターン値（数値）	意味
DCADM_STAT_START_NORMAL	2	OpenTP1 ノードは正常開始中です。
DCADM_STAT_START_RECOVER	3	OpenTP1 ノードは再開始（リラン）中です。
DCADM_STAT_ONLINE	4	OpenTP1 ノードはオンライン中です。
DCADM_STAT_STOP	5	OpenTP1 ノードは正常終了処理中です。
DCADM_STAT_STOPA	6	OpenTP1 ノードは計画停止 A で終了処理中です。
DCADM_STAT_STOPB	7	OpenTP1 ノードは計画停止 B で終了処理中です。
DCADM_STAT_TERM	8	OpenTP1 ノードが停止中です。または異常終了中です。
DCADM_STAT_NOT_UP	9	指定した OpenTP1 ノードとは，次に示す理由で通信できません。 • OpenTP1 ノードの OpenTP1 を，dcsetup コマンドで登録するか，または登録し直す必要があります。 • マルチノード物理定義に指定した値が間違っています（OpenTP1 ノードを登録していないか，指定したホスト名，またはポート番号が間違っています）。 • 通信障害が起こりました（OpenTP1 ノードのマシンの電源を入れていない，またはネットワーク障害が起こりました）。
DCADM_STAT_SWAP	10	系切り替えが起こっています。

リターン値が負の値のとき（エラーが起こったかどうかを示します）

リターン値	リターン値（数値）	意味
DCADMER_COMM	-1851	プロセス間の通信エラーが起こりました。
DCADMER_PARAM	-1852	引数に設定した値が間違っています。
DCADMER_PROTO	-1854	dc_adm_get_nd_status_begin 関数を呼び出していません。
		dc_rpc_open 関数を呼び出していません。
DCADMER_MULTI_DEF	-1864	システム共通定義の multi_node_option オペランドに N を指定しています。
		システムに TP1/Multi が組み込まれていません。
		システムに正しいバージョンの TP1/Multi が組み込まれていません。
DCADMER_NO_MORE_ENTRY	-1865	OpenTP1 ノードは，これ以上ありません。 すべての OpenTP1 ノードの状態を取得し終わりました。

リターン値	リターン値（数値）	意味
DCADMER_REMOTE	-1866	node_id に返されたノード識別子で示す OpenTP1 ノードでは，次に示す理由でマルチノード機能は使えません。 • システム共通定義の multi_node_option オペランドに N を指定しています。 • システムに TP1/Multi が組み込まれていません。 • システムに正しいバージョンの TP1/Multi が組み込まれていません。 • メモリが不足しました。

dc_adm_get_nodeconf_begin

名称

ノード識別子の取得の開始

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

指定したマルチノードサブエリアにある，すべてのノード識別子の取得を開始します。この関数が正常に終了すると，OpenTP1 ノードの個数がリターンされます。

UAP で値を設定する引数

●sub_area

マルチノードサブエリア識別子，または文字列 '*' へのポインタを設定します。マルチノードサブエリア識別子の後ろには，ヌル文字を付けます。文字列 '*' へのポインタを設定した場合は，マルチノードエリアを構成する，すべてのノード識別子を取得できます。

マルチノードサブエリア識別子の長さは，DCADM_SUB_AREA_NAME_SIZE で定義されている最大長以下にしてください。この最大長を超えた名称を設定した場合は，エラーリターンします。

●entry_count

OpenTP1 ノードの個数をリターンする領域へのポインタを設定します。ここに設定した領域へ，sub_area に設定したマルチノードサブエリアにある OpenTP1 ノードの個数をリターンします。sub_area に文字列 '*' へのポインタを設定した場合は，マルチノードエリアを構成する，すべての OpenTP1 ノードの個数がリターンされます。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。entry_count に示す領域に OpenTP1 ノードの個数を格納しました。
DCADMER_PARAM	-1852	引数に設定した値が間違っています。
DCADMER_PROTO	-1854	dc_adm_get_nodeconf_begin 関数は，すでに呼び出しています。
		dc_rpc_open 関数を呼び出していません。
DCADMER_SUBAREA_NOT_EXIST	-1860	sub_area に設定した名称に該当するマルチノードサブエリアはありません。
DCADMER_MEMORY	-1861	メモリが不足しました。
DCADMER_DEF	-1862	マルチノード構成定義に指定した値が間違っています。または，マルチノード物理定義に指定した値が間違っています。
DCADMER_MULTI_DEF	-1864	システム共通定義の multi_node_option オペランドに N を指定しています。
		システムに TP1/Multi が組み込まれていません。
		システムに正しいバージョンの TP1/Multi が組み込まれていません。

dc_adm_get_nodeconf_done

名称

ノード識別子の取得の終了

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

ノード識別子の取得を終了します。この関数は，dc_adm_get_nodeconf_begin 関数のリターン値が DC_OK の場合に，呼び出してください。

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCADMER_PARAM	-1852	引数に設定した値が間違っています。
DCADMER_PROTO	-1854	dc_adm_get_nodeconf_begin 関数を呼び出していません。
		dc_rpc_open 関数を呼び出していません。
DCADMER_MULTI_DEF	-1864	システム共通定義の multi_node_option オペランドに N を指定しています。
		システムに TP1/Multi が組み込まれていません。
		システムに正しいバージョンの TP1/Multi が組み込まれていません。

dc_adm_get_nodeconf_next

名称

ノード識別子の取得

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

この関数を呼び出したユーザサーバが属するマルチノードエリアのすべてのノード，またはマルチノードサブエリアにあるノード識別子を，1 件取得します。

取得するデータは，dc_adm_get_nodeconf_begin 関数を呼び出したタイミングで取得したデータです。

UAP で値を設定する引数

●node_id

ノード識別子を受け取る領域へのポインタを設定します。ノード識別子の最後にはヌル文字を設定します。この領域には，DCADM_NODE_ID_SIZE で定義している長さが必要です。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCADMER_PARAM	-1852	引数に設定した値が間違っています。
DCADMER_PROTO	-1854	dc_adm_get_nodeconf_begin 関数を呼び出していません。
		dc_rpc_open 関数を呼び出していません。

リターン値	リターン値（数値）	意味
DCADMER_MULTI_DEF	-1864	システム共通定義の multi_node_option オペランドに N を指定しています。
		システムに TP1/Multi が組み込まれていません。
		システムに正しいバージョンの TP1/Multi が組み込まれていません。
DCADMER_NO_MORE_ENTRY	-1865	OpenTP1 ノードは，これ以上ありません。すべてのノード識別子の取得を終了しました。

dc_adm_get_node_id

名称

自ノードのノード識別子の取得

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

システム共通定義に指定した，自 OpenTP1 ノードのノード識別子を，node_id で示す領域にリターンします。

UAP で値を設定する引数

●node_id

ノード識別子を受け取る領域へのポインタを設定します。ノード識別子の最後にはヌル文字が設定されます。この領域には，DCADM_NODE_ID_SIZE で定義している長さが必要です。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCADMER_PARAM	-1852	引数に設定した値が間違っています。
DCADMER_PROTO	-1854	dc_rpc_open 関数を呼び出していません。

dc_adm_get_sv_status

名称

指定したユーザサーバのステータスの取得

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

指定したノード識別子にあるユーザサーバの状態を取得します。

UAP で値を設定する引数

●node_id

ノード識別子，または文字列 '*' へのポインタを設定します。ノード識別子の後ろには，ヌル文字を付けます。文字列 '*' へのポインタを設定した場合は，この関数を呼び出した OpenTP1 ノードを指定したと仮定されます。

ノード識別子の長さは，DCADM_NODE_ID_LEN で定義されている長さにしてください。これ以外の長さの名称を設定した場合は，エラーリターンします。

●sv_name

ユーザサーバ名がある領域へのポインタを設定します。ユーザサーバ名の長さは，SERVER_NAME_SIZE で定義されている最大長以下にしてください。この最大長を超えた名称を設定した場合は，エラーリターンします。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値が正の値のとき（ユーザサーバの状態を示します）

リターン値	リターン値（数値）	意味
DCADM_STAT_START_NORMAL	2	ユーザサーバは正常開始中です。
DCADM_STAT_START_RECOVER	3	ユーザサーバは再開始中です。
DCADM_STAT_ONLINE	4	ユーザサーバはオンライン中です。
DCADM_STAT_STOP	5	ユーザサーバは正常終了処理中です。
DCADM_STAT_TERM	8	ユーザサーバが停止中です。または異常終了中です。

リターン値が負の値のとき（エラーが起こったことを示します）

リターン値	リターン値（数値）	意味
DCADMER_COMM	-1851	指定した OpenTP1 ノードとは，次に示す理由で通信できません。 • OpenTP1 ノードの OpenTP1 を，dcsetup コマンドで登録するか，または登録し直す必要があります。 • マルチノード物理定義に指定した値が間違っています（OpenTP1 ノードを登録していないか，指定したホスト名，またはポート番号が間違っています）。 • 通信障害が起こりました（OpenTP1 ノードのマシンの電源を入れていない，またはネットワーク障害が起こりました）。
DCADMER_PARAM	-1852	引数に設定した値が間違っています。
DCADMER_PROTO	-1854	dc_rpc_open 関数を呼び出していません。
DCADMER_MEMORY	-1861	メモリが不足しました。
DCADMER_DEF	-1862	マルチノード構成定義に指定した値が間違っています。
DCADMER_MULTI_DEF	-1864	システム共通定義の multi_node_option オペランドに N を指定しています。または，マルチノード物理定義に指定した値が間違っています。
		システムに TP1/Multi が組み込まれていません。
		システムに正しいバージョンの TP1/Multi が組み込まれていません。
DCADMER_REMOTE	-1866	指定した OpenTP1 ノードでは，次に示す理由でマルチノード機能は使えません。

リターン値	リターン値（数値）	意味
DCADMER_REMOTE	-1866	• システム共通定義の multi_node_option オペランドに N を指定しています。 • システムに TP1/Multi が組み込まれていません。 • システムに正しいバージョンの TP1/Multi が組み込まれていません。 • メモリが不足しました。
DCADMER_NODE_NOT_EXIST	-1867	node_id に設定したノード識別子に該当する OpenTP1 ノードはありません。
DCADMER_SWAP	-1868	系切り替えが起こっているため，ユーザサーバの状態を取得できません。

dc_adm_get_sv_status_begin

名称

ユーザサーバのステータス取得の開始

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

指定したノード識別子にあるユーザサーバの状態の取得を開始します。この関数が正常に終了すると，状態を取得する対象となるユーザサーバの個数をリターンします。

UAP で値を設定する引数

●node_id

ノード識別子，または文字列 '*' へのポインタを設定します。ノード識別子の後ろには，ヌル文字を付けます。文字列 '*' へのポインタを設定した場合は，この関数を呼び出した OpenTP1 ノードを指定したと仮定されます。

ノード識別子の長さは，DCADM_NODE_ID_LEN で定義されている長さにしてください。これ以外の長さの名称を設定した場合は，エラーリターンします。

●entry_count

ユーザサーバの個数をリターンする領域へのポインタを設定します。ここに設定した領域へ，node_id に設定した OpenTP1 ノードにあるユーザサーバの個数をリターンします。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。entry_count に示す領域に，ユーザサーバ数を格納しました。
DCADMER_COMM	-1851	指定した OpenTP1 ノードとは，次に示す理由で通信できません。 • OpenTP1 ノードの OpenTP1 を，dcsetup コマンドで登録するか，または登録し直す必要があります。 • マルチノード物理定義に指定した値が間違っています（OpenTP1 ノードを登録していないか，指定したホスト名，またはポート番号が間違っています）。 • 通信障害が起こりました（OpenTP1 ノードのマシンの電源を入れていない，またはネットワーク障害が起こりました）。
DCADMER_PARAM	-1852	引数に設定した値が間違っています。
DCADMER_PROTO	-1854	dc_adm_get_sv_status_begin 関数は，すでに呼び出しています。
		dc_rpc_open 関数を呼び出していません。
DCADMER_MEMORY	-1861	メモリが不足しました。
DCADMER_DEF	-1862	マルチノード構成定義に指定した値が間違っています。または，マルチノード物理定義に指定した値が間違っています。
DCADMER_MULTI_DEF	-1864	システム共通定義の multi_node_option オペランドに N を指定しています。
		システムに TP1/Multi が組み込まれていません。
		システムに正しいバージョンの TP1/Multi が組み込まれていません。
DCADMER_REMOTE	-1866	指定した OpenTP1 ノードでは，次に示す理由でマルチノード機能は使えません。 • システム共通定義の multi_node_option オペランドに N を指定しています。 • システムに TP1/Multi が組み込まれていません。 • システムに正しいバージョンの TP1/Multi が組み込まれていません。 • メモリが不足しました。
DCADMER_NODE_NOT_EXIST	-1867	node_id に設定したノード識別子に該当する OpenTP1 ノードはありません。
DCADMER_SWAP	-1868	系切り替えが起こっているため，ユーザサーバの状態を取得できません。

dc_adm_get_sv_status_done

名称

ユーザサーバのステータス取得の終了

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

ユーザサーバの状態の取得を終了します。この関数は，dc_adm_get_sv_status_begin 関数のリターン値が DC_OK の場合に呼び出してください。

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCADMER_PARAM	-1852	引数に設定した値が間違っています。
DCADMER_PROTO	-1854	dc_adm_get_sv_status_begin 関数を呼び出していません。
		dc_rpc_open 関数を呼び出していません。
DCADMER_MULTI_DEF	-1864	システム共通定義の multi_node_option オペランドに N を指定しています。
		システムに TP1/Multi が組み込まれていません。
		システムに正しいバージョンの TP1/Multi が組み込まれていません。

dc_adm_get_sv_status_next

名称

ユーザサーバのステータスの取得

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

指定した OpenTP1 ノードにあるユーザサーバの状態を取得します。

取得するデータは，dc_adm_get_sv_status_begin 関数を呼び出したタイミングで取得したデータです。

UAP で値を設定する引数

●sv_name

ユーザサーバ名を受け取る領域へのポインタを設定します。この領域には，SERVER_NAME_SIZE で定義している長さが必要です。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値が正の値のとき（ユーザサーバの状態を示します）

リターン値	リターン値（数値）	意味
DCADM_STAT_START_NORMAL	2	ユーザサーバは正常開始中です。
DCADM_STAT_START_RECOVER	3	ユーザサーバは再開始中です。
DCADM_STAT_ONLINE	4	ユーザサーバはオンライン中です。

リターン値	リターン値（数値）	意味
DCADM_STAT_STOP	5	ユーザサーバは正常終了処理中です。
DCADM_STAT_TERM	8	ユーザサーバが停止中です。または異常終了中です。

リターン値が負の値のとき（エラーが起こったことを示します）

リターン値	リターン値（数値）	意味
DCADMER_PARAM	-1852	引数に設定した値が間違っています。
DCADMER_PROTO	-1854	dc_adm_get_sv_status_begin 関数を呼び出していません。
		dc_rpc_open 関数を呼び出していません。
DCADMER_MULTI_DEF	-1864	システム共通定義の multi_node_option オペランドに N を指定しています。
		システムに TP1/Multi が組み込まれていません。
		システムに正しいバージョンの TP1/Multi が組み込まれていません。
DCADMER_NO_MORE_ENTRY	-1865	ユーザサーバは，これ以上ありません。すべてのユーザサーバの状態の取得を終了しました。

DAM ファイルサービス（dc_dam_～）

DAM ファイルサービスで使える関数について説明します。DAM ファイルサービスの関数を次に示します。

<オンライン環境の場合にだけ使用できる関数>

- dc_dam_close－論理ファイルのクローズ
- dc_dam_end－回復対象外 DAM ファイル使用の終了
- dc_dam_hold－論理ファイルの閉塞
- dc_dam_open－論理ファイルのオープン
- dc_dam_read－論理ファイルからブロックの入力
- dc_dam_release－論理ファイルの閉塞の解除
- dc_dam_rewrite－論理ファイルのブロックの更新
- dc_dam_start－回復対象外 DAM ファイル使用の開始
- dc_dam_status－論理ファイルの状態の参照
- dc_dam_write－論理ファイルへブロックの出力

<オフライン環境の場合にだけ使用できる関数>

- dc_dam_bseek－物理ファイルのブロックの検索
- dc_dam_create－物理ファイルの割り当て
- dc_dam_dget－物理ファイルからブロックの直接入力
- dc_dam_dput－物理ファイルへブロックの直接出力
- dc_dam_get－物理ファイルからブロックの入力
- dc_dam_iclose－物理ファイルのクローズ
- dc_dam_iopen－物理ファイルのオープン
- dc_dam_put－物理ファイルへブロックの出力

DAM ファイルサービスの関数（dc_dam_～）は，TP1/Server Base の UAP でだけ使えます。TP1/LiNK の UAP では，DAM ファイルサービスの関数は使えません。

dc_dam_bseek

名称

物理ファイルのブロックの検索

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

物理ファイルの相対ブロック番号を指定して，該当するブロックに位置づけます。

dc_dam_bseek 関数は，再作成出力要求の dc_dam_iopen 関数を呼び出したあとで使ってください。

該当する相対ブロック番号がファイルにある場合は，相対ブロック番号をそのままリターンします。

物理ファイルのブロックを検索するときは，dc_dam_iopen 関数のリターン値のファイル記述子を設定します。

UAP で値を設定する引数

●fno

ブロックを検索するファイルの，ファイル記述子を設定します。

●blkno

検索する相対ブロック番号を設定します。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
0 または正の整数		0 または正の整数は，相対ブロック番号を示します。
DCDAMER_BADF	-1603	fno に設定したファイル記述子は，正常にオープンして得られた記述子ではありません。
		DAM ファイルをオープンしていません。
DCDAMER_SEQER	-1605	DAM ファイルにアクセスする関数を呼び出す順序が間違っています。
DCDAMER_BNOER	-1606	相対ブロック番号が間違っています。
DCDAMER_PARAM_FLAGS	-1611	flags に設定した値が間違っています。
DCDAMER_IOER	-1620	出力エラーが起こりました。

dc_dam_close

名称

論理ファイルのクローズ

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

論理ファイルをクローズします。

- 回復対象の DAM ファイルの場合

  トランザクション処理の範囲内でオープンした論理ファイルのうち，トランザクションの終了までにクローズしていないものは，DAM サービスが同期点処理時にクローズします。ただし，トランザクションの範囲外（dc_trn_begin 関数を呼び出す前）にオープンした場合や，回復対象外の DAM ファイルは，クローズしません。

  トランザクションを開始する前に論理ファイルをオープンした場合，UAP の処理を終了させる前に必ずクローズしてください。

- 回復対象外の DAM ファイルの場合

  トランザクション処理とは同期しないため，論理ファイルをクローズするときに任意に dc_dam_close 関数を呼び出せます。ただし，dc_dam_end 関数を呼び出す前に，オープンしている論理ファイルを dc_dam_close 関数でクローズしておいてください。

論理ファイルをクローズするときは，dc_dam_open 関数で返されたリターン値のファイル記述子を設定します。

UAP で値を設定する引数

●damfd

クローズするファイルの，ファイル記述子を設定します。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	論理ファイルを正常にクローズしました。
DCDAMER_PROTO	-1600	dc_rpc_open 関数を呼び出していません。
		トランザクションの範囲外でオープンしている DAM ファイルを，トランザクションの範囲内でクローズしています（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。
		ユーザサービス定義の atomic_update オペランドの指定が'N'になっています（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。
		dc_dam_start 関数を呼び出していません（回復対象外の DAM ファイルにアクセスした場合だけリターンされます）。
		次に示すように，UAP を正しくリンケージしていません。 • DAM サービスの関数で TAM ファイルにアクセスする場合に使うライブラリ（-ltdam）を，不当にリンケージしています。 • トランザクション制御用オブジェクトファイルのリソースマネジャ登録が間違っています。
DCDAMER_BADF	-1603	damfd に設定したファイル記述子は，正常にオープンして得られた記述子でありません。
		DAM ファイルをオープンしていません。
DCDAMER_PARAM_FLAGS	-1611	flags に設定した値が間違っています。

dc_dam_create

名称

物理ファイルの割り当て

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

OpenTP1 ファイルシステムに物理ファイルを割り当てます。

物理ファイルの大きさは，（ブロック長＋ 8）×（ブロック数＋ 1）になります。

dc_dam_create 関数を呼び出したあとには，dc_dam_iopen 関数を呼び出す必要はありません。

dc_dam_create 関数を呼び出したあとには，次に示す関数は呼び出せません。

- dc_dam_get 関数
- dc_dam_bseek 関数
- dc_dam_dget 関数
- dc_dam_dput 関数

出力バッファの大きさは（ブロック長＋ 8）×一括処理ブロック数になります。

UAP で値を設定する引数

●fname

OpenTP1 ファイルシステムに作成する物理ファイル名を，（スペシャルファイル名＋ 14）バイト以内のパス名で設定します。

●blksize

物理ファイルのブロック長を設定します。

32760 を超えないセクタ長×n−8（n は正の整数）を満たす値を設定してください。

セクタ長は次のとおりです。

- キャラクタ型スペシャルファイルの場合：filmkfs コマンドの-s オプション指定値
- 通常ファイルの場合：512 バイト

●blknum

物理ファイルのブロック数を設定します。

設定できる範囲は 1～2147483647 です。

●pnum

入出力の単位となる，一括処理ブロック数を設定します。

設定できる範囲は 0～2147483647 です。0 を指定した場合は，10 になります。

●flags

所有者，所有者グループ，ほかの UAP のアクセス権を設定します。次に示す値，または（ ）内に示すビット列で設定します。

DCDAM_READ_OWNER（00400）…所有者の読み出し権を設定

DCDAM_WRITE_OWNER（00200）…所有者の書き込み権を設定

DCDAM_READ_GROUP（00040） …グループの読み出し権を設定

DCDAM_WRITE_GROUP（00020） …グループの書き込み権を設定

DCDAM_READ_OTHERS（00004） …ほかの UAP の読み出し権を設定

DCDAM_WRITE_OTHERS（00002）…ほかの UAP の書き込み権を設定

なお，DCNOFLAGS を設定した場合は，次のように仮定されます。

DCDAM_READ_OWNER（00400）

DCDAM_WRITE_OWNER（00200）

DCDAM_READ_GROUP（00040）

DCDAM_READ_OTHERS（00004）

リターン値

リターン値	リターン値（数値）	意味
0 または正の整数		0 または正の整数は，ファイル記述子を示します。
DCDAMER_NOMEM	-1607	メモリが不足しました。
DCDAMER_OPENED	-1608	設定した物理ファイルはオープン済みです。
DCDAMER_PARAM_FLAGS	-1611	flags に設定した値が間違っています。
DCDAMER_FILEER	-1614	物理ファイル名が間違っています。
DCDAMER_PNUMER	-1615	一括処理ブロック数の値が間違っています。
DCDAMER_EXIST	-1617	すでに同じ名称の物理ファイルが割り当てられています。
DCDAMER_VERSION	-1618	OpenTP1 ファイルシステムのバージョンが，作成時と割り当て時で一致していません。
DCDAMER_IOER	-1620	入出力エラーが起こりました。
DCDAMER_ACCESS	-1628	dc_dam_create 関数を呼び出した UAP は，スペシャルファイルへのアクセス権がありません。
		割り当てようとした DAM ファイルは，セキュリティ機能で保護されています。dc_dam_create 関数を呼び出した UAP には，アクセス権限がありません。
DCDAMER_LBLNER	-1630	ブロック長が適当な値ではありません。
DCDAMER_LBNOER	-1631	ブロック数が適当な値ではありません。
DCDAMER_LFNMER	-1632	物理ファイル名がキャラクタ型スペシャルファイルではありません。またはこのスペシャルファイルに対応する装置がありません。
DCDAMER_LNOINT	-1633	指定した OpenTP1 ファイルが，OpenTP1 ファイルシステムとして初期化されていません。
DCDAMER_LFFOVF	-1634	OpenTP1 ファイルシステムとして初期化したときに指定したファイル数以上の OpenTP1 ファイル（物理ファイル）を割り当てようとしています。
DCDAMER_LFNOVF	-1635	実行しているプロセスでオープンできるファイル数の最大値を超えた値を設定しています。
DCDAMER_USED	-1636	fname に設定した物理ファイルは，現在オンラインで使っています。またはほかのプロセスで使っています。
DCDAMER_SPACE	-1640	OpenTP1 ファイルシステムに物理ファイルを割り当てる分の空き領域がありません。
DCDAMER_NO_ACL	-1646	割り当てようとした DAM ファイルは，セキュリティ機能で保護されています。該当するファイルに対する ACL がありません。

dc_dam_dget

名称

物理ファイルからブロックの直接入力

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

物理ファイルから，設定した相対ブロック番号に該当するブロックを入力します。dc_dam_dget 関数は，再作成出力要求の dc_dam_iopen 関数を呼び出したあとで使ってください。

ブロック長がバッファ長よりも小さい場合は，ブロックを入力してそのブロック長をリターン値として返します。ブロック長がバッファ長よりも大きい場合は，dc_dam_dget 関数はエラーリターンします。

物理ファイルからブロックを直接入力するときは，dc_dam_iopen 関数で返されたリターン値のファイル記述子を設定します。

UAP で値を設定する引数

●fno

ブロックを直接入力するファイルの，ファイル記述子を設定します。

●datadr

入力バッファのアドレスを設定します。

●datalen

入力バッファの長さを設定します。

●blkno

入力するブロックの，相対ブロック番号を設定します。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
正の整数		正の整数は，入力ブロック長を示します。
DCDAMER_BADF	-1603	fno に設定したファイル記述子は，正常にオープンして得られた記述子ではありません。
		DAM ファイルをオープンしていません。
DCDAMER_BUFER	-1604	入力データ長に，ブロック長よりも小さい値を設定しています。
DCDAMER_SEQER	-1605	DAM ファイルにアクセスする関数を呼び出す順序が間違っています。
DCDAMER_BNOER	-1606	相対ブロック番号が間違っています。
DCDAMER_PARAM_FLAGS	-1611	flags に設定した値が間違っています。
DCDAMER_IOER	-1620	入力エラーが起こりました。
DCDAMER_ACCESS	-1628	アクセスしようとした DAM ファイルは，セキュリティ機能で保護されています。dc_dam_dget 関数を呼び出した UAP には，アクセス権限がありません。
DCDAMER_NO_ACL	-1646	アクセスしようとした DAM ファイルは，セキュリティ機能で保護されています。該当するファイルに対する ACL がありません。

dc_dam_dput

名称

物理ファイルへブロックの直接出力

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

物理ファイルへ，設定した相対ブロック番号に該当するブロックを出力します。dc_dam_dput 関数は，再作成出力要求の dc_dam_iopen 関数を呼び出してから使ってください。

出力データ長がブロック長よりも小さい場合は，ブロックへ出力して残りの領域をヌル文字で埋めます。出力データ長がブロック長よりも大きい場合は，dc_dam_dput 関数はエラーリターンします。

物理ファイルへブロックを直接出力するときは，dc_dam_iopen 関数で返されたリターン値のファイル記述子を設定します。

UAP で値を設定する引数

●fno

ブロックを直接出力するファイルの，ファイル記述子を設定します。

●datadr

出力データのアドレスを設定します。

●datalen

出力データの長さを設定します。

●blkno

出力先ブロックの，相対ブロック番号を設定します。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
正の整数		正の整数は，出力ブロック長を示します。
DCDAMER_BADF	-1603	fno に設定したファイル記述子は，正常にオープンして得られた記述子ではありません。
		DAM ファイルをオープンしていません。
DCDAMER_BUFER	-1604	出力データ長に，ブロック長よりも大きい値を設定しています。
DCDAMER_SEQER	-1605	DAM ファイルにアクセスする関数を呼び出す順序が間違っています。
DCDAMER_BNOER	-1606	相対ブロック番号が間違っています。
DCDAMER_PARAM_FLAGS	-1611	flags に設定した値が間違っています。
DCDAMER_IOER	-1620	出力エラーが起こりました。
DCDAMER_ACCESS	-1628	アクセスしようとした DAM ファイルは，セキュリティ機能で保護されています。dc_dam_dput 関数を呼び出した UAP には，アクセス権限がありません。
DCDAMER_NO_ACL	-1646	アクセスしようとした DAM ファイルは，セキュリティ機能で保護されています。該当するファイルに対する ACL がありません。

dc_dam_end

名称

回復対象外 DAM ファイル使用の終了

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

回復対象外の DAM ファイルを使うことを終了します。

dc_dam_start 関数を呼び出した場合は，処理を終了する前に必ず dc_dam_end 関数を呼び出してください。dc_dam_end 関数を呼び出さないと，回復対象外の DAM ファイルへのアクセスに使った資源は UAP が終了するまで解放されないままになります。

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。回復対象外の DAM ファイルの使用を終了しました。
DCDAMER_PROTO	-1600	dc_rpc_open 関数を呼び出していません。
DCDAMER_SEQER	-1605	dc_dam_start 関数を呼び出していません。
DCDAMER_PARAM_FLAGS	-1611	flags に設定した値が間違っています。

dc_dam_get

名称

物理ファイルからブロックの入力

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

OpenTP1 ファイルシステムの物理ファイルから，ブロック単位でデータを順に入力します。
dc_dam_iopen 関数を呼び出したあとで，dc_dam_get 関数を使ってください。

ブロック長がバッファ長よりも小さい場合は，ブロックを入力してそのブロック長をリターン値として返します。ブロック長がバッファ長よりも大きい場合は，エラーリターンします。

物理ファイルからブロックを入力するときは，dc_dam_iopen 関数で返されたリターン値のファイル記述子を設定します。

UAP で値を設定する引数

●fno

ブロックを入力するファイルの，ファイル記述子を設定します。

●datadr

入力バッファのアドレスを設定します。

●datalen

入力バッファの長さを設定します。設定できる範囲は 504～2147483647 です。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
正の整数		正の整数は，入力ブロック長を示します。
DCDAMER_BADF	-1603	fno に設定したファイル記述子は，正常にオープンして得られた記述子ではありません。
		DAM ファイルをオープンしていません。
DCDAMER_BUFER	-1604	ブロック長にバッファ長よりも大きい値を設定しています。
		入力バッファ長に設定した値は，設定できる範囲を超えています。
DCDAMER_SEQER	-1605	DAM ファイルにアクセスする関数を呼び出す順序が間違っています。
DCDAMER_PARAM_FLAGS	-1611	flags に設定した値が間違っています。
DCDAMER_IOER	-1620	入力エラーが起こりました。
DCDAMER_ACCESS	-1628	アクセスしようとした DAM ファイルは，セキュリティ機能で保護されています。dc_dam_get 関数を呼び出した UAP には，アクセス権限がありません。
DCDAMER_EOF	-1637	ファイルの終わりに達しました。
DCDAMER_NO_ACL	-1646	アクセスしようとした DAM ファイルは，セキュリティ機能で保護されています。該当するファイルに対する ACL がありません。

dc_dam_hold

名称

論理ファイルの閉塞

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

論理ファイルを閉塞します。dc_dam_hold 関数を呼び出したあとには，指定した論理ファイルに対するほかの UAP からのアクセス要求は，すべて論理閉塞エラーでリターンします。

- 回復対象の DAM ファイルの場合

  dc_dam_hold 関数を呼び出した場合に，指定した論理ファイルがほかのトランザクション処理で同期点処理中のときは，同期点処理が終了したあとに閉塞します。ただし，同期点処理が完了していなくても，dc_dam_hold 関数はアクセスした UAP にリターンします。

UAP で値を設定する引数

●lfname

閉塞するファイルの論理ファイル名を，1～8 バイトの名称で設定します。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	lfname に設定した論理ファイルは正常に閉塞しました。
DCDAMER_PROTO	-1600	dc_rpc_open 関数を呼び出していません。

リターン値	リターン値（数値）	意味
DCDAMER_PROTO	-1600	ユーザサービス定義の atomic_update オペランドの指定が'N'になっています（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。
		dc_dam_start 関数を呼び出していません（回復対象外の DAM ファイルにアクセスした場合だけリターンされます）。
		次に示すように，UAP を正しくリンケージしていません。 • DAM サービスの関数で TAM ファイルにアクセスする場合に使うライブラリ（-ltdam）を，不当にリンケージしています。 • トランザクション制御用オブジェクトファイルのリソースマネジャ登録が間違っています。
DCDAMER_UNDEF	-1601	設定した論理ファイル名は定義されていません。
DCDAMER_NOMEM	-1607	メモリが不足しました。
DCDAMER_PARAM_LFNAME	-1610	lfname に設定した論理ファイル名が間違っています。
DCDAMER_PARAM_FLAGS	-1611	flags に設定した値が間違っています。
DCDAMER_VERSION	-1618	UAP が，現在稼働している DAM サービスでは動作できないバージョンの DAM ライブラリと結合されています。
DCDAMER_LHOLDED	-1625	lfname に設定した論理ファイル名は，すでに論理閉塞しています。
DCDAMER_OHOLDED	-1626	lfname に設定した論理ファイル名は，すでに障害閉塞しています。
DCDAMER_ACCESS	-1628	閉塞しようとした DAM ファイルは，セキュリティ機能で保護されています。dc_dam_hold 関数を呼び出した UAP には，アクセス権限がありません。
DCDAMER_NO_ACL	-1646	閉塞しようとした DAM ファイルは，セキュリティ機能で保護されています。該当するファイルに対する ACL がありません。

dc_dam_iclose

名称

物理ファイルのクローズ

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

OpenTP1 ファイルシステム上に作成した物理ファイルをクローズします。

データがファイルの終わりまで達していない場合は，ファイルの終わりまでヌル文字のブロックで埋めます。ただし，dc_dam_iopen 関数の flags に設定した値が作成出力要求（DCDAM_INITIALIZE）および dc_dam_create 関数を呼び出している場合にだけヌル文字のブロックで埋めます。

物理ファイルをクローズするときは，dc_dam_create 関数，または dc_dam_iopen 関数で返されたリターン値のファイル記述子を設定します。

UAP で値を設定する引数

●fno

クローズするファイルの，ファイル記述子を設定します。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常にクローズしました。

リターン値	リターン値（数値）	意味
DCDAMER_BADF	-1603	fno に設定したファイル記述子は正常にオープンして得られたファイル記述子ではありません。
		DAM ファイルをオープンしていません。
DCDAMER_PARAM_FLAGS	-1611	flags に設定した値が間違っています。
DCDAMER_IOER	-1620	出力エラーが起こりました。

dc_dam_iopen

名称

物理ファイルのオープン

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

OpenTP1 ファイルシステム上に作成した物理ファイルをオープンします。ただし，オンラインで使っている物理ファイルに対するオープンはできません。

UAP で値を設定する引数

●fname

オープンするファイルの物理ファイル名を（スペシャルファイル名＋ 14 バイト）以内のパス名で設定します。

●pnum

入出力の単位となる，一括処理ブロック数を設定します。

設定できる範囲は 0～2147483647 です。0 を指定した場合は，10 になります。

●flags

作成出力要求か**再作成**（オーバライト）**出力要求**かの種別を設定します。ここに設定した値で，ファイルのクローズ時に，残りの領域をヌル文字のブロックで埋めるかどうかが決まります。ここに設定した値は，dc_dam_put 関数を呼び出したあとで dc_dam_iclose 関数を呼び出して正常終了したときに有効になります。dc_dam_put 関数を呼び出しても，dc_dam_iclose 関数を呼び出さないで UAP の処理を終了した場合は，ヌル文字のブロックで埋めません。

DCDAM_INITIALIZE

　作成出力要求を設定（ヌル文字のブロックで埋めます）

DCDAM_OVERWRITE

再作成出力要求を設定（ヌル文字のブロックで埋めません）

DCNOFLAGS を設定した場合は，DCDAM_OVERWRITE が仮定されます。

リターン値

リターン値	リターン値（数値）	意味
0 または正の整数		0 または正の整数は，ファイル記述子を示します。
DCDAMER_NOMEM	-1607	メモリが不足しました。
DCDAMER_OPENED	-1608	fname に設定した物理ファイルはオープン済みです。
DCDAMER_PARAM_FLAGS	-1611	flags に設定した値が間違っています。
DCDAMER_FILEER	-1614	fname に設定した物理ファイル名が間違っています。
DCDAMER_PNUMER	-1615	一括処理ブロック数の値が間違っています。
DCDAMER_NODAM	-1616	fname に設定した物理ファイルは，DAM ファイルではありません。
DCDAMER_VERSION	-1618	OpenTP1 ファイルシステムのバージョンが，作成時と割り当て時で一致していません。
DCDAMER_NOEXIST	-1619	fname に設定した物理ファイルは存在しません。
DCDAMER_IOER	-1620	入出力エラーが起こりました。
DCDAMER_ACCESS	-1628	dc_dam_iopen 関数を呼び出した UAP には，スペシャルファイルに対するアクセス権がありません。
DCDAMER_LFNMER	-1632	物理ファイル名がキャラクタ型スペシャルファイルではありません。または指定したスペシャルファイルに対応する装置がありません。
DCDAMER_LNOINT	-1633	fname に設定した物理ファイルが，OpenTP1 ファイルシステムとして初期化されていません。
DCDAMER_LFNOVF	-1635	プロセスでオープンできるファイル数の最大値を超えた値を設定しています。
DCDAMER_USED	-1636	fname に設定した物理ファイルは，現在オンラインで使っています。または，ほかのプロセスで使っています。
DCDAMER_ACCESSF	-1638	物理ファイルに対するアクセス権がありません。
DCDAMER_CRUSH	-1639	物理ファイルの破壊を検出しました。

dc_dam_open

名称

論理ファイルのオープン

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

論理ファイルをオープンします。

- 回復対象の DAM ファイルの場合

  論理ファイルに対して，ファイル排他をするかブロック排他をするかを設定します。

  ファイル排他を指定できるのは次の場合です。

  - トランザクションブランチ単位で排他制御する指定で，トランザクションの範囲内で論理ファイルをオープンした場合

  次に示す場合は，ファイル排他を指定できません。ブロック排他で排他制御をしてください。

  - トランザクションの範囲外で論理ファイルをオープンした場合
  - グローバルトランザクション単位で排他制御する指定をした場合

  同じトランザクションブランチ内では，いったん dc_dam_close 関数でクローズした論理ファイルを再びオープンした場合，dc_dam_close 関数を呼び出す前の状態を引き継ぎます。

- 回復対象外の DAM ファイルの場合

  トランザクション処理とは同期しないため，排他に関する制限はありません。

UAP で値を設定する引数

●lfname

オープンするファイルの論理ファイル名を，1～8 バイトの名称で設定します。

●flags

ファイル排他をするかブロック排他をするか，排他エラー時に排他解除待ちをするかどうかを設定します。次の形式で設定します。

```
{DCDAM_FILE_EXCLUSIVE | DCDAM_BLOCK_EXCLUSIVE
                                [ | {DCDAM_WAIT | DCDAM_NOWAIT}] }
```

- フラグ 1

  ファイル排他をするかブロック排他をするかを設定します。

  DCDAM_FILE_EXCLUSIVE…ファイル排他

  DCDAM_BLOCK_EXCLUSIVE…ブロック排他

- フラグ 2

  排他エラーが起こったとき，排他解除待ちをするかどうかを設定します。

  DCDAM_WAIT……排他解除待ちをします。

  DCDAM_NOWAIT…待たないで，エラーリターンします。

どちらも設定しなかった場合は，DCDAM_NOWAIT が仮定されます。

flags の設定方法

flags に設定できる値は，使う DAM ファイルが回復対象かどうかで異なります。

- 回復対象の DAM ファイルの場合

  フラグ 2 の排他解除待ちは，dc_dam_read 関数，dc_dam_write 関数で排他エラーが起こった場合の設定となります。dc_dam_open 関数で排他エラーが起こった場合の設定ではありません。
  dc_dam_open 関数で排他エラーが起こった場合には，無条件に DCDAMER_EXCER でエラーリターンします。

  回復対象の DAM ファイルにアクセスする場合の，flags に設定したフラグと排他の設定内容を次に示します。

フラグ 1	フラグ 2※	flags の設定内容
FILE_EXCLUSIVE	×	ファイル排他
BLOCK_EXCLUSIVE	WAIT	ブロック排他，排他エラー時は解除待ち
	NOWAIT	ブロック排他，排他エラー時はエラーリターン

（凡例）

×：設定できません。

注※

省略した場合は，NOWAIT が仮定されます。

- 回復対象外の DAM ファイルの場合

  フラグ 2 の排他解除待ちは，排他エラーが起こった場合の設定となります。dc_dam_open 関数，dc_dam_read 関数，および dc_dam_write 関数で排他エラーが起こった場合に，フラグ 2 に設定した

値に従って，排他の解除を待つかどうかが決まります。フラグ 2 に DCDAM_NOWAIT を設定，または省略した場合に排他エラーが起こったときは，DCDAMER_EXCER でエラーリターンします。

回復対象外の DAM ファイルにアクセスする場合の，flags に設定したフラグと排他の設定内容を次に示します。

フラグ 1	フラグ 2※	flags の設定内容
FILE_EXCLUSIVE	WAIT	ファイル排他，排他エラー時は解除待ち
	NOWAIT	ファイル排他，排他エラー時はエラーリターン
BLOCK_EXCLUSIVE	WAIT	ブロック排他，排他エラー時は解除待ち
	NOWAIT	ブロック排他，排他エラー時はエラーリターン

注※

省略した場合は，NOWAIT が仮定されます。

回復対象か回復対象外かに関係なく，フラグ 1 にファイル排他を設定するとファイル全体に排他を掛けるため，dc_dam_read 関数，dc_dam_write 関数で排他エラーは起こりません。そのため，排他解除待ちをするかどうかを設定できません。dc_dam_read 関数，dc_dam_write 関数の引数に設定した排他解除待ち種別は無視されます。

リターン値

リターン値	リターン値（数値）	意味
0 または正の整数		0 または正の整数は，ファイル記述子を示します。
DCDAMER_PROTO	-1600	dc_rpc_open 関数を呼び出していません。
		ユーザサービス定義の atomic_update オペランドの指定が'N'になっています（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。
		ユーザサービス定義の atomic_update オペランドの指定が'N'の場合に，dc_dam_start 関数を呼び出していません（回復対象外の DAM ファイルにアクセスした場合だけリターンされます）。
		次に示すように，UAP を正しくリンケージしていません。 • DAM サービスの関数で TAM ファイルにアクセスする場合に使うライブラリ（-ltdam）を，不当にリンケージしています。 • トランザクション制御用オブジェクトファイルのリソースマネジャ登録が間違っています。
		DAM ファイルの排他を，トランザクションの範囲外から指定しています（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。

リターン値	リターン値（数値）	意味
DCDAMER_PROTO	-1600	グローバルトランザクション単位の排他制御で，DAM ファイルにファイル排他を設定しています（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。
DCDAMER_UNDEF	-1601	lfname に設定した論理ファイル名は定義されていません。
DCDAMER_EXCER	-1602	排他エラーが起こりました。
DCDAMER_SEQER	-1605	ユーザサービス定義の atomic_update オペランドの指定が'Y'の場合に，dc_dam_start 関数を呼び出していません（回復対象外の DAM ファイルにアクセスした場合だけリターンされます）。
DCDAMER_NOMEM	-1607	メモリが不足しました。
DCDAMER_OPENED	-1608	lfname に設定した論理ファイルはオープン済みです。
DCDAMER_PARAM_LFNAME	-1610	論理ファイル名に設定した値が間違っています。
DCDAMER_PARAM_FLAGS	-1611	flags に設定した値が間違っています。
DCDAMER_LHOLD	-1621	lfname に設定したファイルは論理閉塞されています。
DCDAMER_OHOLD	-1622	lfname に設定したファイルは障害閉塞されています。
DCDAMER_OPENNUM	-1627	キャラクタ型スペシャルファイルのオープン数が最大値を超えました。
DCDAMER_ACCESS	-1628	キャラクタ型スペシャルファイルにはアクセス権がありません。
DCDAMER_TMERR	-1629	トランザクションサービスでエラーが起こりました（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。
DCDAMER_DLOCK	-1642	デッドロックが起こりました（回復対象外の DAM ファイルにアクセスした場合だけリターンされます）。
DCDAMER_TIMOUT	-1643	ロックサービス定義で指定した待ち時間のタイムアウトのため資源を確保できませんでした（回復対象外の DAM ファイルにアクセスした場合だけリターンされます）。
DCDAMER_LCKOV	-1645	最大同時排他要求数を超えて，排他を要求しています。
DCDAMER_NO_ACL	-1646	オープンしようとした DAM ファイルは，セキュリティ機能で保護されています。該当するファイルに対する ACL がありません。

dc_dam_put

名称

物理ファイルへブロックの出力

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

OpenTP1 ファイルシステム上に作成した物理ファイルにブロック単位でデータを順に出力します。データ長がブロック長よりも小さい場合は，データの後ろをヌル文字で埋めます。大きい場合は，エラーリターンします。

物理ファイルにブロックを出力するときは，dc_dam_create 関数，または dc_dam_iopen 関数で返されたリターン値のファイル記述子を設定します。

UAP で値を設定する引数

●fno

ブロックを出力するファイルの，ファイル記述子を設定します。

●datadr

出力するデータのアドレスを設定します。

●datalen

出力するデータの長さを設定します。設定できる範囲は 504～2147483647 です。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
正の整数		正の整数は，出力するデータの長さ（datalen で指定した値）を示します。
DCDAMER_BADF	-1603	fno に設定したファイル記述子は，正常にオープンして得られた記述子でありません。
		DAM ファイルをオープンしていません。
DCDAMER_BUFER	-1604	データ長にブロック長より大きい値を設定しています。
		出力するデータ長に設定した値は，設定できる範囲を超えています。
DCDAMER_SEQER	-1605	DAM ファイルにアクセスする関数を呼び出す順序が間違っています。
DCDAMER_PARAM_FLAGS	-1611	flags に設定した値が間違っています。
DCDAMER_IOER	-1620	出力エラーが起こりました。
DCDAMER_ACCESS	-1628	アクセスしようとした DAM ファイルは，セキュリティ機能で保護されています。dc_dam_put 関数を呼び出した UAP には，アクセス権限がありません。
DCDAMER_EOF	-1637	ファイルの終わりに達しました。
DCDAMER_NO_ACL	-1646	アクセスしようとした DAM ファイルは，セキュリティ機能で保護されています。該当するファイルに対する ACL がありません。

dc_dam_read

名称

論理ファイルからブロックの入力

形式

ANSI C，C++の形式

```
#include <dcdam.h>
int  dc_dam_read(int damfd,struct DC_DAMKEY *keyptr,int keyno,
                 char *bufadr,int bufsize,DCLONG flags)
```

K&R 版 C の形式

```
#include <dcdam.h>
int dc_dam_read(damfd,keyptr,keyno,bufadr,bufsize,flags)
int             damfd;
struct DC_DAMKEY *keyptr;
int             keyno;
char            *bufadr;
int             bufsize;
DCLONG          flags;
```

機能

指定した論理ファイルから，指定した範囲のブロックを，参照または更新目的で入力します。

- 回復対象の DAM ファイルの場合

  排他する単位は，論理ファイルをオープンしたときの指定に従います。トランザクションの範囲でない処理からでも，dc_dam_read 関数を呼び出せます。ただし，参照目的の入力に限り，排他の指定はできません。

  複数のブロックを一括して指定した場合は，複数のブロックのうちで一つでもエラーが起こると，入力バッファにはブロックを入力しないでエラーを返します。このとき要求があったすべてのブロック排他は解除されます。

  参照目的で入力したあとに，同じブロックに対して更新目的での入力要求をした場合に，この更新目的での入力でエラーが起こると，参照目的での入力の排他も解除されます。

  トランザクション途中でブロックを更新する指定（DAM サービス定義の dam_update_block_over=flush）があっても，次のような場合には，DCDAMER_JNLOV でエラーリターンする場合があります。

  - 一つのトランザクションブランチの中で DAM ファイルのブロックを更新（dc_dam_rewrite 関数）しないで，更新目的のブロック入力（dc_dam_read 関数）をして，そのブロック数が一括更新最大ブロック数（DAM サービス定義の dam_update_block オペランドの値）を超えた場合

回復対象の DAM ファイルのブロックを入力する場合は，トランザクションの処理範囲から dc_dam_read 関数を呼び出してください。

- **回復対象外の DAM ファイルの場合**

  回復対象外の DAM ファイルのブロックを入力する場合は，dc_dam_read 関数を呼び出す条件に制限はありません。

  回復対象外の DAM ファイルに対して，DAM サービス定義の dam_update_block オペランドに指定した値を超えて，更新目的の dc_dam_read 関数を呼び出した場合は，DCDAMER_ACSOV でエラーリターンします。

論理ファイルのブロックを入力するときは，dc_dam_open 関数で返されたリターン値のファイル記述子を設定します。

UAP で値を設定する引数

●damfd

ブロックを入力するファイルの，ファイル記述子を設定します。

●keyptr

参照または更新するブロックの範囲を示す構造体（DAM キー）のアドレスを設定します。構造体にはブロックの範囲を，先頭の相対ブロック番号と最後の相対ブロック番号で設定します。構造体の形式は次のとおりです。

```
struct DC_DAMKEY {
                  int fstblkno;
                  int endblkno;
                  };
```

- fstblkno

  参照または更新するブロックの，先頭の相対ブロック番号を設定します。

- endblkno

  参照または更新するブロックの，最後の相対ブロック番号を設定します。0 を設定した場合は，fstblkno で設定した相対ブロック番号のブロックだけを入力します。

●keyno

keyptr で設定する構造体の数（構造体の配列数）を設定します。

●bufadr

入力バッファのアドレスを設定します。

●bufsize

入力バッファ長を（入力ブロック長 × ブロック数）以上の値で設定します。設定できる範囲は 504～2147483647 です。

●flags

参照要求か更新要求かの種別を設定します。次の形式で設定します。

```
{DCDAM_REFERENCE | DCDAM_MODIFY}
 〔 | {DCDAM_EXCLUSIVE | DCDAM_NOEXCLUSIVE}〕
 〔 | {DCDAM_WAIT | DCDAM_NOWAIT}〕
```

- フラグ 1

  dc_dam_read 関数で入力する種別（参照目的か更新目的か）を設定します。

  DCDAM_REFERENCE…参照目的の入力。

  DCDAM_MODIFY……更新目的の入力。

- フラグ 2

  参照目的の入力の場合，排他をかけるかどうかを指定します。DCDAM_EXCLUSIVE を設定した場合は，同期点の処理まで排他をかけます。

  トランザクションの外から回復対象の DAM ファイルに参照目的でアクセスする場合は，排他する指定はできません。

  フラグ 2 を省略した場合は，DCDAM_NOEXCLUSIVE が仮定されます。

  排他をかけないで dc_dam_read 関数を呼び出した場合，dc_dam_read 関数の処理中にほかの UAP から該当するブロックが更新されることがあります。この場合，dc_dam_read 関数で入力するブロックの内容は，ほかの UAP での更新処理の状態に依存します。したがって，最新のブロックの内容を参照する場合には，必ず DCDAM_EXCLUSIVE を指定してください。

  更新目的の入力の場合，フラグ 2 に値を設定できません（強制的に DCDAM_EXCLUSIVE となります）。

  DCDAM_EXCLUSIVE…排他をします。

  DCDAM_NOEXCLUSIVE…排他をしません。

- フラグ 3

  排他エラーが起こったとき，排他解除待ちをするかどうかを設定します。この値は DCDAM_NOEXCLUSIVE と同時には設定できません。

  damfd に設定したファイル記述子の dc_dam_open 関数が，排他種別にファイル排他を設定していた場合は，ここに設定する値は意味を持ちません。

  DCDAM_WAIT…排他解除待ちをします。

  DCDAM_NOWAIT…待たないで，エラーリターンします。

どちらも設定しなかった場合は次のようになります。

- dc_dam_open 関数で DCDAM_WAIT を設定していれば，排他解除待ち
- dc_dam_open 関数で DCDAM_NOWAIT を設定，または省略している場合はエラーリターン

flags の値に設定したフラグと，排他の設定内容の対応を次に示します。

フラグ 1	フラグ 2※1	フラグ 3※2	flags の設定内容
REFERENCE	EXCLUSIVE	WAIT	参照目的，排他あり，排他エラー時は解除待ち
		NOWAIT	参照目的，排他あり，排他エラー時はエラーリターン
	NOEXCLUSIVE	×	参照目的，排他なし※3
MODIFY	×	WAIT	更新目的，排他エラー時は解除待ち
		NOWAIT	更新目的，排他エラー時はエラーリターン

（凡例）

×：設定できません。

注※1

省略した場合は，NOEXCLUSIVE が仮定されます。

注※2

省略した場合は，dc_dam_open 関数が設定した排他待ち種別の値に従います。

注※3

回復対象の DAM ファイルの場合，トランザクションの範囲でない処理から dc_dam_read 関数を呼び出すときは，フラグ 1 に DCDAM_REFERENCE を設定して，フラグ 2 に DCDAM_NOEXCLUSIVE を設定または省略したときだけ有効です。これ以外の値を設定して dc_dam_read 関数を呼び出した場合は，DCDAMER_PROTO でエラーリターンします。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	すべてのブロック入力は正常に終了しました。
DCDAMER_PROTO	-1600	dc_rpc_open 関数を呼び出していません。
		トランザクションの範囲外で，更新目的の入力，または排他を指定した参照目的の入力をしています（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。
		ユーザサービス定義の atomic_update オペランドの指定が'N'になっています（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。
		dc_dam_start 関数を呼び出していません（回復対象外の DAM ファイルにアクセスした場合だけリターンされます）。
		次に示すように，UAP を正しくリンケージしていません。 • DAM サービスの関数で TAM ファイルにアクセスする場合に使うライブラリ（-ltdam）を，不当にリンケージしています。 • トランザクション制御用オブジェクトファイルのリソースマネジャ登録が間違っています。
DCDAMER_EXCER	-1602	排他エラーが起こりました。

リターン値	リターン値（数値）	意味
DCDAMER_BADF	-1603	damfd に設定したファイル記述子は正常にオープンして得られたファイル記述子ではありません。
		DAM ファイルをオープンしていません。
DCDAMER_BUFER	-1604	すべてのブロックを入力するためには，入力バッファ長に設定した長さは不十分です。
		入力バッファ長に設定した値は，設定できる範囲を超えています。
DCDAMER_BNOER	-1606	相対ブロック番号が間違っています。
DCDAMER_NOMEM	-1607	メモリが不足しました。
DCDAMER_PARAM_KEYNO	-1609	keyno に 1 より小さい値を設定しています。
DCDAMER_PARAM_FLAGS	-1611	flags に指定した値が間違っています。
DCDAMER_JNLOV	-1613	ブロック更新した回数※が，DAM サービス定義の 1 トランザクションで更新できる最大ブロック数に指定した値を超えました（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。
DCDAMER_VERSION	-1618	UAP が，現在稼働している DAM サービスでは動作できないバージョンの DAM ライブラリと結合されています。
DCDAMER_IOER	-1620	入力エラーが起こりました。
DCDAMER_LHOLD	-1621	damfd に設定したファイル記述子のファイルが，論理閉塞されています。
DCDAMER_OHOLD	-1622	damfd に設定したファイル記述子のファイルが，障害閉塞されています。
DCDAMER_ACCESS	-1628	アクセスしようとした DAM ファイルは，セキュリティ機能で保護されています。dc_dam_read 関数を呼び出した UAP には，アクセス権限がありません。
DCDAMER_TMERR	-1629	トランザクションサービスでエラーが起こりました（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。
DCDAMER_DLOCK	-1642	デッドロックが起こりました。
DCDAMER_TIMOUT	-1643	ロックサービス定義で指定した待ち時間のタイムアウトのため，資源を確保できませんでした。
DCDAMER_LCKOV	-1645	最大同時排他要求数を超えて，排他を要求しています。
DCDAMER_ACSOV	-1648	回復対象外の DAM ファイルにアクセスできるブロック数※を超えました（回復対象外の DAM ファイルにアクセスした場合だけリターンされます）。

注※

詳細については，マニュアル「OpenTP1 システム定義」の DAM サービス定義の dam_update_block オペランドを参照してください。

注意事項

リターン値 DCDAMER_JNLOV，DCDAMER_ACSOV でリターンした場合，次のどれかの方法で処置してください。

- DAM サービス定義の dam_update_block オペランドの指定値を大きくする。
- DAM サービス定義の dam_update_block_over オペランドに error を指定して DCDAMER_JNLOV となった場合，dam_update_block_over オペランドの指定を flush に変更する。
- 更新目的で入力するブロック数を DAM サービス定義の dam_update_block オペランドの指定値以下にする。

定義を変更する場合は，マニュアル「OpenTP1 システム定義」の DAM サービス定義の dam_update_block オペランド，および dam_update_block_over オペランドを参照し，注意事項を確認してください。

dc_dam_release

名称

論理ファイルの閉塞の解除

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

dc_dam_hold 関数で論理閉塞された論理ファイル，および障害閉塞されている論理ファイルの閉塞を解除します。

UAP で値を設定する引数

●lfname

閉塞を解除するファイルの論理ファイル名を，1～8 バイトの名称で設定します。

●flags

閉塞解除の種別を設定します。

DCDAM_LOGICAL_RELEASE…論理閉塞を解除します。

DCDAM_OBSTACLE_RELEASE…障害閉塞を解除します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	lfname に設定した論理ファイルの閉塞を正常に解除しました。
DCDAMER_PROTO	-1600	dc_rpc_open 関数を呼び出していません。

リターン値	リターン値（数値）	意味
DCDAMER_PROTO	-1600	ユーザサービス定義の atomic_update オペランドの指定が'N'になっています（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。
		dc_dam_start 関数を呼び出していません（回復対象外の DAM ファイルにアクセスした場合だけリターンされます）。
		次に示すように，UAP を正しくリンケージしていません。 • DAM サービスの関数で TAM ファイルにアクセスする場合に使うライブラリ（-ltdam）を，不当にリンケージしています。 • トランザクション制御用オブジェクトファイルのリソースマネジャ登録が間違っています。
DCDAMER_UNDEF	-1601	lfname に設定した論理ファイルは定義されていません。
DCDAMER_NOMEM	-1607	メモリが不足しました。
DCDAMER_PARAM_LFNAME	-1610	lfname に設定した論理ファイル名が間違っています。
DCDAMER_PARAM_FLAGS	-1611	flags に設定した値が間違っています。
DCDAMER_VERSION	-1618	UAP が，現在稼働している DAM サービスでは動作できないバージョンの DAM ライブラリと結合されています。
DCDAMER_NOEXIST	-1619	lfname で指定した論理ファイルに対応する物理ファイルがありません。
DCDAMER_IOER	-1620	入力エラーが起こりました。
DCDAMER_NOLHOLD	-1623	lfname に設定した論理ファイルは，論理閉塞されていません。
DCDAMER_NOOHOLD	-1624	lfname に設定した論理ファイルは，障害閉塞されていません。
DCDAMER_OPENNUM	-1627	キャラクタ型スペシャルファイルのオープン数が，最大値を超えました。
DCDAMER_ACCESS	-1628	キャラクタ型スペシャルファイルにはアクセス権がありません。
		アクセスしようとした DAM ファイルは，セキュリティ機能で保護されています。dc_dam_release 関数を呼び出した UAP には，アクセス権限がありません。
DCDAMER_LFNMER	-1632	物理ファイルがキャラクタ型スペシャルファイルではありません。または，指定したスペシャルファイルに対応する装置がありません。
DCDAMER_LNOINT	-1633	lfname で指定した論理ファイルに対応する物理ファイルが，OpenTP1 ファイルシステムとして初期化されていません。
DCDAMER_ACCESSF	-1638	lfname で指定した論理ファイルに対応する物理ファイルに対するアクセス権がありません。

リターン値	リターン値（数値）	意味
DCDAMER_NO_ACL	-1646	閉塞を解除しようとした DAM ファイルは，セキュリティ機能で保護されています。該当するファイルに対する ACL がありません。

dc_dam_rewrite

名称

論理ファイルのブロックの更新

形式

ANSI C，C++の形式

```
#include <dcdam.h>
int  dc_dam_rewrite(int damfd,struct DC_DAMKEY *keyptr,
                    int keyno,
                    char *bufadr,int bufsize,DCLONG flags)
```

K&R 版 C の形式

```
#include <dcdam.h>
int  dc_dam_rewrite(damfd,keyptr,keyno,bufadr,bufsize,flags)
int             damfd;
struct DC_DAMKEY *keyptr;
int             keyno;
char            *bufadr;
int             bufsize;
DCLONG          flags;
```

機能

dc_dam_read 関数で入力したブロックを更新目的で出力します。または更新要求を取り消します。ブロックを更新するタイミングを次に示します。

- 回復対象の DAM ファイルの場合

  更新したデータは DAM サービス専用共用メモリ上に蓄えておき，トランザクションがコミットしたときに，実際にファイルを更新します。ただし，ディファード更新を指定した DAM ファイルの場合は，トランザクションのコミットとは非同期に更新されます。

- 回復対象外の DAM ファイルの場合

  dc_dam_rewrite 関数がリターンした時点で，DAM ファイルが更新されます。

複数のブロックを一括して指定したときに，指定したブロックのうちで一つでもエラーになると，処理を中断してエラーリターンします。このとき更新はしません。

論理ファイルのブロックを更新するときは，dc_dam_open 関数で返されたリターン値のファイル記述子を設定します。

UAP で値を設定する引数

●damfd

ブロックを更新するファイルの，ファイル記述子を設定します。

●keyptr

更新するブロックの範囲を示す構造体（DAM キー）のアドレスを設定します。構造体にはブロックの範囲を，先頭の相対ブロック番号と最後の相対ブロック番号で設定します。構造体の形式は次のとおりです。

```
struct DC_DAMKEY {
                   int fstblkno;
                   int endblkno;
                  };
```

・fstblkno

更新するブロックの先頭の相対ブロック番号を設定します。

・endblkno

更新するブロックの，最後の相対ブロック番号を設定します。0 を設定した場合は，fstblkno で設定した相対ブロック番号のブロックだけを更新します。

●keyno

keyptr で設定する構造体の数（構造体の配列数）を設定します。

●bufadr

更新データのアドレスを設定します。

●bufsize

更新データ長を（更新ブロック長×更新ブロック数）以上の値で設定します。設定できる範囲は 504～2147483647 です。

●flags

更新要求か更新要求の取り消しかの更新種別を，次のどちらかで設定します。

DCDAM_UPDATE

更新要求

DCDAM_CANCEL

更新要求の取り消し

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	すべてのブロック更新は正常に終了しました。
DCDAMER_PROTO	-1600	dc_rpc_open 関数を呼び出していません。
		トランザクションの範囲外で，dc_dam_rewrite 関数を呼び出しています（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。
		ユーザサービス定義の atomic_update オペランドの指定が'N'になっています（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。
		dc_dam_start 関数を呼び出していません（回復対象外の DAM ファイルにアクセスした場合だけリターンされます）。
		次に示すように，UAP を正しくリンケージしていません。 • DAM サービスの関数で TAM ファイルにアクセスする場合に使うライブラリ（-ltdam）を，不当にリンケージしています。 • トランザクション制御用オブジェクトファイルのリソースマネジャ登録が間違っています。
DCDAMER_BADF	-1603	damfd に設定したファイル記述子は正常にオープンして得られたファイル記述子ではありません。
		DAM ファイルをオープンしていません。
DCDAMER_BUFER	-1604	更新データ長（更新ブロック長×更新ブロック数）が短過ぎます。
		更新データ長に設定した値は，設定できる範囲を超えています。
DCDAMER_SEQER	-1605	更新目的の dc_dam_read 関数を呼び出していません。
DCDAMER_BNOER	-1606	相対ブロック番号が間違っています。
DCDAMER_NOMEM	-1607	メモリが不足しました。
DCDAMER_PARAM_KEYNO	-1609	keyno に 1 より小さい値を設定しています。
DCDAMER_PARAM_FLAGS	-1611	flags に設定した値が間違っています。
DCDAMER_JNLOV	-1613	ブロック更新した回数※が，DAM サービス定義の 1 トランザクションで更新できる最大ブロック数に指定した値を超えました（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。
DCDAMER_IOER	-1620	出力エラーが起こりました（回復対象外の DAM ファイルにアクセスした場合だけリターンされます）。
DCDAMER_LHOLD	-1621	damfd に設定したファイルが，論理閉塞されています。

リターン値	リターン値（数値）	意味
DCDAMER_OHOLD	-1622	damfd に設定したファイルが，障害閉塞されています。
DCDAMER_TMERR	-1629	トランザクションサービスでエラーが起こりました。
DCDAMER_BUFOV	-1641	更新データ長（更新ブロック長×更新ブロック数）が長過ぎます。

注※

詳細については，マニュアル「OpenTP1 システム定義」の DAM サービス定義の dam_update_block オペランドを参照してください。

注意事項

リターン値 DCDAMER_JNLOV でリターンした場合，次のどれかの方法で処置してください。

- DAM サービス定義の dam_update_block オペランドの指定値を大きくする。
- DAM サービス定義の dam_update_block_over オペランドに error を指定して DCDAMER_JNLOV となった場合，dam_update_block_over オペランドの指定を flush に変更する。
- 更新するブロック数を DAM サービス定義の dam_update_block オペランドの指定値以下にする。

定義を変更する場合は，マニュアル「OpenTP1 システム定義」の DAM サービス定義の dam_update_block オペランド，および dam_update_block_over オペランドを参照し，注意事項を確認してください。

dc_dam_start

名称

回復対象外 DAM ファイル使用の開始

形式

ANSI C，C++の形式

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

K&R 版 C の形式

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

機能

回復対象外の DAM ファイルを使うことを宣言します。回復対象外の DAM ファイルを使うときは，dc_dam_open 関数を呼び出す前に，dc_dam_start 関数を呼び出します。dc_dam_start 関数は，UAP プロセスごとに呼び出してください。

dc_dam_start 関数が正常に終了すると，回復対象外の DAM ファイルにアクセスする環境が設定されます。

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。回復対象外の DAM ファイルを使える状態になりました。
DCDAMER_PROTO	-1600	dc_rpc_open 関数を呼び出していません。
DCDAMER_NOMEM	-1607	メモリが不足しました。
DCDAMER_PARAM_FLAGS	-1611	flags に設定した値が間違っています。
DCDAMER_VERSION	-1618	UAP が，現在稼働している DAM サービスでは動作できない DAM ライブラリと結合されています。

リターン値	リターン値（数値）	意味
DCDAMER_STARTED	-1647	dc_dam_start 関数は，すでに呼び出しています。

dc_dam_status

名称

論理ファイルの状態の参照

形式

ANSI C，C++の形式

```
#include <dcdam.h>
int  dc_dam_status(char *lfname,struct DC_DAMSTAT *stbuf,
                   int phyfilno,DCLONG flags)
```

K&R 版 C の形式

```
#include <dcdam.h>
int  dc_dam_status(lfname,stbuf,phyfilno,flags)
char       *lfname;
struct  DC_DAMSTAT  *stbuf;
int        phyfilno;
DCLONG     flags;
```

機能

論理ファイルの現在の状態を，構造体 C_DAMSTAT にリターンします。リターンする内容を次に示します。

- 論理ファイルのブロック数
- 論理ファイルのブロック長
- 論理ファイルに対応した物理ファイル名
- 論理ファイルの現在の状態（閉塞されているかどうか）
- DAM サービス定義で指定した論理ファイルの属性
- DAM サービス定義で指定した論理ファイルのセキュリティ属性

dc_dam_status 関数は，dc_dam_open 関数で論理ファイルをオープンする前でも，オープンしたあとでも呼び出せます。

論理ファイルの状態を参照するときは，論理ファイル名を設定します。

UAP で値を設定する引数

●lfname

論理ファイル名を，1～8 バイトの名称で設定します。

●stbuf

論理ファイルの状態を受け取る構造体 DC_DAMSTAT のアドレスを設定します。構造体には，dc_dam_status 関数に設定した論理ファイルの状態が返されます。

●phyfilno

DAM サービスで使う領域です。ヌル文字（0）を設定します。

●flags

DCNOFLAGS を設定します。

OpenTP1 から値が返される引数

●stbuf

論理ファイルの状態を示す情報が，構造体 DC_DAMSTAT で返されます。構造体の形式は次のとおりです。

```
struct  DC_DAMSTAT {
                    int   st_block_len;
                    int   st_block_num;
                    char  st_file_ph_name[64];
                    char  st_file_stat;
                    char  st_file_def;
                    char  st_file_sec;
                    char  st_filler_1;
                    int   st_file_inf;
                    };
```

- st_block_len
  論理ファイルのブロック長が返されます。
- st_block_num
  論理ファイルのブロック数が返されます。
- st_file_ph_name
  論理ファイルに対応した物理ファイル名が返されます。
- st_file_stat
  論理ファイルの現在の状態が，次に示す値のどれかで返されます。
  DCDAM_ST_NOT_HOLD … 論理ファイルへアクセスできます。
  DCDAM_ST_HOLD_LOG … 論理ファイルは，論理閉塞されています。
  DCDAM_ST_HOLD_OBS … 論理ファイルは，障害閉塞されています。
  DCDAM_ST_HOLD_REQ … 論理ファイルへ閉塞要求中です。
- st_file_def
  DAM サービス定義で指定した論理ファイルの属性が，次に示す値のどれかで返されます。

DCDAM_ST_QUICK…ディファード更新処理の対象でない DAM ファイルです。
DCDAM_ST_DEFERRED… ディファード更新処理対象の DAM ファイルです。
DCDAM_ST_NORECOVER… 回復対象外の DAM ファイルです。
DCDAM_ST_CACHELESS…キャッシュレスアクセス指定の回復対象外の DAM ファイルです。

- st_file_sec
  DAM サービス定義で指定した論理ファイルのセキュリティ属性が，次に示す値のどれかで返されます。
  DCDAM_ST_NON… セキュリティの指定はありません。
  DCDAM_ST_SEC… セキュリティの指定があります。
- st_filler_1
  予備領域 1（ヌル文字（0）が設定されます）。
- st_file_inf
  予備領域 2（-1 が設定されます）。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	構造体 DC_DAMSTAT に，論理ファイルの状態を正常に設定しました。
DCDAMER_PROTO	-1600	dc_rpc_open 関数を呼び出していません。
		ユーザサービス定義の atomic_update オペランドの指定が'N'になっています（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。
		dc_dam_start 関数を呼び出していません（回復対象外の DAM ファイルにアクセスした場合だけリターンされます）。
DCDAMER_UNDEF	-1601	lfname に設定した論理ファイル名は定義されていません。
DCDAMER_NOMEM	-1607	メモリが不足しました。
DCDAMER_PARAM_LFNAME	-1610	lfname に設定した論理ファイル名が間違っています。
DCDAMER_PARAM_FLAGS	-1611	flags に設定した値が間違っています。
DCDAMER_PARAM_ERROR	-1612	引数に設定した値が間違っています。
		stbuf に設定した値が間違っています。
DCDAMER_VERSION	-1618	UAP が，現在稼働している DAM サービスでは動作できないバージョンの DAM ライブラリと結合されています。
DCDAMER_ACCESS	-1628	状態を参照しようとした DAM ファイルは，セキュリティ機能で保護されています。dc_dam_status 関数を呼び出した UAP には，アクセス権限がありません。

リターン値	リターン値（数値）	意味
DCDAMER_NO_ACL	-1646	状態を参照しようとした DAM ファイルは，セキュリティ機能で保護されています。該当するファイルに対する ACL がありません。

注意事項

dc_dam_status 関数を呼び出すと，DAM サービスは情報を取得するための排他制御をします。そのため，dc_dam_status 関数を頻繁に使うと，排他解除待ち時間が発生してスループットが低下することがあります。オンライン中に DAM ファイルの状態を参照するのは，必要最小限にしてください。

dc_dam_write

名称

論理ファイルへブロックの出力

形式

ANSI C，C++の形式

```
#include <dcdam.h>
int  dc_dam_write(int damfd,struct DC_DAMKEY *keyptr,int keyno,
                  char *bufadr,int bufsize,DCLONG flags)
```

K&R 版 C の形式

```
#include <dcdam.h>
int  dc_dam_write(damfd,keyptr,keyno,bufadr,bufsize,flags)
int             damfd;
struct DC_DAMKEY *keyptr;
int             keyno;
char            *bufadr;
int             bufsize;
DCLONG          flags;
```

機能

指定したブロックを出力します。ブロックを出力するタイミングを次に示します。

- 回復対象の DAM ファイルの場合

  更新したデータは DAM サービス専用共用メモリ上に蓄えておき，トランザクションがコミットしたときに，実際にファイルを更新します。ただし，ディファード出力を指定した DAM ファイルの場合は，トランザクションのコミットとは非同期に出力されます。

- 回復対象外の DAM ファイルの場合

  dc_dam_write 関数がリターンした時点で，DAM ファイルが出力されます。

複数のブロックを一括して出力要求したときに，指定したブロックのうちで一つでもエラーになると，処理を中断してエラーリターンします。このときはブロックを出力しません。

論理ファイルのブロックを出力するときは，dc_dam_open 関数で返されたリターン値のファイル記述子を設定します。

UAP で値を設定する引数

●damfd

ブロックを出力するファイルの，ファイル記述子を設定します。

●keyptr

出力するブロックの範囲を示す構造体（DAM キー）のアドレスを設定します。構造体にはブロックの範囲を，先頭の相対ブロック番号と最後の相対ブロック番号で設定します。構造体の形式は次のとおりです。

```
struct DC_DAMKEY {
                  int fstblkno;
                  int endblkno;
                  };
```

- fstblkno

  出力するブロックの先頭の相対ブロック番号を設定します。

- endblkno

  出力するブロックの，最後の相対ブロック番号を設定します。0 を設定した場合は，fstblkno で設定した相対ブロック番号のブロックだけを出力します。

●keyno

keyptr で設定する構造体の数（構造体の配列数）を設定します。

●bufadr

更新データのアドレスを設定します。

●bufsize

出力データ長を（出力ブロック長×出力ブロック数）以上の値で設定します。設定できる範囲は 504～2147483647 です。

●flags

排他エラーが起こったとき，排他解除待ちをするかどうかを設定します。

DCDAM_WAIT… 排他解除待ちをします。

DCDAM_NOWAIT… 待たないで，エラーリターンします。

DCNOFLAGS… dc_dam_open 関数の flags に設定した値に従います。

DCNOFLAGS を設定した場合は次のようになります。

- dc_dam_open 関数で DCDAM_WAIT を設定していれば，排他解除待ち
- dc_dam_open 関数で DCDAM_NOWAIT を設定，または省略している場合はエラーリターン

damfd に設定したファイル記述子の dc_dam_open 関数が，排他種別にファイル排他を設定していた場合は，ここに設定する値は意味を持ちません。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	すべてのブロック出力は正常に終了しました。
DCDAMER_PROTO	-1600	dc_rpc_open 関数を呼び出していません。
		トランザクションの範囲外で，dc_dam_write 関数を呼び出しています（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。
		ユーザサービス定義の atomic_update オペランドの指定が'N'になっています（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。
		dc_dam_start 関数を呼び出していません（回復対象外の DAM ファイルにアクセスした場合だけリターンされます）。
		次に示すように，UAP を正しくリンケージしていません。 • DAM サービスの関数で TAM ファイルにアクセスする場合に使うライブラリ（-ltdam）を，不当にリンケージしています。 • トランザクション制御用オブジェクトファイルのリソースマネジャ登録が間違っています。
DCDAMER_EXCER	-1602	排他エラーが起こりました。
DCDAMER_BADF	-1603	damfd に設定したファイル記述子は正常にオープンして得られたファイル記述子ではありません。
		DAM ファイルをオープンしていません。
DCDAMER_BUFER	-1604	出力データ長（出力ブロック長×出力ブロック数）が短過ぎます。
		出力データ長に設定した値は，設定できる範囲を超えています。
DCDAMER_SEQER	-1605	関数を呼び出す順序が間違っています。
DCDAMER_BNOER	-1606	相対ブロック番号が間違っています。
DCDAMER_NOMEM	-1607	メモリが不足しました。
DCDAMER_PARAM_KEYNO	-1609	keyno に 1 より小さい値を設定しています。
DCDAMER_PARAM_FLAGS	-1611	flags に設定した値が間違っています。
DCDAMER_JNLOV	-1613	ブロック更新した回数※が，DAM サービス定義の 1 トランザクションで更新できる最大ブロック数に指定した値を超えました（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。
DCDAMER_IOER	-1620	出力エラーが起こりました（回復対象外の DAM ファイルにアクセスした場合だけリターンされます）。

リターン値	リターン値（数値）	意味
DCDAMER_LHOLD	-1621	damfd に設定したファイルが，論理閉塞されています。
DCDAMER_OHOLD	-1622	damfd に設定したファイルが，障害閉塞されています。
DCDAMER_ACCESS	-1628	アクセスしようとした DAM ファイルは，セキュリティ機能で保護されています。dc_dam_write 関数を呼び出した UAP には，アクセス権限がありません。
DCDAMER_TMERR	-1629	トランザクションサービスでエラーが起こりました（回復対象の DAM ファイルにアクセスした場合だけリターンされます）。
DCDAMER_BUFOV	-1641	出力データ長（出力ブロック長×出力ブロック数）が長過ぎます。
DCDAMER_DLOCK	-1642	デッドロックが起こりました。
DCDAMER_TIMOUT	-1643	ロックサービス定義で指定した待ち時間のタイムアウトのため，資源を確保できませんでした。
DCDAMER_LCKOV	-1645	最大同時排他要求数を超えて，排他を要求しています。
DCDAMER_ACSOV	-1648	アクセスできる最大ブロック数※を超えました（回復対象外の DAM ファイルにアクセスした場合だけリターンされます）。

注※

詳細については，マニュアル「OpenTP1 システム定義」の DAM サービス定義の dam_update_block オペランドを参照してください。

注意事項

リターン値 DCDAMER_JNLOV，DCDAMER_ACSOV がリターンした場合，次のどれかの方法で処置してください。

- DAM サービス定義の dam_update_block オペランドの指定値を大きくする。
- DAM サービス定義の dam_update_block_over オペランドに error を指定して DCDAMER_JNLOV となった場合，dam_update_block_over オペランドの指定を flush に変更する。
- 出力するブロック数は，DAM サービス定義の dam_update_block オペランドの指定値以下にする。
- dc_dam_rewrite 関数で更新されていないブロックがある場合は，dc_dam_write 関数を呼び出す前に更新する。

定義を変更する場合は，マニュアル「OpenTP1 システム定義」の DAM サービス定義の dam_update_block オペランド，および dam_update_block_over オペランドを参照し，注意事項を確認してください。

IST サービス（dc_ist_～）

IST テーブルへアクセスする関数について説明します。IST サービスの関数を次に示します。

- dc_ist_close － IST テーブルのクローズ
- dc_ist_open － IST テーブルのオープン
- dc_ist_read － IST テーブルからレコードの入力
- dc_ist_write － IST テーブルへレコードの出力

IST サービスの関数（dc_ist_～）は，TP1/Server Base の UAP でだけ使えます。TP1/LiNK の UAP では，IST サービスの関数は使えません。

dc_ist_close

名称

IST テーブルのクローズ

形式

ANSI C，C++の形式

```
#include <dcist.h>
int  dc_ist_close(int istid,DCLONG flags)
```

K&R 版 C の形式

```
#include <dcist.h>
int  dc_ist_close(istid,flags)
int     istid;
DCLONG  flags;
```

機能

指定した IST テーブルをクローズします。

UAP で値を設定する引数

●istid

クローズする IST テーブルの，テーブル記述子を設定します。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	IST テーブルを正常にクローズしました。
DCISTER_PROTO	-3800	IST テーブルへアクセスする関数を呼び出す順序が間違っています。
DCISTER_BADID	-3803	istid に設定したテーブル記述子は正常にオープンして得られたテーブル記述子ではありません。
		IST テーブルをオープンしていません。
DCISTER_PARAM_FLAGS	-3811	flags に設定した値が間違っています。

dc_ist_open

名称

IST テーブルのオープン

形式

ANSI C，C++の形式

```
#include <dcist.h>
int  dc_ist_open(char *istname,DCLONG flags)
```

K&R 版 C の形式

```
#include <dcist.h>
int  dc_ist_open(istname,flags)
char    *istname;
DCLONG  flags;
```

機能

指定した IST テーブルをオープンします。IST テーブルを正常にオープンできると，テーブル記述子がリターンされます。

UAP で値を設定する引数

●istname

オープンする IST テーブル名を，1～8 バイトの名称で設定します。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
0 または正の整数		0 または正の整数は，テーブル記述子を示します。
DCISTER_PROTO	-3800	IST テーブルへアクセスする関数を呼び出す順序が間違っています。
DCISTER_UNDEF	-3801	istname に設定した IST テーブル名は定義されていません。
DCISTER_NOMEM	-3807	メモリが不足しました。
DCISTER_OPENED	-3808	istname に設定した IST テーブル名はオープン済みです。

リターン値	リターン値（数値）	意味
DCISTER_PARAM_TBLNAME	-3810	IST テーブル名に設定した値の長さが間違っています。
DCISTER_PARAM_FLAGS	-3811	flags に設定した値が間違っています。

dc_ist_read

名称

IST テーブルからレコードの入力

形式

ANSI C，C++の形式

```
#include <dcist.h>
int  dc_ist_read(int istid,struct DC_ISTKEY *keyptr,int keyno,
                 char *bufadr,int bufsize,DCLONG flags)
```

K&R 版 C の形式

```
#include <dcist.h>
int  dc_ist_read(istid,keyptr,keyno,bufadr,bufsize,flags)
int     istid;
struct  DC_ISTKEY  *keyptr;
int     keyno;
char    *bufadr;
int     bufsize;
DCLONG   flags;
```

機能

指定した IST テーブルから指定した範囲のレコードを，参照の目的で入力します。複数のレコードを一括して指定した場合に，指定したレコードのうち一つでもエラーが起こると，入力バッファにはレコードを入力しないで dc_ist_read 関数はエラーリターンします。

IST テーブルからレコードを入力するときは，dc_ist_open 関数で返されたリターン値のテーブル記述子を設定します。

UAP で値を設定する引数

●istid

アクセスする IST テーブルの，テーブル記述子を設定します。

●keyptr

参照するレコードの相対レコード番号の範囲を示す構造体（IST キー）のアドレスを設定します。構造体には，レコードの範囲を先頭と最後の相対ブロック番号で設定します。構造体の形式は次のとおりです。

```
struct  DC_ISTKEY {
                  int  fstrecno;
                  int  endrecno;
                  };
```

- fstrecno
  アクセスするレコードの，先頭の相対レコード番号を設定します。
- endrecno
  アクセスするレコードの，最後の相対レコード番号を設定します。0 を設定した場合は，fstrecno で設定した相対レコード番号のレコードだけを入力します。

●keyno

keyptr で設定する構造体の数（構造体の配列数）を設定します。

●bufadr

入力バッファのアドレスを設定します。

●bufsize

入力バッファ長を設定します。入力バッファ長には，（入力レコード長 × 入力レコード数）以上の値を設定してください。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	指定したレコードは，すべて正常に入力しました。
DCISTER_PROTO	-3800	IST テーブルへアクセスする関数を呼び出す順序が間違っています。
DCISTER_BADID	-3803	istid に設定したテーブル記述子は正常にオープンして得られたテーブル記述子ではありません。
		IST テーブルをオープンしていません。
DCISTER_BUFER	-3804	すべてのレコードを入力するには，bufsize に設定した入力バッファ長が足りません。
DCISTER_RNOER	-3806	相対レコード番号が間違っています。
DCISTER_NOMEM	-3807	メモリが不足しました。
DCISTER_PARAM_KEYNO	-3809	keyno に設定した値が，1 未満です。
DCISTER_PARAM_FLAGS	-3811	flags に設定した値が間違っています。

dc_ist_write

名称

IST テーブルへレコードの出力

形式

ANSI C，C++の形式

```
#include <dcist.h>
int  dc_ist_write(int istid,struct DC_ISTKEY *keyptr,int keyno,
                  char *bufadr,int bufsize,DCLONG flags)
```

K&R 版 C の形式

```
#include <dcist.h>
int  dc_ist_write(istid,keyptr,keyno,bufadr,bufsize,flags)
int     istid;
struct  DC_ISTKEY *keyptr;
int     keyno;
char    *bufadr;
int     bufsize;
DCLONG   flags;
```

機能

指定した範囲のレコードを，IST テーブルに出力します。複数のレコードを一括して指定した場合に，指定したレコードのうち一つでもエラーが起こると，出力バッファにはレコードを出力しないで dc_ist_write 関数はエラーリターンします。

dc_ist_write 関数が正常に終了すると，自ノードのレコードの内容が更新されます。他ノードの IST テーブルへは，この関数が正常に終了してから，時間をおいて更新されます。

IST テーブルへレコードを出力するときは，dc_ist_open 関数で返されたリターン値のテーブル記述子を設定します。

UAP で値を設定する引数

●istid

アクセスする IST テーブルの，テーブル記述子を設定します。

●keyptr

出力するレコードの相対レコード番号の範囲を示す構造体（IST キー）のアドレスを設定します。構造体には，レコードの範囲を先頭と最後の相対ブロック番号で設定します。構造体の形式は次のとおりです。

```
struct  DC_ISTKEY {
                  int  fstrecno;
```

```
                int  endrecno;
                };
```

- fstrecno
  アクセスするレコードの，先頭の相対レコード番号を設定します。
- endrecno
  アクセスするレコードの，最後の相対レコード番号を設定します。0 を設定した場合は，fstrecno で設定した相対レコード番号のレコードだけを出力します。

●keyno

keyptr で設定する構造体の数（構造体の配列数）を設定します。

●bufadr

出力する更新データがあるバッファのアドレスを設定します。

●bufsize

出力バッファ長を設定します。出力バッファ長には，（出力レコード長 × 出力レコード数）の値を設定してください。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	指定したレコードは，すべて正常に出力しました。
DCISTER_PROTO	-3800	IST テーブルへアクセスする関数を呼び出す順序が間違っています。
DCISTER_BADID	-3803	istid に設定したテーブル記述子は正常にオープンして得られたテーブル記述子ではありません。
		IST テーブルをオープンしていません。
DCISTER_BUFER	-3804	すべてのレコードを出力するには，bufsize に設定した出力バッファ長が足りません。
DCISTER_RNOER	-3806	相対レコード番号が間違っています。
DCISTER_NOMEM	-3807	メモリが不足しました。
DCISTER_PARAM_KEYNO	-3809	keyno に設定した値が，1 未満です。
DCISTER_PARAM_FLAGS	-3811	flags に設定した値が間違っています。

リターン値	リターン値（数値）	意味
DCISTER_BUFOV	-3841	出力バッファ長が，出力するすべてのレコード長の合計よりも長過ぎます。

ユーザジャーナルの取得（dc_jnl_～）

ユーザジャーナルを取得する関数について説明します。ユーザジャーナルの取得の関数を次に示します。

- dc_jnl_ujput － ユーザジャーナルの取得

ユーザジャーナルの取得の関数（dc_jnl_～）は，TP1/Server Base の UAP でだけ使えます。TP1/LiNK の UAP では，ユーザジャーナルの関数は使えません。

dc_jnl_ujput

名称

ユーザジャーナルの取得

形式

ANSI C，C++の形式

```
#include <dcjnl.h>
int  dc_jnl_ujput(char *data,DCULONG dsize,DCLONG ujcode,
                  DCLONG flags)
```

K&R 版 C の形式

```
#include <dcjnl.h>
int dc_jnl_ujput(data,dsize,ujcode,flags)
char     *data;
DCULONG  dsize;
DCLONG   ujcode;
DCLONG   flags;
```

機能

UAP の履歴情報であるユーザジャーナル（UJ）をシステムジャーナルファイル（system_jnl_file）に取得します。dc_jnl_ujput 関数 1 回の呼び出しで取得する UJ の単位を UJ レコードといいます。

dc_jnl_ujput 関数を呼び出しても，すぐにはシステムジャーナルファイルに出力されません。ジャーナルバッファに空きがなくなったときか，トランザクション処理が正常終了した同期点で，システムジャーナルファイルに UJ レコードが出力されます。

dc_jnl_ujput 関数は，dc_rpc_open 関数の呼び出し後から dc_rpc_close 関数の呼び出しまでの間で呼び出せます。出力済みの UJ レコードは，dc_jnl_ujput 関数を呼び出したトランザクションで障害が発生しても，ロールバック（部分回復）で無効にできません。dc_jnl_ujput 関数を呼び出したトランザクションをロールバックしても，UJ レコードはシステムジャーナルファイルへ出力されます。

UAP で値を設定する引数

●data

取得する UAP の履歴情報を設定します。UAP の履歴情報として有効になるデータは，dsize に設定した長さです。

●dsize

取得する UAP の履歴情報の長さを設定します。設定できる長さは 1 から（取得先のシステムジャーナルファイルサービス定義の jnl_max_datasize オペランドの値－8）までです。

●ujcode

UJ コードを，0～255 の値で設定します。

●flags

UJ レコードを取得する時点で，システムジャーナルファイルに UJ レコードを出力するかどうかを，次に示す形式で設定します。

DCJNL_FLUSH

UJ レコードを取得する時点で，システムジャーナルファイルに UJ レコードを出力します。トランザクション内で UJ レコードが取得されている場合，この設定は無視されます。

DCNOFLAGS

UJ レコードを取得する時点では，システムジャーナルファイルに UJ レコードを出力しません。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCJNLER_PARAM	-1101	パラメタの形式が間違っています。
DCJNLER_SHORT	-1102	ユーザジャーナルの長さ（dsize の値）に，0 以下のデータ長を設定しています。
DCJNLER_LONG	-1103	ユーザジャーナルの長さ（dsize の値）に，設定できる範囲以上の値を設定しています。
DCJNLER_PROTO	-1105	dc_rpc_open 関数が呼び出されていません。または，該当するシステムの実行環境がジャーナルファイルレスモードのため，dc_jnl_ujput 関数が動作できません。

注意事項

トランザクション外 UJ レコードは，ジャーナルバッファに空きがなくなったとき，またはほかのアプリケーションのトランザクションが正常終了した同期点（コミットした時点）で，システムジャーナルファイルに出力されます。トランザクションが発生しないアプリケーションで UJ レコードを取得する場合は，flags に DCJNL_FLUSH を設定した dc_jnl_ujput 関数を，適切なタイミングで呼び出してください。

資源の排他制御（dc_lck_～）

任意のユーザファイルを排他制御する関数について説明します。資源の排他制御の関数を次に示します。

- dc_lck_get － 資源の排他
- dc_lck_release_all － 全資源の排他の解除
- dc_lck_release_byname － 資源名称を指定した排他の解除

資源の排他制御の関数（dc_lck_～）は，TP1/Server Base の UAP でだけ使えます。TP1/LiNK の UAP では，資源の排他制御の関数は使えません。

dc_lck_get

名称

資源の排他

形式

ANSI C，C++の形式

```
#include <dclck.h>
int  dc_lck_get(char *name,DCLONG lockmode,DCLONG ownerflag,
                DCLONG flags)
```

K&R 版 C の形式

```
#include <dclck.h>
int  dc_lck_get(name,lockmode,ownerflag,flags)
char      *name;
DCLONG    lockmode;
DCLONG    ownerflag;
DCLONG    flags;
```

機能

UAP で使う資源の排他を指定します。排他の管理は，OpenTP1 のトランザクションマネジャで管理するグローバルトランザクション単位で処理されます。

ここで指定した排他は，排他を解除する関数（dc_lck_release_all 関数，dc_lck_release_byname 関数）を呼び出すか，dc_lck_get 関数を呼び出したグローバルトランザクションの同期点取得後に解除されます。

UAP で値を設定する引数

●name

排他する資源の名称を，16 バイトの英数字文字列で設定します。ここで設定した資源名称を基に，ロックサービスで排他を管理します。16 バイト未満で，ヌル文字が出た場合は，そこまでの値を資源名称と見なします。16 バイトを超えた値が設定された場合は，16 バイト目までを資源名称と見なして，超えた部分を切り捨てます。

ロックサービスでは，文字列の内容についてはチェックしません。論理的に正しい名称を設定してください。資源名称に英数字以外の値を使った場合は，デッドロック情報，タイムアウト情報，および lckls コマンドの表示が乱れることがあります。

●lockmode

排他制御モードを設定します。排他制御モードは次のどちらか一方を設定します。重複して設定できません。

DCLCK_PR

資源を参照します。ほかの UAP には参照だけを許可し，更新は禁止します。

DCLCK_EX

資源を更新します。ほかの UAP には参照も更新も禁止します。

●ownerflag

DCLCK_OWNER_MIGRATE を設定します。

●flags

資源の排他に関するフラグを設定します。設定できる値を次に示します。

DCLCK_WAIT

ほかの UAP との資源を競合した場合に，資源の解放待ちにします。競合した場合にこのフラグが設定されてないときは，エラーリターンします。

DCLCK_TEST

資源が使えるかどうかをテストするときに設定します。このフラグを設定したときは，dc_lck_get 関数が正常終了しても，name に設定した資源は確保されていないので注意してください。

DCNOFLAGS

フラグを設定しません。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCLCKER_PARAM	-401	引数に設定した値が間違っています。
DCLCKER_WAIT	-450	ほかの UAP が，name に名称を設定した資源を使っています。
DCLCKER_DLOCK	-452	デッドロックが起こりました。
DCLCKER_TIMOUT	-453	OpenTP1 のロックサービス定義で指定した待ち時間のタイムアウトで，資源を確保できませんでした。
DCLCKER_MEMORY	-454	排他制御用のテーブルが不足しています。
DCLCKER_OUTOFTRN	-455	トランザクション処理でない UAP から指定しています。
DCLCKER_VERSION	-457	OpenTP1 のライブラリとロックサービスのバージョンが一致していません。

dc_lck_release_all

名称

全資源の排他の解除

形式

ANSI C，C++の形式

```
#include <dclck.h>
int  dc_lck_release_all(DCLONG ownerflag,
                        DCLONG flags)
```

K&R 版 C の形式

```
#include <dclck.h>
int  dc_lck_release_all(ownerflag,flags)
DCLONG    ownerflag;
DCLONG    flags;
```

機能

dc_lck_get 関数で指定した資源の排他をすべて解除します。同期点取得前に排他を解除するときに，dc_lck_release_all 関数を呼び出します。

排他をしたグローバルトランザクションが終了したときに，OpenTP1 のロックサービスによって自動的に排他は解除されます。このときは，UAP で排他を解除する必要はありません。

UAP で値を設定する引数

●ownerflag

DCLCK_OWNER_MIGRATE を設定します。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCLCKER_PARAM	-401	引数に設定した値が間違っています。
DCLCKER_OUTOFTRN	-455	トランザクション処理でない UAP から dc_lck_release_all 関数を呼び出しています。

リターン値	リターン値（数値）	意味
DCLCKER_NOTHING	-456	この関数を呼び出したトランザクションでは，資源を確保していません。
DCLCKER_VERSION	-457	OpenTP1 のライブラリとロックサービスのバージョンが一致していません。

dc_lck_release_byname

名称

資源名称を指定した排他の解除

形式

ANSI C，C++の形式

```
#include <dclck.h>
int  dc_lck_release_byname(char *name,DCLONG ownerflag,DCLONG flags)
```

K&R 版 C の形式

```
#include <dclck.h>
int  dc_lck_release_byname(name,ownerflag,flags)
char      *name;
DCLONG    ownerflag;
DCLONG    flags;
```

機能

dc_lck_get 関数で指定した資源の排他を，資源名称を指定して解除します。同期点取得前に排他を解除するときに，dc_lck_release_byname 関数を呼び出します。

排他をしたグローバルトランザクションが終了したときに，ロックサービスによって自動的に排他は解除されます。このときは，UAP で排他を解除する必要はありません。

UAP で値を設定する引数

●name

排他の指定を解除する資源名称を設定します。資源名称は dc_lck_get 関数で設定した名称と同じ値を設定してください。

●ownerflag

DCLCK_OWNER_MIGRATE を設定します。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCLCKER_PARAM	-401	引数に設定した値が間違っています。
DCLCKER_OUTOFTRN	-455	トランザクション処理でない UAP から dc_lck_release_byname 関数を呼び出しています。
DCLCKER_NOTHING	-456	解除を指定した資源名称に該当する資源がありません。
DCLCKER_VERSION	-457	OpenTP1 のライブラリとロックサービスのバージョンが一致していません。

監査ログの出力（dc_log_audit_～）

UAP から監査ログを出力する関数について説明します。監査ログを出力する関数を次に示します。

- dc_log_audit_print － 監査ログの出力

dc_log_audit_print

名称

監査ログの出力

形式

ANSI C，C++の形式

```
#include <dclog.h>
int  dc_log_audit_print(char *msgid,char *compid,DCLONG ctgry,
                        DCLONG result,DCLONG op,char *msg,DCLONG flags)
```

K&R 版 C の形式

```
#include <dclog.h>
int  dc_log_audit_print(msgid,compid,ctgry,result,op,msg,flags)
char      *msgid;
char      *compid;
DCLONG    ctgry;
DCLONG    result;
DCLONG    op;
char      *msg;
DCLONG    flags;
```

機能

引数に設定した情報に，OpenTP1 でヘッダ情報，通番，日時，発生プログラム名，発生プロセス ID，発生場所，サブジェクト識別情報，オブジェクト情報，オブジェクトロケーション情報，リクエスト送信元ホスト，ロケーション識別情報を付けて，監査ログファイルに出力します。出力する監査ログの発生プログラム名は OpenTP1 です。監査ログの出力処理でエラーが発生した場合，エラーメッセージを標準エラー出力および syslog に出力します。

OpenTP1 では，dc_log_audit_print 関数で使うメッセージ ID 用に，34000 から 34999 までの範囲の番号を割り当てています。UAP から出力するメッセージ ID の番号には，34000 から 34999 までの範囲の値を付与してください。

監査ログの出力項目については，マニュアル「OpenTP1 プログラム作成の手引」を参照してください。

UAP で値を設定する引数

●msgid

監査ログごとに付けられる識別子（メッセージ ID）を設定します。「KFCAn1n2n3n4n5-x」の形式（11 文字）で，最後にヌル文字を付けて設定します。UAP から出力する通番（n1n2n3n4n5 の部分）には，34000 から 34999 までの間の数値を設定します。x の部分には，出力する監査ログの内容によってメッセージの種類（「E」，「W」，または「I」）を指定してください。

●compid

dc_log_audit_print 関数を呼び出した UAP を識別する任意の値（要求元プログラム ID）を設定します。使用できる文字は，数字，英字，または記号の 2 文字で，最後はヌル文字で終わらせて設定します。監査ログには，先頭にアスタリスク（*）が付いて，「*AA」の形式で出力されます（AA は compid で指定した文字列です)。

●ctgry

監査ログに出力する「監査事象種別」の値を設定します。次のどれかを設定します。

DCLOG_CTG_STARTSTOP…「起動・停止」の監査事象を表します。

DCLOG_CTG_AUTH…「識別・認証」の監査事象を表します。

DCLOG_CTG_ACCESS…「アクセス制御」の監査事象を表します。

DCLOG_CTG_CONFIG…「構成定義」の監査事象を表します。

DCLOG_CTG_FAIL…「障害」の監査事象を表します。

DCLOG_CTG_LINK…「リンク状態」の監査事象を表します。

DCLOG_CTG_EXTERNAL…「外部サービス」の監査事象を表します。

DCLOG_CTG_CONTENT…「重要情報アクセス」の監査事象を表します。

DCLOG_CTG_MAINTAIN…「保守」の監査事象を表します。

DCLOG_CTG_ANORMALY…「異常事象」の監査事象を表します。

DCLOG_CTG_MANAGE…「管理動作」の監査事象を表します。

監査事象種別の詳細については，マニュアル「OpenTP1 運用と操作」を参照してください。

●result

監査ログに出力する「監査事象結果」の値を設定します。次のどれかを設定します。

DCLOG_RES_SUCCESS…事象の成功を表します。

DCLOG_RES_FAIL…事象の失敗を表します。

DCLOG_RES_OCCUR…成功または失敗の分類がない事象の発生を表します

●op

監査ログに出力する「動作情報」の値を設定します。ctgry に指定した「監査事象種別」と対応した，次の予約語のどれかを設定します。NULL を指定した場合はこの項目は出力されません。

表 2-1　監査事象種別と予約語の対応

監査事象種別	予約語	意味
DCLOG_CTG_STARTSTOP（起動・停止）	DCLOG_OP_START	開始・起動
	DCLOG_OP_STOP	終了・停止
DCLOG_CTG_AUTH（識別・認証）	DCLOG_OP_LOGIN	ログイン
	DCLOG_OP_LOGOUT	ログアウト
	DCLOG_OP_LOGON	ログオン
	DCLOG_OP_LOGOFF	ログオフ
	DCLOG_OP_DISABLE	アカウントの無効化
DCLOG_CTG_ACCESS（アクセス制御）	DCLOG_OP_ENFORCE	実施
DCLOG_CTG_CONFIG（構成定義）	DCLOG_OP_REFER	参照
	DCLOG_OP_ADD	追加
	DCLOG_OP_UPDATE	更新
	DCLOG_OP_DELETE	削除
DCLOG_CTG_FAIL（障害）	DCLOG_OP_OCCUR	発生
DCLOG_CTG_LINK（リンク状態）	DCLOG_OP_UP	リンク活性
	DCLOG_OP_DOWN	リンク非活性
DCLOG_CTG_EXTERNAL（外部サービス）	DCLOG_OP_REQ	要求
	DCLOG_OP_RES	応答
	DCLOG_OP_SEND	発信
	DCLOG_OP_RECV	受信
DCLOG_CTG_CONTENT（重要情報アクセス）	DCLOG_OP_REFER	参照
	DCLOG_OP_ADD	追加
	DCLOG_OP_UPDATE	更新
	DCLOG_OP_DELETE	削除
DCLOG_CTG_MAINTAIN（保守）	DCLOG_OP_INSTALL	インストール
	DCLOG_OP_UNINSTALL	アンインストール
	DCLOG_OP_UPDATE	更新（アップデート）
	DCLOG_OP_BACKUP	バックアップ
	DCLOG_OP_MAINTAIN	保守作業
DCLOG_CTG_ANORMALY	DCLOG_OP_OCCUR	発生

監査事象種別	予約語	意味
（異常事象）	DCLOG_OP_OCCUR	発生
DCLOG_CTG_MANAGE（管理動作）	DCLOG_OP_INVOKE	（管理者の）呼び出し
	DCLOG_OP_NOTIFY	（管理者への）通知

●msg

監査ログに出力する「自由記述」の値を設定した領域のアドレスを指定します。NULL を指定した場合はこの項目を出力しません。

使用できる文字は数字，英字，記号，スペース，引用符（"），およびコンマ（,）です。最大 1024 文字で，最後はヌル文字で終わらせて設定します。このヌル文字は文字列の長さに数えません。

msg の表示内容には指定した情報の先頭と末尾に引用符（"）が付与されます。また，文字列中に引用符（"）がある場合は，その文字の前に引用符（"）が付与されます。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DCLOG_AUDIT_OFF	1	監査ログの出力が無効になっています。次に示すことが考えられます。 • ログサービス定義の log_audit_out オペランドに N を指定しているか，または log_audit_out オペランドの指定がありません。 • ユーザサービス定義，またはユーザサービスデフォルト定義のどちらかで log_audit_out_suppress オペランドに Y が指定されています。
		次のどれかの定義ファイルで，msgid で指定したメッセージ ID が log_audit_message オペランドに指定されていません。 • ログサービス定義 • ユーザサービス定義 • ユーザサービスデフォルト定義
		規定された範囲外のメッセージを指定しています。
DC_OK	0	正常に終了しました。
DCLOGER_PARAM_ARGS	-1900	引数に設定した値が間違っています。
DCLOGER_DEFFILE	-1904	定義解析に失敗しました。
DCLOGER_PROTO	-1999	dc_rpc_open 関数が発行されていません。

リターン値	リターン値（数値）	意味
DCLOGER_FATAL	-1997	上記以外のエラーが発生しました。

注意事項

dc_rpc_mainloop 関数が終了したあとで dc_log_audit_print 関数を呼び出した場合，リクエスト送信元ホストには，最後に要求した dc_rpc_call 関数または dc_rpc_call_to 関数発行元ノードのアドレスが取得されます。

メッセージログの出力（dc_log～）

UAP からメッセージログを出力する関数について説明します。メッセージログの出力の関数を次に示します。

- dc_logprint － メッセージログの出力

メッセージログの出力の関数（dc_log_～）は，TP1/Server Base と TP1/LiNK のどちらの UAP でも使えます。

dc_logprint

名称

メッセージログの出力

形式

ANSI C，C++の形式

```
#include <dclog.h>
int  dc_logprint(char *msgid,char *pgm_id,
                 char *string,char *info,
                 DCLONG color,DCLONG flags)
```

K&R 版 C の形式

```
#include <dclog.h>
int  dc_logprint(msgid,pgm_id,string,info,color,flags)
char      *msgid;
char      *pgm_id;
char      *string;
char      *info;
DCLONG    color;
DCLONG    flags;
```

機能

引数に設定した文字列に，OpenTP1 で行ヘッダ，OpenTP1 識別子，日時，要求元ノード名，要求元プログラム ID，メッセージ ID を付けて，メッセージログファイルに出力します。

OpenTP1 では，dc_logprint 関数で使うメッセージ ID 用に，05000 から 06999 までの範囲の番号を割り当てています。UAP から出力するメッセージ ID の番号には，05000 から 06999 までの範囲の値を付けてください。

障害が起こって UAP からメッセージログが出力できない場合でも，dc_logprint 関数が DC_OK で正常に終了することがあります。そのためメッセージログが抜ける場合がありますが，メッセージログの抜けはメッセージログに付けるメッセージログ通番で確認できます。

一つのプロセスから複数回 dc_logprint 関数を呼び出した場合は，メッセージログファイルへの出力順序は保証されます。ただし，複数のプロセスから別々に dc_logprint 関数を呼び出した場合は，呼び出した順にメッセージログファイルに出力されない場合があります。

通信障害（DCLOGER_COMM），およびログサービス未起動（DCLOGER_NOT_UP）のエラーが起こった場合は，UAP から出力したメッセージを，その UAP プロセス上で編集して，標準エラー出力に出力します。このとき，メッセージの終わりには，次に示すエラー要因を示すコードを付けます。

- E1

ログサービスが未起動であるため，メッセージログファイルに出力できなかったメッセージを示します。

- E2

  通信障害のためメッセージログファイルに出力できなかったメッセージを示します。

  （例）

  KFCA05201-I　SPP1：サービス要求を受け取りました。（E1）

  KFCA05410-I　SPP1：更新処理を開始します。（E2）

E1，E2 以外のエラーを検出した場合，OpenTP1 はエラーの原因を示すメッセージログに dc_logprint 関数に指定したメッセージ ID の番号を付けて，標準エラー出力に出力します。

UAP で値を設定する引数

●msgid

メッセージログごとに付けられる識別子（メッセージ ID）を設定します。「KFCAn1n2n3n4n5-x」の形式（11 文字）で，最後にヌル文字を付けて設定します。UAP から出力する通番（n1n2n3n4n5 の部分）には，05000 から 06999 までの間の数値を設定します。

●pgm_id

dc_logprint 関数を呼び出した UAP を識別する値（要求元プログラム ID）を，ユーザ任意で設定します。英数字 2 文字で，最後はヌル文字で終わらせて設定します。

●string

メッセージログファイルにメッセージログとして出力したい任意の文字列を設定します。文字列の上限は各 OS で次のとおりです。なお，文字列の最後はヌル文字で終わらせてください。

- Linux：444 バイト
- Linux 以外の OS：222 バイト

●info

NULL を設定します。

●color

dc_logprint 関数で設定したメッセージログを NETM の操作支援端末に出力する場合の，表示色を設定します。次のどれかを設定します。

1…緑

2…赤

3…白

4…青

5…紫

6…空色

7…黄色

上記以外の数値やヌル文字を設定した場合は，緑が仮定されます。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCLOGER_PARAM_ARGS	-1900	引数に設定した値が間違っています。
DCLOGER_COMM	-1901	通信障害が発生したか，または dc_rpc_open 関数が発行されていません。
DCLOGER_MEMORY	-1902	メモリが不足しました。
DCLOGER_DEFFILE	-1904	システムの環境設定が間違っています。
DCLOGER_NOT_UP	-1905	ログサービスが起動していません。
DCLOGER_HEADER	-1906	ログサービスがメッセージログに付ける情報を取得したときに，障害が起こりました。

注意事項

ログ出力量が多い場合は，dc_logprint 関数のリターンが遅くなります。例えば，障害発生時にメッセージ出力量が激しく多くなると，トランザクション処理時間が延びてしまいます。これはスローダウンの要因になりますので，注意してください。

メッセージ送受信（dc_mcf_～）

メッセージ送受信形態の通信をするときに使用する関数について説明します。メッセージ送受信の関数を次に示します。

- dc_mcf_adltap － アプリケーションに関するタイマ起動要求の削除
- dc_mcf_ap_info － アプリケーション情報通知
- dc_mcf_ap_info_uoc － UOC へのアプリケーション情報通知
- dc_mcf_close － MCF 環境のクローズ
- dc_mcf_commit － MHP のコミット
- dc_mcf_contend － 継続問い合わせ応答の終了
- dc_mcf_execap － アプリケーションプログラムの起動
- dc_mcf_mainloop － MHP のサービス開始
- dc_mcf_open － MCF 環境のオープン
- dc_mcf_receive － メッセージの受信
- dc_mcf_recvsync － 同期型のメッセージの受信※
- dc_mcf_reply － 応答メッセージの送信※
- dc_mcf_resend － メッセージの再送※
- dc_mcf_rollback － MHP のロールバック
- dc_mcf_send － メッセージの送信※
- dc_mcf_sendrecv － 同期型のメッセージの送受信※
- dc_mcf_sendsync － 同期型のメッセージの送信※
- dc_mcf_tactcn － コネクションの確立※
- dc_mcf_tactle － 論理端末の閉塞解除※
- dc_mcf_tdctcn － コネクションの解放※
- dc_mcf_tdctle － 論理端末の閉塞※
- dc_mcf_tdlqle － 論理端末の出力キュー削除
- dc_mcf_tempget － 一時記憶データの受け取り
- dc_mcf_tempput － 一時記憶データの更新
- dc_mcf_timer_cancel － ユーザタイマ監視の取り消し
- dc_mcf_timer_set － ユーザタイマ監視の設定
- dc_mcf_tlscn － コネクションの状態取得※

- dc_mcf_tlscom － MCF 通信サービスの状態取得
- dc_mcf_tlsle － 論理端末の状態取得※
- dc_mcf_tlsln － サーバ型コネクションの確立要求の受付状態取得※
- dc_mcf_tofln － サーバ型コネクションの確立要求の受付終了※
- dc_mcf_tonln － サーバ型コネクションの確立要求の受付開始※

注※

詳細については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

メッセージ送受信の関数（dc_mcf_～）は，TP1/Server Base の UAP でだけ使えます。TP1/LiNK の UAP では，メッセージ送受信の関数は使えません。

dc_mcf_adltap

名称

アプリケーションに関するタイマ起動要求の削除

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int dc_mcf_adltap (DCLONG action, dcmcf_adltapopt *apopt,
                   char *resv01, DCLONG *resv02,
                   char *resv03, char *resv04)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int dc_mcf_adltap (action, apopt, resv01, resv02, resv03, resv04)
DCLONG             action;
dcmcf_adltapopt    *apopt;
char               *resv01;
DCLONG             *resv02;
char               *resv03;
char               *resv04;
```

機能

指定されたアプリケーションに関するタイマ要求を削除し，アプリケーションの起動を停止します。ただし，ans 型，および cont 型のアプリケーションに関するタイマ要求は削除できません。

UAP で値を設定する引数

●action

アプリケーション名で指定することを示す，DCMCFAP を設定します。

●apopt

この関数の対象となったコネクションの情報を，構造体 dcmcf_adltapopt に設定します。

構造体の形式を次に示します。

```
typedef struct {
                DCLONG    mcfid;          …アプリケーション起動
                                            プロセス識別子
                char      resv01[4];      …予備領域
                char      idnam[9];       …アプリケーション名
                char      resv02[7];      …予備領域
                char      resv03[112];    …予備領域
```

```
char      resv04[376];   …予備領域
} dcmcf_adltapopt;
```

- mcfid
  処理対象のアプリケーションを持つアプリケーション起動サービスのアプリケーション起動プロセス識別子※を設定します。設定できる範囲は 1～239 です。

  注※

  MCF 環境定義（mcftenv -s）で指定するアプリケーション起動プロセス識別子は 16 進数とみなしてください。

  例えば，アプリケーション起動プロセス識別子が 10 の場合，16 を設定してください。
- resv01
  領域をヌル文字で埋めます。
- idnam
  起動を停止するアプリケーションのアプリケーション名を設定します。アプリケーション名は 8 バイト以内で設定し，文字列の最後にヌル文字を付けます。
- resv02，resv03，resv04
  領域をヌル文字で埋めます。

●resv01，resv02，resv03，resv04

NULL を設定します。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。
DCMCFRTN_71001	-12001	MCF が開始処理中のため，dc_mcf_adltap 関数が受け付けられません。
DCMCFRTN_71002	-12002	MCF が終了処理中のため，dc_mcf_adltap 関数が受け付けられません。
DCMCFRTN_71004	-12004	dc_mcf_adltap 関数の処理中にメモリ不足が発生しました。
DCMCFRTN_71005	-12005	通信障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71006	-12006	内部障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71007	-12007	指定されたアプリケーション名は登録されていません。
		タイマ起動要求されていないアプリケーション名を指定しています。

リターン値	リターン値（数値）	意味
DCMCFRTN_71007	-12007	問合せ応答型または継続問い合わせ応答型のアプリケーション名を指定しています。
DCMCFRTN_71009	-12009	dc_mcf_adltap 関数が，該当するアプリケーション起動プロセスではサポートされていません。
DCMCFRTN_71010	-12010	指定されたアプリケーションに関するタイマ起動要求の削除を要求しましたが，受け付けられませんでした。原因については，メッセージログファイルを参照してください。
DCMCFRTN_72050	-13050	action に DCMCFAP が指定されていません。
		action に未サポートのフラグを設定しています。
DCMCFRTN_72051	-13051	apopt に NULL が設定されています。
DCMCFRTN_72052	-13052	resv01 に NULL が設定されていません。
DCMCFRTN_72053	-13053	resv02 に NULL が設定されていません。
DCMCFRTN_72054	-13054	resv03 に NULL が設定されていません。
DCMCFRTN_72055	-13055	resv04 に NULL が設定されていません。
DCMCFRTN_72061	-13061	dcmcf_adltapopt の mcfid に 0 以下または 240 以上の値が設定されています。
DCMCFRTN_72062	-13062	dcmcf_adltapopt の resv01 がヌル文字で埋められていません。
DCMCFRTN_72063	-13063	dcmcf_adltapopt の idnam の先頭がヌル文字です。
DCMCFRTN_72064	-13064	dcmcf_adltapopt の resv02 がヌル文字で埋められていません。
DCMCFRTN_72065	-13065	dcmcf_adltapopt の resv03 がヌル文字で埋められていません。
DCMCFRTN_72067	-13067	dcmcf_adltapopt の resv04 がヌル文字で埋められていません。
DCMCFRTN_72073	-13073	dcmcf_adltapopt の idnam に設定された文字数が 9 バイト以上です。
DCMCFRTN_72074	-13074	dcmcf_adltapopt の idnam に設定された文字列中に不正な文字があります。

dc_mcf_ap_info

名称

アプリケーション情報通知

形式

ANSI C，C++の形式

```
#include<dcmcf.h>
int dc_mcf_ap_info(DCLONG flags, char *mcfid, char *apname,
                   struct DC_MCFAPINFO *apinfo, char *resv01,
                   DCLONG resv02)
```

K&R 版 C の形式

```
#include<dcmcf.h>
int dc_mcf_ap_info(flags, mcfid, apname, apinfo, resv01, resv02)
DCLONG  flags;
char  *mcfid;
char  *apname;
struct DC_MCFAPINFO  *apinfo;
char  *resv01;
DCLONG  resv02;
```

機能

MHP からアプリケーションに関するさまざまな情報を取得できます。

通知の対象となるアプリケーションは，dc_mcf_ap_info 関数を呼び出した MHP 自身のアプリケーションか，ほかの MHP のアプリケーションです。dc_mcf_ap_info 関数が正常終了した場合だけ，アプリケーション情報は有効となります。

UAP で値を設定する引数

●flags

参照するアプリケーションによって，次のフラグを設定します。

DCMCFMYSELF

dc_mcf_ap_info 関数を呼び出した MHP に関するアプリケーション情報を取得する場合に設定します。

DCMCFOTHER

アプリケーション定義が登録されている MCF 通信サービスのプロセス識別子とアプリケーション名によって，特定のアプリケーションに関するアプリケーション情報を取得する場合に設定します。

●mcfid

- flags に DCMCFMYSELF を設定した場合

NULL を設定します。

- flags に DCMCFOTHER を設定した場合
  参照するアプリケーションの定義が登録されている MCF 通信プロセス識別子またはアプリケーション起動プロセス識別子を文字列で設定します。文字列の最後にはヌル文字を付けます。

●apname

- flags に DCMCFMYSELF を設定した場合
  NULL を設定します。
- flags に DCMCFOTHER を設定した場合
  参照したいアプリケーション名を設定します。アプリケーション名の最後にはヌル文字を付けます。
  ただし，エラーイベント名(ERREVT1，ERREVT2，ERREVT3，ERREVT4)を設定した場合，mcf_ap_type には，アプリケーション定義の省略時解釈値である非応答型(DCMCF_NOANS)が設定されます。

●apinfo

アプリケーション情報を取得する領域 DC_MCFAPINFO のアドレスを設定します。

●resv01

NULL を設定します。

●resv02

DCNOFLAGS を設定します。

OpenTP1 から値が返される引数

●apinfo

アプリケーション情報が，構造体 DC_MCFAPINFO で返されます。

構造体の形式は次のとおりです。

```
struct  DC_MCFAPINFO  {
                        char mcf_apinfo[4];
                        DCLONG mcf_resv00;
                        char mcf_ap_name[9];
                        char mcf_ap_mcfid[3];
                        char mcf_resv01[4];
                        DCLONG mcf_ap_stat;
                        DCLONG mcf_ap_type;
                        char mcf_sg_name[32];
                        DCLONG mcf_sg_stat;
                        DCLONG mcf_sg_hold;
                        char mcf_sv_name[32];
                        DCLONG mcf_sv_stat;
                        DCLONG mcf_ap_ntmetim;
                        DCLONG mcf_ap_tempsize;
```

```
    DCLONG mcf_ap_msgcnt;
    DCLONG mcf_ap_trnmode;
    DCLONG mcf_ap_quekind;
    char mcf_resv02[72];
}
```

- mcf_apinfo
  MCF で使用する領域です。
- mcf_resv00
  MCF で使用する領域です。
- mcf_ap_name
  通知対象のアプリケーション名が設定されます。
- mcf_ap_mcfid
  通知対象のアプリケーションが登録されている MCF 通信サービスのプロセス識別子が設定されます。
- mcf_resv01
  MCF で使用する領域です。
- mcf_ap_stat
  アプリケーションの閉塞，閉塞解除の状態がフラグで設定されます。
  DCMCF_IN_DACT…入力閉塞状態
  DCMCF_SC_DACT…スケジュール閉塞状態
  DCMCF_DACTSTAT…入力・スケジュール閉塞状態
  DCMCF_ACTSTAT…閉塞解除状態
- mcf_ap_type
  アプリケーションの型がフラグで設定されます（MCF アプリケーション定義 mcfaalcap の-n オプションの type オペランドの型が設定されます）。
  DCMCF_ANS…応答型
  DCMCF_NOANS…非応答型
  DCMCF_CONT…継続問い合わせ型
  ただし，flags に DCMCFOTHER を設定し，apname にエラーイベント名（ERREVT1，ERREVT2，ERREVT3，ERREVT4）を設定した場合は，実際の型が通知されません。アプリケーション定義の省略時解釈値である非応答型（DCMCF_NOANS）が設定されます。
- mcf_sg_name
  アプリケーションに対応したサービスグループ名が設定されます。
- mcf_sg_stat
  サービスグループの閉塞，閉塞解除の状態がフラグで設定されます。
  入力閉塞状態： DCMCF_IN_DACT
  スケジュール閉塞状態： DCMCF_SC_DACT
  入力・スケジュール閉塞状態： DCMCF_DACTSTAT

閉塞解除状態： DCMCF_ACTSTAT

- mcf_sg_hold
  サービスグループの保留，保留解除の状態がフラグで設定されます。
  入力保留状態： DCMCF_IN_HOLD
  スケジュール保留状態： DCMCF_SC_HOLD
  入力・スケジュール保留状態： DCMCF_HOLDSTAT
  保留解除状態： DCMCF_RLSSTAT
- mcf_sv_name
  アプリケーションに対応したサービス名が設定されます。
- mcf_sv_stat
  サービスの閉塞，閉塞解除の状態がフラグで設定されます。
  入力閉塞状態： DCMCF_IN_DACT
  スケジュール閉塞状態： DCMCF_SC_DACT
  入力・スケジュール閉塞状態： DCMCF_DACTSTA
  閉塞解除状態： DCMCF_ACTSTAT
- mcf_ap_ntmetim
  非トランザクション MHP の限界経過時間が設定されます。
  ただし，mcf_ap_trnmode が DCMCF_TRN の場合は，0 が設定されます（MCF アプリケーション定義 mcfaalcap の-v オプションの ntmetim オペランドの値が設定されます。ただし，MCF アプリケーション定義を省略した場合は，MCF マネジャ定義 mcfmuap の-v オプションの ntmetim オペランドの値が設定されます）。
- mcf_ap_tempsize
  継続問い合わせ応答用の一時記憶データ格納用領域サイズが設定されます。
  ただし，mcf_ap_type が DCMCF_CONT 以外の場合は，0 が設定されます（MCF アプリケーション定義 mcfaalcap の-n オプションの tempsize オペランドの値が設定されます）。
- mcf_ap_msgcnt
  入力メッセージ最大格納数が設定されます（MCF アプリケーション定義 mcfaalcap の-n オプションの msgcnt オペランドの値が設定されます）。
- mcf_ap_trnmode
  アプリケーションのトランザクション属性がフラグで設定されます（MCF アプリケーション定義 mcfaalcap の-n オプションの trnmode オペランドの値が設定されます）。
  トランザクションとして管理します： DCMCF_TRN
  トランザクションとして管理しません： DCMCF_NONTRN
- mcf_ap_quekind
  受信メッセージの割り当て先がフラグで設定されます（MCF アプリケーション定義 mcfaalcap の-g オプションの quekind オペランドの値が設定されます）。

ディスクキューに割り当てる場合： DCMCF_DISK
メモリキューに割り当てる場合： DCMCF_MEMORY

- mcf_resv02
  MCF で使用する領域です。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。
DCMCFRTN_72000	-13000	MHP サービス以外から呼び出されました。
DCMCFRTN_72001	-13001	アプリケーション名の設定が不正です。 アプリケーション名とプロセス識別子の組み合わせが不正です。
DCMCFRTN_72016	-13016	パラメタに不正な値があります。
上記以外		プログラムの破壊などによる，予期しないエラーが発生しました。

注意事項

ERREVT1，ERREVT2，ERREVT3，ERREVT4 は，同時に複数起動した場合，同名のエラーイベントであっても異なるアプリケーションの型を持つことがあります。したがって，dc_mcf_ap_info 関数を呼び出した MHP 以外のエラーイベント(ERREVT1，ERREVT2，ERREVT3，ERREVT4)に対する型の通知は行いません。この場合，MCF アプリケーション定義の省略時解釈値である「非応答型」を通知します。

dc_mcf_ap_info_uoc

名称

UOC へのアプリケーション情報通知

形式

ANSI C，C++の形式

```
#include<dcmcf.h>
int dc_mcf_ap_info_uoc(DCLONG flags, char *apname,
                       struct DC_MCFAPINFO_UOC *apinfo)
```

K&R 版 C の形式

```
#include<dcmcf.h>
int dc_mcf_ap_info_uoc(flags, apname, apinfo)
DCLONG flags;
char *apname;
struct DC_MCFAPINFO_UOC *apinfo;
```

機能

引数 apname で指定したアプリケーションの定義内容（アプリケーション属性定義）やアプリケーションの状態（dc_mcf_ap_info_uoc 関数を呼び出した時点での状態）などの情報（アプリケーション情報）を apinfo で指定した struct DC_MCFAPINFO_UOC に設定します。dc_mcf_ap_info_uoc 関数が正常終了した場合だけ，アプリケーション情報は有効となります。

UOC が動作する通信サービスから起動できないユーザアプリケーション，またはシステムイベントを指定した場合は，不正なアプリケーションが指定されたと見なし，DCMCFRTN_72001 でエラーリターンします。

dc_mcf_ap_info_uoc 関数が使用できる UOC は，入力メッセージ編集 UOC（アプリケーション名決定 UOC）だけです。そのほかの UOC からは使用できません。そのほかの UOC から使用した場合，動作は保証しません。

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

●apname

取得したいアプリケーション名を指定します。アプリケーション名の最後にはヌル文字を付けます。

●apinfo

アプリケーション情報を取得する領域 DC_MCFAPINFO_UOC のアドレスを指定します。

OpenTP1 から値が返される引数

●apinfo

アプリケーション情報が，構造体 DC_MCFAPINFO で返されます。

構造体の形式は次のとおりです。

```
struct  DC_MCFAPINFO_UOC  {
                                char mcf_apinfo[4];
                                DCLONG mcf_resv00;
                                char mcf_ap_name[9];
                                char mcf_ap_mcfid[3];
                                char mcf_resv01[4];
                                DCLONG mcf_ap_stat;
                                DCLONG mcf_ap_type;
                                DCLONG mcf_ap_msgcnt;
                                char mcf_sg_name[32];
                                DCLONG mcf_sg_stat;
                                DCLONG mcf_sg_hold;
                                DCLONG mcf_sg_msgcnt;
                                char mcf_sv_name[32];
                                DCLONG mcf_sv_stat;
                                DCLONG mcf_ap_ntmetim;
                                DCLONG mcf_ap_tempsize;
                                DCLONG mcf_ap_max_msgcnt;
                                DCLONG mcf_ap_trnmode;
                                DCLONG mcf_ap_quekind;
                                char mcf_resv02[64];
                               };
```

- mcf_apinfo
  MCF で使用する領域です。
- mcf_resv00
  MCF で使用する領域です。
- mcf_ap_name
  通知対象のアプリケーション名が設定されます。
- mcf_ap_mcfid
  通知対象のアプリケーションが登録されている MCF 通信サービスのプロセス識別子が設定されます。
- mcf_resv01
  MCF で使用する領域です。
- mcf_ap_stat
  アプリケーションの閉塞，閉塞解除の状態がフラグで設定されます。

入力閉塞状態： DCMCF_IN_DACT

スケジュール閉塞状態： DCMCF_SC_DACT

入力・スケジュール閉塞状態： DCMCF_DACTSTAT

閉塞解除状態： DCMCF_ACTSTAT

- mcf_ap_type
  アプリケーションの型がフラグで設定されます（MCF アプリケーション定義 mcfaalcap の-n オプションの type オペランドの型が設定されます）。

  応答型： DCMCF_ANS

  非応答型： DCMCF_NOANS

  継続問い合わせ型： DCMCF_CONT

  ただし，flags に DCMCFOTHER を設定し，apname にエラーイベント名（ERREVT1，ERREVT2，ERREVT3，ERREVT4）を設定した場合は，実際の型が通知されません。アプリケーション定義の省略時解釈値である非応答型（DCMCF_NOANS）が設定されます。
- mcf_ap_msgcnt
  このアプリケーションに入力されたメッセージの滞留数が設定されます。
- mcf_sg_name
  アプリケーションに対応したサービスグループ名が設定されます。
- mcf_sg_stat
  サービスグループの閉塞，閉塞解除の状態がフラグで設定されます。

  入力閉塞状態： DCMCF_IN_DACT

  スケジュール閉塞状態： DCMCF_SC_DACT

  入力・スケジュール閉塞状態： DCMCF_DACTSTAT

  閉塞解除状態： DCMCF_ACTSTAT
- mcf_sg_hold
  サービスグループの保留，保留解除の状態がフラグで設定されます。

  入力保留状態： DCMCF_IN_HOLD

  スケジュール保留状態： DCMCF_SC_HOLD

  入力・スケジュール保留状態： DCMCF_HOLDSTAT

  保留解除状態： DCMCF_RLSSTAT
- mcf_sg_msgcnt
  このサービスグループに入力されたメッセージの滞留数が設定されます。
- mcf_sv_name
  アプリケーションに対応したサービス名が設定されます。
- mcf_sv_stat
  サービスの閉塞，閉塞解除の状態がフラグで設定されます。

  入力閉塞状態： DCMCF_IN_DACT

スケジュール閉塞状態： DCMCF_SC_DACT

入力・スケジュール閉塞状態： DCMCF_DACTSTA

閉塞解除状態： DCMCF_ACTSTAT

- mcf_ap_ntmetim

  非トランザクション MHP の限界経過時間が設定されます。

  ただし，mcf_ap_trnmode が DCMCF_TRN の場合は，0 が設定されます（MCF アプリケーション定義 mcfaalcap の-v オプションの ntmetim オペランドの値が設定されます。ただし，MCF アプリケーション定義を省略した場合は，MCF マネジャ定義 mcfmuap の-v オプションの ntmetim オペランドの値が設定されます）。

- mcf_ap_tempsize

  継続問い合わせ応答用の一時記憶データ格納用領域サイズが設定されます。

  ただし，mcf_ap_type が DCMCF_CONT 以外の場合は，0 が設定されます（MCF アプリケーション定義 mcfaalcap の-n オプションの tempsize オペランドの値が設定されます）。

- mcf_ap_max_msgcnt

  入力メッセージ最大格納数が設定されます（MCF アプリケーション定義 mcfaalcap の-n オプションの msgcnt オペランドの値が設定されます）。

- mcf_ap_trnmode

  アプリケーションのトランザクション属性がフラグで設定されます（MCF アプリケーション定義 mcfaalcap の-n オプションの trnmode オペランドの値が設定されます）。

  トランザクションとして管理します： DCMCF_TRN

  トランザクションとして管理しません： DCMCF_NONTRN

- mcf_ap_quekind

  受信メッセージの割り当て先がフラグで設定されます（MCF アプリケーション定義 mcfaalcap の-g オプションの quekind オペランドの値が設定されます）。

  ディスクキューに割り当てる場合： DCMCF_DISK

  メモリキューに割り当てる場合： DCMCF_MEMORY

- mcf_resv02

  MCF で使用する領域です。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。
DCMCFRTN_72000	-13000	MHP サービス以外から呼び出されました。
DCMCFRTN_72001	-13001	アプリケーション名の設定が不正です。 指定したアプリケーションの情報を取得できません。

リターン値	リターン値（数値）	意味
DCMCFRTN_72016	-13016	パラメタに不正な値があります。
上記以外		プログラムの破壊などによる，予期しないエラーが発生しました。

注意事項

1. dc_mcf_ap_info_uoc 関数を使用できる UOC は入力メッセージ編集 UOC（アプリケーション名決定 UOC）だけです。dc_mcf_ap_info_uoc 関数がそのほかの UOC で呼び出されたかどうかのチェックはできないので，そのほかの UOC での動作は保証しません。UOC については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。
2. UOC へのアプリケーション情報通知関数で取得できるアプリケーションの情報は，UOC が動作する通信サービスから起動できるユーザアプリケーションだけです。また，システムイベント（MCF アプリケーション定義 mcfaalcap の-n オプションの kind オペランドに mcf を指定した定義）に関するアプリケーション情報は通知しません。
3. UAP トレースは取得されません。

dc_mcf_close

名称

MCF 環境のクローズ

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
void  dc_mcf_close(DCLONG flags)
```

K&R 版 C の形式

```
#include <dcmcf.h>
void  dc_mcf_close(flags)
DCLONG      flags;
```

機能

MCF の機能を使う環境をクローズします。dc_mcf_open 関数を呼び出した UAP で，メイン関数で dc_rpc_close 関数を呼び出す前に，プロセスで 1 回だけ呼び出します。

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

リターン値

dc_mcf_close 関数のリターン値はありません。

dc_mcf_commit

名称

MHP のコミット

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int  dc_mcf_commit(DCLONG action)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int   dc_mcf_commit(action)
DCLONG      action;
```

機能

MHP から開始するグローバルトランザクションで，ルートトランザクションブランチとして，トランザクションを構成するトランザクションブランチの UAP，トランザクションサービス，およびリソースマネジャに，処理が正常に終了したこと（コミット）を知らせます。

dc_mcf_commit 関数が正常に終了すると，新しいグローバルトランザクションが開始します。

グローバルトランザクションが複数のトランザクションブランチから構成されるとき（関数を呼び出した MHP だけでないとき）は，それぞれのトランザクションブランチの処理結果がコミットとならないかぎりコミットされません。また，複数のリソースマネジャで構成されるときも同様に，それぞれのリソースマネジャの処理結果がコミットとならないかぎりコミットされません。コミットされない場合は，すべてのトランザクションブランチがロールバックされて，リターン値 DCMCFRTN_ROLLBACK でエラーリターンします。

MCF アプリケーション定義で非応答型（type=noans）と指定した MHP からだけ，dc_mcf_commit 関数を呼び出せます。それ以外の型の MHP から呼び出した場合は，リターン値 DCMCFRTN_72000 でエラーリターンします。また，MHP 以外の UAP で呼び出した場合も同様に，リターン値 DCMCFRTN_72000 でエラーリターンします。

UAP で値を設定する引数

●action

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。このリターン値が返った場合，dc_mcf_commit 関数を呼び出したプロセスでは，新しいトランザクションが開始しています。
DCMCFRTN_ROLLBACK	-11906	トランザクションは，コミットできないでロールバックしました。このリターン値が返った場合，dc_mcf_commit 関数を呼び出したプロセスでは，新しいトランザクションが開始しています。
DCMCFRTN_HEURISTIC	-11907	dc_mcf_commit 関数を呼び出したグローバルトランザクションは，ヒューリスティック決定のため，あるトランザクションブランチはコミットとなって，また，あるトランザクションブランチはロールバックとなりました。このリターン値が返った場合，dc_mcf_commit 関数を呼び出したプロセスでは，新しいトランザクションが開始しています。
DCMCFRTN_HAZARD	-11908	グローバルトランザクションのトランザクションブランチがヒューリスティックに完了しました。しかし，障害のため，ヒューリスティックに完了したトランザクションブランチの同期点の結果が判明しません。このリターン値が返った場合，dc_mcf_commit 関数を呼び出したプロセスでは，新しいトランザクションが開始しています。 トランザクションサービス定義で，trn_extend_function オペランドに 00000001 を指定し，1 相コミット時にリソースマネジャからのリターン値が XAER_NOTA の場合も，DCMCFRTN_HAZARD を返します。
DCMCFRTN_72000	-13000	＜ MHP の実行でリターンした場合＞ dc_mcf_commit 関数を呼び出した位置が間違っています。MHP で先頭セグメントを受信する dc_mcf_receive 関数を呼び出す前に，dc_mcf_commit 関数を呼び出しています。 MCF アプリケーション定義で非応答型（type=noans）と指定していない MHP から，dc_mcf_commit 関数を呼び出しています。 非トランザクション属性の MHP から，dc_mcf_commit 関数を呼び出しています。
		＜ SPP の実行でリターンした場合＞ SPP から dc_mcf_commit 関数は呼び出せません。
DCMCFRTN_72016	-13016	action に設定した値が間違っています。
上記以外		プログラムの破壊などによる，予期しないエラーが発生しました。

注意事項

dc_mcf_commit 関数が正常に終了しても，入力メッセージは入力キューから削除されません。そのため，MHP がスケジュールし直されたあとに該当するメッセージを処理し続ける場合，どの時点までコミットしていたかは不定です。MHP が再びスケジュールされるのは次の場合です。

1. MCF イベントが通知されて，MCF イベント処理用 MHP をスケジュールしたとき
2. システムが異常終了して，OpenTP1 が該当する処理の MHP を再スケジュールしたとき

スケジュールし直された MHP で該当するメッセージを処理し続ける場合，コミットした範囲はユーザで管理してください。

dc_mcf_contend

名称

継続問い合わせ応答の終了

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int  dc_mcf_contend(DCLONG action,char *resv01)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int  dc_mcf_contend(action,resv01)
DCLONG   action;
char   *resv01;
```

機能

継続問い合わせ応答を終了します。継続問い合わせ応答を終了させるときは，その MHP で呼び出した dc_mcf_reply 関数の nextap がヌル文字列であること，および継続問い合わせ応答型のアプリケーションを起動させる dc_mcf_execap 関数を呼び出していないことが前提です。次起動アプリケーションを dc_mcf_reply 関数の nextap に設定している場合，または継続問い合わせ応答型のアプリケーションを起動させる dc_mcf_execap 関数を呼び出している場合は，dc_mcf_contend 関数はエラーリターンします。

dc_mcf_contend 関数を呼び出したあとは，一時記憶データにアクセスする関数（dc_mcf_tempget 関数，dc_mcf_tempput 関数）は使えません。

UAP で値を設定する引数

●action

DCNOFLAGS を設定します。

●resv01

ヌル文字列を設定します。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。

リターン値	リターン値（数値）	意味
DCMCFRTN_72000	-13000	＜MHP の実行でリターンした場合＞ 先頭セグメントを受信する dc_mcf_receive 関数を呼び出す前に，dc_mcf_contend 関数を呼び出しています。
		＜SPP の実行でリターンした場合＞ SPP では dc_mcf_contend 関数を呼び出せません。
DCMCFRTN_72016	-13016	action に設定した値が間違っています。
		resv01 に設定した値がヌル文字列になっていません。
DCMCFRTN_72101	-13101	継続問い合わせ応答型でないアプリケーションで，dc_mcf_contend 関数を呼び出しています。
DCMCFRTN_72107	-13107	dc_mcf_contend 関数を 2 回以上呼び出しています。
DCMCFRTN_72111	-13111	次起動アプリケーション（nextap）を設定して dc_mcf_reply 関数を呼び出したあと，dc_mcf_contend 関数を呼び出しています。
		継続問い合わせ応答型のアプリケーション名を設定して dc_mcf_execap 関数を呼び出したあと，dc_mcf_contend 関数を呼び出しています。
上記以外		プログラムの破壊などによる，予期しないエラーが発生しました。

dc_mcf_execap

名称

アプリケーションプログラムの起動

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int  dc_mcf_execap(DCLONG action,DCLONG commform,char *resv01,
                   DCLONG active,char *apnam,char *comdata,
                   DCLONG cdataleng)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int  dc_mcf_execap(action,commform,resv01,active,apnam,comdata,
                   cdataleng)
DCLONG      action;
DCLONG      commform;
char        *resv01;
DCLONG      active;
char        *apnam;
char        *comdata;
DCLONG      cdataleng;
```

機能

UAP（SPP，または MHP）から，apnam に設定したアプリケーション名の MHP を起動させます。トランザクション終了後またはサービス関数終了後，すぐに起動させることも，設定した時間経過後に起動させることもできます。詳細については，マニュアル「OpenTP1 プログラム作成の手引」の「アプリケーションプログラムの起動」の説明を参照してください。

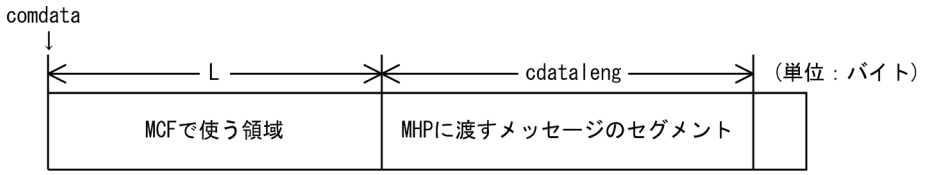
SPP から dc_mcf_execap 関数を呼び出した場合は，SPP がトランザクションとして処理していることと，その SPP のメイン関数で dc_mcf_open 関数を呼び出していることが前提です。

MHP から dc_mcf_execap 関数で MHP を起動させる場合，起動された MHP で受け取るメッセージ入力元の論理端末名称は，最初に受信したメッセージ中の名称になります。さらに，その MHP から dc_mcf_execap 関数を呼び出した場合も，受け取るメッセージ入力元の論理端末名称は，最初にメッセージを受信したときの名称が引き渡されます。

SPP から dc_mcf_execap 関数で MHP を起動する場合，起動された MHP で受け取るメッセージ入力元の論理端末名称は「*」となります。さらに，その MHP から dc_mcf_execap 関数を呼び出した場合も，受け取るメッセージ入力元の論理端末名称は「*」となります。

送信できるメッセージの一つのセグメント長は，32 キロバイトまでです。ただし，プロトコルによって，実際の最大長が小さいことがあります。詳細については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

起動させる MHP に渡すメッセージのセグメント形式を次に示します。L は，バッファ形式 1 の場合は 8 バイト，バッファ形式 2 の場合は 4 バイトです。



UAP で値を設定する引数

●action

起動させる MHP に渡すセグメントが論理メッセージの最終セグメントかどうか，MHP をいつ起動させるか，および使うバッファ形式を，次の形式で設定します。

```
{DCMCFESI | DCMCFEMI} 〔|{DCMCFJUST|DCMCFINTV|DCMCFTIME}〕
                                                〔|{DCMCFBUF1|DCMCFBUF2}〕
```

DCMCFESI

先頭セグメント，または中間セグメントを渡すときに設定します。この値を設定した dc_mcf_execap 関数を呼び出した場合は，そのあとに必ず action に DCMCFEMI を設定した dc_mcf_execap 関数を呼び出してください。

DCMCFEMI

最終セグメントを渡すときに設定します。

論理メッセージが単一セグメントの場合も，DCMCFEMI を設定します。

先頭セグメント，または中間セグメントの送信後，メッセージの送信の終了を連絡する場合にも，DCMCFEMI を設定します。

DCMCFJUST

即時起動のときに設定します。このとき，active に設定した値は無視されます。

DCMCFINTV

経過時間指定のタイマ起動のときに設定します。dc_mcf_execap 関数を呼び出してから active に設定した時間だけ経過したあとに，MHP を起動させます。

DCMCFTIME

時刻指定のタイマ起動の場合に設定します。active に設定した時刻に，MHP を起動させます。

DCMCFBUF1

バッファ形式 1 を使うときに設定します。

DCMCFBUF2

バッファ形式 2 を使うときに設定します。

●commform

DCNOFLAGS を設定します。

●resv01

ヌル文字列を設定します。

●active

- **経過時間指定のタイマ起動（action に DCMCFINTV を設定）の場合**

  dc_mcf_execap 関数を呼び出してから，何秒後に MHP を起動させるかを設定します。設定できる秒数の範囲は，1 から 360000 まで（1 秒から 100 時間まで）です。

- **時刻指定のタイマ起動（action に DCMCFTIME を設定）の場合**

  apnam に設定した MHP を起動させる時刻を設定します。0 時 0 分 0 秒を基準にして，起動時刻を秒数で算出して設定します。時刻はローカルタイムで指定します。

  （時刻の設定例）

  午後 2 時 30 分 30 秒（ローカルタイム）に起動させる場合：

  14 × 3600 + 30 × 60 + 30 = 52230

  …52230 を active に設定します。

設定できる値の範囲は，0（0 時 0 分 0 秒に開始）から 86399（23 時 59 分 59 秒に開始）までです。

active に設定した値は，タイマ起動の場合にだけ有効となります。即時起動の場合は，active に設定した値は無視されます。

注意事項

- 経過時間指定のタイマ起動の場合

  時間監視の精度は秒単位です。また，タイマ定義（mcfttim -t）の btim オペランドで指定する時間監視間隔で起動するかどうかを監視しています。このため，active に設定した経過時間と実際に起動する時間には秒単位の誤差が生じます。そのため，タイミングによっては，設定した監視時間よりも短い時間で起動することがあります。監視時間が小さくなるほど，誤差の影響を受けやすくなりますので，監視時間は 3（単位：秒）以上の値の設定を推奨します。

- 時刻指定のタイマ起動の場合

  時間監視の精度は秒単位です。また，タイマ定義（mcfttim -t）の btim オペランドで指定する時間監視間隔で起動するかどうかを監視しています。このため，active に設定した時刻と実際に起動する時刻には秒単位の誤差が生じます。

●apnam

起動させる MHP のアプリケーション名を設定します。アプリケーション名は最大 8 バイトです。アプリケーション名の最後にはヌル文字を付けます。

●comdata

起動させる MHP に渡す，メッセージのセグメントの内容を設定します。先頭セグメント，または中間セグメントの送信後，メッセージの送信の終了を連絡する場合にも，必ず設定してください。

●cdataleng

起動させる MHP に渡すセグメントの長さを設定します。先頭セグメント，または中間セグメントの送信後，メッセージの送信の終了を連絡する場合には，0 を設定します。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。
DCMCFRTN_71002	-12002	メッセージキューへの入出力処理時に障害が起こりました。
		メッセージキューが閉塞されています。
		メッセージキューが割り当てられていません。
		セグメント長に 32000 バイトを超える値を設定しています。
		MCF が終了処理中のため，apnam に設定した MHP を起動できません。
DCMCFRTN_71003	-12003	メッセージキューが満杯です。
DCMCFRTN_71004	-12004	メッセージを格納するバッファをメモリ上に確保できませんでした。
DCMCFRTN_71108	-12108	apnam に設定したアプリケーション名の MHP を起動しようとしましたが，起動させようとした MHP の管理テーブルが確保できませんでした。
		プロセスのローカルメモリが不足しています。
DCMCFRTN_72000	-13000	＜ MHP の実行でリターンした場合＞ action に DCMCFFRST を設定した dc_mcf_receive 関数を呼び出す前に，dc_mcf_execap 関数を呼び出しています。
		＜ SPP の実行でリターンした場合＞ トランザクションでない SPP の処理から，dc_mcf_execap 関数を呼び出しています。
DCMCFRTN_72001	-13001	設定したアプリケーション名は，MCF で定義していません。
		アプリケーション名が間違っています。

リターン値	リターン値（数値）	意味
DCMCFRTN_72001	-13001	MCF マネジャ定義の通信サービス定義（mcfmcname 定義コマンド）に，アプリケーション起動プロセス名を指定していません。
		アプリケーション起動プロセスに対応する MCF アプリケーション定義の環境定義（mcfaenv 定義コマンドの-p オプション）に，アプリケーション起動プロセス識別子を指定していません。
		アプリケーション環境定義（mcfaenv 定義コマンドの-p オプション）で指定したアプリケーション起動プロセス識別子と，アプリケーション起動プロセスの通信構成定義（mcftenv 定義コマンド）で指定する識別子が一致していません。
		＜非応答型の MHP を起動する場合＞ • 起動先アプリケーションのアプリケーション属性定義の，論理端末（mcfaalcap 定義コマンドの-n オプションの lname オペランド）に値を指定していません。 • 起動先アプリケーションのアプリケーション属性定義に指定した論理端末を，アプリケーション起動プロセスの通信構成定義（mcftalcle 定義コマンド）に定義していません。 • 起動先アプリケーションのアプリケーション属性定義に指定した論理端末が，一方送信型（mcftalcle -t=send）ではありません。 • 起動先アプリケーションのアプリケーション属性定義で指定した論理端末は，アプリケーション起動を使えません。
		＜応答型および継続問い合わせ応答型の MHP を起動する場合＞ • 起動先アプリケーションのアプリケーション属性定義に，内部通信路（mcfaalcap 定義コマンドの-n オプションの cname オペランド）を指定していません。 • 起動先アプリケーションのアプリケーション属性定義に指定した内部通信路を，アプリケーション起動プロセスの通信構成定義（mcftpsvr 定義コマンドの-c オプション）に定義していません。 • アプリケーション起動プロセスの通信構成定義（mcftalcle 定義コマンド）に，問い合わせ型論理端末（mcftalcle -t=request）を指定していません。
		＜ SPP からアプリケーションを起動する場合＞ • アプリケーション起動プロセス識別子を，起動元の UAP のユーザサービス定義またはユーザサービスデフォルト定義の mcf_psv_id オペランドに指定していません。 • 起動元の UAP のユーザサービス定義またはユーザサービスデフォルト定義の mcf_psv_id オペランドに指定しているアプリケーション起動プロセス識別子が，アプリケーション起動プロセスの通信構成定義（mcftenv 定義コマンド

リターン値	リターン値（数値）	意味
DCMCFRTN_72001	-13001	の-s オプション），およびアプリケーション環境定義（mcfaenv 定義コマンドの-p オプション）で指定しているアプリケーション起動プロセス識別子と一致していません。 • 起動元の UAP のユーザサービス定義またはユーザサービスデフォルト定義の mcf_mgrid オペランドに指定している MCF マネジャ識別子が，アプリケーション起動プロセスが属している MCF マネジャの識別子と一致していません。
DCMCFRTN_72005	-13005	action で DCMCFESI を設定した dc_mcf_execap 関数で，メッセージのセグメント長に 1 バイト未満の値を設定しています。
DCMCFRTN_72007	-13007	dc_mcf_reply 関数をすでに呼び出した 応答型（type=ans）の MHP から，応答型の MHP を dc_mcf_execap 関数で起動させています。
		dc_mcf_reply 関数をすでに呼び出した 継続問い合わせ応答型（type=cont）の MHP から，継続問い合わせ応答型の MHP を dc_mcf_execap 関数で起動させています。
DCMCFRTN_72009	-13009	応答型（type=ans）の MHP から，dc_mcf_execap 関数で応答型の MHP を 2 回以上起動させています。
		継続問い合わせ応答型（type=cont）の MHP から，dc_mcf_execap 関数で継続問い合わせ応答型の MHP を 2 回以上起動させています。
DCMCFRTN_72011	-13011	応答型（type=ans）でない MHP から，dc_mcf_execap 関数で応答型の MHP を起動させています。
		継続問い合わせ応答型（type=cont）でない MHP から，dc_mcf_execap 関数で継続問い合わせ応答型の MHP を起動させています。
DCMCFRTN_72016	-13016	action に設定できない値を設定しています。
		resv01 に設定した値がヌル文字になっていません。
		action に設定したアプリケーション起動の方法が間違っています。
		設定した引数が間違っています。
DCMCFRTN_72024	-13024	commform に DCNOFLAGS を設定していません。
DCMCFRTN_72026	-13026	action のセグメント種別（最終セグメント DCMCFEMI，または最終セグメント以外 DCMCFESI）に設定した値が間違っています。
DCMCFRTN_72041	-13041	＜メッセージが単一セグメントの場合＞ cdataleng に 0 バイト，またはマイナス値を設定しています。
		＜メッセージが複数セグメントの場合＞

リターン値	リターン値（数値）	意味
DCMCFRTN_72041	-13041	action に DCMCFESI を設定した dc_mcf_execap 関数を呼び出さないで，メッセージの送信の終了を連絡しています。
DCMCFRTN_72044	-13044	dc_mcf_contend 関数をすでに呼び出した 継続問い合わせ応答型（type=cont）の MHP から，dc_mcf_execap 関数で 継続問い合わせ応答型の MHP を起動させています。
DCMCFRTN_72108	-13108	active に設定できる範囲の値を超えています。
DCMCFRTN_72109	-13109	MCF アプリケーション定義で継続問い合わせ応答型（type=cont）と指定した MHP を，タイマ起動の dc_mcf_execap 関数で起動させようとしています。
DCMCFRTN_77001	-18001	起動しようとしたアプリケーションに対応する論理端末（LE）は，処理中で使えません。または使える論理端末がありません。
上記以外		プログラムの破壊などによる，予期しないエラーが発生しました。

注意事項

1. MCF マネジャ定義の UAP 共通定義（mcfmuap -c order）の指定によって，アプリケーションプログラムの起動順序が異なりますので注意してください。
2. 一つのサービス関数で，TAM または DAM ファイルの更新と dc_mcf_execap 関数の呼び出しをして，起動先のアプリケーションで更新後の TAM または DAM ファイルを参照させたい場合，TAM または DAM ファイルを排他ありで参照してください。排他なしで参照した場合，更新前のデータが入力されることがあります。

dc_mcf_mainloop

名称

MHP のサービス開始

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int  dc_mcf_mainloop(DCLONG flags)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int  dc_mcf_mainloop(flags)
DCLONG      flags;
```

機能

この関数を呼び出したプロセスで実行中の，サービスグループに含まれるサービス関数へのサービス要求の受け付けを開始します。dc_mcf_mainloop 関数は，OpenTP1 から終了要求を受けるまでリターンしません。

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	OpenTP1 からの終了要求を受けました。dc_mcf_mainloop 関数を呼び出した UAP は，すぐに自分のプロセスの終了処理をして，dc_mcf_close 関数と dc_rpc_close 関数を呼び出して exit()してください。
DCMCFER_INVALID_ARGS	-11900	設定した引数が間違っています。
DCMCFER_PROTO	-11901	dc_mcf_mainloop 関数を呼び出す前に，dc_rpc_open 関数を呼び出していません。
DCMCFER_FATAL	-11902	サービスを開始できませんでした。
DCMCFER_NOMEM	-11903	メモリが不足しました。

dc_mcf_open

名称

MCF 環境のオープン

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int  dc_mcf_open(DCLONG flags)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int  dc_mcf_open(flags)
DCLONG      flags;
```

機能

MCF の機能を使う環境を構築します。MCF の関数を使う UAP では，必ず呼び出します。

dc_mcf_open 関数は，dc_rpc_open 関数を呼び出したあとに，メイン関数で必ず呼び出します。dc_mcf_mainloop 関数（SPP の場合は dc_rpc_mainloop 関数）を呼び出す前に，プロセスで 1 回だけ呼び出します。dc_mcf_open 関数を呼び出す位置を次に示します。

```
dc_rpc_open()
dc_mcf_open()
dc_mcf_mainloop()(SPPの場合はdc_rpc_mainloop())
   :
   :
dc_mcf_close()
dc_rpc_close()
```

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCMCFER_INVALID_ARGS	-11900	flags に設定した値が間違っています。

リターン値	リターン値（数値）	意味
DCMCFER_PROTO	-11901	dc_rpc_open 関数を呼び出していません。
		dc_mcf_open 関数はすでに呼び出しています。
DCMCFER_FATAL	-11902	初期化処理に失敗しました。
DCMCFER_NOMEM	-11903	メモリが不足しました。

dc_mcf_receive

名称

メッセージの受信

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int  dc_mcf_receive(DCLONG action,DCLONG commform,char *termnam,
                    char *resv01,char *recvdata,DCLONG *rdataleng,
                    DCLONG inbufleng,DCLONG *time)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int  dc_mcf_receive(action,commform,termnam,resv01,recvdata,
                    rdataleng,inbufleng,time)
DCLONG    action;
DCLONG    commform;
char    *termnam;
char    *resv01;
char    *recvdata;
DCLONG    *rdataleng;
DCLONG    inbufleng;
DCLONG    *time;
```

機能

メッセージのうち，一つのセグメントを受け取ります。一つの論理メッセージを受信するときは，セグメントの数だけ dc_mcf_receive 関数を呼び出します。

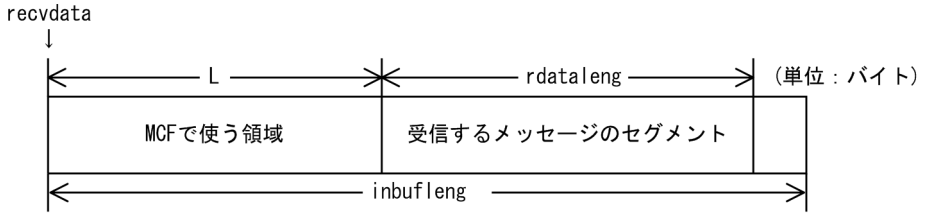
dc_mcf_receive 関数では，次に示すメッセージを受信できます。

- 相手システムから通信プロトコルを介して送られたメッセージ
- 自システムから通知された MCF イベント
- 自システムの UAP からアプリケーション起動（dc_mcf_execap 関数）で送られたメッセージ
- 自システムで mcfuevt コマンドを実行して送られたメッセージ
- ユーザタイマ監視を設定したときに指定したメッセージ

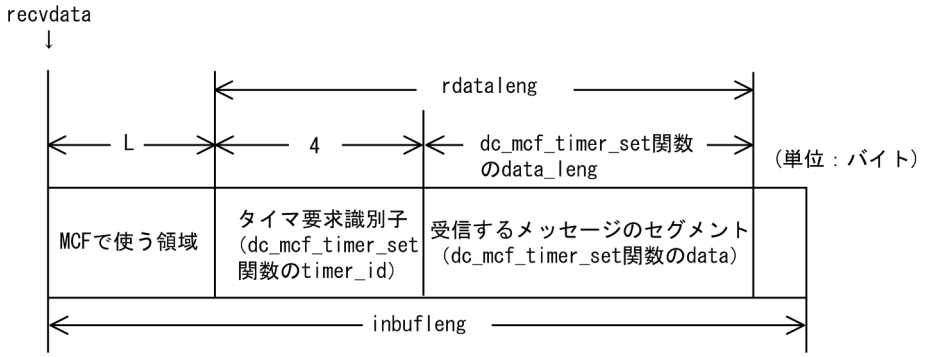
相手システムから通信プロトコルを介して送られたメッセージを受信する場合，通信プロトコルによって dc_mcf_receive 関数の文法が異なります。相手システムからのメッセージを受信する dc_mcf_receive 関数の文法については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

受信できる一つのセグメントの最大長は，1 メガバイトまでです。ただし，プロトコルによって，実際の最大長が小さいことがあります。詳細については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

セグメントを受信する領域の形式を次に示します。L は，バッファ形式 1 の場合は 8 バイト，バッファ形式 2 の場合は 4 バイトです。



また，ユーザタイマ監視を設定したときに指定したメッセージを受信する場合の形式を次に示します。L は，バッファ形式 1 の場合は 8 バイト，バッファ形式 2 の場合は 4 バイトです。



UAP で値を設定する引数

●action

メッセージの先頭セグメントを受信するかどうか，および使うバッファ形式を，次の形式で設定します。

```
{DCMCFFRST|DCMCFSEG} 〔|{DCMCFBUF1|DCMCFBUF2}〕
```

DCMCFFRST

先頭セグメントを受信するときに設定します。

メッセージが単一セグメントの場合も，DCMCFFRST を設定します。

DCMCFSEG

中間セグメントおよび最終セグメントを受信するときに設定します。

DCMCFBUF1

バッファ形式 1 を使うときに設定します。通常，バッファ形式 1 を使います。

DCMCFBUF2

バッファ形式 2 を使うときに設定します。

●commform

DCNOFLAGS を設定します。

●termnam【中間セグメント，最終セグメントを受信するとき】

入力元の論理端末名称を設定します。先頭セグメントを受信したときに返された論理端末名称を設定します。論理端末名称の最後にはヌル文字を付けます。

●resv01

NULL またはヌル文字列を設定します。

TP1/NET/XMAP3 の論理端末でメッセージを受信する場合の注意事項

resv01 には XMAP3 のドローで指定した次画面名（マップ名）が返されます。詳細については，マニュアル「OpenTP1 プロトコル TP1/NET/XMAP3 編」を参照してください。

●recvdata

セグメントを受信する領域を設定します。自システムから送信したメッセージの場合，受け取れるセグメントの最大長は 32000 バイトです。相手システムから送信されたメッセージの場合，受け取れるセグメントの最大長は，通信プロトコル対応製品別で異なります。

dc_mcf_receive 関数が終了すると，メッセージのうちセグメントが一つ返されます。

●inbufleng

セグメントを受信する領域の長さを設定します。

OpenTP1 から値が返される引数

●termnam【先頭セグメントを受信するとき】

入力元の論理端末名称が返されます。

中間セグメントおよび最終セグメントを受信するときは，ここで返された論理端末名称を termnam に設定します。

●recvdata

受信したセグメントの内容が返されます。

●rdataleng

受信したセグメントの長さが返されます。

●time

メッセージを受信した時刻が，1970 年 1 月 1 日 0 時 0 分 0 秒からの通算の秒数で返されます。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。
DCMCFRTN_71000	-12000	先頭セグメントを受信する dc_mcf_receive 関数を 2 回以上呼び出しています。中間セグメントおよび最終セグメントを受信する場合は，action に DCMCFSEG を設定して dc_mcf_receive 関数を呼び出してください。
DCMCFRTN_71001	-12001	メッセージの最終セグメントを受信したあとで，次のセグメントを受信する dc_mcf_receive 関数を呼び出しています。直前に呼び出した dc_mcf_receive 関数でメッセージはすべて受信しました。 このリターン値が返されたあとに，再び dc_mcf_receive 関数を呼び出した場合は，リターン値 DCMCFRTN_72000 が返されます。
DCMCFRTN_71002	-12002	メッセージキューへの入力処理時に障害が起こりました。
		メッセージキューが閉塞されています。
DCMCFRTN_72000	-13000	＜ MHP の実行でリターンした場合＞ 先頭セグメントを受信する dc_mcf_receive 関数を呼び出す前に，中間セグメントおよび最終セグメントを受信する dc_mcf_receive 関数を呼び出しています。先頭セグメントを受信する場合は，action に DCMCFFRST を設定して dc_mcf_receive 関数を呼び出してください。 リターン値 DCMCFRTN_71001 が返されたあとに，再び dc_mcf_receive 関数を呼び出しています。
		＜ SPP の実行でリターンした場合＞ SPP では dc_mcf_receive 関数を呼び出せません。
DCMCFRTN_72001	-13001	termnam に設定した論理端末名称が間違っています。
DCMCFRTN_72013	-13013	受信領域の長さを超えるセグメントを受信しました。受信領域の長さを超えた部分は切り捨てられました。
DCMCFRTN_72016	-13016	action に設定した値が間違っています。
		resv01 に設定した値が間違っています。
		引数に設定した値に間違いがあります。
DCMCFRTN_72024	-13024	commform に設定した値が間違っています。
DCMCFRTN_72025	-13025	action に設定したセグメント種別（DCMCFFRST または DCMCFSEG）の値が間違っています。
DCMCFRTN_72036	-13036	セグメントを受信する領域の長さが不足しています。バッファ形式 1 の場合は 9 バイト以上，バッファ形式 2 の場合は 5 バイト以上の領域を確保してください。

リターン値	リターン値（数値）	意味
上記以外		プログラムの破壊などによる，予期しないエラーが発生しました。

dc_mcf_recvsync

名称

同期型のメッセージの受信

形式

形式については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

機能

相手システムから送信された論理メッセージを，稼働中の UAP の処理で受信します。UAP から dc_mcf_recvsync 関数を呼び出すと，入力キューに格納されているメッセージのうち，関数に設定した論理端末名称からのメッセージを受信します。該当するメッセージがない場合は，メッセージを受信するまで dc_mcf_recvsync 関数は待ちます。このように，UAP からの dc_mcf_recvsync 関数の呼び出しと同期して，論理メッセージを受信します。

dc_mcf_recvsync 関数では，セグメント単位で論理メッセージを受信します。論理メッセージが一つのセグメントから構成される場合は，dc_mcf_recvsync 関数を 1 回呼び出して受信します。論理メッセージが複数のセグメントから構成される場合は，必要なセグメントの数だけ dc_mcf_recvsync 関数を呼び出して，一つの論理メッセージを受信します。

受信できる一つのセグメントの最大長は，1 メガバイトまでです。ただし，プロトコルによって，実際の最大長が小さいことがあります。詳細については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

dc_mcf_recvsync 関数がセグメントを受信する MCF の領域は，MCF で使う領域と，受信するメッセージのセグメントから構成されます。

引数に設定する値と関数のリターン値は，使う通信プロトコルによって異なります。詳細については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

dc_mcf_reply

名称

応答メッセージの送信

形式

形式については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

機能

相手システムへ，論理メッセージを応答送信します。dc_mcf_reply 関数は，dc_mcf_receive 関数で受信したメッセージを論理端末へ応答送信する関数です。

dc_mcf_reply 関数は，アプリケーションの型が ans 型，cont 型，または noans 型の MHP から呼び出せます。noans 型の MHP から呼び出す場合は，非応答型の MHP からの問い合わせ応答をするための設定（UAP 共通定義（mcfmuap）の-c オプションで noansreply オペランドに yes を指定）が必要です。

dc_mcf_reply 関数では，セグメント単位で論理メッセージを応答送信します。受信した論理メッセージが一つのセグメントから構成される場合は，dc_mcf_reply 関数を 1 回呼び出して応答送信します。受信した論理メッセージが複数のセグメントから構成される場合は，必要なセグメントの数だけ dc_mcf_reply 関数を呼び出して，一つの論理メッセージを応答送信します。

MCF で管理するアプリケーション（MHP のサービス関数）では，dc_mcf_reply 関数で論理メッセージを最後まで応答送信して，MHP が正常に終了してから，MCF がメッセージを送信します。このように，dc_mcf_reply 関数は MHP の処理とは非同期にメッセージを応答送信します。

送信できるメッセージの一つのセグメント長は，32 キロバイトまでです。ただし，プロトコルによって，実際の最大長が小さいことがあります。詳細については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

dc_mcf_reply 関数がセグメントを応答送信する MCF の領域は，MCF で使う領域と，応答送信するメッセージのセグメントから構成されます。

引数に設定する値と関数のリターン値は，使う通信プロトコルによって異なります。詳細については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

dc_mcf_resend

名称

メッセージの再送

形式

形式については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

機能

相手システムへ，送信済みの論理メッセージを再び送信します。再送するメッセージは，送信済みのメッセージとは別の，新しいメッセージとして扱います。どのメッセージを再送するかは，次に示す送信済みメッセージの情報で選択できます。

- 出力先の論理端末名称
- メッセージ出力通番
- メッセージ種別（一般分岐，優先分岐）

dc_mcf_resend 関数を使うノードでは，送信済みメッセージを保持するキュー（ディスクキュー）を使うことが前提です。

再送対象としたメッセージが送信されていない場合は，dc_mcf_resend 関数はエラーリターンします。出力キュー内に対象のメッセージがない場合もエラーリターンします。

引数に設定する値と関数のリターン値は，使う通信プロトコルによって異なります。詳細はマニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

注意事項

MCF マネジャ定義の UAP 共通定義（mcfmuap -c order）の指定によって，メッセージの再送順序が異なりますので注意してください。

dc_mcf_rollback

名称

MHP のロールバック

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int  dc_mcf_rollback(DCLONG action)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int  dc_mcf_rollback(action)
DCLONG      action;
```

機能

トランザクション属性を定義した MHP のサービスプログラムの開始から，dc_mcf_rollback 関数までの処理を取り消します。action に DCMCFRTRY を設定した場合は，MHP の開始から dc_mcf_rollback 関数までの処理を取り消して，取り消した処理の MHP をスケジュールし直します。

UAP で値を設定する引数

●action

部分回復の種別を設定します。次のどれか一つを設定します。

DCMCFRTRY

MHP を開始した時点から，dc_mcf_rollback 関数までの処理を取り消して，MHP を異常終了します。取り消した処理はスケジュールし直します（受信メッセージを，該当する入力キューの最後に格納し，MHP をスケジュールし直します）。

MHP は異常終了しますが，UAP 異常終了通知イベント（ERREVT3）は起動しません。また，アプリケーション異常終了限界回数にはカウントしません。アプリケーション属性定義（mcfaalcap）で異常終了時に閉塞する指定をしていても，アプリケーション，サービスグループ，およびサービスは閉塞しません。

DCMCFRRTN

MHP を開始した時点から，dc_mcf_rollback 関数までの処理を取り消して，リターンします。DCMCFRRTN を設定した dc_mcf_rollback 関数が正常に終了したあとの処理は，別のトランザクションとして処理します。

DCMCFNRTN

MHP を開始した時点から，dc_mcf_rollback 関数までの処理を取り消します。dc_mcf_rollback 関数からリターンしないで，MHP を異常終了します。

このとき，UAP 異常終了通知イベント（ERREVT3）を起動します。また，アプリケーション異常終了限界回数にカウントします。アプリケーション属性定義（mcfaalcap）の指定によっては，アプリケーション，サービスグループ，およびサービスを閉塞します。

設定する値とシステムの動作の関係を，次の表に示します。

表 2-2　action に設定する値とシステムの動作の関係（dc_mcf_rollback 関数）

設定値	MHP の動作	ERREVT3 の起動	各種閉塞処理
DCMCFRTRY	異常終了します。	起動しません。	閉塞しません。
DCMCFRRTN	関数がリターンします。	起動しません。	閉塞しません。
DCMCFNRTN	異常終了します。	起動します。	アプリケーション属性定義（mcfaalcap）の指定によっては，アプリケーション，サービスグループ，およびサービスが閉塞することがあります。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。
DCMCFRTN_72000	-13000	＜ MHP の実行でリターンした場合＞ dc_mcf_rollback 関数を呼び出した位置が間違っています。MHP で先頭セグメントを受信する dc_mcf_receive 関数を呼び出す前に，action に DCMCFRRTN を設定した dc_mcf_rollback 関数を呼び出しています。 非トランザクション属性の MHP から，dc_mcf_rollback 関数を呼び出しています。
		＜ SPP の実行でリターンした場合＞ SPP では，dc_mcf_rollback 関数は呼び出せません。
DCMCFRTN_72027	-13027	action に設定した値が間違っています。
上記以外		プログラムの破壊などによる，予期しないエラーが発生しました。

dc_mcf_send

名称

メッセージの送信

形式

形式については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

機能

相手システムへ，一方送信の論理メッセージを送信します。

dc_mcf_send 関数では，セグメント単位で論理メッセージを送信します。送信する論理メッセージが一つのセグメントから構成される場合は，dc_mcf_send 関数を 1 回呼び出して送信します。送信する論理メッセージが複数のセグメントから構成される場合は，必要なセグメントの数だけ dc_mcf_send 関数を呼び出して，一つの論理メッセージを送信します。

MCF で管理するアプリケーション（MHP のサービス関数），または SPP は dc_mcf_send 関数は UAP の処理とは非同期にメッセージを送信します。

送信できるメッセージの一つのセグメント長は，32 キロバイトまでです。ただし，プロトコルによって，実際の最大長が小さいことがあります。詳細については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

dc_mcf_send 関数がセグメントを送信する MCF の領域は，MCF で使う領域と，送信するメッセージのセグメントから構成されます。

引数に設定する値と関数のリターン値は，使う通信プロトコルによって異なります。詳細については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

注意事項

MCF マネジャ定義の UAP 共通定義（mcfmuap -c order）の指定によって，メッセージの送信順序が異なりますので注意してください。

dc_mcf_sendrecv

名称

同期型のメッセージの送受信

形式

形式については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

機能

相手システムへ，稼働中の UAP の処理で論理メッセージを送信して，応答を受信します。UAP から dc_mcf_sendrecv 関数を呼び出してから，関数に指定した論理端末にメッセージを送信し終わり，応答が返ってくるまで，dc_mcf_sendrecv 関数は待ちます。このように，UAP からの dc_mcf_sendrecv 関数の呼び出しと同期して，論理メッセージを送受信します。

メッセージの最終セグメントを dc_mcf_sendrecv 関数で送信して，MCF がメッセージを送信すると，dc_mcf_sendrecv 関数は応答待ちの状態になります。

dc_mcf_sendrecv 関数では，セグメント単位で論理メッセージを送信します。論理メッセージが一つのセグメントから構成される場合は，dc_mcf_sendrecv 関数を 1 回呼び出して送信します。論理メッセージが複数のセグメントから構成される場合は，必要なセグメントの数だけ dc_mcf_sendrecv 関数を呼び出して，一つの論理メッセージを送信します。

論理端末からの応答メッセージのセグメントを MCF ですべて受け取ると，最終セグメントを送信した dc_mcf_sendrecv 関数で，先頭セグメントだけを受信します。中間セグメント以降は，dc_mcf_recvsync 関数で受信します。

受信できる一つのセグメントの最大長は，1 メガバイトまでです。また，送信できるメッセージの一つのセグメント長は，32 キロバイトまでです。ただし，プロトコルによって，実際の最大長が小さいことがあります。詳細については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

dc_mcf_sendrecv 関数がセグメントを送信する MCF の領域は，MCF で使う領域と，送信するメッセージのセグメントから構成されます。

引数に設定する値と関数のリターン値は，使う通信プロトコルによって異なります。詳細については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

dc_mcf_sendsync

名称

同期型のメッセージの送信

形式

形式については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

機能

相手システムへ，稼働中の UAP の処理で論理メッセージを送信します。UAP から dc_mcf_sendsync 関数を呼び出してから，出力キューにメッセージが書き込まれ関数に指定した論理端末にメッセージを送信し終わるまで，dc_mcf_sendsync 関数は待ちます。このように，UAP からの dc_mcf_sendsync 関数の呼び出しと同期して，論理メッセージを送信します。

dc_mcf_sendsync 関数では，セグメント単位で論理メッセージを送信します。論理メッセージが一つのセグメントから構成される場合は，dc_mcf_sendsync 関数を 1 回呼び出して送信します。論理メッセージが複数のセグメントから構成される場合は，必要なセグメントの数だけ dc_mcf_sendsync 関数を呼び出して，一つの論理メッセージを送信します。

送信できるメッセージの一つのセグメント長は，32 キロバイトまでです。ただし，プロトコルによって，実際の最大長が小さいことがあります。詳細については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

dc_mcf_sendsync 関数がセグメントを送信する MCF の領域は，MCF で使う領域と，送信するメッセージのセグメントから構成されます。

引数に設定する値と関数のリターン値は，使う通信プロトコルによって異なります。詳細については，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

dc_mcf_tactcn

名称

コネクションの確立

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int dc_mcf_tactcn (DCLONG action, dcmcf_tactcnopt *cnopt,
                   char *proinf, DCLONG *resv02, char *resv03,
                   char *resv04)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int dc_mcf_tactcn (action, cnopt, proinf, resv02, resv03, resv04)
DCLONG             action;
dcmcf_tactcnopt    *cnopt;
char               *proinf;
DCLONG             *resv02;
char               *resv03;
char               *resv04;
```

機能

コネクションを確立します。

なお，dc_mcf_tactcn 関数の正常終了は，コネクション確立要求をプロトコル製品が正常に受け付けたことを意味します。このため，相手システムとのコネクションの確立が正常に完了したことを示すものではありません。

dc_mcf_tactcn 関数の呼び出し後にコネクションに関する何らかの処理をする場合は，dc_mcf_tlscn 関数を用いてコネクションの状態を確認してください。

UAP で値を設定する引数

●action

確立するコネクションの指定方法，および通信プロトコルに依存する機能を使用するかどうかを，次の形式で設定します。

```
{DCMCFLE | DCMCFCN}  〔 | DCMCFPRO〕
```

DCMCFLE

　確立するコネクションを論理端末名称で指定するときに設定します。

DCMCFCN

確立するコネクションをコネクション ID で指定するときに設定します。

DCMCFPRO

通信プロトコルに依存する機能を使用するときに設定します。

●cnopt

この関数の対象となったコネクションの情報を，構造体 dcmcf_tactcnopt に設定します。

構造体の形式を次に示します。

```
typedef struct {
                DCLONG        mcfid;          …MCF通信プロセス識別子
                char          resv01[4];      …予備領域
                char          idnam[9];       …論理端末名称,
                                                コネクションID
                char          resv02[7];      …予備領域
                char          resv03[112];    …予備領域
                char          scnnam[9];      …MCF使用領域
                char          resv04[7];      …予備領域
                char          yournam[9];     …MCF使用領域
                char          resv05[7];      …予備領域
                char          hostnam[143];   …MCF使用領域
                char          resv06[17];     …予備領域
                char          resv07[184];    …予備領域
                } dcmcf_tactcnopt;
```

- mcfid

  処理対象のコネクションを持つ MCF 通信サービスの MCF 通信プロセス識別子※を設定します。設定できる範囲は 0～239 です。

  論理端末名称を使用してコネクションの確立を要求する場合は，無効となります。

  0 を指定すると，該当するコネクション ID が属する MCF 通信サービスを検索します。MCF 通信サービスが多い構成や UAP からこの関数を多数発行する場合は，MCF 通信プロセス識別子の指定をお勧めします。

  注※

  MCF 環境定義（mcftenv -s）で指定する MCF 通信プロセス識別子は 16 進数とみなしてください。例えば，MCF 通信プロセス識別子が 10 の場合，16 を設定してください。

- resv01

  領域をヌル文字で埋めます。

- idnam

  確立するコネクションの論理端末名称，またはコネクション ID を設定します。論理端末名称，またはコネクション ID は 8 バイト以内で設定し，文字列の最後にヌル文字を付けます。

- resv02，resv03，scnnam，resv04，yournam，resv05，hostnam，resv06，resv07

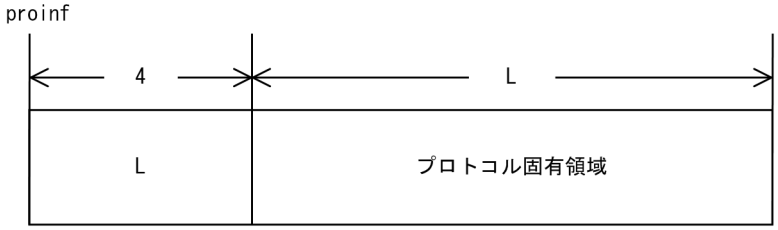
  領域をヌル文字で埋めます。

●proinf

プロトコル固有領域を設定します。

通信プロトコルに依存する機能を使用しないときは，NULL を設定します。

プロトコル固有領域の形式を次に示します。



プロトコル固有領域は 1024 バイトまで設定できます。

設定する値は使う通信プロトコルによって異なります。詳細は，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

●resv02，resv03，resv04

NULL を設定します。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。
DCMCFRTN_71001	-12001	MCF が開始処理中のため，dc_mcf_tactcn 関数が受け付けられません。
DCMCFRTN_71002	-12002	MCF が終了処理中のため，dc_mcf_tactcn 関数が受け付けられません。
DCMCFRTN_71004	-12004	dc_mcf_tactcn 関数の処理中にメモリ不足が発生しました。
DCMCFRTN_71005	-12005	通信障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71006	-12006	内部障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71007	-12007	指定されたコネクション名は登録されていません。
DCMCFRTN_71008	-12008	指定された論理端末名称は登録されていません。
DCMCFRTN_71009	-12009	dc_mcf_tactcn 関数が，該当する MCF 通信プロセスではサポートされていません。

リターン値	リターン値（数値）	意味
DCMCFRTN_71010	-12010	MCF 通信プロセスにコネクションの確立を要求しましたが，受け付けられませんでした。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71011	-12011	コネクションが削除されているため，dc_mcf_tactcn 関数が受け付けられません。
DCMCFRTN_71014	-12014	TP1/NET/NCSB，もしくは TP1/NET/X25-Extended の論理端末名称を指定しています。または，TP1/NET/OSI-TP，もしくは TP1/NET/TCP/IP のコネクショングループ名を指定しています。
DCMCFRTN_72050	-13050	action に未サポートのフラグを設定しています。
DCMCFRTN_72051	-13051	cnopt に NULL が設定されています。
DCMCFRTN_72052	-13052	＜ action に DCMCFPRO を設定していない場合＞ proinf に NULL が設定されていません。
		＜ action に DCMCFPRO を設定した場合＞ proinf が示すプロトコル固有領域長 L に 0 未満，または 1025 以上の値を設定しています。
DCMCFRTN_72053	-13053	resv02 に NULL が設定されていません。
DCMCFRTN_72054	-13054	resv03 に NULL が設定されていません。
DCMCFRTN_72055	-13055	resv04 に NULL が設定されていません。
DCMCFRTN_72060	-13060	action には DCMCFLE と DCMCFCN を同時に設定できません。
DCMCFRTN_72061	-13061	dcmcf_tactcnopt の mcfid に 0 未満または 240 以上の値が設定されています。
DCMCFRTN_72062	-13062	dcmcf_tactcnopt の resv01 がヌル文字で埋められていません。
DCMCFRTN_72063	-13063	dcmcf_tactcnopt の idnam の先頭がヌル文字です。
DCMCFRTN_72064	-13064	dcmcf_tactcnopt の resv02 がヌル文字で埋められていません。
DCMCFRTN_72065	-13065	dcmcf_tactcnopt の resv03 がヌル文字で埋められていません。
DCMCFRTN_72066	-13066	dcmcf_tactcnopt の scnnam がヌル文字で埋められていません。
DCMCFRTN_72067	-13067	dcmcf_tactcnopt の resv04 がヌル文字で埋められていません。
DCMCFRTN_72068	-13068	dcmcf_tactcnopt の yournam がヌル文字で埋められていません。

リターン値	リターン値（数値）	意味
DCMCFRTN_72069	-13069	dcmcf_tactcnopt の resv05 がヌル文字で埋められていません。
DCMCFRTN_72070	-13070	dcmcf_tactcnopt の hostnam がヌル文字で埋められていません。
DCMCFRTN_72071	-13071	dcmcf_tactcnopt の resv06 がヌル文字で埋められていません。
DCMCFRTN_72072	-13072	dcmcf_tactcnopt の resv07 がヌル文字で埋められていません。
DCMCFRTN_72073	-13073	dcmcf_tactcnopt の idnam に設定された文字数が 9 バイト以上です。
DCMCFRTN_72074	-13074	dcmcf_tactcnopt の idnam に設定された文字列中に不正な文字があります。

dc_mcf_tactle

名称

論理端末の閉塞解除

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int dc_mcf_tactle (DCLONG action, dcmcf_tactleopt *leopt,
                   char *proinf, DCLONG *resv02,
                   char *resv03, char *resv04)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int dc_mcf_tactle (action, leopt, proinf, resv02, resv03, resv04)
DCLONG             action;
dcmcf_tactleopt    *leopt;
char               *proinf;
DCLONG             *resv02;
char               *resv03;
char               *resv04;
```

機能

論理端末の閉塞を解除します。

なお，dc_mcf_tactle 関数の正常終了は，論理端末の閉塞解除要求をプロトコル製品が正常に受け付けたことを意味します。このため，論理端末の閉塞解除が正常に完了したことを示すものではありません。

dc_mcf_tactle 関数の呼び出し後に論理端末に関する何らかの処理をする場合は，dc_mcf_tlsle 関数を用いて論理端末の状態を確認してください。

UAP で値を設定する引数

●action

閉塞解除する論理端末の指定方法，および通信プロトコルに依存する機能を使用するかどうかを，次の形式で設定します。

```
DCMCFLE〔|DCMCFPRO〕
```

DCMCFLE

論理端末名称で指定するときに設定します。

DCMCFPRO

通信プロトコルに依存する機能を使用するときに設定します。

●leopt

この関数の対象となった論理端末の情報を，構造体 dcmcf_tactleopt に設定します。

構造体の形式を次に示します。

```
typedef struct {
                DCLONG    mcfid;          …MCF通信プロセス識別子
                char      resv01[4];      …予備領域
                char      idnam[9];       …論理端末名称
                char      resv02[7];      …予備領域
                char      resv03[112];    …予備領域
                char      resv04[376];    …予備領域
                } dcmcf_tactleopt;
```

- mcfid

  処理対象の論理端末を持つ MCF 通信サービスの MCF 通信プロセス識別子※を設定します。設定できる範囲は 0～239 です。

  0 を指定すると，該当する論理端末名称が属する MCF 通信サービスを検索します。MCF 通信サービスが多い構成や UAP からこの関数を多数発行する場合は，MCF 通信プロセス識別子の指定をお勧めします。

  注※

  MCF 環境定義（mcftenv -s）で指定する MCF 通信プロセス識別子は 16 進数とみなしてください。

  例えば，MCF 通信プロセス識別子が 10 の場合，16 を設定してください。

- resv01

  領域をヌル文字で埋めます。

- idnam

  閉塞解除する論理端末の論理端末名称を設定します。論理端末名称は 8 バイト以内で設定し，文字列の最後にヌル文字を付けます。

- resv02，resv03，resv04

  領域をヌル文字で埋めます。

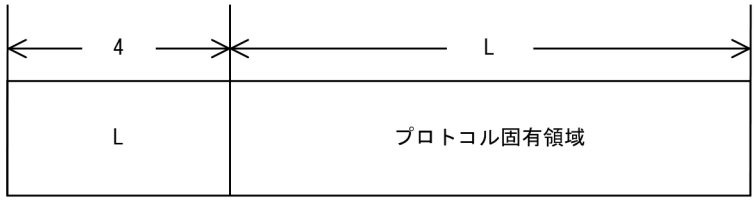
●proinf

プロトコル固有領域を設定します。

通信プロトコルに依存する機能を使用しないときは，NULL を設定します。

プロトコル固有領域の形式を次に示します。

proinf



プロトコル固有領域は 1024 バイトまで設定できます。

設定する値は使う通信プロトコルによって異なります。詳細は，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

●resv02，resv03，resv04

NULL を設定します。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。
DCMCFRTN_71001	-12001	MCF が開始処理中のため，dc_mcf_tactle 関数が受け付けられません。
DCMCFRTN_71002	-12002	MCF が終了処理中のため，dc_mcf_tactle 関数が受け付けられません。
DCMCFRTN_71004	-12004	dc_mcf_tactle 関数の処理中にメモリ不足が発生しました。
DCMCFRTN_71005	-12005	通信障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71006	-12006	内部障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71008	-12008	指定された論理端末名称は登録されていません。
DCMCFRTN_71009	-12009	dc_mcf_tactle 関数が，該当する MCF 通信プロセスではサポートされていません。
DCMCFRTN_71010	-12010	MCF 通信プロセスに論理端末の閉塞の解除を要求しましたが，受け付けられませんでした。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71011	-12011	論理端末が削除されているため，dc_mcf_tactle 関数が受け付けられません。
DCMCFRTN_72050	-13050	action に DCMCFLE が指定されていません。
		action に未サポートのフラグを設定しています。
DCMCFRTN_72051	-13051	leopt に NULL が設定されています。

リターン値	リターン値（数値）	意味
DCMCFRTN_72052	-13052	＜ action に DCMCFPRO を設定していない場合＞ proinf に NULL が設定されていません。
		＜ action に DCMCFPRO を設定した場合＞ proinf が示すプロトコル固有領域長 L に 0 未満，または 1025 以上の値を設定しています。
DCMCFRTN_72053	-13053	resv02 に NULL が設定されていません。
DCMCFRTN_72054	-13054	resv03 に NULL が設定されていません。
DCMCFRTN_72055	-13055	resv04 に NULL が設定されていません。
DCMCFRTN_72061	-13061	dcmcf_tactleopt の mcfid に 0 未満または 240 以上の値が設定されています。
DCMCFRTN_72062	-13062	dcmcf_tactleopt の resv01 がヌル文字で埋められていません。
DCMCFRTN_72063	-13063	dcmcf_tactleopt の idnam の先頭がヌル文字です。
DCMCFRTN_72064	-13064	dcmcf_tactleopt の resv02 がヌル文字で埋められていません。
DCMCFRTN_72065	-13065	dcmcf_tactleopt の resv03 がヌル文字で埋められていません。
DCMCFRTN_72067	-13067	dcmcf_tactleopt の resv04 がヌル文字で埋められていません。
DCMCFRTN_72073	-13073	dcmcf_tactleopt の idnam に設定された文字数が 9 バイト以上です。
DCMCFRTN_72074	-13074	dcmcf_tactleopt の idnam に設定された文字列中に不正な文字があります。

dc_mcf_tdctcn

名称

コネクションの解放

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int dc_mcf_tdctcn (DCLONG action, dcmcf_tdctcnopt *cnopt,
                   char *proinf, DCLONG *resv02, char *resv03,
                   char *resv04)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int dc_mcf_tdctcn (action, cnopt, proinf, resv02, resv03, resv04)
DCLONG               action;
dcmcf_tdctcnopt      *cnopt;
char                 *proinf;
DCLONG               *resv02;
char                 *resv03;
char                 *resv04;
```

機能

コネクションを解放します。

なお，dc_mcf_tdctcn 関数の正常終了は，コネクション解放要求をプロトコル製品が正常に受け付けたことを意味します。このため，相手システムとのコネクションの解放が正常に完了したことを示すものではありません。

dc_mcf_tdctcn 関数の呼び出し後にコネクションに関する何らかの処理をする場合は，dc_mcf_tlscn 関数を用いてコネクションの状態を確認してください。

UAP で値を設定する引数

●action

解放するコネクションの指定方法，および通信プロトコルに依存する機能を使用するかどうかを，次の形式で設定します。

```
{DCMCFLE | DCMCFCN}  〔 | DCMCFFRC〕  〔 | DCMCFPRO〕
```

DCMCFLE

解放するコネクションを論理端末名称で指定するときに設定します。

DCMCFCN

解放するコネクションをコネクション ID で指定するときに設定します。

DCMCFFRC

コネクションを強制的に解放するときに設定します。

DCMCFPRO

通信プロトコルに依存する機能を使用するときに設定します。

●cnopt

この関数の対象となったコネクションの情報を，構造体 dcmcf_tdctcnopt に設定します。

構造体の形式を次に示します。

```
typedef struct {
                DCLONG    mcfid;          …MCF通信プロセス識別子
                char      resv01[4];      …予備領域
                char      idnam[9];       …論理端末名称,
                                            コネクションID
                char      resv02[7];      …予備領域
                char      resv03[112];    …予備領域
                char      scnnam[9];      …MCF使用領域
                char      resv04[7];      …予備領域
                char      resv05[360];    …予備領域
                } dcmcf_tdctcnopt;
```

- mcfid

  処理対象のコネクションを持つ MCF 通信サービスの MCF 通信プロセス識別子※を設定します。設定できる範囲は 0～239 です。

  論理端末名称を使用してコネクションの解放を要求する場合は，無効となります。

  0 を指定すると，該当するコネクション ID が属する MCF 通信サービスを検索します。MCF 通信サービスが多い構成や UAP からこの関数を多数発行する場合は，MCF 通信プロセス識別子の指定をお勧めします。

  注※

  MCF 環境定義（mcftenv -s）で指定する MCF 通信プロセス識別子は 16 進数とみなしてください。例えば，MCF 通信プロセス識別子が 10 の場合，16 を設定してください。

- resv01

  領域をヌル文字で埋めます。

- idnam

  解放するコネクションの論理端末名称，またはコネクション ID を設定します。論理端末名称，またはコネクション ID は 8 バイト以内で設定し，文字列の最後にヌル文字を付けます。

- resv02，resv03，scnnam，resv04，resv05

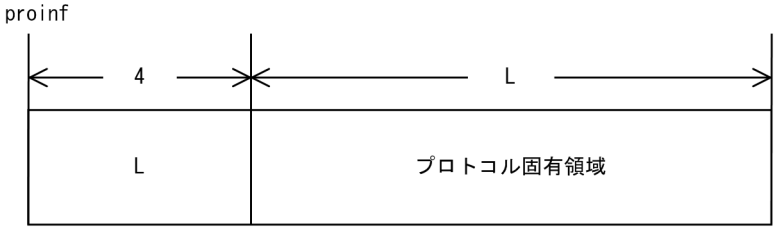
  領域をヌル文字で埋めます。

●proinf

プロトコル固有領域を設定します。

通信プロトコルに依存する機能を使用しないときは，NULL を設定します。

プロトコル固有領域の形式を次に示します。



プロトコル固有領域は 1024 バイトまで設定できます。

設定する値は使う通信プロトコルによって異なります。詳細は，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

●resv02，resv03，resv04

NULL を設定します。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。
DCMCFRTN_71001	-12001	MCF が開始処理中のため，dc_mcf_tdctcn 関数が受け付けられません。
DCMCFRTN_71002	-12002	MCF が終了処理中のため，dc_mcf_tdctcn 関数が受け付けられません。
DCMCFRTN_71004	-12004	dc_mcf_tdctcn 関数の処理中にメモリ不足が発生しました。
DCMCFRTN_71005	-12005	通信障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71006	-12006	内部障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71007	-12007	指定されたコネクション名は登録されていません。
DCMCFRTN_71008	-12008	指定された論理端末名称は登録されていません。
DCMCFRTN_71009	-12009	dc_mcf_tdctcn 関数が，該当する MCF 通信プロセスではサポートされていません。

リターン値	リターン値（数値）	意味
DCMCFRTN_71010	-12010	MCF 通信プロセスにコネクションの解放を要求しましたが，受け付けられませんでした。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71011	-12011	コネクションが削除されているため，dc_mcf_tdctcn 関数が受け付けられません。
DCMCFRTN_71014	-12014	TP1/NET/NCSB，もしくは TP1/NET/X25-Extended の論理端末名称を指定しています。または，TP1/NET/OSI-TP，もしくは TP1/NET/TCP/IP のコネクショングループ名を指定しています。
DCMCFRTN_72050	-13050	action に未サポートのフラグを設定しています。
DCMCFRTN_72051	-13051	cnopt に NULL が設定されています。
DCMCFRTN_72052	-13052	＜ action に DCMCFPRO を設定していない場合＞ proinf に NULL が設定されていません。
		＜ action に DCMCFPRO を設定した場合＞ proinf が示すプロトコル固有領域長 L に 0 未満，または 1025 以上の値を設定しています。
DCMCFRTN_72053	-13053	resv02 に NULL が設定されていません。
DCMCFRTN_72054	-13054	resv03 に NULL が設定されていません。
DCMCFRTN_72055	-13055	resv04 に NULL が設定されていません。
DCMCFRTN_72060	-13060	action には DCMCFLE と DCMCFCN を同時に設定できません。
DCMCFRTN_72061	-13061	dcmcf_tdctcnopt の mcfid に 0 未満または 240 以上の値が設定されています。
DCMCFRTN_72062	-13062	dcmcf_tdctcnopt の resv01 がヌル文字で埋められていません。
DCMCFRTN_72063	-13063	dcmcf_tdctcnopt の idnam の先頭がヌル文字です。
DCMCFRTN_72064	-13064	dcmcf_tdctcnopt の resv02 がヌル文字で埋められていません。
DCMCFRTN_72065	-13065	dcmcf_tdctcnopt の resv03 がヌル文字で埋められていません。
DCMCFRTN_72066	-13066	dcmcf_tdctcnopt の scnnam がヌル文字で埋められていません。
DCMCFRTN_72067	-13067	dcmcf_tdctcnopt の resv04 がヌル文字で埋められていません。
DCMCFRTN_72069	-13069	dcmcf_tdctcnopt の resv05 がヌル文字で埋められていません。

リターン値	リターン値（数値）	意味
DCMCFRTN_72073	-13073	dcmcf_tdctcnopt の idnam に設定された文字数が 9 バイト以上です。
DCMCFRTN_72074	-13074	dcmcf_tdctcnopt の idnam に設定された文字列中に不正な文字があります。

dc_mcf_tdctle

名称

論理端末の閉塞

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int dc_mcf_tdctle (DCLONG action, dcmcf_tdctleopt *leopt,
                   char *proinf, DCLONG *resv02,
                   char *resv03, char *resv04)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int dc_mcf_tdctle (action, leopt, proinf, resv02, resv03, resv04)
DCLONG             action;
dcmcf_tdctleopt    *leopt;
char               *proinf;
DCLONG             *resv02;
char               *resv03;
char               *resv04;
```

機能

論理端末を閉塞します。

なお，dc_mcf_tdctle 関数の正常終了は，論理端末の閉塞要求をプロトコル製品が正常に受け付けたことを意味します。このため，論理端末の閉塞が正常に完了したことを示すものではありません。

dc_mcf_tdctle 関数の呼び出し後に論理端末に関する何らかの処理をする場合は，dc_mcf_tlsle 関数を用いて論理端末の状態を確認してください。

UAP で値を設定する引数

●action

閉塞する論理端末の指定方法，および通信プロトコルに依存する機能を使用するかどうかを，次の形式で設定します。

```
DCMCFLE〔|DCMCFPRO〕
```

DCMCFLE

論理端末名称で指定するときに設定します。

DCMCFPRO

通信プロトコルに依存する機能を使用するときに設定します。

●leopt

この関数の対象となったコネクションの情報を，構造体 dcmcf_tdctleopt に設定します。

構造体の形式を次に示します。

```
typedef struct {
                DCLONG    mcfid;          …MCF通信プロセス識別子
                char      resv01[4];      …予備領域
                char      idnam[9];       …論理端末名称
                char      resv02[7];      …予備領域
                char      resv03[112];    …予備領域
                char      resv04[376];    …予備領域
                } dcmcf_tdctleopt;
```

- mcfid

  処理対象の論理端末を持つ MCF 通信サービスの MCF 通信プロセス識別子※を設定します。設定できる範囲は 0～239 です。

  0 を指定すると，該当する論理端末名称が属する MCF 通信サービスを検索します。MCF 通信サービスが多い構成や UAP からこの関数を多数発行する場合は，MCF 通信プロセス識別子の指定をお勧めします。

  注※

  MCF 環境定義（mcftenv -s）で指定する MCF 通信プロセス識別子は 16 進数とみなしてください。

  例えば，MCF 通信プロセス識別子が 10 の場合，16 を設定してください。

- resv01

  領域をヌル文字で埋めます。

- idnam

  閉塞する論理端末の論理端末名称を設定します。論理端末名称は 8 バイト以内で設定し，文字列の最後にヌル文字を付けます。

- resv02，resv03，resv04

  領域をヌル文字で埋めます。

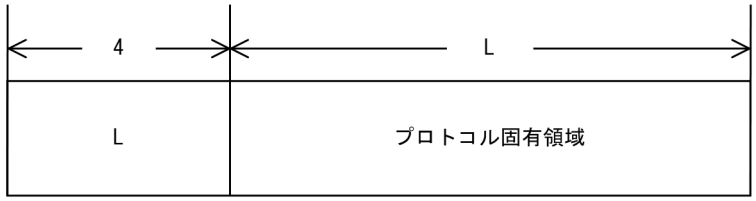
●proinf

プロトコル固有領域を設定します。

通信プロトコルに依存する機能を使用しないときは，NULL を設定します。

プロトコル固有領域の形式を次に示します。

proinf



プロトコル固有領域は 1024 バイトまで設定できます。

設定する値は使う通信プロトコルによって異なります。詳細は，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

●resv02，resv03，resv04

NULL を設定します。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。
DCMCFRTN_71001	-12001	MCF が開始処理中のため，dc_mcf_tdctle 関数が受け付けられません。
DCMCFRTN_71002	-12002	MCF が終了処理中のため，dc_mcf_tdctle 関数が受け付けられません。
DCMCFRTN_71004	-12004	dc_mcf_tdctle 関数の処理中にメモリ不足が発生しました。
DCMCFRTN_71005	-12005	通信障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71006	-12006	内部障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71008	-12008	指定された論理端末名称は登録されていません。
DCMCFRTN_71009	-12009	dc_mcf_tdctle 関数が，該当する MCF 通信プロセスではサポートされていません。
DCMCFRTN_71010	-12010	MCF 通信プロセスに論理端末の閉塞を要求しましたが，受け付けられませんでした。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71011	-12011	論理端末が削除されているため，dc_mcf_tdctle 関数が受け付けられません。
DCMCFRTN_72050	-13050	action に DCMCFLE が指定されていません。
		action に未サポートのフラグを設定しています。
DCMCFRTN_72051	-13051	leopt に NULL が設定されています。

リターン値	リターン値（数値）	意味
DCMCFRTN_72052	-13052	＜ action に DCMCFPRO を設定していない場合＞ proinf に NULL が設定されていません。
		＜ action に DCMCFPRO を設定した場合＞ proinf が示すプロトコル固有領域長 L に 0 未満，または 1025 以上の値を設定しています。
DCMCFRTN_72053	-13053	resv02 に NULL が設定されていません。
DCMCFRTN_72054	-13054	resv03 に NULL が設定されていません。
DCMCFRTN_72055	-13055	resv04 に NULL が設定されていません。
DCMCFRTN_72061	-13061	dcmcf_tdctleopt の mcfid に 0 未満または 240 以上の値が設定されています。
DCMCFRTN_72062	-13062	dcmcf_tdctleopt の resv01 がヌル文字で埋められていません。
DCMCFRTN_72063	-13063	dcmcf_tdctleopt の idnam の先頭がヌル文字です。
DCMCFRTN_72064	-13064	dcmcf_tdctleopt の resv02 がヌル文字で埋められていません。
DCMCFRTN_72065	-13065	dcmcf_tdctleopt の resv03 がヌル文字で埋められていません。
DCMCFRTN_72067	-13067	dcmcf_tdctleopt の resv04 がヌル文字で埋められていません。
DCMCFRTN_72073	-13073	dcmcf_tdctleopt の idnam に設定された文字数が 9 バイト以上です。
DCMCFRTN_72074	-13074	dcmcf_tdctleopt の idnam に設定された文字列中に不正な文字があります。

dc_mcf_tdlqle

名称

論理端末の出力キュー削除

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int dc_mcf_tdlqle (DCLONG action, dcmcf_tdlqleopt *leopt,
                   char *resv01, DCLONG *resv02,
                   char *resv03, char *resv04)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int dc_mcf_tdlqle (action, leopt, resv01, resv02, resv03, resv04)
DCLONG             action;
dcmcf_tdlqleopt    *leopt;
char               *resv01;
DCLONG             *resv02;
char               *resv03;
char               *resv04;
```

機能

論理端末の出力キューを削除します。

出力キューを正常に削除すると，未処理送信メッセージ廃棄通知イベント（ERREVTA）を通知します。

UAP で値を設定する引数

●action

論理端末名称で指定することを示す，DCMCFLE を設定します。

●leopt

この関数の対象となったコネクションの情報を，構造体 dcmcf_tdlqleopt に設定します。

構造体の形式を次に示します。

```
typedef struct {
                DCLONG    mcfid;          …MCF通信プロセス識別子
                char      resv01[4];      …予備領域
                char      idnam[9];       …論理端末名称
                char      resv02[7];      …予備領域
                char      resv03[112];    …予備領域
```

```
        char      resv04[376];   …予備領域
        } dcmcf_tdlqleopt;
```

- mcfid

  処理対象の論理端末を持つ MCF 通信サービスの MCF 通信プロセス識別子※を設定します。設定できる範囲は 0～239 です。

  0 を指定すると，該当する論理端末名称が属する MCF 通信サービスを検索します。MCF 通信サービスが多い構成や UAP から dc_mcf_tdlqle 関数を多数発行する場合は，MCF 通信プロセス識別子の指定をお勧めします。

  注※

  MCF 環境定義（mcftenv -s）で指定する MCF 通信プロセス識別子は 16 進数とみなしてください。例えば，MCF 通信プロセス識別子が 10 の場合，16 を設定してください。

- resv01

  領域をヌル文字で埋めます。

- idnam

  出力キューを削除する論理端末の論理端末名称を設定します。論理端末名称は 8 バイト以内で設定し，文字列の最後にヌル文字を付けます。

- resv02，resv03，resv04

  領域をヌル文字で埋めます。

●resv01，resv02，resv03，resv04

NULL を設定します。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。
DCMCFRTN_71001	-12001	MCF が開始処理中のため，dc_mcf_tdlqle 関数が受け付けられません。
DCMCFRTN_71002	-12002	MCF が終了処理中のため，dc_mcf_tdlqle 関数が受け付けられません。
DCMCFRTN_71004	-12004	dc_mcf_tdlqle 関数の処理中にメモリ不足が発生しました。
DCMCFRTN_71005	-12005	通信障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71006	-12006	内部障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71008	-12008	指定された論理端末名称は登録されていません。

リターン値	リターン値（数値）	意味
DCMCFRTN_71009	-12009	dc_mcf_tdlqle 関数が，該当する MCF 通信プロセスではサポートされていません。
DCMCFRTN_71010	-12010	MCF 通信プロセスに論理端末の出力キューの削除を要求しましたが，受け付けられませんでした。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71011	-12011	論理端末が削除されているため，dc_mcf_tdlqle 関数が受け付けられません。
DCMCFRTN_71017	-12017	論理端末が閉塞されていないため，dc_mcf_tdlqle 関数が受け付けられません。
DCMCFRTN_71018	-12018	セッションが終了されていないため，dc_mcf_tdlqle 関数が受け付けられません。
DCMCFRTN_71019	-12019	代行送信中のため，dc_mcf_tdlqle 関数が受け付けられません。
DCMCFRTN_72050	-13050	action に DCMCFLE が指定されていません。
		action に未サポートのフラグを設定しています。
DCMCFRTN_72051	-13051	leopt に NULL が設定されています。
DCMCFRTN_72052	-13052	resv01 に NULL が設定されていません。
DCMCFRTN_72053	-13053	resv02 に NULL が設定されていません。
DCMCFRTN_72054	-13054	resv03 に NULL が設定されていません。
DCMCFRTN_72055	-13055	resv04 に NULL が設定されていません。
DCMCFRTN_72061	-13061	dcmcf_tdlqleopt の mcfid に 0 未満または 240 以上の値が設定されています。
DCMCFRTN_72062	-13062	dcmcf_tdlqleopt の resv01 がヌル文字で埋められていません。
DCMCFRTN_72063	-13063	dcmcf_tdlqleopt の idnam の先頭がヌル文字です。
DCMCFRTN_72064	-13064	dcmcf_tdlqleopt の resv02 がヌル文字で埋められていません。
DCMCFRTN_72065	-13065	dcmcf_tdlqleopt の resv03 がヌル文字で埋められていません。
DCMCFRTN_72067	-13067	dcmcf_tdlqleopt の resv04 がヌル文字で埋められていません。
DCMCFRTN_72073	-13073	dcmcf_tdlqleopt の idnam に設定された文字数が 9 バイト以上です。
DCMCFRTN_72074	-13074	dcmcf_tdlqleopt の idnam に設定された文字列中に不正な文字があります。

dc_mcf_tempget

名称

一時記憶データの受け取り

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int  dc_mcf_tempget(DCLONG action,char *getdata,DCLONG gtempleng,
                    DCLONG *gdataleng,char *resv01)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int  dc_mcf_tempget(action,getdata,gtempleng,gdataleng,resv01)
DCLONG       action;
char         *getdata;
DCLONG       gtempleng;
DCLONG       *gdataleng;
char         *resv01;
```

機能

継続問い合わせ応答用一時記憶領域に格納されている一時記憶データを受け取ります。

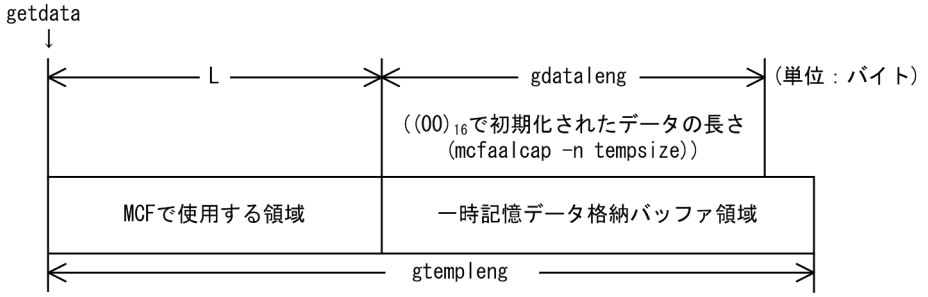
gtempleng の長さ（9～32008 バイト（バッファ形式 1 の場合），または 7～32006 バイト（バッファ形式 2 の場合））を超える一時記憶データがある場合，超えた部分については切り捨てます。

dc_mcf_tempget 関数実行時，gtempleng から 8（バッファ形式 1 の場合），または 6（バッファ形式 2 の場合）を減算した値と比べて一時記憶データ長が短い場合，一時記憶データを受け取り領域に設定します。残りの受け取り領域については何も設定しません。

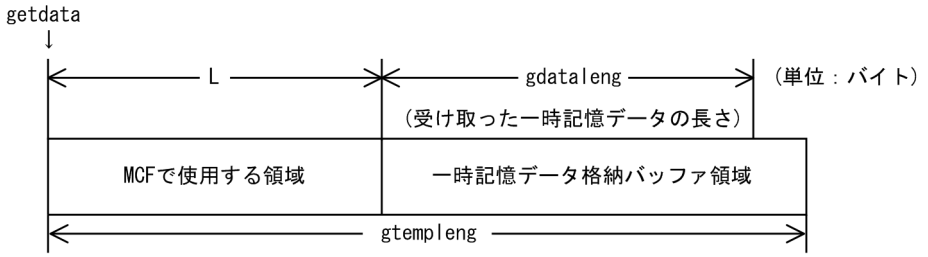
dc_mcf_tempget 関数実行時，初期状態（継続問い合わせ応答開始後，dc_mcf_tempput 関数を 1 回も実行していない状態）の場合，アプリケーション属性定義（mcfaalcap -n）の tempsize オペランドで指定した長さの$(00)_{16}$の一時記憶データがあるものとして，dc_mcf_tempget 関数を実行します。

受け取り領域の形式を次に示します。L は，バッファ形式 1 では 8 バイト，バッファ形式 2 では 6 バイトです。

- dc_mcf_tempput 関数未実行（継続問い合わせ応答開始後，dc_mcf_tempput 関数を実行していない（初期状態））



- dc_mcf_tempput 関数実行済み（継続問い合わせ応答開始後，dc_mcf_tempput 関数を 1 回以上実行）



UAP で値を設定する引数

●action

使用するバッファ形式を設定します。

```
{DCMCFBUF1|DCMCFBUF2}
```

DCMCFBUF1

バッファ形式 1 を使用する場合に設定します。

DCMCFBUF2

バッファ形式 2 を使用する場合に設定します。

●getdata

受け取り領域を設定します。

処理終了後，getdata には OpenTP1 から値が返されます。

●gtempleng

バッファ形式 1 の場合，受け取り領域長を 9～32008 バイトで設定します。

バッファ形式 2 の場合，受け取り領域長を 7～32006 バイトで設定します。

●resv01

ヌル文字列を設定します。

OpenTP1 から値が返される引数

●getdata

受け取った一時記憶データが返されます。初期状態の場合，継続問い合わせ応答用一時記憶領域の長さ（アプリケーション属性定義（mcfaalcap -n）の tempsize オペランドの指定値）分だけ$(00)_{16}$が埋められます。

●gdataleng

前回更新した一時記憶データの長さが返されます。初期状態の場合，継続問い合わせ応答用一時記憶領域の長さ（アプリケーション属性定義（mcfaalcap -n）の tempsize オペランドの指定値）が返されます。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。
DCMCFRTN_72000	-13000	SPP では dc_mcf_tempget 関数を呼び出せません。
DCMCFRTN_72013	-13013	gtempleng の設定値を超える一時記憶データを受け取りました。gtempleng の設定値を超えた部分は切り捨てられました。
DCMCFRTN_72016	-13016	action に設定した値が間違っています。
		resv01 に設定した値が間違っています。
		引数に設定した値に間違いがあります。
DCMCFRTN_72036	-13036	gtempleng の設定値が不足しています。バッファ形式 1 の場合は 9 バイト以上，バッファ形式 2 の場合は 7 バイト以上の領域を確保してください。
DCMCFRTN_72101	-13101	継続問い合わせ応答型でないアプリケーションで，dc_mcf_tempget 関数を呼び出しています。
DCMCFRTN_72106	-13106	先頭セグメントを受信する dc_mcf_receive 関数を呼び出す前に，dc_mcf_tempget 関数を呼び出しています。
DCMCFRTN_72107	-13107	dc_mcf_contend 関数を呼び出したあとで，dc_mcf_tempget 関数を呼び出しています。
上記以外		プログラムの破壊などによる，予期しないエラーが発生しました。

dc_mcf_tempput

名称

一時記憶データの更新

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int  dc_mcf_tempput(DCLONG action,char *putdata,DCLONG pdataleng,
                    char *resv01)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int  dc_mcf_tempput(action,putdata,pdataleng,resv01)
DCLONG    action;
char      *putdata;
DCLONG    pdataleng;
char      *resv01;
```

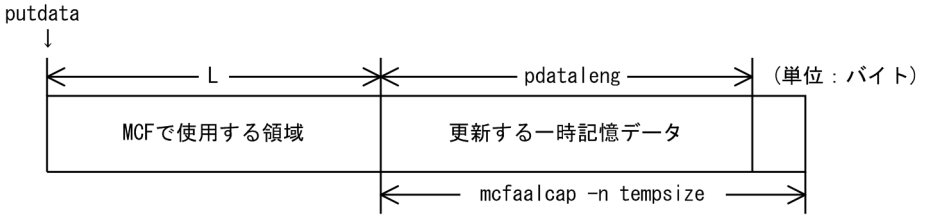
機能

継続問い合わせ応答用一時記憶領域に格納されている一時記憶データを更新します。

更新要求をする前に，必ず受け取り要求をしてください。

アプリケーション属性定義（mcfaalcap -n）の tempsize オペランドには，更新する一時記憶データ長（pdataleng）以上の値を指定してください。

更新する領域の形式を次に示します。L は，バッファ形式 1 では 8 バイト，バッファ形式 2 では 6 バイトです。



UAP で値を設定する引数

●action

使用するバッファ形式を設定します。

```
{DCMCFBUF1|DCMCFBUF2}
```

DCMCFBUF1

バッファ形式 1 を使用する場合に設定します。

DCMCFBUF2

バッファ形式 2 を使用する場合に設定します。

●putdata

一時記憶データの更新データが格納されている領域を設定します。

●pdataleng

一時記憶データの更新データ長を設定します。

●resv01

ヌル文字列を設定します。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。
DCMCFRTN_71103	-12103	一時記憶データを更新するための領域をメモリ上に確保できませんでした。
DCMCFRTN_72000	-13000	SPP では dc_mcf_tempput 関数を呼び出せません。
DCMCFRTN_72016	-13016	action に設定した値が間違っています。
		resv01 に設定した値が間違っています。
		引数に設定した値に間違いがあります。
DCMCFRTN_72035	-13035	pdataleng に設定した更新データの長さが，アプリケーション属性定義（mcfaalcap -n）の tempsize オペランドで定義した長さを超えています。
		pdataleng に 0 バイト，またはマイナス値を設定しています。
DCMCFRTN_72101	-13101	継続問い合わせ応答型でないアプリケーションで，dc_mcf_tempput 関数を呼び出しています。
DCMCFRTN_72105	-13105	dc_mcf_tempget 関数を呼び出す前に，dc_mcf_tempput 関数を呼び出しています。
DCMCFRTN_72106	-13106	先頭セグメントを受信する dc_mcf_receive 関数を呼び出す前に，dc_mcf_tempput 関数を呼び出しています。
DCMCFRTN_72107	-13107	dc_mcf_contend 関数を呼び出したあとで，dc_mcf_tempput 関数を呼び出しています。

リターン値	リターン値（数値）	意味
上記以外		プログラムの破壊などによる，予期しないエラーが発生しました。

dc_mcf_timer_cancel

名称

ユーザタイマ監視の取り消し

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int dc_mcf_timer_cancel(DCLONG flags,DCLONG timer_id,char *lename)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int dc_mcf_timer_cancel(flags,timer_id,lename)
DCLONG    flags;
DCLONG    timer_id;
char      *lename;
```

機能

dc_mcf_timer_set 関数で設定したユーザタイマ監視を取り消します。

設定したユーザタイマ監視は，dc_mcf_timer_cancel 関数が正常にリターンした時点で取り消されています。

また，この関数を呼び出した時点で，設定したユーザタイマ監視がタイムアウトしていて，MHP がすでに起動されている場合には，dc_mcf_timer_cancel 関数が DCMCFER_PARAM_TIM_ID でエラーリターンします。

dc_mcf_timer_cancel 関数は，ユーザサーバからだけ呼び出せます。

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

●timer_id

ユーザタイマ監視を設定したときと同じタイマ要求識別子を指定します。

●lename

ユーザタイマ監視を設定したときと同じ論理端末名称を指定します。論理端末名称の最後にはヌル文字を付けます。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCMCFER_PARAM_TIM_ID	-11910	timer_id に指定したタイマ要求識別子は登録されていません。
		すでにタイムアウトが発生して MHP が起動されているか，またはすでにユーザタイマ監視が取り消されています。
DCMCFER_PARAM_FLAGS	-11911	flags に設定した値が間違っています。
DCMCFER_PARAM_LENAME	-11912	lename に設定した値が間違っています。
DCMCFER_NO_DEFINE	-11916	要求された機能は MCF で定義されていません。

dc_mcf_timer_set

名称

ユーザタイマ監視の設定

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int dc_mcf_timer_set(DCLONG flags,DCLONG timer,DCLONG timer_id,
                     char *lename,
                     char *apname,char *data,DCLONG data_leng,
                     DCLONG resv01,DCLONG resv02)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int dc_mcf_timer_set(flags, timer, timer_id, lename,
                     apname, data,
                     data_leng, resv01, resv02)
DCLONG    flags;
DCLONG    timer;
DCLONG    timer_id;
char      *lename;
char      *apname;
char      *data;
DCLONG    data_leng;
DCLONG    resv01;
DCLONG    resv02;
```

機能

ユーザで任意の時間監視をするには，UAP から dc_mcf_timer_set 関数でユーザタイマ監視を設定します。dc_mcf_timer_set 関数を呼び出すには，MCF 通信構成定義 mcfttim の-p オプションに usertime=yes を指定する必要があります。

dc_mcf_timer_set 関数は，ユーザサーバからだけ呼び出せます。

timer に指定した時間（単位：秒）を経過した（タイムアウトが発生した）場合，lename で指定した論理端末からイベントを生成し，apname に設定したアプリケーション名の MHP を起動させます。MHP から dc_mcf_timer_set 関数を呼び出す場合，lename を省略できます。省略した場合，入力元論理端末を仮定します。

タイムアウト発生時に起動させる MHP は，非応答型（noans 型）の MHP でなくてはなりません。この MHP にメッセージが渡されたときの形式は「dc_mcf_receive」を参照してください。

dc_mcf_timer_set 関数で設定したユーザタイマ監視は，同一の timer_id および lename を指定した dc_mcf_timer_cancel 関数を呼び出すことで取り消せます。

時間監視は，dc_mcf_timer_set 関数を呼び出した直後から行います。

同時に時間監視できる数は，MCF 通信構成定義 mcfttim の-p オプションの timereqno オペランドに指定した最大タイマ監視要求数までです。

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

●timer

dc_mcf_timer_set 関数を呼び出してから，何秒後に MHP を起動させるかを設定します。設定できる秒数は，1 から 360000 まで（1 秒から 100 時間まで）です。

> **注意事項**
>
> 時間監視の精度は秒単位です。また，タイマ定義（mcfttim -t）の btim オペランドで指定する時間監視間隔でタイムアウトが発生したかどうかを監視しています。このため，timer に設定した監視時間と実際にタイムアウトを検出する時間には秒単位の誤差が生じます。そのため，タイミングによっては，設定した監視時間よりも短い時間で起動することがあります。監視時間が小さくなるほど，誤差の影響を受けやすくなりますので，監視時間は 3（単位：秒）以上の値の設定を推奨します。

●timer_id

タイマ要求識別子を設定します。

timer_id はこのタイマを一意に識別するための情報です。timer_id は，lename で指定した論理端末内で必ずユニークになるようにしてください。

●lename

タイムアウトが発生したときにイベントを生成する論理端末名称を 8 バイト以内で指定します。論理端末名称の最後にはヌル文字を付けます。論理端末名称は，アプリケーション起動サービスの論理端末を指定してもかまいません。

MHP からこの関数を呼び出す場合は，論理端末名称を省略できます。省略する場合には，ヌル文字列を指定します。省略した場合，入力元論理端末を仮定します。

●apname

起動させる MHP のアプリケーション名を指定します。このアプリケーションの属性は，lename で指定した論理端末を持つ MCF 通信サーバの MCF 通信構成定義 mcftenv -a オプションで指定した MCF アプリケーション定義内のアプリケーション属性定義（mcfaalcap）に定義してください。アプリケーション名は最大 8 バイトです。アプリケーション名の最後にはヌル文字を付けます。MHP は非応答型（noans 型）の MHP でなければなりません。指定するアプリケーション名はユーザイベントでなければなりません。

●data

起動させる MHP に渡す，メッセージのセグメントの内容を設定します。複数のセグメントを設定できません。起動させる MHP に渡すセグメントがない場合は，ヌル文字を設定します。

●data_leng

起動させる MHP に渡すセグメントの長さを設定します。起動させる MHP に渡すセグメントがない場合は，0 を設定します。

設定できる値の範囲は，0～256 です。設定できる値の最大値は MCF 通信構成定義 mcfttim の-p オプションの msgsize オペランドに指定した最大メッセージ長に依存します。

●resv01

DCNOFLAGS を設定します。

●resv02

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCMCFER_INVALID_ARGS	-11900	引数に設定した値が間違っています。
DCMCFER_PARAM_TIMER	-11909	timer に設定した値が間違っています。
DCMCFER_PARAM_TIM_ID	-11910	timer_id に指定したタイマ要求識別子は，すでに登録されています。
DCMCFER_PARAM_FLAGS	-11911	flags に設定した値が間違っています。
DCMCFER_PARAM_LENAME	-11912	lename に設定した値が間違っています。
DCMCFER_PARAM_APNAME	-11913	apname に設定した値が間違っています。
DCMCFER_PARAM_LENG	-11914	data_leng に設定した値が間違っています。
DCMCFER_PARAM_DATA	-11915	data に設定した値が間違っています。
DCMCFER_NO_DEFINE	-11916	要求された機能は MCF で定義されていません。
DCMCFER_NO_TIMER_ENT	-11917	タイマ登録領域に空きがないためユーザタイマ監視が設定できません。タイマ登録領域を確保するため，MCF 通信構成定義 mcfttim の-p オプションの timereqno オペランドの値を見直してください。必要に応じて，MCF マネジャ定義 mcfmcomn の-p オプション，およびシステム環境定義 static_shmpool_size オペランドの値を確認してください。

dc_mcf_tlscn

名称

コネクションの状態取得

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int dc_mcf_tlscn (DCLONG action, dcmcf_tlscnopt *cnopt,
                  char *resv01, DCLONG *resv02,
                  char *resv03, DCLONG *infcnt,
                  dcmcf_cninf *inf, char *resv04)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int dc_mcf_tlscn (action, cnopt, resv01, resv02, resv03, infcnt,
                  inf, resv04)
DCLONG            action;
dcmcf_tlscnopt    *cnopt;
char              *resv01;
DCLONG            *resv02;
char              *resv03;
DCLONG            *infcnt;
dcmcf_cninf       *inf;
char              *resv04;
```

機能

コネクションの状態を取得します。

UAP で値を設定する引数

●action

状態を取得するコネクションの指定方法を次の形式で設定します。

```
{DCMCFLE | DCMCFCN}
```

DCMCFLE

状態を取得するコネクションを論理端末名称で指定するときに設定します。

DCMCFCN

状態を取得するコネクションをコネクション ID で指定するときに設定します。

●cnopt

この関数の対象となったコネクションの情報を，構造体 dcmcf_tlscnopt に設定します。

構造体の形式を次に示します。

```
typedef struct {
                DCLONG    mcfid;        …MCF通信プロセス識別子
                char      resv01[4];    …予備領域
                char      idnam[9];     …論理端末名称,
                                          コネクションID
                char      resv02[7];    …予備領域
                char      resv03[112];  …予備領域
                char      resv04[376];  …予備領域
                } dcmcf_tlscnopt;
```

- mcfid

  処理対象のコネクションを持つ MCF 通信サービスの MCF 通信プロセス識別子※を設定します。設定できる範囲は 0～239 です。

  論理端末名称を使用してコネクションの状態取得を要求する場合は，無効となります。

  0 を指定すると，該当するコネクション ID が属する MCF 通信サービスを検索します。MCF 通信サービスが多い構成や UAP からこの関数を多数発行する場合は，MCF 通信プロセス識別子の指定をお勧めします。

  注※

  MCF 環境定義（mcftenv -s）で指定する MCF 通信プロセス識別子は 16 進数とみなしてください。例えば，MCF 通信プロセス識別子が 10 の場合，16 を設定してください。

- resv01

  領域をヌル文字で埋めます。

- idnam

  状態を取得するコネクションの論理端末名称，またはコネクション ID を設定します。論理端末名称，またはコネクション ID は 8 バイト以内で設定し，文字列の最後にヌル文字を付けます。

- resv02，resv03，resv04

  領域をヌル文字で埋めます。

●resv01，resv02，resv03

NULL を設定します。

●infcnt

コネクション状態を格納する領域 dcmcf_cninf の個数として，1 を設定します。

処理終了後は，該当するコネクションの個数が返されます。

●inf

コネクション状態を格納する領域 dcmcf_cninf を設定します。

「構造体 dcmcf_cninf のサイズ×infcnt」バイト数分の領域が必要です。

●resv04

NULL を設定します。

OpenTP1 から値が返される引数

●infcnt

この関数の対象となったコネクションの個数が返されます。

●inf

この関数の対象となったコネクションの情報が，構造体 dcmcf_cninf で返されます。

構造体の形式を次に示します。

```
typedef struct {
                char     idnam[9];     …コネクションID
                char     resv01[7];    …予備領域
                char     pnam[4];      …プロトコル種別
                DCLONG   status;       …コネクション状態
                char     resv02[40];   …予備領域
                } dcmcf_cninf;
```

- idnam
  要求したコネクションのコネクション ID が設定されます。
- resv01
  領域がヌル文字で埋められます
- pnam
  要求したコネクションのプロトコル種別が設定されます。

  'UA '
  　TP1/NET/User Agent（OSAS/UA プロトコル）

  'TP '
  　TP1/NET/OSI-TP（OSI TP プロトコル）

  'XP '
  　TP1/NET/XMAP3

  'NIF'
  　TP1/NET/OSAS-NIF（NIF プロトコル）

'SL2'

TP1/NET/SLU - TypeP2（SLUTYPE-P プロトコル（2 次局））

'TCP'

TP1/NET/TCP/IP（TCP/IP プロトコル）

- status

要求したコネクションの状態として，次の値が設定されます。

DCMCF_CNST_ACT

コネクションが確立されていることを示します。

DCMCF_CNST_ACT_B

コネクションが確立処理中であることを示します。

DCMCF_CNST_DCT

コネクションが解放されていることを示します。

DCMCF_CNST_DCT_B

コネクションが解放処理中であることを示します。

- resv02

領域がヌル文字で埋められます。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。
DCMCFRTN_71001	-12001	MCF が開始処理中のため，dc_mcf_tlscn 関数が受け付けられません。
DCMCFRTN_71004	-12004	dc_mcf_tlscn 関数の処理中にメモリ不足が発生しました。
DCMCFRTN_71005	-12005	通信障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71006	-12006	内部障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71007	-12007	指定されたコネクション名は登録されていません。
DCMCFRTN_71008	-12008	指定された論理端末名称は登録されていません。
DCMCFRTN_71009	-12009	dc_mcf_tlscn 関数が，該当する MCF 通信プロセスではサポートされていません。
DCMCFRTN_71010	-12010	MCF 通信プロセスにコネクションの状態取得を要求しましたが，受け付けられませんでした。原因については，メッセージログファイルを参照してください。

リターン値	リターン値（数値）	意味
DCMCFRTN_71011	-12011	コネクションが削除されているため，dc_mcf_tlscn 関数が受け付けられません。
DCMCFRTN_71014	-12014	TP1/NET/NCSB，もしくは TP1/NET/X25-Extended の論理端末名称を指定しています。または，TP1/NET/OSI-TP，もしくは TP1/NET/TCP/IP のコネクショングループ名を指定しています。
DCMCFRTN_72050	-13050	action に未サポートのフラグを設定しています。
DCMCFRTN_72051	-13051	cnopt に NULL が設定されています。
DCMCFRTN_72052	-13052	resv01 に NULL が設定されていません。
DCMCFRTN_72053	-13053	resv02 に NULL が設定されていません。
DCMCFRTN_72054	-13054	resv03 に NULL が設定されていません。
DCMCFRTN_72055	-13055	resv04 に NULL が設定されていません。
DCMCFRTN_72056	-13056	infcnt に NULL が設定されています。
DCMCFRTN_72057	-13057	inf に NULL が設定されています。
DCMCFRTN_72060	-13060	action には DCMCFLE と DCMCFCN を同時に設定できません。
DCMCFRTN_72061	-13061	dcmcf_tlscnopt の mcfid に 0 未満または 240 以上の値が設定されています。
DCMCFRTN_72062	-13062	dcmcf_tlscnopt の resv01 がヌル文字で埋められていません。
DCMCFRTN_72063	-13063	dcmcf_tlscnopt の idnam の先頭がヌル文字です。
DCMCFRTN_72064	-13064	dcmcf_tlscnopt の resv02 がヌル文字で埋められていません。
DCMCFRTN_72065	-13065	dcmcf_tlscnopt の resv03 がヌル文字で埋められていません。
DCMCFRTN_72067	-13067	dcmcf_tlscnopt の resv04 がヌル文字で埋められていません。
DCMCFRTN_72073	-13073	dcmcf_tlscnopt の idnam に設定された文字数が 9 バイト以上です。
DCMCFRTN_72074	-13074	dcmcf_tlscnopt の idnam に設定された文字列中に不正な文字があります。
DCMCFRTN_72076	-13076	infcnt に 1 が設定されていません。

dc_mcf_tlscom

名称

MCF 通信サービスの状態取得

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int dc_mcf_tlscom (DCLONG action, char *resv01, DCLONG *infcnt,
                   dcmcf_svinf *inf, char *resv02)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int dc_mcf_tlscom (action, resv01, infcnt, inf, resv02)
DCLONG             action;
char               *resv01;
DCLONG             *infcnt;
dcmcf_svinf        *inf;
char               *resv02;
```

機能

MCF 通信サービスまたはアプリケーション起動サービスの状態を取得します。

UAP で値を設定する引数

●action

DCNOFLAGS を設定します。

●resv01

NULL を設定します。

●infcnt

MCF 通信サービスまたはアプリケーション起動サービスの状態を格納する領域 dcmcf_svinf の個数を設定します。

処理終了後は，該当する MCF 通信サービスの個数が返されます。

●inf

MCF 通信サービスまたはアプリケーション起動サービスの状態を格納する領域 dcmcf_svinf を設定します。

「構造体 dcmcf_svinf のサイズ×infcnt」バイト数分の領域が必要です。

●resv02

NULL を設定します。

OpenTP1 から値が返される引数

●infcnt

MCF サービスに登録されている MCF 通信サービスまたはアプリケーション起動サービスの個数が返されます。

●inf

MCF サービスに登録されている MCF 通信サービスまたはアプリケーション起動サービスの情報が構造体 dcmcf_svinf で返されます。

構造体の形式を次に示します。

```
typedef struct {
                DCLONG    mcfid;        …MCF通信プロセス識別子,
                                          アプリケーション起動
                                          プロセス識別子
                char      svname[9];    …MCF通信サービス名
                char      resv01[7];    …予備領域
                char      pnam[20];     …プロトコル種別
                DCLONG    status;       …MCF通信サービスの状態
                char      resv02[20];   …予備領域
                } dcmcf_svinf;
```

- mcfid
  MCF 通信プロセス識別子またはアプリケーション起動プロセス識別子が設定されます。
- svname
  MCF 通信サービス名が設定されます。
- resv01
  領域がヌル文字で埋められます。
- pnam
  プロトコル種別が設定されます。

  'MCF△△△△△△△△△△△△△△△△'
  　TP1/Message Control のアプリケーション起動サービス

  'User△Agent△△△△△△△△△△'
  　TP1/NET/User Agent（OSAS/UA プロトコル）

  'TP△△△△△△△△△△△△△△△△△△'
  　TP1/NET/OSI-TP（OSI TP プロトコル）

'XMAP3△△△△△△△△△△△△△△△'

TP1/NET/XMAP3

'OSAS-NIF△△△△△△△△△△△△'

TP1/NET/OSAS-NIF（NIF/OSI プロトコル）

'NET/SLUP2△△△△△△△△△△△△'

TP1/NET/SLU - TypeP2（SLUTYPE-P プロトコル（2 次局））

'TCP/IP△△△△△△△△△△△△△△'

TP1/NET/TCP/IP（TCP/IP プロトコル）

'UDP/IP△△△△△△△△△△△△△△'

TP1/NET/UDP（UDP プロトコル）

- status

  MCF 通信サービスまたはアプリケーション起動サービスの状態として，次の値が設定されます。

  DCMCF_SVST_OFLN

  サービスが停止されていることを示します。

  DCMCF_SVST_START

  サービスが準備中であることを示します。

  DCMCF_SVST_ONLN

  サービスが開始されている，または終了準備中であることを示します。

  DCMCF_SVST_PREEND

  サービスが部分停止の終了準備中であることを示します。

  DCMCF_SVST_END

  サービスが終了中であることを示します。

- resv02

  領域がヌル文字で埋められます。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。
DCMCFRTN_71001	-12001	MCF が開始処理中のため，dc_mcf_tlscom 関数が受け付けられません。
DCMCFRTN_71004	-12004	dc_mcf_tlscom 関数の処理中にメモリ不足が発生しました。
DCMCFRTN_71005	-12005	通信障害が発生しました。原因については，メッセージログファイルを参照してください。

リターン値	リターン値（数値）	意味
DCMCFRTN_71006	-12006	内部障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_72013	-13013	MCF 通信サービスまたはアプリケーション起動サービスの数が infcnt の指定値を超えました。infcnt の指定値を超えた MCF 通信サービスまたはアプリケーション起動サービスの情報は切り捨てられました。
DCMCFRTN_72050	-13050	action に DCNOFLAGS が設定されていません。
DCMCFRTN_72052	-13052	resv01 に NULL が設定されていません。
DCMCFRTN_72053	-13053	resv02 に NULL が設定されていません。
DCMCFRTN_72056	-13056	infcnt に NULL が設定されています。
DCMCFRTN_72057	-13057	inf に NULL が設定されています。
DCMCFRTN_72076	-13076	infcnt に 0 以下の値が設定されています。

dc_mcf_tlsle

名称

論理端末の状態取得

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int dc_mcf_tlsle (DCLONG action, dcmcf_tlsleopt *leopt,
                  char *resv01, DCLONG *resv02,
                  char *resv03, DCLONG *infcnt,
                  dcmcf_leinf2 *inf, char *resv04)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int dc_mcf_tlsle (action, leopt, resv01, resv02, resv03, infcnt,
                  inf, resv04)
DCLONG            action;
dcmcf_tlsleopt    *leopt;
char              *resv01;
DCLONG            *resv02;
char              *resv03;
DCLONG            *infcnt;
dcmcf_leinf2      *inf;
char              *resv04;
```

機能

論理端末の状態を取得します。

UAP で値を設定する引数

●action

論理端末名称で指定することを示す，DCMCFLE を設定します。

●leopt

この関数の対象となったコネクションの情報を，構造体 dcmcf_tlsleopt に設定します。

構造体の形式を次に示します。

```
typedef struct {
                DCLONG    mcfid;        …MCF通信プロセス識別子
                char      resv01[4];    …予備領域
                char      idnam[9];     …論理端末名称
                char      resv02[7];    …予備領域
                char      resv03[112];  …予備領域
```

```
        char      resv04[376];  …予備領域
        } dcmcf_tlsleopt;
```

- mcfid

  処理対象の論理端末を持つ MCF 通信サービスの MCF 通信プロセス識別子※を設定します。設定できる範囲は 0～239 です。

  0 を指定すると，該当する論理端末名称が属する MCF 通信サービスを検索します。MCF 通信サービスが多い構成や UAP からこの関数を多数発行する場合は，MCF 通信プロセス識別子の指定をお勧めします。

  注※

  MCF 環境定義（mcftenv -s）で指定する MCF 通信プロセス識別子は 16 進数とみなしてください。例えば，MCF 通信プロセス識別子が 10 の場合，16 を設定してください。

- resv01

  領域をヌル文字で埋めます。

- idnam

  状態を取得する論理端末の論理端末名称を設定します。論理端末名称は 8 バイト以内で設定し，文字列の最後にヌル文字を付けます。

- resv02，resv03，resv04

  領域をヌル文字で埋めます。

●resv01，resv02，resv03

NULL を設定します。

●infcnt

論理端末の状態を格納する領域 dcmcf_leinf2 の個数として，1 を設定します。

処理終了後は，該当する論理端末の個数が返されます。

●inf

論理端末の状態を格納する領域 dcmcf_leinf2 を設定します。

「構造体 dcmcf_leinf2 のサイズ×infcnt」バイト数分の領域が必要です。

●resv04

NULL を設定します。

OpenTP1 から値が返される引数

●infcnt

この関数の対象となった論理端末の個数が返されます。

●inf

この関数の対象となった論理端末の情報が構造体 dcmcf_leinf2 で返されます。

構造体の形式を次に示します。

```
typedef struct {
                char      idnam[9];    …論理端末名称
                char      resv01[7];   …予備領域
                char      resv02[4];   …予備領域
                DCLONG    status;      …論理端末状態
                char      resv03[40];  …予備領域
                } dcmcf_leinf2;
```

- idnam
  要求した論理端末の論理端末名称が設定されます。
- resv01，resv02
  領域がヌル文字で埋められます。
- status
  要求した論理端末の状態として，次の値が設定されます。
  DCMCF_LEST_ACT
  　論理端末が閉塞解除されていることを示します。
  DCMCF_LEST_DCT
  　論理端末が閉塞されていることを示します。
- resv03
  領域がヌル文字で埋められます。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。
DCMCFRTN_71001	-12001	MCF が開始処理中のため，dc_mcf_tlsle 関数が受け付けられません。
DCMCFRTN_71004	-12004	dc_mcf_tlsle 関数の処理中にメモリ不足が発生しました。
DCMCFRTN_71005	-12005	通信障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71006	-12006	内部障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71008	-12008	指定された論理端末名称は登録されていません。

リターン値	リターン値（数値）	意味
DCMCFRTN_71009	-12009	dc_mcf_tlsle 関数が，該当する MCF 通信プロセスではサポートされていません。
DCMCFRTN_71010	-12010	MCF 通信プロセスに論理端末の状態取得を要求しましたが，受け付けられませんでした。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71011	-12011	論理端末が削除されているため，dc_mcf_tlsle 関数が受け付けられません。
DCMCFRTN_72050	-13050	action に DCMCFLE が指定されていません。
		action に未サポートのフラグを設定しています。
DCMCFRTN_72051	-13051	leopt に NULL が設定されています。
DCMCFRTN_72052	-13052	resv01 に NULL が設定されていません。
DCMCFRTN_72053	-13053	resv02 に NULL が設定されていません。
DCMCFRTN_72054	-13054	resv03 に NULL が設定されていません。
DCMCFRTN_72055	-13055	resv04 に NULL が設定されていません。
DCMCFRTN_72056	-13056	infcnt に NULL が設定されています。
DCMCFRTN_72057	-13057	inf に NULL が設定されています。
DCMCFRTN_72061	-13061	dcmcf_tlsleopt の mcfid に 0 未満または 240 以上の値が設定されています。
DCMCFRTN_72062	-13062	dcmcf_tlsleopt の resv01 がヌル文字で埋められていません。
DCMCFRTN_72063	-13063	dcmcf_tlsleopt の idnam の先頭がヌル文字です。
DCMCFRTN_72064	-13064	dcmcf_tlsleopt の resv02 がヌル文字で埋められていません。
DCMCFRTN_72065	-13065	dcmcf_tlsleopt の resv03 がヌル文字で埋められていません。
DCMCFRTN_72067	-13067	dcmcf_tlsleopt の resv04 がヌル文字で埋められていません。
DCMCFRTN_72073	-13073	dcmcf_tlsleopt の idnam に設定された文字数が 9 バイト以上です。
DCMCFRTN_72074	-13074	dcmcf_tlsleopt の idnam に設定された文字列中に不正な文字があります。
DCMCFRTN_72076	-13076	infcnt に 1 が設定されていません。

dc_mcf_tlsln

名称

サーバ型コネクションの確立要求の受付状態取得

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int dc_mcf_tlsln (DCLONG action, DCLONG mcfid, char *resv01,
                  DCLONG *infcnt, dcmcf_lninf *inf, char *resv02)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int dc_mcf_tlsln (action, mcfid, resv01, infcnt, inf, resv02)
DCLONG         action;
DCLONG         mcfid;
char           *resv01;
DCLONG         *infcnt;
dcmcf_lninf    *inf;
char           *resv02;
```

機能

サーバ型コネクションの確立要求の受付状態を取得します。

UAP で値を設定する引数

●action

DCNOFLAGS を設定します。

●mcfid

処理対象の MCF 通信サービスの MCF 通信プロセス識別子※を設定します。設定できる範囲は 1～239 です。

注※

MCF 環境定義（mcftenv -s）で指定する MCF 通信プロセス識別子は 16 進数とみなしてください。

例えば，MCF 通信プロセス識別子が 10 の場合，16 を設定してください。

●resv01

NULL を設定します。

●infcnt

サーバ型コネクションの確立要求の受付状態を格納する領域 dcmcf_lninf の個数として，1 を設定します。

処理終了後は，該当する MCF 通信サービスの個数が返されます。

●inf

サーバ型コネクションの確立要求の受付状態を格納する領域 dcmcf_lninf を設定します。

「構造体 dcmcf_lninf のサイズ×infcnt」バイト数分の領域が必要です。

●resv02

NULL を設定します。

OpenTP1 から値が返される引数

●infcnt

この関数の対象となった MCF 通信サービスの個数が返されます。

●inf

この関数の対象となった MCF 通信サービスのサーバ型コネクションの確立要求の受付状態が構造体 dcmcf_lninf で返されます。

構造体の形式を次に示します。

```
typedef struct {
                DCLONG    status;       …受付状態
                char      resv01[60];   …予備領域
                } dcmcf_lninf;
```

- status

  サーバ型コネクションの確立要求の受付状態として，次の値が設定されます。

  DCMCF_LNST_LISTEN

  　サーバ型コネクションの確立要求の受付が開始されていることを示します。

  DCMCF_LNST_RETRY

  　サーバ型コネクションの確立要求の受付が開始処理中であることを示します。

  DCMCF_LNST_ONLN_W

  　サーバ型コネクションの確立要求の受付が開始要求待ち状態であることを示します。

  DCMCF_LNST_INIT

  　サーバ型コネクションの確立要求の受付が終了したことを示します。

  それぞれの状態のときに使用できる関数の関係を次の表に示します。

status の設定値	使用できるライブラリ関数	
	dc_mcf_tonln	dc_mcf_tofln
DCMCF_LNST_LISTEN	×	○
DCMCF_LNST_RETRY	×	○
DCMCF_LNST_ONLN_W	○	○
DCMCF_LNST_INIT	○	×

（凡例）

○：使用できます。

×：使用できません。

- resv01

  領域がヌル文字で埋められます。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。
DCMCFRTN_71001	-12001	MCF が開始処理中のため，dc_mcf_tlsln 関数が受け付けられません。
DCMCFRTN_71002	-12002	MCF が終了処理中のため，dc_mcf_tlsln 関数が受け付けられません。
DCMCFRTN_71004	-12004	dc_mcf_tlsln 関数の処理中にメモリ不足が発生しました。
DCMCFRTN_71005	-12005	通信障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71006	-12006	内部障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71009	-12009	dc_mcf_tlsln 関数が，該当する MCF 通信プロセスではサポートされていません。
DCMCFRTN_71010	-12010	MCF 通信プロセスにサーバ型コネクションの確立要求の受付状態取得を要求しましたが，受け付けられませんでした。原因については，メッセージログファイルを参照してください。
DCMCFRTN_72050	-13050	action に DCNOFLAGS が設定されていません。
DCMCFRTN_72052	-13052	resv01 に NULL が設定されていません。
DCMCFRTN_72053	-13053	resv02 に NULL が設定されていません。
DCMCFRTN_72056	-13056	infcnt に NULL が設定されています。
DCMCFRTN_72057	-13057	inf に NULL が設定されています。

リターン値	リターン値（数値）	意味
DCMCFRTN_72061	-13061	mcfid に 0 以下または 240 以上の値が設定されています。
DCMCFRTN_72076	-13076	infcnt に 1 が設定されていません。

dc_mcf_tofln

名称

サーバ型コネクションの確立要求の受付終了

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int dc_mcf_tofln (DCLONG action, DCLONG mcfid, char *resv01,
                  char *resv02)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int dc_mcf_tofln (action, mcfid, resv01, resv02)
DCLONG    action;
DCLONG    mcfid;
char      *resv01;
char      *resv02;
```

機能

サーバ型コネクションの確立要求の受付を終了します。

UAP で値を設定する引数

●action

DCNOFLAGS を設定します。

●mcfid

処理対象の MCF 通信サービスの MCF 通信プロセス識別子※を設定します。設定できる範囲は 1～239 です。

注※

MCF 環境定義（mcftenv -s）で指定する MCF 通信プロセス識別子は 16 進数とみなしてください。

例えば，MCF 通信プロセス識別子が 10 の場合，16 を設定してください。

●resv01，resv02

NULL を設定します。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。
DCMCFRTN_71001	-12001	MCF が開始処理中のため，dc_mcf_tofln 関数が受け付けられません。
DCMCFRTN_71002	-12002	MCF が終了処理中のため，dc_mcf_tofln 関数が受け付けられません。
DCMCFRTN_71004	-12004	dc_mcf_tofln 関数の処理中にメモリ不足が発生しました。
DCMCFRTN_71005	-12005	通信障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71006	-12006	内部障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71009	-12009	dc_mcf_tofln 関数が，該当する MCF 通信プロセスではサポートされていません。
DCMCFRTN_71010	-12010	MCF 通信プロセスにサーバ型コネクションの確立要求の受付終了を要求しましたが，受け付けられませんでした。原因については，メッセージログファイルを参照してください。
DCMCFRTN_72050	-13050	action に DCNOFLAGS が設定されていません。
DCMCFRTN_72052	-13052	resv01 に NULL が設定されていません。
DCMCFRTN_72053	-13053	resv02 に NULL が設定されていません。
DCMCFRTN_72061	-13061	mcfid に 0 以下または 240 以上の値が設定されています。

dc_mcf_tonln

名称

サーバ型コネクションの確立要求の受付開始

形式

ANSI C，C++の形式

```
#include <dcmcf.h>
int dc_mcf_tonln (DCLONG action, DCLONG mcfid, char *resv01,
                  char *resv02)
```

K&R 版 C の形式

```
#include <dcmcf.h>
int dc_mcf_tonln (action, mcfid, resv01, resv02)
DCLONG    action;
DCLONG    mcfid;
char      *resv01;
char      *resv02;
```

機能

サーバ型コネクションの確立要求の受付を開始します。

UAP で値を設定する引数

●action

DCNOFLAGS を設定します。

●mcfid

処理対象の MCF 通信サービスの MCF 通信プロセス識別子※を設定します。設定できる範囲は 1～239 です。

注※

MCF 環境定義（mcftenv -s）で指定する MCF 通信プロセス識別子は 16 進数とみなしてください。

例えば，MCF 通信プロセス識別子が 10 の場合，16 を設定してください。

●resv01，resv02

NULL を設定します。

リターン値

リターン値	リターン値（数値）	意味
DCMCFRTN_00000	0	正常に終了しました。
DCMCFRTN_71001	-12001	MCF が開始処理中のため，dc_mcf_tonln 関数が受け付けられません。
DCMCFRTN_71002	-12002	MCF が終了処理中のため，dc_mcf_tonln 関数が受け付けられません。
DCMCFRTN_71004	-12004	dc_mcf_tonln 関数の処理中にメモリ不足が発生しました。
DCMCFRTN_71005	-12005	通信障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71006	-12006	内部障害が発生しました。原因については，メッセージログファイルを参照してください。
DCMCFRTN_71009	-12009	dc_mcf_tonln 関数が，該当する MCF 通信プロセスではサポートされていません。
DCMCFRTN_71010	-12010	MCF 通信プロセスにサーバ型コネクションの確立要求の受付開始を要求しましたが，受け付けられませんでした。原因については，メッセージログファイルを参照してください。
DCMCFRTN_72050	-13050	action に DCNOFLAGS が設定されていません。
DCMCFRTN_72052	-13052	resv01 に NULL が設定されていません。
DCMCFRTN_72053	-13053	resv02 に NULL が設定されていません。
DCMCFRTN_72061	-13061	mcfid に 0 以下または 240 以上の値が設定されています。

性能検証用トレース（dc_prf_～）

性能検証用トレースで使える関数について説明します。性能検証用トレースの関数を次に示します。

- dc_prf_get_trace_num－性能検証用トレース取得通番の通知
- dc_prf_utrace_put－ユーザ固有の性能検証用トレースの取得

性能検証用トレースの関数（dc_prf_～）は，TP1/Server Base の UAP，および TP1/LiNK の UAP で使えます。ただし，この機能は，TP1/Extension 1 をインストールしていることが前提です。TP1/Extension 1 をインストールしていない場合の動作は保証できませんので，ご了承ください。

dc_prf_get_trace_num

名称

性能検証用トレース取得通番の通知

形式

ANSI C，C++の形式

```
#include <dcprf.h>
int dc_prf_get_trace_num(unsigned short *trace_num,
                         DCLONG flags)
```

K&R 版 C の形式

```
#include <dcprf.h>
int dc_prf_get_trace_num(trace_num, flags)
unsigned short  *trace_num;
DCLONG      flags;
```

機能

dc_prf_get_trace_num 関数を呼び出す前に取得した，最新の性能検証用トレース（prf トレース）のプロセス内取得通番を，関数呼び出し元に通知します。

dc_prf_get_trace_num 関数を呼び出したプロセスで一度も性能検証用トレースを取得していない場合，プロセス内取得番号は 0 となります。

UAP で値を設定する引数

●trace_num

性能検証用トレース取得通番を設定するエリアの先頭ポインタを設定します。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCPRFER_PARAM	-4601	引数に指定した値に誤りがあります。

dc_prf_utrace_put

名称

ユーザ固有の性能検証用トレースの取得

形式

ANSI C，C++の形式

```
#include <dcprf.h>
int dc_prf_utrace_put(unsigned short event_id,
                      unsigned short datalen,
                      char *buffaddr, DCLONG flags)
```

K&R 版 C の形式

```
#include <dcprf.h>
int dc_prf_utrace_put(event_id, datalen, buffaddr, flags)
unsigned short  event_id;
unsigned short  datalen;
char            *buffaddr;
DCLONG          flags;
```

機能

ユーザ固有の性能検証用トレース（prf トレース）を取得します。

UAP で値を設定する引数

●event_id

取得するイベントのイベント ID を設定します。使用できるイベント ID の範囲は 0x0001～0x0040 です。

●datalen

取得するトレースデータのデータ長を設定します。設定できるデータ長は 4 バイト以上 256 バイト以下です。また，このデータ長は 4 バイトの倍数でなければなりません。

●buffaddr

取得するトレースデータの設定されているバッファの先頭ポインタを設定します。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCPRFER_PARAM	-4601	引数に指定した値に誤りがあります。

注意事項

dc_prf_utrace_put 関数がリターン値「DC_OK」を返してもトレースが正しく取得されているとは限りません。これは，トレースの取得処理で，排他を使用しないため複数のプロセスから同時に取得要求が出された場合，データが消失してしまうことがあるためです。

リモート API 機能（dc_rap_～）

リモート API 機能で，コネクションの確立・解放をユーザが管理する場合に使用する関数について説明します。リモート API 機能の関数を次に示します。

- dc_rap_connect － rap リスナーとのコネクションの確立
- dc_rap_disconnect － rap リスナーとのコネクションの解放

リモート API 機能の関数（dc_rap_～）は，TP1/Server Base の UAP，および TP1/LiNK の UAP で使えます。

dc_rap_connect

名称

rap リスナーとのコネクションの確立

形式

ANSI C，C++の形式

```
#include <dcrap.h>
int dc_rap_connect(char *target_host, DCLONG target_port,
                   DCRAP_SV_ID *sv_id, DCLONG rflags)
```

K&R 版 C の形式

```
#include <dcrap.h>
int dc_rap_connect(target_host, target_port, sv_id, rflags)
char        *target_host;
DCLONG      target_port;
DCRAP_SV_ID *sv_id;
DCLONG      rflags;
```

機能

rap リスナーと rap クライアントとの間にコネクションを確立します。

コネクションを確立する rap リスナーは target_host 上に target_port で起動されている rap リスナーです。

UAP で値を設定する引数

●target_host　～((1～255 文字の英数字，ピリオド，およびハイフン))

rap リスナーが起動されている OpenTP1 ノードのホスト名を設定します。

●target_port　～<符号なし整数>((1～65535))

rap リスナーの使用しているウェルノウンポートのポート番号を設定します。

●rflags

DCNOFLAGS を設定します。

OpenTP1 から値が返される引数

●sv_id

正常終了，または DCRAPER_ALREADY_CONNECT の場合，サービス ID が返されます。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常終了しました。rap リスナーとのコネクションが確立されました。
DCRAPER_PARAM	-5501	引数が間違っています。
DCRAPER_PROTO	-5502	プロトコル不正です。要因としては次のことが考えられます。 • dc_rpc_open 関数が呼び出されていません。 • ユーザサービス定義の rpc_rap_auto_connect オペランドに'Y'が指定されていますが，dc_rap_connect 関数が呼び出されました。 • ユーザサービスネットワーク定義の dcsvgdef 定義コマンドで，-w オプションが指定されていません。
DCRAPER_NOMEMORY	-5503	メモリ不足が発生しました。
DCRAPER_NETDOWN	-5505	rap リスナーとの通信でネットワーク障害が発生しました。
DCRAPER_TIMEDOUT	-5506	rap リスナーとの通信でタイムアウトが発生しました。
DCRAPER_NOSOCKET	-5507	ソケット不足が発生しました。
DCRAPER_NOHOSTNAME	-5508	ホスト名称が解決できません。
DCRAPER_MAX_CONNECTION	-5517	一つのプロセスから dc_rap_connect 関数を呼び出せる上限値を超えました。
DCRAPER_NOMEMORY_SV	-5520	rap リスナーまたは rap サーバでメモリ不足が発生しました。
DCRAPER_SHUTDOWN	-5521	rap リスナーは停止中です。
DCRAPER_NOCONTINUE	-5522	続行できない障害が発生しました。障害の要因として次のことが考えられます。 • 予期しないメッセージを受信しました。 • 予期しない相手からのメッセージを受信しました。
DCRAPER_SYSCALL	-5523	システムコールで予期しないエラーが発生しました。
DCRAPER_NOSERVICE	-5528	rap リスナーは開始処理中，または停止処理中です。
DCRAPER_ALREADY_CONNECT	-5529	すでに rap リスナーとのコネクションは確立しています。
DCRAPER_UNKNOWN_NODE	-5531	接続されていないネットワーク上の rap リスナーに対してコネクションを確立しようとしています。
DCRAPER_TIMEOUT_SV	-5532	rap リスナーサービス定義の rap_watch_time オペランドに指定したメッセージ送受信監視時間内にコネクションが確立できませんでした。
DCRAPER_PANIC_SV	-5533	rap リスナーでシステム障害が発生しました。
DCRAPER_MAX_CONNECTION_SV	-5534	rap リスナーの管理する rap クライアントとのコネクション要求受付可能最大数を超えました。

注意事項

DCRAPER_ALREADY_CONNECT 以外のリターン値で，dc_rap_connect 関数がエラーリターンした場合，rap リスナーとのコネクションは確立されていません。

UAP トレースに取得されるエラー要因コードは次のとおりです。

0：エラーなし。

1：dc_rpc_open 関数が呼び出されていません。

3：ホスト名称の設定に誤りがあります。

4：ポート番号の設定に誤りがあります。

5：サービス ID 格納エリアが設定されていません。

6：ユーザサービス定義の rpc_rap_auto_connect オペランドの指定値が Y の場合に，dc_rap_connect 関数が呼び出されました。または，ユーザサービスネットワーク定義が定義されていません。

7：コネクション確立後に発行した dc_rpc_call がエラーとなった場合は，dc_rap_disconnect でコネクションを解放したあと，再度 dc_rap_connect でコネクションを確立させてください。

dc_rap_disconnect

名称

rap リスナーとのコネクションの解放

形式

ANSI C，C++の形式

```
#include <dcrap.h>
int dc_rap_disconnect(DCRAP_SV_ID sv_id, DCLONG rflags)
```

K&R 版 C の形式

```
#include <dcrap.h>
int dc_rap_disconnect(sv_id, rflags)
DCRAP_SV_ID sv_id;
DCLONG      rflags;
```

機能

rap リスナーと rap クライアントとの間に確立されているコネクションを解放します。

UAP で値を設定する引数

●sv_id

dc_rap_connect 関数で受け取ったサービス ID を設定します。

●rflags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常終了しました。rap リスナーとのコネクションが解放されました。
DCRAPER_PARAM	-5501	引数が間違っています。要因としては次のことが考えられます。 • dc_rap_connect 関数で受け取ったサービス ID ではありません。
DCRAPER_PROTO	-5502	プロトコル不正です。要因としては次のことが考えられます。 • dc_rpc_open 関数が呼び出されていません。

リターン値	リターン値（数値）	意味
DCRAPER_PROTO	-5502	• ユーザサービス定義の rpc_rap_auto_connect オペランドに'Y'が指定されていますが，dc_rap_disconnect 関数が呼び出されました。 • ユーザサービスネットワーク定義の dcsvgdef 定義コマンドで，-w オプションが指定されていません。
DCRAPER_NOMEMORY	-5503	メモリ不足が発生しました。
DCRAPER_NETDOWN	-5505	rap リスナーとの通信でネットワーク障害が発生しました。
DCRAPER_TIMEDOUT	-5506	rap リスナーとの通信でタイムアウトが発生しました。
DCRAPER_SHUTDOWN	-5521	rap リスナーは停止中です。
DCRAPER_NOCONTINUE	-5522	続行できない障害が発生しました。障害の要因として次のことが考えられます。 • 予期しないメッセージを受信しました。 • 予期しない相手からのメッセージを受信しました。
DCRAPER_SYSCALL	-5523	システムコールで予期しないエラーが発生しました。

注意事項

DCRAPER_PARAM，DCRAPER_PROTO 以外のリターン値で，dc_rap_disconnect 関数がエラーリターンした場合，rap リスナーとのコネクションは解放されています。

UAP トレースに取得されるエラー要因コードは次のとおりです。

0：エラーなし。

1：dc_rpc_open 関数が呼び出されていません。

3：ユーザサービス定義の rpc_rap_auto_connect オペランドの指定値で次のようになります。

Y の場合：dc_rap_disconnect 関数が呼び出されました。

N の場合：dc_rap_connect 関数を呼び出してコネクションを確立する前に，dc_rap_disconnect 関数が呼び出されました。

リモートプロシジャコール（dc_rpc_～）

クライアント/サーバ形態の通信をするときに使う，OpenTP1 のリモートプロシジャコールの関数について説明します。リモートプロシジャコールに関する関数を次に示します。

- dc_rpc_call－遠隔サービスの要求
- dc_rpc_call_to－通信先を指定した遠隔サービスの呼び出し
- DCRPC_BINDTBL_SET，DCRPC_DIRECT_SCHEDULE－DCRPC_BINDING_TBL 構造体の設定
- dc_rpc_close－アプリケーションプログラムの終了
- dc_rpc_cltsend－CUP への一方通知
- dc_rpc_discard_further_replies－処理結果の受信の拒否
- dc_rpc_discard_specific_reply－特定の処理結果の受信の拒否
- dc_rpc_get_callers_address－クライアント UAP のノードアドレスの取得
- dc_rpc_get_error_descriptor－エラーが発生した非同期応答型 RPC 要求の記述子の取得
- dc_rpc_get_gateway_address－ゲートウェイのノードアドレスの取得
- dc_rpc_get_service_prio－サービス要求のスケジュールプライオリティの参照
- dc_rpc_get_watch_time－サービスの応答待ち時間の参照
- dc_rpc_mainloop－SPP のサービス開始
- dc_rpc_open－アプリケーションプログラムの開始
- dc_rpc_poll_any_replies－処理結果の非同期受信
- dc_rpc_service_retry－サービス関数のリトライ
- dc_rpc_set_service_prio－サービス要求のスケジュールプライオリティの設定
- dc_rpc_set_watch_time－サービスの応答待ち時間の更新

リモートプロシジャコールの関数（dc_rpc_～）は，TP1/Server Base と TP1/LiNK のどちらの UAP でも使えます。

dc_rpc_call

名称

遠隔サービスの要求

形式

ANSI C，C++の形式

```
#include <dcrpc.h>
int  dc_rpc_call(char *group,char *service,char *in,
                 DCULONG *in_len,char *out,DCULONG *out_len,
                 DCLONG flags)
```

K&R 版 C の形式

```
#include <dcrpc.h>
int dc_rpc_call(group,service,in,in_len,out,out_len,flags)
char      *group;
char      *service;
char      *in;
DCULONG   *in_len;
char      *out;
DCULONG   *out_len;
DCLONG    flags;
```

機能

SPP のサービスを要求します。dc_rpc_call 関数を使うときは，要求するサービスがどのノードにあるかを意識する必要はありません。

サービスを要求するときには，**「サービスグループ名」**と**「サービス名」**を dc_rpc_call 関数の引数に設定します。この名称に該当するサービス関数へサービスが要求されます。

dc_rpc_call 関数を呼び出す UAP は，トランザクションとして実行していても，していなくてもかまいません。トランザクションとして実行している処理から dc_rpc_call 関数でサービスを要求するときは，要求するサービスの処理はトランザクションブランチとして稼働します。dc_rpc_call 関数を使う場合，サーバ UAP があるノードの OpenTP1 が稼働していることが前提です。

dc_rpc_call 関数を実行して応答を待っている間にシグナルを受信しても，関数はリターンされません。

リターン値の一覧のあとに，次の説明を掲載しています。dc_rpc_call 関数の詳しい説明を知りたいときに参照してください。

(1)dc_rpc_call 関数の引数について

(2)dc_rpc_call 関数がエラーになる場合

(3)dc_rpc_call 関数がエラーになるタイミング

(4)dc_rpc_call 関数がエラーになったときに再実行する指定

(5)サービス要求に優先度を付ける場合

(6)リターン値 DCRPCER_NO_SUCH_SERVICE_GROUP と DCRPCER_NET_DOWN の違い

(7)リターン値 DCRPCER_SERVICE_TERMINATED をリターンさせる指定

(8)エラーリターン値と同期点処理の関係

(9)サービスを要求するときの注意

(10)ドメイン修飾をしてサービスを要求する場合

UAP で値を設定する引数

●group

SPP の**サービスグループ名**を，31 バイト以内のアスキー文字列で設定します。文字列の最後にはヌル文字を設定してください。このヌル文字は文字列の長さに数えません。

ドメイン修飾をしてサービスを要求するときは，サービスグループ名の後ろに**@**（アットマーク）と **DNS のドメイン名**を付けて，文字列の最後にはヌル文字を設定してください。

●service

SPP の**サービス名**を，31 バイト以内のアスキー文字列で設定します。文字列の最後にはヌル文字を設定してください。このヌル文字は文字列の長さに数えません。

●in

サービスの入力パラメタを設定します。

●in_len

サービスの入力パラメタ長を設定します。1 から DCRPC_MAX_MESSAGE_SIZE※までの範囲の長さが設定できます。DCRPC_MAX_MESSAGE_SIZE は，dcrpc.h で定義してあります。

注※

rpc_max_message_size オペランドを使用した場合，DCRPC_MAX_MESSAGE_SIZE の値（1 メガバイト）ではなく，rpc_max_message_size オペランドに指定した値になります。

●out

サービス関数から返ってくる応答を格納する領域を設定します。out で指す領域へ，サービス関数からの応答が設定されます。

●out_len

サービスの応答の長さを設定します。1 から DCRPC_MAX_MESSAGE_SIZE※までの範囲の長さが設定できます。DCRPC_MAX_MESSAGE_SIZE は，dcrpc.h で定義してあります。

注※

rpc_max_message_size オペランドを使用した場合，DCRPC_MAX_MESSAGE_SIZE の値（1 メガバイト）ではなく，rpc_max_message_size オペランドに指定した値になります。

非応答型 RPC の場合も，サービスの応答の長さを設定した領域のアドレスを指定する必要があります。この場合，サービスの応答の長さには，0 を設定します。

●flags

RPC の形態とオプションを，次に示す形式で設定します。

```
{DCNOFLAGS|DCRPC_NOWAIT|DCRPC_NOREPLY|DCRPC_CHAINED}〔|DCRPC_TPNOTRAN〕〔|DCRPC_DOMAIN〕
```

DCNOFLAGS

同期応答型 RPC

DCRPC_NOWAIT

非同期応答型 RPC

DCRPC_NOREPLY

非応答型 RPC

DCRPC_CHAINED

連鎖 RPC

DCRPC_TPNOTRAN

トランザクション処理からのサービス要求で，要求先の処理をトランザクションにしない場合に設定します。DCRPC_TPNOTRAN を設定すると，トランザクションの処理からのサービス要求でも，サービス関数の処理はトランザクションになりません。

DCRPC_DOMAIN

サービスグループ名をドメイン修飾した場合に指定します。ドメイン修飾をした RPC はトランザクションブランチにできません。そのため，トランザクションの処理から dc_rpc_call 関数を使う場合は，必ず DCRPC_TPNOTRAN と一緒に指定してください。

DCRPC_TPNOTRAN と DCRPC_DOMAIN は，RPC の形態に付けて設定します。

（例 1）

同期応答型 RPC でトランザクションにしないサービス要求をする場合，flags には「DCNOFLAGS | DCRPC_TPNOTRAN」と設定します。

（例 2）

トランザクションの処理から，同期応答型 RPC でドメイン修飾をしたサービス要求をする場合，flags には「DCNOFLAGS|DCRPC_TPNOTRAN|DCRPC_DOMAIN」と設定します。

サーバ UAP から値が返される引数

●out

サービス関数が設定した，応答が返されます。

●out_len

サービス関数が設定した，応答の長さが返されます。

リターン値

ここで示すリターン値は，OpenTP1 が返す値です。サービス関数から返される値ではありません。

リターン値	リターン値（数値）	意味
0 または正の整数		正常に終了しました。非同期応答型 RPC の場合は，正の整数は記述子です。
DCRPCER_INVALID_ARGS	-301	引数に設定した値が間違っています。
DCRPCER_PROTO	-302	dc_rpc_open 関数を呼び出していません。
		非オートコネクトモード（ユーザサービス定義の rpc_rap_auto_connect オペランドに N を指定）でリモート API 機能を使用時に dc_rap_connect 関数を呼び出していません。
DCRPCER_NO_BUFS	-304	メモリが不足しました。または，サービス要求先 SPP のメッセージ格納バッファプール（message_store_buflen オペランド）が不足したため，サービス要求を受け付けられませんでした。 サービス要求先 SPP のユーザサービス定義，またはユーザサービスデフォルト定義の message_store_buflen オペランドの指定値を見直してください。
DCRPCER_NET_DOWN	-306	通信障害が起こりました。 ネットワークに障害が発生していないか確認してください。
		リモート API 機能を使用している場合は，dc_rpc_call 関数の応答待ち時間を満了した可能性があります。 dc_rpc_call 関数の応答待ち時間の設定（watch_time オペランド，dc_rpc_set_watch_time 関数の引数）を見直してください。 リモート API 機能使用時に dc_rpc_call 関数の応答待ち時間を満了した場合のリターン値を DCRPCER_TIMED_OUT に変更するには，ユーザサービス定義に rap_extend_function

リターン値	リターン値（数値）	意味
DCRPCER_NET_DOWN	-306	オペランドを指定してください。rap_extend_function オペランドについてはマニュアル「OpenTP1 システム定義」を参照してください。
DCRPCER_TIMED_OUT	-307	dc_rpc_call 関数の応答待ち時間を満了しました。 dc_rpc_call 関数の応答待ち時間の設定（watch_time オペランド，dc_rpc_set_watch_time 関数の引数）を見直してください。
		サービス要求先 SPP が，サービス関数の実行中に異常終了しました。 サービス要求先 SPP が異常終了した要因を調査してください。
		接続済みのコネクションでメッセージの吸い込みが発生しました。サービス要求先の OpenTP1 がオンラインであることを確認し，再度 dc_rpc_call 関数を呼び出してください。
DCRPCER_MESSAGE_TOO_BIG	-308	in_len に設定した入力パラメタ長が，最大値を超えています。 in_len の設定値を見直してください。
DCRPCER_REPLY_TOO_BIG	-309	サービス要求先 SPP のサービス関数で設定した応答の長さ（out_len）が，dc_rpc_call 関数の応答の長さ（out_len）を超えました。 サービス要求先 SPP のサービス関数での応答の長さ（out_len）の設定を見直してください。
DCRPCER_NO_SUCH_SERVICE_GROUP	-310	group に設定したサービスグループ名が不正であるか，group に設定したサービスグループのサービス要求先 SPP が起動されていません。 group の設定を見直すか，group に設定したサービスグループのサービス要求先 SPP を起動してください。
DCRPCER_NO_SUCH_SERVICE	-311	service に設定したサービス名が不正であるか，サービス要求先 SPP で service に設定したサービス名がユーザサービス定義ファイルの service オペランドに設定されていません。 service の設定を見直すか，service に設定したサービス名をサービス要求先 SPP の service オペランドに設定してください。
DCRPCER_SERVICE_CLOSED	-312	group で設定したサービスグループのサービス要求先 SPP は，サーバ閉塞またはサービス閉塞しています。 閉塞要因を調査し，閉塞を解除してください。
DCRPCER_SERVICE_TERMINATING	-313	サービス要求先 SPP は，終了処理中です。
DCRPCER_SERVICE_NOT_UP	-314	group に設定したサービスグループ名のサービス要求先 SPP は起動していません。または，サービス要求送信処理で通信障害が起きたおそれがあります。

リターン値	リターン値（数値）	意味
DCRPCER_SERVICE_NOT_UP	-314	group に設定したサービスグループ名のサービス要求先 SPP を起動してください。すでに起動している場合は，ネットワーク障害が発生していないかを確認してください。
		サービス要求の応答待ち時間（watch_time オペランド，dc_rpc_set_watch_time 関数の引数）に 0 を指定している場合に，サービス関数を実行中のサービス要求先 SPP が異常終了しました。 サービス要求先 SPP が異常終了した要因を調査してください。
DCRPCER_OLTF_NOT_UP	-315	サービス要求先 SPP の OpenTP1 が起動していません。OpenTP1 が停止処理中であるか，サービス要求送信処理で通信障害が起きたおそれがあります。 サービス要求先 SPP の OpenTP1 を起動するか，ネットワーク障害が発生していないかどうかを確認してください。
DCRPCER_SYSERR_AT_SERVER	-316	サービス要求先 SPP で，システムエラー（内部矛盾）が起こりました。
DCRPCER_NO_BUFS_AT_SERVER	-317	サービス要求先 SPP で，メモリが不足しました。
DCRPCER_SYSERR	-318	サービス要求元 UAP で，システムエラー（内部矛盾）が起こりました。
DCRPCER_INVALID_REPLY	-319	サービス要求先 SPP のサービス関数で設定する応答の長さ（out_len）が，1 から DCRPC_MAX_MESSAGE_SIZE※で定義されている値の範囲にありません。 サービス要求先 SPP のサービス関数内の応答の長さ（out_len）の設定を見直してください。
DCRPCER_OLTF_INITIALIZING	-320	サービス要求先 SPP の OpenTP1 は，開始処理中です。
DCRPCER_NO_BUFS_RB	-323	サービス要求元 UAP，またはサービス要求先 SPP で，メモリが不足しました。このリターン値が返った場合，トランザクションブランチはロールバックします。 サービス要求元 UAP，またはサービス要求先 SPP で，不要なメモリを確保していないか見直してください。
DCRPCER_SYSERR_RB	-324	サービス要求元 UAP で，システムエラー（内部矛盾）が起こりました。このリターン値が返った場合，トランザクションブランチはロールバックします。
DCRPCER_SYSERR_AT_SERVER_RB	-325	サービス要求先 SPP で，システムエラー（内部矛盾）が起こりました。このリターン値が返った場合，トランザクションブランチはロールバックします。
DCRPCER_REPLY_TOO_BIG_RB	-326	サービス要求先 SPP のサービス関数で設定した応答の長さ（out_len）が，dc_rpc_call 関数の応答の長さ（out_len）を超えました。このリターン値が返った場合，トランザクションブランチはロールバックします。

リターン値	リターン値（数値）	意味
DCRPCER_REPLY_TOO_BIG_RB	-326	サービス要求先 SPP のサービス関数で，応答の長さ（out_len）の設定を見直してください。
DCRPCER_TRNCHK	-327	ノード間負荷バランス機能，およびノード間負荷バランス拡張機能を使用している場合に，同一サービスグループ名のサービス要求先 SPP のトランザクション属性（atomic_update オペランド）が不一致になっています。または，負荷を分散する先のノードにある OpenTP1 のバージョンが，要求元 UAP の OpenTP1 のバージョンよりも古いため，ノード間負荷バランス機能，およびノード間負荷バランス拡張機能を実行できません。 このリターン値が返るのは，ノード間負荷バランス機能およびノード間負荷バランス拡張機能を使用している SPP にサービスを要求した場合だけです。 ノード間負荷バランス機能，およびノード間負荷バランス拡張機能を使用する SPP のトランザクション属性（atomic_update オペランド），または OpenTP1 のバージョンを見直してください。
		dcsvgdef 定義コマンドを使用して，非トランザクション属性（ユーザサービス定義の atomic_update オペランドに'N'を指定，またはシステム共通定義の jnl_fileless_option オペランドに'Y'を指定）のユーザサーバに対し，dc_rpc_call 関数の flags に DCRPC_TPNOTRAN との論理和を指定しないでサービスを要求しました。 dcsvgdef 定義コマンドまたは，dc_rpc_call 関数の flags を見直してください
DCRPCER_NO_SUCH_DOMAIN	-328	group のドメイン修飾をしたサービスグループ名のドメイン名が間違っています。 ドメイン名を見直してください。
DCRPCER_NO_PORT	-329	group にドメイン修飾をしてサービスを要求しましたが，ドメイン代表スケジュールサービスのポート番号が見つかりません。 システム共通定義の domain_masters_port オペランドの設定，および/etc/services のドメイン代表スケジュールサービスのポート番号の設定を見直してください。
DCRPCER_SERVER_BUSY	-356	サービス要求先 SPP（ソケット受信型サーバ）が，サービス要求を受け取れません。 サービス要求先 SPP のユーザサービス定義，またはユーザサービスデフォルト定義の max_socket_msg オペランド，max_socket_msglen オペランドの設定を見直してください。
DCRPCER_TESTMODE	-366	オンラインテスタを使用している環境で，テストモードの UAP から，テストモードでない SPP へサービスを要求しています。または，テストモードでない UAP からテストモードの SPP へサービスを要求しています。 UAP のテストモードの設定を見直してください。

リターン値	リターン値（数値）	意味
DCRPCER_NOT_TRN_EXTEND	-367	トランザクション属性の連鎖 RPC を実行したあとで，flags に DCRPC_TPNOTRAN を設定し，dc_rpc_call 関数でサービスを要求しています。
DCRPCER_SECCHK	-370	サービス要求先 SPP は，セキュリティ機能で保護されています。 dc_rpc_call 関数を実行したサービス要求元 UAP には，サービス要求先 SPP へのアクセス権限がありません。サービス要求先 SPP のアクセス権限を見直してください。
DCRPCER_TRNCHK_EXTEND	-372	サービス要求先 SPP の OpenTP1 で，同時に起動できるトランザクションブランチの数を超えました。そのため，トランザクションブランチを開始できません。 トランザクションサービス定義の trn_tran_process_count オペランドの設定を見直してください。
		サービス要求元 UAP が，一つのトランザクションブランチから開始できる子トランザクションブランチの最大数を超えました。そのため，トランザクションブランチを開始できません。 トランザクションサービス定義の trn_max_subordinate_count オペランドの設定を見直してください。
		トランザクション内のドメイン修飾をしたサービス要求で，flags に DCRPC_TPNOTRAN を設定していません。
		サービス要求先 SPP で，リソースマネジャ（RM）にエラーが発生しました。そのため，トランザクションブランチを開始できません。 リソースマネジャ（RM）のエラー要因を解決したあと，再度実行してください。
		TP1/LiNK の「システム環境設定」ウィンドウの「トランザクション機能」が［あり］に設定されていません。 TP1/LiNK の「システム環境設定」の「トランザクション機能」の設定を見直してください。
DCRPCER_SERVICE_TERMINATED	-378	サービス要求先 SPP が，サービス関数の実行中に異常終了しました。 サービス要求先 SPP のサービス関数処理を見直してください。 このリターン値が返るのは，ユーザサービス定義の rpc_extend_function オペランドに"00000001"を指定したサービス要求元 UAP の場合だけです。rpc_extend_function オペランドに"00000000"を指定，またはオペランドを省略した場合は，このリターン値ではなく，DCRPCER_TIMED_OUT，または DCRPCER_SERVICE_NOT_UP が返ります。

注※

rpc_max_message_size オペランドを使用した場合，DCRPC_MAX_MESSAGE_SIZE の値（1 メガバイト）ではなく，rpc_max_message_size オペランドに指定した値になります。

(1)dc_rpc_call 関数の引数について

dc_rpc_call 関数の引数について説明します。

サーバ UAP に渡す値

サービスを要求するときは，サービス関数から返ってくる応答の領域（out）を確保しておきます。クライアント UAP では，dc_rpc_call 関数に次の値を設定しておきます。

- 入力パラメタ（in）
- 入力パラメタ長（in_len）
- 応答の長さ（out_len）

  入力パラメタ，入力パラメタ長，応答の長さは，クライアント UAP の dc_rpc_call 関数で設定した値が，そのままサービス関数に渡されます。文字コードや数字の表記形式を変えたい場合は，クライアント UAP，または要求されたサービス関数の処理で変換してください。応答を返さないサービス関数のサービスを要求するときは，応答の長さを設定しても無視されます。

  入力パラメタ長と応答の長さの最大値は，それぞれヘッダファイル dcrpc.h の DCRPC_MAX_MESSAGE_SIZE※で宣言しています。最大値を確認する場合は，dcrpc.h の内容を参照してください。

  注※

  rpc_max_message_size オペランドを使用した場合，DCRPC_MAX_MESSAGE_SIZE の値（1 メガバイト）ではなく，rpc_max_message_size オペランドに指定した値になります。

サーバ UAP から戻ってくる値

サービス関数の処理が終了して応答が戻ってくると，次の値を参照できます。

- サービス関数の応答（out）
- サービス関数の応答の長さ（out_len）

out_len はサービス関数から実際に返ってきた応答の長さです。out と out_len を参照できる場合を次に示します。

- 同期応答型 RPC，連鎖 RPC の場合

  dc_rpc_call 関数がリターンしたあと。
- 非同期応答型 RPC の場合

  dc_rpc_poll_any_replies 関数が該当する応答を受け取ってリターンしたあと。out_len は参照できません。
- 非応答型 RPC の場合

out と out_len は参照できません。

dc_rpc_call 関数，または dc_rpc_poll_any_replies 関数がエラーリターンした場合には，out と out_len は参照できません。

返ってきた応答が，クライアント UAP で確保した応答の領域（out）よりも大きい場合は，DCRPCER_REPLY_TOO_BIG でエラーリターンします。

flags に設定する値

flags に設定した値と dc_rpc_call 関数の実行結果について説明します。

- 同期応答型 RPC（flags に DCNOFLAGS を設定）

  dc_rpc_call 関数は，応答が返るか，通信先でエラーが起こるまでリターンしません。

- 非同期応答型 RPC（flags に DCRPC_NOWAIT を設定）

  dc_rpc_call 関数は，すぐにリターンします。ただし，応答を参照できるのは，dc_rpc_poll_any_replies 関数で非同期に応答を受信したあとです。応答（out）の領域は，次に示す方法で非同期応答型 RPC を終了するまで，解放しないでください。

  - dc_rpc_poll_any_replies 関数で応答を受け取る
  - dc_rpc_discard_further_replies 関数で応答の受信を拒否
  - トランザクションの処理からサービスを要求した場合は，コミット，またはロールバックした

  トランザクションの処理で非同期応答型 RPC を使う場合，同期点処理（コミット，またはロールバック）前に dc_rpc_poll_any_replies 関数で応答を受け取ってください。**同期点処理後には，dc_rpc_poll_any_replies 関数で応答は受信できません。**dc_rpc_poll_any_replies 関数が受信する応答を特定する場合は，dc_rpc_call 関数がリターンしたときに返された正の整数（記述子）を，dc_rpc_poll_any_replies 関数の引数に設定します。受信する応答を特定する場合は，dc_rpc_call 関数のリターン値を保持しておいてください。

  なお，非トランザクションの処理で，同期点処理後に応答を受け取りたい場合は，システムサービス定義の rpc_extend_function オペランドで該当するオプションを指定する必要があります。

  rpc_extend_function については，マニュアル「OpenTP1 システム定義」を参照してください。

- 非応答型 RPC（flags に DCRPC_NOREPLY を設定）

  dc_rpc_call 関数は，サービス関数の処理が終わるのを待たないで，すぐにリターンします。サービス関数は応答を返さないサービスと見なされます。そのため，サービス関数が実行されたかどうかは，サービスを要求した UAP からはわかりません。DCRPC_NOREPLY を設定した場合は，応答（out）と応答の長さ（out_len）の値は参照できません。

- 連鎖 RPC（flags に DCRPC_CHAINED を設定）

  dc_rpc_call 関数は，応答が返るか，通信先でエラーが起こるまでリターンしません。連鎖 RPC で同じサービスグループに属するサービスを複数回要求する場合は，最初の要求時と同じプロセスで実行できます。

  連鎖 RPC を使う場合には，次に示す制限があります。

1. 2 回目以降の dc_rpc_call 関数では，ユーザサーバおよびサービスの閉塞を検出できません。
2. 2 回目以降の dc_rpc_call 関数でサービス関数の処理で異常が起こった場合には，ユーザサーバ全体が閉塞します。サービス単位では閉塞しません。

(2)dc_rpc_call 関数がエラーになる場合

dc_rpc_call 関数のエラーリターンする理由について説明します。

サーバ UAP があるノードの OpenTP1 が稼働していない場合

サービスを要求される OpenTP1 が稼働していない場合は，dc_rpc_call 関数は次に示すリターン値のどれかでエラーリターンします。

- DCRPCER_NET_DOWN
- DCRPCER_SERVICE_NOT_UP
- DCRPCER_OLTF_NOT_UP
- DCRPCER_OLTF_INITIALIZING

サーバ UAP が稼働していない場合

サーバ UAP がマルチサーバの場合，異常終了中，または部分回復処理中であっても，サービス要求は OpenTP1 で新しく起動されたプロセスで実行されます。ただし，次に示す場合は，dc_rpc_call 関数はエラーリターンします。

1. 閉塞している SPP へはサービスを要求できません。サービスグループが閉塞されている場合は，DCRPCER_SERVICE_CLOSED でエラーリターンします。
2. ユーザサーバの停止コマンド（dcsvstop コマンド）または OpenTP1 の停止コマンド（dcstop コマンド）で，SPP が終了処理中または終了している場合は，次に示すどれかのステータスコードでエラーリターンします。
   - DCRPCER_SERVICE_TERMINATING
   - DCRPCER_SERVICE_CLOSED
   - DCRPCER_NO_SUCH_SERVICE_GROUP

   これらのうち，どのリターン値が返るかは，dc_rpc_call 関数を呼び出したタイミングで決まります。
3. OpenTP1 が開始処理中の場合は，DCRPCER_OLTF_INITIALIZING でエラーリターンします。この場合は，サーバ UAP または OpenTP1 が起動完了したあとに正常にサービスを要求できることがあります。OpenTP1 は，メッセージ ID KFCA01809-I のメッセージログが出力されると起動を完了するので，このメッセージが表示されてから再びサービスを要求してください。

ノード間負荷バランス機能およびノード間負荷バランス拡張機能の環境でサービスを要求した場合

ノード間負荷バランス機能およびノード間負荷バランス拡張機能の環境では，該当するサービスのスケジュールが閉塞していると，OpenTP1 が自動的にほかのノードにサービス要求を転送します。ただし，次に示す条件の場合 dc_rpc_call 関数は DCRPCER_TRNCHK でエラーリターンします。

1. トランザクション処理の場合，転送しようとした先のノードにあるサービスのトランザクション属性が，閉塞していたサービスと一致していないとき。
2. 転送しようとした先のノードにある OpenTP1 のバージョンが，サービスを要求した OpenTP1 のノードのバージョンよりも低いとき。

これらのエラーリターンが起こった場合は，次に示す処置をしてください。

1. ノード間負荷バランス機能およびノード間負荷バランス拡張機能を構成する複数の SPP のトランザクション属性を一致させてください。
2. ノード間負荷バランス機能およびノード間負荷バランス拡張機能を構成する複数の OpenTP1 のバージョンを一致させてください。

ソケット受信型サーバへサービスを要求する場合

ソケット受信型サーバでは，ユーザサービス定義の max_socket_msg オペランドと max_socket_msglen オペランドの指定で，データの輻輳制御をしています。そのため，定義した値を超えた場合，サービス要求を受信できない場合があります。このとき dc_rpc_call 関数は，DCRPCER_SERVER_BUSY でエラーリターンします。この値が返った場合は，クライアント UAP はしばらく時間をおいてから再実行すれば，サービスを要求できることがあります。

連鎖 RPC を使った場合

トランザクションとして処理している連鎖 RPC を使っている UAP から，同じサーバ UAP へトランザクションでない dc_rpc_call 関数を呼び出すと，DCRPCER_NOT_TRN_EXTEND でエラーリターンします。

オンラインテスタを使っている場合

オンラインテスタを使っている場合に，テストモードの UAP とテストモードでない UAP 間で dc_rpc_call 関数を呼び出すと，DCRPCER_TESTMODE でエラーリターンします。

セキュリティ機能を使っている場合

dc_rpc_call 関数を呼び出したときに，目的のサービスがセキュリティ機能で保護されていて，dc_rpc_call 関数を呼び出したクライアント UAP に SPP へのアクセス権がない場合は，DCRPCER_SECCHK でエラーリターンします。

(3)dc_rpc_call 関数がエラーになるタイミング

サービスを要求された SPP が異常終了した場合，クライアント UAP でエラーが返るタイミングについて説明します。

- 同期応答型 RPC または連鎖 RPC（flags に DCNOFLAGS または DCRPC_CHAINED を設定）

  サービスを実行する SPP が処理の終わる前に異常終了すると，DCRPCER_TIMED_OUT でエラーリターンします。クライアント UAP のユーザサービス定義 watch_time オペランドに無限を指定している場合は，DCRPCER_SERVICE_NOT_UP でエラーリターンします。
- 非同期応答型 RPC（flags に DCRPC_NOWAIT を設定）

サービスを実行する SPP が処理の終わる前に異常終了すると，上記のリターン値が dc_rpc_poll_any_replies 関数にエラーリターンします。

- 非応答型 RPC（flags に DCRPC_NOREPLY を設定）
  サーバ UAP の異常終了をクライアント UAP では検知できません。

クライアント UAP の時間監視でエラーになる場合

次に示す場合には，クライアント UAP のユーザサービス定義の watch_time オペランドに指定した時間が経過したあとで，DCRPCER_TIMED_OUT でエラーリターンします。

- SPP があるノードの OpenTP1 全体が異常終了した場合
- サービス要求のデータがサーバ UAP に届く前，またはサーバ UAP の処理が完了してからクライアント UAP に結果が届く前に障害が起こった場合

(4)dc_rpc_call 関数がエラーになったときに再実行する指定

開始処理中，または系切り替え中などでサービス要求先の OpenTP1 が起動していない場合でも，dc_rpc_call 関数をエラーとしないで，OpenTP1 で要求先検索，およびサービス要求送信を再実行させられます。

要求先検索，およびサービス要求送信を再実行させる場合は，システム共通定義の rpc_retry オペランドに Y を指定してください。要求先検索とサービス要求送信の再実行回数，および再実行間隔は，システム共通定義の rpc_retry_count オペランドと rpc_retry_interval オペランドで指定します。システム共通定義で指定した再実行回数を超えた場合は，dc_rpc_call 関数は次に示すどれかのステータスコードでエラーリターンします。

- DCRPCER_INVALID_ARGS
- DCRPCER_NET_DOWN
- DCRPCER_NO_SUCH_SERVICE_GROUP
- DCRPCER_SERVICE_NOT_UP
- DCRPCER_OLTF_NOT_UP
- DCRPCER_OLTF_INITIALIZING

(5)サービス要求に優先度を付ける場合

サービス要求のスケジュールプライオリティは，dc_rpc_call 関数を呼び出す直前に，dc_rpc_set_service_prio 関数を呼び出して設定します。プライオリティを設定しない場合は，スケジュールサービスの省略時の解釈で，サービス要求の優先度が決まります。

(6)リターン値 DCRPCER_NO_SUCH_SERVICE_GROUP と DCRPCER_NET_DOWN の違い

上記のリターン値は，該当するサービスグループ名のユーザサーバが見つからなかった場合にリターンされます。

- DCRPCER_NO_SUCH_SERVICE_GROUP
  システム共通定義の all_node オペランドに指定した，すべてのノードを探して見つからなかったことを示します。
- DCRPCER_NET_DOWN
  探している途中で，all_node オペランドに指定した一つ以上のノードとの通信で障害が起こったことを示します。これは，該当する OpenTP1 システムがなかった場合も含みます。

(7)リターン値 DCRPCER_SERVICE_TERMINATED をリターンさせる指定

サービスを要求された SPP が，処理を完了する前に異常終了したことを DCRPCER_TIMED_OUT または DCRPCER_SERVICE_NOT_UP 以外のリターン値で判別したい場合には，ユーザサービス定義の rpc_extend_function オペランドに"00000001"を指定します。この指定をすると，上記のエラー時に DCRPCER_SERVICE_TERMINATED がリターンされるようになります。rpc_extend_function オペランドに"00000000"を指定するか，またはオペランドを省略した場合は，DCRPCER_SERVICE_TERMINATED は返らないで，DCRPCER_TIMED_OUT または DCRPCER_SERVICE_NOT_UP がリターンされます。

(8)エラーリターン値と同期点処理の関係

dc_rpc_call 関数のリターン値と同期点処理（コミット，ロールバック）の関係について説明します。ここで説明する内容は，サービス要求がトランザクション処理になる場合に該当します。トランザクションでないサービス要求（flags に DCRPC_TPNOTRAN を設定した場合も含む）には該当しません。

dc_rpc_call 関数がエラーリターンしてもコミットとなる場合

サービスを要求されたサービス関数の異常終了や，ノードの障害，ネットワーク障害の場合でも，DCRPCER_TIMED_OUT がリターンすることがあります。クライアント UAP がトランザクション処理でない場合は，DCRPCER_TIMED_OUT が返っても，要求したサービスがあるサービス関数は正常に終了していて，データベースへの更新などが実行されているときもあります。

ロールバック処理が必要なエラーリターン値

トランザクション処理から呼び出した dc_rpc_call 関数がエラーリターンした場合，リターン値によっては，必ずトランザクションがロールバック（サーバ UAP が rollback_only 状態）になります。この場合，コミットの関数，またはロールバックの関数のどちらを使っても，必ずロールバックになります。必ずロールバックになる dc_rpc_call 関数のリターン値を次に示します。

- DCRPCER_INVALID_REPLY
- DCRPCER_NO_BUFS_AT_SERVER
- DCRPCER_NO_SUCH_SERVICE
- DCRPCER_REPLY_TOO_BIG_RB

(9)サービスを要求するときの注意

1. dc_rpc_call 関数に設定するサービスグループ名およびサービス名は，サーバ UAP の環境設定で指定しておいてください。間違ったサービスグループ名またはサービス名を dc_rpc_call 関数に設定してサービスを要求すると，DCRPCER_NO_SUCH_SERVICE_GROUP または DCRPCER_NO_SUCH_SERVICE でエラーリターンします。

   応答を返さないサービス関数の場合は， DCRPCER_NO_SUCH_SERVICE ではリターンしません。

2. サーバ UAP は，クライアント UAP とは別のプロセスで実行します。そのため，通常の関数の呼び出しや手続き呼び出しとは，次の点が異なります。

   - OS がクライアント UAP のプロセスに与えている属性は，サーバ UAP へは引き継がれません（例えば，環境変数，スケジュールの優先順位（nice 値）など）。
   - クライアント UAP のノードで設定した OpenTP1 の環境設定は，サーバ UAP の OpenTP1 へは引き継がれません（例えば，トランザクション属性の指定有無，トランザクションブランチの限界経過時間，スケジュールの優先順位など）。

3. 入力パラメタ（in）およびサービス関数の応答（out）に，同じバッファ領域を設定しないでください。

4. flags に DCRPC_NOREPLY を設定した場合には，次に示すリターン値は返りません。

   - **起こらないエラー**

     DCRPCER_REPLY_TOO_BIG

     DCRPCER_INVALID_REPLY

   - **起こっても検出できないエラー**

     DCRPCER_NO_SUCH_SERVICE

     DCRPCER_SERVICE_TERMINATING

     DCRPCER_SYSERR_AT_SERVER

     DCRPCER_NO_BUFS_AT_SERVER

     DCRPCER_SECCHK

   また，エラー発生時に OpenTP1 にメッセージを出力することもありません。エラーを検出しなければならない場合，flags に DCNOFLAGS を設定する（同期応答型 RPC）ことを検討してください。

5. トランザクションの処理から dc_rpc_call 関数でサービスグループを呼び出すと，トランザクションが決着するまで一つの SPP を占有します。同じトランザクションの処理から，dc_rpc_call 関数で同じサービスを複数回使う場合は，次に示すようにしてください。

   - 使う回数に応じて，ユーザサービス定義の balance_count オペランドおよび parallel_count オペランドの指定値を見積もり直す。
   - プロセスが増えないように，連鎖 RPC でサービスを要求する。

   balance_count オペランドおよび parallel_count オペランドの指定値に誤りがあると，トランザクションが異常終了したり，デッドロックが起こったりする場合があります。

6. 非同期応答型 RPC を使う場合には，dc_rpc_poll_any_replies 関数ですべての非同期の応答を受信するか，dc_rpc_discard_further_replies 関数で非同期の応答を拒否するまで，サーバ UAP（SPP）を

占有する場合があります。これは，トランザクションの処理の場合にもトランザクションの処理でない場合にも起こります。非同期応答型 RPC を使う場合には，使う回数に応じて，常駐プロセス数の指定を増やしておいてください。

非同期応答型 RPC は，SPP を占有すること以外にも多くの資源を必要とします。UAP の処理効率が下がったり，必要ない SPP が開始されたりすることを防ぐため，非同期応答型 RPC の dc_rpc_call 関数を使ったあとには，確実に応答を受信するか，受信を拒否するかしてください。

7. 非同期応答型 RPC で連続して複数回使ったあとで応答を受け取る場合には，非同期応答型 RPC のサービスを要求するときに，それぞれ異なる応答格納領域（out）を設定してください。同じ領域を設定すると，あとから来た応答が上書きされて，応答を正しく受け取れなくなります。
8. 非同期応答型 RPC でサービス要求したサーバ UAP（SPP）は，非同期応答型 RPC を実行したプロセスが dc_rpc_poll_any_replies 関数を発行したかどうかにかかわらず，サービス関数実行後にすぐに応答を送信します。dc_rpc_poll_any_replies 関数を発行しないで，複数回の非同期応答型 RPC を大量に実行すると，SPP から送信される応答が TCP/IP のバッファにたまり，SPP の応答送信処理が失敗する場合があります。SPP で応答送信処理が失敗すると，非同期応答型 RPC 発行元では，dc_rpc_poll_any_replies 関数を発行しても，SPP から応答を受信することはできません。
9. トランザクション属性の非同期応答型 RPC，または非応答型 RPC を大量に実行すると，SPP から送信されるトランザクションについてのメッセージを受信できなくなり，トランザクションがロールバックすることがあります。

(10)ドメイン修飾をしてサービスを要求する場合

ドメイン修飾したサービスグループ名を設定すると，DNS のドメイン内にある OpenTP1 のサービスを要求できます。ドメイン修飾をしたサービスグループ名は，サービスグループ名の後ろに，@と DNS のドメイン名を付けて設定します。

ドメイン修飾をしてサービスを要求するときの注意

1. ドメイン修飾をしてサービスを要求する場合は，dc_rpc_call 関数の flags に DCRPC_DOMAIN を設定します。DCRPC_DOMAIN を設定しないでドメイン修飾をしたサービスグループ名を設定すると，dc_rpc_call 関数は DCRPCER_NO_SUCH_SERVICE_GROUP でエラーリターンします。
2. ドメイン修飾をした RPC の場合，dc_rpc_call 関数を呼び出したプロセスがトランザクションの処理でも，トランザクションを拡張できません。そのため，トランザクションの処理からドメイン修飾をしてサービスを要求する場合は，flags に DCRPC_TPNOTRAN を指定して，トランザクションを拡張しないようにしてください。なお，ドメイン名に自ドメインを設定した場合でも，トランザクションを拡張できません。
3. ドメイン修飾をした RPC では，キュー受信型サーバにだけサービスを要求できます。ソケット受信型サーバへは，ドメイン修飾をしたサービス要求はできません。
4. ドメイン修飾をしたサービス要求では，namdomainsetup コマンドで登録したホストで起動されるドメイン代表スケジュールサービスへサービス要求を送信します。ドメイン代表スケジュールサービスのポート番号は，/etc/services から取得します。サービス要求の送信で障害が起こった場合は，namdomainsetup コマンドで複数のホスト名を登録していれば，順次送信を試みます。ドメイン修飾

をした RPC が正常に終了しても，ドメイン代表スケジュールサービスへの送信で障害が起こっている場合もあります。

ドメイン修飾をしたサービス要求をする前の準備

ドメイン修飾をした RPC では，次に示す環境設定をしてください。

1. DNS のドメインデータファイルへ，ドメイン代表スケジュールサービスを起動するホスト名を登録します。登録するときは，namdomainsetup コマンドを使います。
2. ドメイン修飾をしたサービス要求をする OpenTP1 を起動するホストの/etc/services へ，ドメイン代表スケジュールサービスのポート番号を設定します。ポート番号は，次の形式で記述してください。

   ```
   OpenTP1scd ポート番号/tcp
   ```

3. ドメイン代表スケジュールサービスを起動する OpenTP1 のスケジュールサービス定義の scd_port オペランドに，ドメイン代表スケジュールサービスのウェルノウンポートを指定します。

注意事項

システム共通定義の all_node オペランドで指定したドメイン以外の OpenTP1 システムにトランザクショナル RPC を行う場合，自ドメインおよび他ドメイン内のすべての OpenTP1 システムのノード識別子(システム共通定義の node_id オペランド)は一意にする必要があります。また，すべての OpenTP1 システムは，バージョン 03-02 以降にする必要があります。これらの条件を満たしていないと，トランザクションが正しく回復できなくなる場合があります。

dc_rpc_call_to

名称

通信先を指定した遠隔サービスの呼び出し

形式

ANSI C，C++の形式

```
#include <dcrpc.h>
int dc_rpc_call_to(struct DCRPC_BINDING_TBL *direction,
                   char *group,
                   char *service,char *in, DCULONG *in_len,
                   char *out, DCULONG *out_len, DCLONG flags)
```

K&R 版 C の形式

```
#include <dcrpc.h>
int dc_rpc_call_to(*direction, *group, *service, *in, *in_len,
                   *out, *out_len, flags)
struct DCRPC_BINDING_TBL  *direction;
char      *group;
char      *service;
char      *in;
DCULONG   *in_len;
char      *out;
DCULONG   *out_len;
DCLONG    flags;
```

機能

特定の SPP にサービスを要求します。dc_rpc_call 関数と同様に「サービスグループ名」と「サービス名」を引数に設定するのに加え，ホスト名またはノード識別子を指定した DCRPC_BINDING_TBL 構造体を引数に設定します。DCRPC_BINDING_TBL 構造体に指定したホスト名またはノード識別子は，サービス要求先を特定する検索のキーとして使用します。この設定に該当するサービス関数へサービスが要求されます。

ただし，ドメイン修飾をしてサービスを要求できません。それ以外は，dc_rpc_call 関数の機能と変わりません。

なお，この機能は，TP1/Extension 1 をインストールしていることが前提です。TP1/Extension 1 をインストールしていない場合の動作は保証できませんので，ご了承ください。

UAP で値を設定する引数

●direction

サービス要求先を特定する検索キーを格納する DCRPC_BINDING_TBL 構造体のアドレスを指定します。検索のキーはホスト名またはノード識別子のどちらかです。

DCRPC_BINDING_TBL 構造体の形式を次に示します。

```
struct DCRPC_BINDING_TBL {
    char *nid ;           /*ノード識別子格納アドレス*/
    char *hostnm ;        /*ホスト名格納アドレス*/
    short portno ;        /*ポート番号*/
    short filler1 ;       /*予備*/
    DCLONG flags ;        /*属性*/
    DCLONG filler2[4] ;   /*予備*/
};
```

- nid

  ノード識別子を検索のキーにする場合，サービス要求先のノード識別子を格納した領域のアドレスを指定します。文字列の最後にはヌル文字を設定してください。このヌル文字は文字列の長さに数えません。

  ノード識別子は，システム共通定義の node_id オペランドに指定した名称で，かつサービス要求先ノードのホスト名がグローバルドメイン※内にあることが前提です。

  ノード識別子を検索のキーにしない場合には，nid にアドレス 0 を指定してください。

- hostnm

  ホスト名を検索のキーにする場合，サービス要求先ノードのホスト名を格納した領域のアドレスを指定します。指定できるホスト名は 1 から 255 文字までの文字列です。文字列に使用できる文字は，英数字記号，ピリオド，およびハイフンです（ただし，IP アドレス形式は除きます）。文字列の最後にはヌル文字を設定してください。このヌル文字は文字列の長さに数えません。指定するホスト名は，/etc/hosts ファイルまたは DNS などで，IP アドレスとのマッピングができる名称です。

  なお，サービス要求先ノードのホスト名はグローバルドメイン※に指定されていても，指定されていなくても，どちらでもかまいません。

  ホスト名を検索のキーにしない場合には，hostnm にアドレス 0 を指定してください。

- portno

  ホスト名を検索のキーとする場合，サービス要求先ノードのネームサービスのポート番号（システム共通定義 name_port に指定した値）を指定します。portno の指定値は，DCRPC_BINDING_TBL 構造体の flags に DCRPC_NAMPORT を指定した場合だけ有効です。

  portno に 0 を指定した場合，または DCRPC_BINDING_TBL 構造体の flags に DCNOFLAGS を指定した場合は，サービス要求元のネームサービスのポート番号と，サービス要求先のネームサービスのポート番号が同じであることが前提となります。

  また，ノード識別子を検索のキーとする場合は，portno に指定した値は無視します。

- flags

  DCNOFLAGS を指定します。

portno に値を指定した場合は，DCRPC_NAMPORT を指定してください。

なお，filler1，filler2 は拡張性を考慮して設けたエリアなので，値を設定する必要はありません（filler1，filler2 のメンバ名称は使用しないでください）。

注※

ここでのグローバルドメインとは，次のノード名の集合を指します。

システム共通定義の name_domain_file_use オペランドに N を指定している場合

システム共通定義の all_node オペランド，all_node_ex オペランドで指定したノード名の集合です。

システム共通定義の name_domain_file_use オペランドに Y を指定している場合

ドメイン定義ファイルに指定したノード名の集合です。なお，ドメイン定義ファイルは次の場所に格納します。

- all_node のドメイン定義ファイル
  $DCCONFPATH/dcnamndディレクトリ下
- all_node_ex のドメイン定義ファイル
  $DCCONFPATH/dcnamndexディレクトリ下

dc_rpc_call_to 関数の direction に指定する DCRPC_BINDING_TBL 構造体は，「DCRPC_BINDTBL_SET 関数」または「DCRPC_DIRECT_SCHEDULE 関数」を使用して作成することもできます。詳細については，「DCRPC_BINDTBL_SET，DCRPC_DIRECT_SCHEDULE」を参照してください。

●group

SPP の**サービスグループ名**を，31 バイト以内のアスキー文字列で設定します。文字列の最後にはヌル文字を設定してください。このヌル文字は文字列の長さに数えません。

●service

SPP の**サービス名**を，31 バイト以内のアスキー文字列で設定します。文字列の最後にはヌル文字を設定してください。このヌル文字は文字列の長さに数えません。

●in

サービスの入力パラメタを設定します。

●in_len

サービスの入力パラメタ長を設定します。1 から DCRPC_MAX_MESSAGE_SIZE※までの範囲の長さが設定できます。DCRPC_MAX_MESSAGE_SIZE は，dcrpc.h で定義してあります。

注※

rpc_max_message_size オペランドを使用した場合，DCRPC_MAX_MESSAGE_SIZE の値（1 メガバイト）ではなく，rpc_max_message_size オペランドに指定した値になります。

●out

サービス関数から返ってくる応答を格納する領域を設定します。out で指す領域へ，サービス関数からの応答が設定されます。

●out_len

サービスの応答の長さを設定します。1 から DCRPC_MAX_MESSAGE_SIZE※までの範囲の長さが設定できます。DCRPC_MAX_MESSAGE_SIZE は，dcrpc.h で定義してあります。

注※

rpc_max_message_size オペランドを使用した場合，DCRPC_MAX_MESSAGE_SIZE の値（1 メガバイト）ではなく，rpc_max_message_size オペランドに指定した値になります。

非応答型 RPC の場合も，サービスの応答の長さを設定した領域のアドレスを指定する必要があります。この場合，サービスの応答の長さには，0 を設定します。

●flags

RPC の形態とオプションを次に示す形式で設定します。

```
{DCNOFLAGS|DCRPC_NOWAIT|DCRPC_NOREPLY|DCRPC_CHAINED}〔|DCRPC_TPNOTRAN〕
```

DCNOFLAGS

同期応答型 RPC

DCRPC_NOWAIT

非同期応答型 RPC

DCRPC_NOREPLY

非応答型 RPC

DCRPC_CHAINED

連鎖 RPC

DCRPC_TPNOTRAN

トランザクション処理からのサービス要求で，要求先の処理をトランザクションにしない場合に設定します。または DCRPC_DIRECT_SCHEDULE 関数を使用して DCRPC_BINDING_TBL 構造体を作成し，非トランザクション属性（ユーザサービス定義の atomic_update オペランドに'N'を指定，またはシステム共通定義の jnl_fileless_option オペランドに'Y'を指定）のユーザサーバにサービス要求を行う場合に設定します。

この値は RPC の形態との論理和で設定してください。

（例）

同期応答型 RPC でトランザクションにしないサービス要求をする場合，flags には「DCNOFLAGS ｜ DCRPC_TPNOTRAN」と設定します。

サーバ UAP から値が返される引数

●out

サービス関数が設定した，応答が返されます。

●out_len

サービス関数が設定した，応答の長さが返されます。

リターン値

dc_rpc_call 関数のリターン値を参照してください。

なお，dc_rpc_call 関数のリターン値のうち，dc_rpc_call_to 関数ではエラー要因に追加があるリターン値を次に示します。

リターン値	リターン値（数値）	意味
DCRPCER_INVALID_ARGS	-301	引数に設定した値が間違っています。
		DCRPC_BINDING_TBL 構造体の hostnm に指定されたホスト名が，/etc/hosts ファイルまたは DNS などで，IP アドレスとのマッピングができません。
		dc_rpc_call_to 関数の第 1 引数に指定した DCRPC_BINDING_TBL 構造体が，DCRPC_DIRECT_SCHEDULE 関数で作成されていて，DCRPC_DIRECT_SCHEDULE 関数の hostnm に 0 を指定しています。
		dc_rpc_call_to 関数の第 8 引数に，指定できない DCRPC_DOMAIN を指定しています。
DCRPCER_NO_SUCH_SERVICE_GROUP	-310	group に設定したサービスグループは定義されていません。または，group に設定したサービスグループがサポートしていない機能を使用して，dc_rpc_call_to 関数を実行しています。
		DCRPC_BINDING_TBL 構造体の nid に指定されたノード識別子が，グローバルドメイン※内にありません。
DCRPCER_SERVICE_NOT_UP	-314	DCRPC_DIRECT_SCHEDULE 関数を使用して DCRPC_BINDING_TBL 構造体を作成し，ソケット受信型（ユーザサービス定義の receive_from オペランドに socket を指定）ユーザサーバをサービス要求先として，dc_rpc_call_to 関数を呼び出しました。
DCRPCER_TRNCHK_EXTEND	-372	同時に起動できるトランザクションブランチの数を超えたため，トランザクションブランチを開始できません。

リターン値	リターン値（数値）	意味
DCRPCER_TRNCHK_EXTEND	-372	一つのトランザクションブランチから開始できる子トランザクションブランチの最大数を超えたため，トランザクションブランチを開始できません。
		トランザクション内でドメイン修飾をしたサービス要求で，flags に DCRPC_TPNOTRAN を設定していません。
		リソースマネジャ(RM)でエラーが発生したため，トランザクションブランチを開始できません。
		DCRPC_DIRECT_SCHEDULE 関数を使用して DCRPC_BINDING_TBL 構造体を作成し，非トランザクション属性（ユーザサービス定義の atomic_update オペランドに'N'を指定，またはシステム共通定義の jnl_fileless_option オペランドに'Y'を指定）のユーザサーバに対し，dc_rpc_call_to 関数の flags に DCRPC_TPNOTRAN との論理和を指定しないで dc_rpc_call_to 関数を呼び出しました。
		TP1/LiNK の「システム環境設定」ウィンドウの［トランザクション機能］が［あり］に設定されていません。

注※

ここでのグローバルドメインとは，次のノード名の集合を指します。

システム共通定義の name_domain_file_use オペランドに N を指定している場合

システム共通定義の all_node オペランド，all_node_ex オペランドで指定したノード名の集合です。

システム共通定義の name_domain_file_use オペランドに Y を指定している場合

ドメイン定義ファイルに指定したノード名の集合です。なお，ドメイン定義ファイルは次の場所に格納します。

- all_node のドメイン定義ファイル
  $DCCONFPATH/dcnamndディレクトリ下
- all_node_ex のドメイン定義ファイル
  $DCCONFPATH/dcnamndexディレクトリ下

その他関連事項

dc_rpc_call 関数の項目を参照してください。

注意事項

1. 一つのマシン内に複数の LAN アダプタが接続されているマルチホームドホスト形態で，システム共通定義の my_host に指定したホスト名と異なる自マシン内のほかのホスト名を，DCRPC_BINDING_TBL 構造体の hostnm，DCRPC_BINDING_SET 関数の hostnm，または

DCRPC_DIRECT_SCHEDULE 関数の hostnm に指定してはいけません。指定した場合は，**動作の保証はしません**。

2. ホスト名とノード識別子を同時に DCRPC_BINDING_TBL 構造体に指定した場合，ホスト名の指定が有効になり，ノード識別子の指定は無視します。
3. DCRPC_BINDING_TBL 構造体に，ホスト名，ノード識別子ともに 0 を指定した場合は，dc_rpc_call 関数とまったく同様に動作します。
4. スケジュールサービスの管理するユーザサーバに直接サービスを要求する場合は，必ず DCRPC_DIRECT_SCHEDULE 関数を使用して DCRPC_BINDING_TBL 構造体を作成してください。
5. DCRPC_DIRECT_SCHEDULE 関数で作成した DCRPC_BINDING_TBL 構造体を指定して dc_rpc_call_to 関数を呼び出す場合，サービス要求先の OpenTP1 のバージョンは 03-02 以降でなければなりません。03-02 以降でない場合は，**動作の保証はしません**。
6. 次の場合，dc_rpc_call_to 関数が DCRPCER_TIMED_OUT でリターンすることがあります。

   システム共通定義の all_node オペランドで指定していないノードのサービスグループに対して，ホスト名を検索のキーに使用して dc_rpc_call_to 関数を呼び出したあとに，このノード上の OpenTP1 を停止または再開始し，同じサービスグループに対して再度ホスト名を検索のキーに使用して dc_rpc_call_to 関数を呼び出した場合
7. DCRPC_DIRECT_SCHEDULE 関数を使用して作成した DCRPC_BINDING_TBL 構造体を dc_rpc_call_to 関数の direction に指定して，dc_rpc_call_to 関数を呼び出した場合，rpc_retry オペランドは無効になります。
8. DCRPC_DIRECT_SCHEDULE 関数を使用して作成した DCRPC_BINDING_TBL 構造体を dc_rpc_call_to 関数の direction に指定して，dc_rpc_call_to 関数を呼び出した場合，性能検証用トレースは取得できますが，呼び出し先の UAP の性能検証用トレース情報とはリンクしていません。クライアント UAP で取得した性能検証用トレースの通番は，サーバ UAP に引き継ぎません。

DCRPC_BINDTBL_SET，DCRPC_DIRECT_SCHEDULE

名称

DCRPC_BINDING_TBL 構造体の設定

形式

ANSI C，C++の形式

```
#include <dcrpc.h>
void DCRPC_BINDTBL_SET(struct DCRPC_BINDING_TBL *direction,
                       char *nid, char *hostnm,
                       short portno, DCLONG flags)

void DCRPC_DIRECT_SCHEDULE(struct DCRPC_BINDING_TBL *direction,
                           char *hostnm, short scdport,
                           DCLONG flags)
```

K&R 版 C の形式

```
#include <dcrpc.h>
void DCRPC_BINDTBL_SET(*direction, *nid, *hostnm, portno, flags)
struct DCRPC_BINDING_TBL *direction;
char      *nid;
char      *hostnm;
short     portno;
DCLONG    flags;

void DCRPC_DIRECT_SCHEDULE(*direction, *hostnm, scdport, flags)
struct DCRPC_BINDING_TBL  *direction;
char      *hostnm;
short     scdport;
DCLONG    flags;
```

機能

dc_rpc_call_to 関数の第 1 引数に指定する DCRPC_BINDING_TBL 構造体を次のどちらかの関数を使用して作成します。

- DCRPC_BINDTBL_SET 関数

  指定されたサービス要求先ノードのノード識別子(nid)，または指定されたサービス要求先ノードのホスト名（hostnm）を DCRPC_BINDING_TBL 構造体に設定し，dc_rpc_call_to 関数の第 1 引数を作成します。

- DCRPC_DIRECT_SCHEDULE 関数

  指定されたサービス要求先ノードのホスト名（hostnm）と，指定されたスケジュールサービスのポート番号（scdport）を DCRPC_BINDING_TBL 構造体に設定し，dc_rpc_call_to 関数の第 1 引数を作成します。

DCRPC_DIRECT_SCHEDULE 関数で作成した DCRPC_BINDING_TBL 構造体を指定して dc_rpc_call_to 関数を呼び出すと，OpenTP1 は指定されたスケジュールサービスの管理するユーザサーバに，直接サービス要求を送信します。ただし，DCRPC_DIRECT_SCHEDULE 関数で作成した DCRPC_BINDING_TBL 構造体は，次に示すユーザサーバへのサービス要求にしか使用できません。

- キュー受信型（ユーザサービス定義の receive_from オペランドに queue を指定）のユーザサーバ
- TP1/EE のサーバ UAP
- アプリケーションサーバの TP1 インバウンドアダプタ（TP1 インバウンドアダプタについては，マニュアル「Cosminexus V9 アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」を参照してください。）

また，DCRPC_DIRECT_SCHEDULE 関数で作成した DCRPC_BINDING_TBL 構造体を指定して dc_rpc_call_to 関数を呼び出す場合は，サービス要求先の OpenTP1 のバージョンやユーザサーバのトランザクション属性を認識しなければならないなど，多くの注意事項があります。詳細については，dc_rpc_call_to 関数の注意事項を参照してください。

UAP で値を設定する引数

●direction

dc_rpc_call_to 関数の第 1 引数で使用する DCRPC_BINDING_TBL 構造体のアドレスを設定します。

●nid

DCRPC_BINDTBL_SET 関数で，ノード識別子をサービス要求先を特定する検索のキーとする場合，ノード識別子を格納した領域のアドレスを指定します。文字列の最後にはヌル文字を設定してください。このヌル文字は文字列の長さに数えません。

ノード識別子は，システム共通定義の node_id オペランドに指定した名称で，かつサービス要求先ノードのホスト名がグローバルドメイン※内にあることが前提です。

ノード識別子を検索のキーにしない場合には，nid にアドレス 0 を指定してください。

注※

ここでのグローバルドメインとは，次のノード名の集合を指します。

システム共通定義の name_domain_file_use オペランドに N を指定している場合

システム共通定義の all_node オペランド，all_node_ex オペランドで指定したノード名の集合です。

システム共通定義の name_domain_file_use オペランドに Y を指定している場合

ドメイン定義ファイルに指定したノード名の集合です。なお，ドメイン定義ファイルは次の場所に格納します。

- all_node のドメイン定義ファイル
  $DCCONFPATH/dcnamndディレクトリ下
- all_node_ex のドメイン定義ファイル

$DCCONFPATH/dcnamndexディレクトリ下

●hostnm

サービス要求先ノードのホスト名を格納した領域のアドレスを指定します。指定できるホスト名は 1 から 255 文字までの文字列です。文字列の最後にはヌル文字を設定してください。このヌル文字は文字列の長さに数えません。指定するホスト名は，/etc/hosts ファイルまたは DNS などで，IP アドレスとのマッピングができる名称です。

なお，サービス要求先ノードのホスト名はグローバルドメイン※に指定されていても，指定されていなくても，どちらでもかまいません。

DCRPC_BINDTBL_SET 関数で，ホスト名をサービス要求先を特定する検索のキーにしない場合には，hostnm にアドレス 0 を指定してください。

DCRPC_DIRECT_SCHEDULE 関数では，hostnm は必ず指定してください。
DCRPC_DIRECT_SCHEDULE 関数の hostnm にアドレス 0 を指定した場合は，この DCRPC_BINDING_TBL 構造体を指定して呼び出した dc_rpc_call_to 関数が，DCRPCER_INVALID_ARGS でエラーリターンします。

注※

ここでのグローバルドメインとは，次のノード名の集合を指します。

システム共通定義の name_domain_file_use オペランドに N を指定している場合

システム共通定義の all_node オペランド，all_node_ex オペランドで指定したノード名の集合です。

システム共通定義の name_domain_file_use オペランドに Y を指定している場合

ドメイン定義ファイルに指定したノード名の集合です。なお，ドメイン定義ファイルは次の場所に格納します。

- all_node のドメイン定義ファイル
  $DCCONFPATH/dcnamndディレクトリ下
- all_node_ex のドメイン定義ファイル
  $DCCONFPATH/dcnamndexディレクトリ下

●portno

- DCRPC_BINDTBL_SET 関数でホスト名を検索のキーとする場合
  サービス要求先の OpenTP1 システムのネームサービスのポート番号（システム共通定義の name_port オペランドで指定）を指定します。サービス要求先のネームサービスのポート番号が，サービス要求元のネームサービスのポート番号と同じ場合は，0 を指定します。
- DCRPC_BINDTBL_SET 関数でノード識別子を検索のキーとする場合
  portno に 0 を指定します。サービス要求先のポート番号（システム共通定義の all_node オペランドで指定）を省略している場合は，サービス要求先のネームサービスのポート番号（システム共通定義の name_port オペランドで指定）と，サービス要求元のネームサービスのポート番号が同じでなければなりません。

●scdport

DCRPC_DIRECT_SCHEDULE 関数の scdport には，サービス要求先 OpenTP1 システムのスケジュールサービスのポート番号（サービス要求先のスケジュールサービス定義の scd_port オペランドで指定）を指定します。0 を指定した場合は，送信先ポート番号の省略値として，サービス要求元のスケジュールサービス定義に指定されている scd_port オペランドの値を仮定します。したがって，
DCRPC_DIRECT_SCHEDULE 関数の scdport に 0 を指定する場合，サービス要求元の OpenTP1 は，スケジュールサービス定義に scd_port オペランドが指定され，起動されていなければなりません。

●flags

DCNOFLAGS を指定します。

その他関連事項

dc_rpc_call_to 関数を参照してください。

注意事項

1. DCRPC_BINDTBL_SET 関数，DCRPC_DIRECT_SCHEDULE 関数は，dc_rpc_call_to 関数の第 1 引数に指定する，DCRPC_BINDING_TBL 構造体を設定するための関数です。
2. DCRPC_BINDTBL_SET 関数，DCRPC_DIRECT_SCHEDULE 関数の引数の指定値チェックや，指定方法によってどのような動作をするかについては，作成した DCRPC_BINDING_TBL 構造体を指定して呼び出した dc_rpc_call_to 関数の動作に現れます。dc_rpc_call_to 関数については，「2. リモートプロシジャコール（dc_rpc_～）」の dc_rpc_call_to を参照してください。
3. DCRPC_BINDTBL_SET 関数，DCRPC_DIRECT_SCHEDULE 関数では，UAP トレースは取得しません。

dc_rpc_close

名称

アプリケーションプログラムの終了

形式

ANSI C，C++の形式

```
#include <dcrpc.h>
void  dc_rpc_close(DCLONG flags)
```

K&R 版 C の形式

```
#include <dcrpc.h>
void dc_rpc_close(flags)
DCLONG     flags;
```

機能

OpenTP1 の各種関数を使うための環境をクローズします。dc_rpc_close 関数を呼び出したあとは，OpenTP1 の関数は使えません。

dc_rpc_close 関数はメイン関数で呼び出します。プロセスで 1 回だけ呼び出してください。

dc_rpc_close 関数は，OpenTP1 に UAP が正常に終了したことを知らせる働きをします。dc_rpc_close 関数を呼び出さないで UAP を終了すると，異常終了と見なされて，サービスグループの閉塞やプロセス再起動の対象になることがあります。また，OpenTP1 で管理する各種資源が解放されないままになって，そのあとの処理に悪影響を与える場合があります。OpenTP1 で使うすべての UAP では，dc_rpc_open 関数を呼び出したら，exit()で終了する前に dc_rpc_close 関数を必ず呼び出してください。

dc_rpc_close 関数は，dc_rpc_open 関数がエラーリターンしたときにも必ず呼び出してください。

dc_rpc_close 関数を呼び出したあとは，同じ UAP で再び dc_rpc_open 関数を呼び出せません。

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

リターン値

dc_rpc_close 関数のリターン値はありません。

dc_rpc_cltsend

名称

CUP への一方通知

形式

ANSI C，C++の形式

```
#include <dcrpc.h>
int  dc_rpc_cltsend(char *hostname,unsigned short port,
                    char *msg,
                    DCLONG len,DCLONG flags)
```

K&R 版 C の形式

```
#include <dcrpc.h>
int  dc_rpc_cltsend(hostname,port,msg,len,flags)
char       *hostname;
unsigned  short  port;
char       *msg;
DCLONG     len;
DCLONG     flags;
```

機能

CUP へ一方的にデータを送ります。hostname と port で示すホストのポート番号に対応するプロセス（CUP）に，msg で示すデータを len で示す長さだけ送ります。送信できるデータの長さは，0 から DCRPC_MAX_MESSAGE_SIZE※で示すバイト長までです。

注※

rpc_max_message_size オペランドを使用した場合，DCRPC_MAX_MESSAGE_SIZE の値（1 メガバイト）ではなく，rpc_max_message_size オペランドに指定した値になります。

dc_rpc_cltsend 関数で送信したデータは，TP1/Client が提供する次の API で受け取ることができます。

- TP1/Client/W，TP1/Client/P の場合
  dc_clt_accept_notification 関数
  dc_clt_accept_notification_s 関数
  dc_clt_chained_accept_notification 関数
  dc_clt_chained_accept_notification_s 関数
  CBLDCCLS('NOTIFY ')
  CBLDCCLS('A-NOTIFY')
  CBLDCCLS('EXNACPT ')
  CBLDCCLS('EXNCACPT')

- TP1/Client/J の場合
  acceptNotification メソッド
  acceptNotificationChained メソッド
- TP1/Client for .NET Framework の場合
  AcceptNotification メソッド
  AcceptNotificationChained メソッド

TP1/Client のライブラリが提供する API については，次の各製品マニュアルを参照してください。

- マニュアル「OpenTP1 クライアント使用の手引 TP1/Client/W，TP1/Client/P 編」
- マニュアル「OpenTP1 クライアント使用の手引 TP1/Client/J 編」
- マニュアル「TP1/Client for .NET Framework 使用の手引」

UAP で値を設定する引数

●hostname

データを送信するホスト名を設定します。指定できるホスト名は 1 から 255 文字までの文字列です。文字列の後ろには，ヌル文字を設定してください。

●port

データを送信するポート番号を設定します。

●msg

送信するデータを設定します。

●len

送信するデータの長さを設定します。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCRPCER_INVALID_ARGS	-301	引数に設定した値が間違っています。
DCRPCER_PROTO	-302	dc_rpc_open 関数を呼び出していません。
DCRPCER_NO_BUFS	-304	メモリが不足しました。

リターン値	リターン値（数値）	意味
DCRPCER_NET_DOWN	-306	ネットワークに障害が起こりました。
DCRPCER_MESSAGE_TOO_BIG	-308	送信するデータの長さが，DCRPC_MAX_MESSAGE_SIZE※を超えています。
DCRPCER_SERVICE_NOT_UP	-314	送信先プロセスがありません。
		ネットワークに障害が起こりました。

注※

rpc_max_message_size オペランドを使用した場合，DCRPC_MAX_MESSAGE_SIZE の値（1 メガバイト）ではなく，rpc_max_message_size オペランドに指定した値になります。

注意事項

1. dc_rpc_cltsend 関数の正常リターンは，RPC の通信プロトコル（TCP/IP）レベルでの送信（データ受け渡し）が完了したことを意味します。そのため，dc_rpc_cltsend 関数が正常にリターンしても，CUP がデータを正常に受信していないおそれがあります。したがって，CUP が受信可能な TP1/Client の API を呼び出していることが明らかな場合にだけ，dc_rpc_cltsend 関数を発行してください。
2. CUP が稼働していない場合は，DCRPCER_SERVICE_NOT_UP でエラーリターンします。
3. dc_rpc_cltsend 関数によるデータ送信先に指定できるのは CUP だけです。SPP のプロセスや自プロセスへは，データを送信できません。
4. hostname と port が示す 1 つの CUP に対し，複数のプロセスから同時にデータを送信する場合，送信するデータの長さは 32,656 バイトを超えないようにしてください。次の条件がすべて重なった場合，CUP が受信データを破棄してしまうことがあります。
   - 1 つの CUP に対し，複数のプロセスから同時にデータを送信する。
   - 1 つの CUP に対し，複数のプロセスから同時に送信するデータの中に，32,656 バイトを超えるデータが存在する。
   - 32,656 バイトを超えるデータの送信処理中に，一時クローズ処理が実行される。

dc_rpc_discard_further_replies

名称

処理結果の受信の拒否

形式

ANSI C，C++の形式

```
#include <dcrpc.h>
void  dc_rpc_discard_further_replies(DCLONG flags)
```

K&R 版 C の形式

```
#include <dcrpc.h>
void dc_rpc_discard_further_replies(flags)
DCLONG     flags;
```

機能

非同期応答型 RPC（dc_rpc_call 関数の flags に DCRPC_NOWAIT を設定）で，まだ返ってきていない応答を，これ以上受信しないことを示す関数です。この関数を呼び出したあとは，応答が返ってきても受信しないで捨てられます。

非同期応答型 RPC の結果をこれ以上受信しない場合は，必ず dc_rpc_discard_further_replies 関数を呼び出してください。呼び出さないと，dc_rpc_poll_any_replies 関数が不要な応答を受信してしまうことがあります。

dc_rpc_discard_further_replies 関数を使う場合を次に示します。

- 応答待ち時間切れになったあと，結果を保持しておくバッファを解放する場合
- 非同期応答型 RPC を複数回使って，そのうち最初の応答だけ必要な場合

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

リターン値

dc_rpc_discard_further_replies 関数のリターン値はありません。

dc_rpc_discard_specific_reply

名称

特定の処理結果の受信の拒否

形式

ANSI C，C++の形式

```
#include <dcrpc.h>
int  dc_rpc_discard_specific_reply (int des, DCLONG flags)
```

K&R 版 C の形式

```
#include <dcrpc.h>
int  dc_rpc_discard_specific_reply (des, flags)
int        des ;
DCLONG     flags ;
```

機能

非同期応答型 RPC（dc_rpc_call 関数の flags に DCRPC_NOWAIT を設定）で，まだ返ってきていない特定の応答を，これ以上受信しないことを示す関数です。受信を拒否する非同期応答を特定するには，des に，非同期応答型 RPC がリターンした際に返した記述子を設定します。この関数を呼び出したあとに返ってきた応答の中で，設定した記述子と同じ記述子を持つ応答は受信しないで捨てられます。

UAP で値を設定する引数

●des

非同期応答型 RPC の dc_rpc_call 関数（flags に DCRPC_NOWAIT を設定）が正常に終了したときに返された，記述子を設定します。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCRPCER_INVALID_ARGS	-301	引数に設定した値が間違っています。
DCRPCER_PROTO	-302	dc_rpc_open 関数を呼び出していません。

リターン値	リターン値（数値）	意味
DCRPCER_INVALID_DES	-322	des に設定した記述子は存在しません。要因として次のことが考えられます。 • 設定した記述子に対応する非同期応答型 RPC を行っていない。 • 非同期応答型 RPC の応答がすでに受信されている，または受信が拒否されている。

dc_rpc_get_callers_address

名称

クライアント UAP のノードアドレスの取得

形式

ANSI C，C++の形式

```
#include <dcrpc.h>
void  dc_rpc_get_callers_address(DCULONG *node,DCLONG flags)
```

K&R 版 C の形式

```
#include <dcrpc.h>
void  dc_rpc_get_callers_address(node,flags)
DCULONG     *node;
DCLONG      flags;
```

機能

クライアント UAP のプロセスが稼働するノードアドレスを，サーバ UAP で取得します。
dc_rpc_get_callers_address 関数からリターンしたアドレス値で，クライアント UAP のセキュリティチェックができます。

dc_rpc_get_callers_address 関数で返されたアドレスを使って，サービスの応答やエラーの応答などは送信できません。

dc_rpc_get_callers_address 関数は，サービス関数から呼び出してください。サービス関数以外から呼び出した場合の処理は保証しません。

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

OpenTP1 から値が返される引数

●node

クライアント UAP のノードアドレスが返されます。

リターン値

dc_rpc_get_callers_address 関数のリターン値はありません。

注意事項

次の条件が重なった場合，dc_rpc_get_callers_address 関数が返すクライアント UAP のノードアドレスは，実際にクライアント UAP が通信で使用したノードアドレスと異なる場合があります。

- リモート API 機能を使用して，サービス要求を受け付けた。
- クライアント UAP が存在するホストがマルチホームドホスト形態である。

dc_rpc_get_error_descriptor

名称

エラーが発生した非同期応答型 RPC 要求の記述子の取得

形式

ANSI C，C++の形式

```
#include <dcrpc.h>
int  dc_rpc_get_error_descriptor(DCLONG flags)
```

K&R 版 C の形式

```
#include <dcrpc.h>
int  dc_rpc_get_error_descriptor(flags)
DCLONG     flags
```

機能

非同期応答を特定しない dc_rpc_poll_any_replies 関数がエラーリターンした直後に呼び出すことで，エラーが発生した非同期応答型 RPC 要求に対応する記述子を取得します。

記述子を取得できるのは，SPP 側でエラーが発生した場合だけです。

dc_rpc_poll_any_replies 関数の呼び出し側でエラーが発生した場合には，この関数で記述子を取得できません。

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
正の整数		dc_rpc_poll_any_replies 関数が戻したエラーに対応する非同期応答型 RPC 要求の記述子を取得しました。
0		dc_rpc_poll_any_replies が戻したエラーに対応する非同期応答型 RPC 要求の記述子を取得できませんでした。
DCRPCER_INVALID_ARGS	-301	引数が間違っています。

dc_rpc_get_gateway_address

名称

ゲートウェイのノードアドレスの取得

形式

ANSI C，C++の形式

```
#include <dcrpc.h>
int dc_rpc_get_gateway_address(DCULONG *node,DCLONG flags)
```

K&R 版 C の形式

```
#include <dcrpc.h>
int dc_rpc_get_gateway_address(node, flags)
DCULONG    *node;
DCLONG     flags;
```

機能

アプリケーションゲートウェイ型ファイアウォールなどの，ゲートウェイを介してクライアント UAP からのサービス要求を受信したとき，サーバ UAP でゲートウェイのノードアドレスを取得します。

リモート API 機能を使用してサービスを要求した場合，サーバ UAP で，ゲートウェイのノードアドレスを取得できます。

dc_rpc_get_gateway_address 関数で返されたアドレスを使って，サービスの応答やエラーの応答などの送信はできません。

dc_rpc_get_gateway_address 関数は，サービス関数から呼び出してください。サービス関数以外から呼び出した場合の処理は保証しません。

UAP で値を設定する引数

●node

ゲートウェイのノードアドレスが返される領域のアドレスを設定します。

●flags

DCNOFLAGS を設定します。

OpenTP1 から値が返される引数

●node

ゲートウェイのノードアドレスが返されます。

リモート API 機能を使用していない場合には 0 が設定されます。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCRPCER_INVALID_ARGS	-301	引数に設定した値が間違っています。
DCRPCER_PROTO	-302	サービス関数から呼び出されていません。

dc_rpc_get_service_prio

名称

サービス要求のスケジュールプライオリティの参照

形式

ANSI C，C++の形式

```
#include <dcrpc.h>
int  dc_rpc_get_service_prio(void)
```

K&R 版 C の形式

```
#include <dcrpc.h>
int  dc_rpc_get_service_prio()
```

機能

dc_rpc_set_service_prio 関数で設定した，サービス要求のスケジュールプライオリティを参照します。この関数がリターンするスケジュールプライオリティの値は，UAP が再び dc_rpc_set_service_prio 関数を呼び出すまで変わりません。

次に示す場合，dc_rpc_get_service_prio 関数は省略時仮定値（4）をリターンします。

- UAP で dc_rpc_set_service_prio 関数を呼び出していない場合
- dc_rpc_set_service_prio 関数の引数 prio に 0 を設定して呼び出した場合

リターン値

リターン値	意味
正の整数	dc_rpc_set_service_prio 関数で設定したスケジュールプライオリティを，1～8 の範囲で示します。
上記以外	プログラムの破壊などによる，予期しないエラーが起こりました。

dc_rpc_get_watch_time

名称

サービスの応答待ち時間の参照

形式

ANSI C，C++の形式

```
#include <dcrpc.h>
int  dc_rpc_get_watch_time(void)
```

K&R 版 C の形式

```
#include <dcrpc.h>
int  dc_rpc_get_watch_time()
```

機能

現在のサービス要求の応答待ち時間を参照します。この関数は，dc_rpc_set_watch_time 関数で応答待ち時間を一時的に変更する前に，元の値を退避するために使います。

この関数は，dc_rpc_set_watch_time 関数で変更したサービス応答待ち時間をリターンします。変更していない場合は，次に示す値をリターンします。

- TP1/Server Base の場合：システム共通定義の watch_time オペランドの値
- TP1/LiNK の場合：180 秒

この関数で得られる値は，OpenTP1 の dc_rpc_call 関数に対して有効です。

リターン値

リターン値	リターン値（数値）	意味
正の整数		現在のサービス応答待ち時間を示します。
DC_OK	0	サービス応答待ち時間は，無制限に待ち続ける指定です。
上記以外		プログラムの破壊などによる，予期しないエラーが起こりました。

dc_rpc_mainloop

名称

SPP のサービス開始

形式

ANSI C，C++の形式

```
#include <dcrpc.h>
int  dc_rpc_mainloop(DCLONG flags)
```

K&R 版 C の形式

```
#include <dcrpc.h>
int dc_rpc_mainloop(flags)
DCLONG     flags;
```

機能

このプロセスで実行中の SPP にあるサービス関数へのサービス要求を受け付けます。dc_rpc_mainloop 関数は，メイン関数で呼び出します。プロセスで 1 回だけ呼び出してください。

dc_rpc_mainloop 関数は，OpenTP1 からの終了要求を受けるまでリターンしません。OpenTP1 からの終了要求を受けるとは次のような場合です。

- 次に示す OpenTP1 の終了コマンドを実行したために，終了処理に入ったとき

  dcstop コマンド（正常終了）

  dcstop -n コマンド（強制正常終了）

  dcstop -a コマンド（計画停止 A）

  dcstop -b コマンド（計画停止 B）
- dc_rpc_mainloop 関数を呼び出している SPP のプロセスを，次に示すサーバの終了コマンドを実行したために，終了処理に入ったとき

  dcsvstop コマンド（正常終了）
- dc_rpc_mainloop 関数を呼び出している SPP のプロセスが，ユーザサービス定義の最大プロセス数の指定を超えたために，OpenTP1 からプロセスを終了させられるとき
- SPP を非常駐プロセスで実行しているときは，サービスの処理を終了したとき
- SPP をマルチサーバで負荷分散しているときは，該当するサービスグループへのサービス要求が減少したとき

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	OpenTP1 からの終了要求を受けました。SPP はすぐに終了処理をして，dc_rpc_close 関数を呼び出して exit() してください。
DCRPCER_INVALID_ARGS	-301	引数が間違っています。
DCRPCER_PROTO	-302	dc_rpc_open 関数を呼び出していません。
		dc_rpc_mainloop 関数か，dc_mcf_mainloop 関数はすでに呼び出しています。
DCRPCER_FATAL	-303	SPP のサービスを開始できませんでした。

注意事項

dc_rpc_mainloop 関数は，OpenTP1 からの終了要求を受けるとリターンします。ただし，次のような場合は dc_rpc_mainloop 関数がリターンしないままプロセスが終了します。

1. OpenTP1 の強制停止コマンド（dcstop -f コマンド）や，サーバの強制停止コマンド（dcsvstop -f コマンド）を実行したために，SPP が終了処理に入ったとき
2. UAP または OpenTP1 の不良が原因でプロセスが異常終了したとき
3. サービス関数から abort や exit が実行されたとき
4. ハードウェアやオペレーティングシステム，または OpenTP1 そのものが障害になったとき

上記のような場合，dc_rpc_mainloop 関数が正常に終了したあとで，終了処理をするように SPP を作成してあっても，その処理は実行されないので注意してください。

dc_rpc_open

名称

アプリケーションプログラムの開始

形式

ANSI C，C++の形式

```
#include <dcrpc.h>
int  dc_rpc_open(DCLONG flags)
```

K&R 版 C の形式

```
#include <dcrpc.h>
int dc_rpc_open(flags)
DCLONG    flags;
```

機能

OpenTP1 の各種関数を使う準備をします。

dc_rpc_open 関数はメイン関数で呼び出します。すべての OpenTP1 の関数を呼び出す前に，プロセスで 1 回だけ呼び出してください。メイン関数での初期化手順を次に示します。

1. プロセス間通信のためのエントリポイントを開きます。
2. OpenTP1 で使う共用メモリを取得します。
3. OpenTP1 に UAP の開始を通知して，プロセスの監視を要求します。
4. そのほか，設定した UAP の環境に従って，使う OpenTP1 の各機能の初期化処理をします。

UAP にトランザクション属性を指定している場合は，そのノードで OpenTP1 のトランザクションサービスとプロセスサービスが実行中であることが前提になります。dc_rpc_open 関数は，OpenTP1 が OS の開始に伴って，または dcstart コマンドによって正常開始したあとでないと実行できません。OpenTP1 が正常開始する前に dc_rpc_open 関数を呼び出すと，DCRPCER_OLTF_NOT_UP でエラーリターンします。この場合は，dc_rpc_call 関数などの OpenTP1 の関数は使えません。

UAP トレースは，dc_rpc_open 関数が正常に終了したあとに呼び出したすべての OpenTP1 の関数について取得されます。そのため，dc_rpc_open 関数がエラーリターンした場合の UAP トレースは，取得されている場合も取得されていない場合もあります。

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCRPCER_INVALID_ARGS	-301	引数が間違っています。
DCRPCER_PROTO	-302	dc_rpc_open 関数はすでに呼び出しています。
DCRPCER_FATAL	-303	初期化に失敗しました。以降，OpenTP1 の関数は使えません。
DCRPCER_OLTF_NOT_UP	-315	UAP があるノードの OpenTP1 が実行していません。
DCRPCER_STANDBY_END	-369	待機系のユーザサーバが，待機の終了を要求されました。
DCRPCER_SEC_INIT	-371	セキュリティ機能を使った OpenTP1 が，セキュリティ環境の初期化処理でエラーになりました。

指定例

```
#include  <dcrpc.h>
main(){
if(dc_rpc_open(DCNOFLAGS) < 0){
  fputs("cannot begin usrserv1",stderr);
goto RPC_CLOSE;
}
if(dc_rpc_mainloop(DCNOFLAGS) < 0)
  fputs("cannot begin usrserv1",stderr);
/* サービス関数が呼び出されて実行されます。 */
/* この間，メイン関数には制御は戻りません */
RPC_CLOSE:
dc_rpc_close(DCNOFLAGS);
}
```

注意事項

dc_rpc_open 関数は，定義ファイルで設定した内容に従って，UAP が使う OpenTP1 の各機能の環境設定（初期化）をします。

初期化処理では，UAP プロセスでオープンするファイルディスクリプタの最大数を OS に対して設定します。したがって，dc_rpc_open 関数の発行後，UAP プロセスでオープンするファイルディスクリプタの最大数を OS に対して再設定（変更）しないでください。変更した場合の動作は保証できません。

dc_rpc_poll_any_replies

名称

処理結果の非同期受信

形式

ANSI C，C++の形式

```
#include <dcrpc.h>
int  dc_rpc_poll_any_replies(int des,DCLONG timeout,
                             DCLONG flags)
```

K&R 版 C の形式

```
#include <dcrpc.h>
int  dc_rpc_poll_any_replies(des,timeout,flags)
int       des;
DCLONG    timeout;
DCLONG    flags;
```

機能

非同期応答型 RPC（dc_rpc_call 関数の flags に DCRPC_NOWAIT を設定）でサービス要求した結果を受信します。

受信する非同期応答を特定する場合は，flags に DCRPC_SPECIFIC_MSG を設定します。このフラグを設定した場合は，des に設定した記述子をリターンした非同期応答型 RPC の応答を受信します。

受信する非同期応答を特定しない場合は，flags に DCNOFLAGS を設定します。このとき，des に設定した値は無視されます。flags に DCNOFLAGS を設定した dc_rpc_poll_any_replies 関数が正常に終了すると，受信した非同期応答の記述子と同じ値をリターンします。

dc_rpc_poll_any_replies 関数は，次のどちらかの場合にリターンします。

- 非同期応答型 RPC の応答を受信した場合
- timeout に指定した応答待ち時間切れになった場合

dc_rpc_poll_any_replies 関数が正常に終了すると，非同期応答型 RPC を使った dc_rpc_call 関数に設定した応答を格納する領域に，受信した応答が設定されます。

リターン値の一覧のあとに，次の説明を掲載しています。dc_rpc_poll_any_replies 関数の詳しい説明を知りたいときに参照してください。

(1)dc_rpc_poll_any_replies 関数の引数　timeout について

(2)dc_rpc_poll_any_replies 関数がエラーになるタイミング

(3)リターン値 DCRPCER_SERVICE_TERMINATED をリターンさせる指定

(4)エラーリターン値と同期点処理の関係

(5)dc_rpc_poll_any_replies 関数で応答が受け取れない場合

(6)dc_rpc_poll_any_replies 関数を使うときの注意

UAP で値を設定する引数

●des

非同期応答型 RPC の dc_rpc_call 関数（flags に DCRPC_NOWAIT を設定）が正常に終了したときに返された，記述子を設定します。flags に DCNOFLAGS を設定した場合は，ここに設定した値は無視されます。

●timeout

非同期応答型 RPC の dc_rpc_call 関数の結果が返ってくるまでの待ち時間を，秒単位またはミリ秒単位で設定します。指定できる値の範囲は，-1 から DCLONG 型で表せる最大の数までです。

dc_rpc_poll_any_replies 関数で非同期応答を受信する場合は，UAP に設定した応答待ち時間を参照しません。

0 を設定した場合は，flags に DCNOFLAGS，または DCRPC_SPECIFIC_MSG を設定したとき，応答が返っていないと DCRPCER_TIMED_OUT で すぐにエラーリターンします。flags に DCRPC_WAIT_MILLISEC を設定したときは 50 ミリ秒として処理します。

-1 を設定した場合は，応答が返るまで待ち続けます。

●flags

次に示す形式で設定します。

```
{DCNOFLAGS|DCRPC_SPECIFIC_MSG} 〔|DCRPC_WAIT_MILLISEC〕
```

DCNOFLAGS

この dc_rpc_poll_any_replies 関数で受信する非同期の応答を特定しません。

DCRPC_SPECIFIC_MSG

des に設定した記述子をリターンした，非同期応答型 RPC の応答を受信します。

DCRPC_WAIT_MILLISEC

timeout で設定した待ち時間の単位をミリ秒にします。

リターン値

リターン値	リターン値（数値）	意味
正の整数		受信した非同期応答の記述子を示します。正の整数は，flags に DCNOFLAGS を設定した dc_rpc_poll_any_replies 関数が正常に終了した場合に返されます。
DC_OK	0	正常に終了しました。DC_OK は，flags に DCRPC_SPECIFIC_MSG を設定した dc_rpc_poll_any_replies 関数が正常に終了した場合に返されます。
DCRPCER_INVALID_ARGS	-301	引数に設定した値が間違っています。
DCRPCER_PROTO	-302	dc_rpc_open 関数を呼び出していません。
DCRPCER_NO_BUFS	-304	メモリが不足しました。
DCRPCER_NET_DOWN	-306	ネットワークに障害が起こりました。
DCRPCER_TIMED_OUT	-307	dc_rpc_call 関数または dc_gwf_call 関数の処理が時間切れ（タイムアウト）になりました。
		サービスを要求された SPP が，処理を完了する前に異常終了しました。
DCRPCER_MESSAGE_TOO_BIG	-308	dc_rpc_call 関数または dc_gwf_call 関数の in_len に設定した入力パラメタ長が，最大値を超えています。
DCRPCER_REPLY_TOO_BIG	-309	返ってきた応答が，クライアント UAP で用意した領域に入り切りません。
DCRPCER_NO_SUCH_SERVICE_GROUP	-310	dc_rpc_call 関数または dc_gwf_call 関数の group に設定したサービスグループは，定義されていません。または，group に設定したサービスグループがサポートしていない機能を使用して，dc_rpc_call 関数もしくは dc_gwf_call 関数を実行しています。
DCRPCER_NO_SUCH_SERVICE	-311	サービスを要求した SPP には，service に設定したサービス名が定義されていません。
DCRPCER_SERVICE_CLOSED	-312	dc_rpc_call 関数または dc_gwf_call 関数の service に設定したサービス名があるサービスグループは，閉塞しています。
DCRPCER_SERVICE_TERMINATING	-313	dc_rpc_call 関数または dc_gwf_call 関数の service に設定したサービスは，終了処理中です。
DCRPCER_SERVICE_NOT_UP	-314	dc_rpc_call 関数または dc_gwf_call 関数の service に設定したサービスの UAP プロセスが，稼働していません。
		timeout に-1 を設定した場合に，サービスを要求された SPP が，処理を完了する前に異常終了しました。
DCRPCER_OLTF_NOT_UP	-315	dc_rpc_call 関数または dc_gwf_call 関数の service に設定したサービスがあるノードの OpenTP1 が稼働していません。

リターン値	リターン値（数値）	意味
DCRPCER_OLTF_NOT_UP	-315	異常終了，停止中，終了処理中，および通信障害が起こったことが考えられます。
DCRPCER_SYSERR_AT_SERVER	-316	dc_rpc_call 関数または dc_gwf_call 関数に設定したサービスで，システムエラーが起こりました。
DCRPCER_NO_BUFS_AT_SERVER	-317	dc_rpc_call 関数または dc_gwf_call 関数に設定したサービスで，メモリが不足しました。
DCRPCER_SYSERR	-318	システムエラーが起こりました。
DCRPCER_INVALID_REPLY	-319	サービス関数が OpenTP1 に返した応答の長さが，1 から DCRPC_MAX_MESSAGE_SIZE※で定義されている値の範囲にありません。
DCRPCER_OLTF_INITIALIZING	-320	サービスを要求されたノードにある OpenTP1 は，開始処理中です。
DCRPCER_ALL_RECEIVED	-321	非同期応答型 RPC で要求したサービスの処理結果は，すべて受信しました。
DCRPCER_INVALID_DES	-322	des に設定した記述子は存在しません。このリターン値は，flags に DCRPC_SPECIFIC_MSG を設定した場合に返されます。
DCRPCER_NO_BUFS_RB	-323	メモリが不足しました。このリターン値が返った場合は，トランザクションブランチをコミットできません。
DCRPCER_SYSERR_RB	-324	システムエラーが起こりました。このリターン値が返った場合は，トランザクションブランチをコミットできません。
DCRPCER_SYSERR_AT_SERVER_RB	-325	設定したサービスで，システムエラーが起こりました。このリターン値が返った場合は，トランザクションブランチをコミットできません。
DCRPCER_REPLY_TOO_BIG_RB	-326	返ってきた応答が，クライアント UAP で用意した領域に入り切りません。このリターン値が返った場合は，トランザクションブランチをコミットできません。
DCRPCER_TRNCHK	-327	ノード間負荷バランス機能およびノード間負荷バランス拡張機能の環境で，複数の SPP のトランザクション属性が一致していません。このリターン値は，ノード間負荷バランス機能およびノード間負荷バランス拡張機能を使っている SPP にサービスを要求した場合にだけリターンされます。
DCRPCER_NO_SUCH_DOMAIN	-328	ドメイン修飾をしたサービスグループ名の，ドメイン名が間違っています。
DCRPCER_NO_PORT	-329	ドメイン修飾をしてサービスを要求しましたが，ドメイン代表スケジュールサービスのポート番号が見つかりません。
DCRPCER_SERVER_BUSY	-356	サービスを要求されたソケット受信型サーバが，サービス要求を受け取れません。

リターン値	リターン値（数値）	意味
DCRPCER_TESTMODE	-366	オンラインテスタを使っている環境で，テストモードの UAP からテストモードでない SPP へサービスを要求しています。または，テストモードでない UAP からテストモードの SPP へサービスを要求しています。
DCRPCER_SECCHK	-370	サービスを要求された SPP は，セキュリティ機能で保護されています。dc_rpc_call 関数または dc_gwf_call 関数でサービスを要求した UAP には，SPP へのアクセス権限がありません。
DCRPCER_TRNCHK_EXTEND	-372	同時に起動できるトランザクションブランチの数を超えたため，トランザクションブランチを開始できません。
		一つのトランザクションブランチから開始できる子トランザクションブランチの最大数を超えたため，トランザクションブランチを開始できません。
		リソースマネジャ(RM)でエラーが発生したため，トランザクションブランチを開始できません。
DCRPCER_SERVICE_TERMINATED	-378	サービスを要求された SPP が，処理を完了する前に異常終了しました。このリターン値は，ユーザサービス定義の rpc_extend_function オペランドに"00000001"を指定したクライアント UAP の場合にだけリターンされます。rpc_extend_function オペランドに"00000000"を指定，またはオペランドを省略した場合は，このリターン値は返らないで，DCRPCER_TIMED_OUT または DCRPCER_SERVICE_NOT_UP がリターンされます。

注※

rpc_max_message_size オペランドを使用した場合，DCRPC_MAX_MESSAGE_SIZE の値（1 メガバイト）ではなく，rpc_max_message_size オペランドに指定した値になります。

(1)dc_rpc_poll_any_replies 関数の引数 timeout について

非同期受信の監視時間は，応答が返るたびにリセットされます。そのため，受信する非同期応答を特定（flags に DCRPC_SPECIFIC_MSG を設定）した場合は，timeout に設定した時間を経過しても，応答を受信できる場合があります。また，timeout に設定した時間を経過しても，DCRPCER_TIMED_OUT でエラーリターンしない場合もあります。

(2)dc_rpc_poll_any_replies 関数がエラーになるタイミング

サービスを要求された SPP が異常終了した場合，クライアント UAP でエラーが返るタイミングについて説明します。

サービスを実行する SPP が処理の終わる前に異常終了すると，dc_rpc_poll_any_replies 関数は DCRPCER_TIMED_OUT でエラーリターンします。dc_rpc_poll_any_replies 関数の引数 timeout に-1 を設定している場合は，DCRPCER_SERVICE_NOT_UP でエラーリターンします。

dc_rpc_poll_any_replies 関数の時間監視でエラーになる場合

次に示す場合には，dc_rpc_poll_any_replies 関数の引数 timeout に設定した時間が経過したあとで，DCRPCER_TIMED_OUT でエラーリターンします。

- SPP があるノードの OpenTP1 全体が異常終了した場合
- サービス要求のデータがサーバ UAP に届く前，または サーバ UAP の処理が完了してからクライアント UAP に結果が届く前に障害が起こった場合

(3)リターン値 DCRPCER_SERVICE_TERMINATED をリターンさせる指定

サービスを要求された SPP が，処理を完了する前に異常終了したことを DCRPCER_TIMED_OUT または DCRPCER_SERVICE_NOT_UP 以外のリターン値で判別したい場合には，ユーザサービス定義の rpc_extend_function オペランドに"00000001"を指定します。この指定をすると，上記のエラー時に DCRPCER_SERVICE_TERMINATED がリターンされるようになります。rpc_extend_function オペランドに"00000000"を指定するか，またはオペランドを省略した場合は，DCRPCER_SERVICE_TERMINATED は返らないで，DCRPCER_TIMED_OUT または DCRPCER_SERVICE_NOT_UP がリターンされます。

(4)エラーリターン値と同期点処理の関係

dc_rpc_poll_any_replies 関数のリターン値と同期点処理（コミット，ロールバック）の関係について説明します。ここで説明する内容は，サービス要求がトランザクション処理になる場合に該当します。トランザクションでないサービス要求（dc_rpc_call 関数の flags に DCRPC_TPNOTRAN を設定した場合も含む）には該当しません。

dc_rpc_poll_any_replies 関数がエラーリターンしてもコミットとなる場合

サービスを要求されたサービス関数の異常終了や，ノードの障害，ネットワーク障害の場合でも，DCRPCER_TIMED_OUT がリターンすることがあります。クライアント UAP がトランザクション処理でない場合は，DCRPCER_TIMED_OUT が返っても，要求したサービスがあるサービス関数は正常終了していて，データベースへの更新などが実行されているときもあります。

ロールバック処理が必要なエラーリターン値

トランザクション処理から呼び出した dc_rpc_poll_any_replies 関数がエラーリターンした場合，リターン値によっては，必ずトランザクションがロールバック（サーバ UAP が rollback_only 状態）になります。この場合，コミットの関数，またはロールバックの関数のどちらを使っても，必ずロールバックになります。必ずロールバックになる dc_rpc_poll_any_replies 関数のリターン値を次に示します。

- DCRPCER_INVALID_REPLY
- DCRPCER_NO_BUFS_AT_SERVER
- DCRPCER_NO_SUCH_SERVICE
- DCRPCER_REPLY_TOO_BIG_RB

(5)dc_rpc_poll_any_replies 関数で応答が受け取れない場合

非同期応答型 RPC でサービスを要求した UAP が次に示す関数を呼び出すと，dc_rpc_poll_any_replies 関数で応答を受け取れません。

1. dc_rpc_discard_further_replies 関数で，非同期応答の受信を拒否した場合
2. トランザクションの処理の場合，同期点処理の関数でコミット またはロールバックした場合

上記の関数を使ったあとで返ってきた応答は，破棄されます。非同期応答型 RPC では，上記の関数を呼び出す前に，必要な非同期の応答を dc_rpc_poll_any_replies 関数ですべて受け取ってください。

(6)dc_rpc_poll_any_replies 関数を使うときの注意

1. dc_rpc_poll_any_replies 関数を待ち時間 0 と設定（引数 timeout に 0 を設定）して呼び出すと，マルチスレッド環境のスケジューリングの関係で，応答が到着していても受信できない場合があります。そのため，すべての応答を受信するまで，待ち時間に 0 を設定した dc_rpc_poll_any_replies 関数を呼び出す UAP は，無限ループになることがあるので注意してください。
2. 記述子を特定しない dc_rpc_poll_any_replies 関数がエラーリターンした場合，エラーとなった応答の記述子を特定できません。dc_rpc_poll_any_replies 関数がエラーリターンしたときに該当する記述子がわかるようにしておきたい場合は，flags に DCRPC_SPECIFIC_MSG を設定しておいてください。

dc_rpc_service_retry

名称

サービス関数のリトライ

形式

ANSI C，C++の形式

```
#include <dcrpc.h>
int  dc_rpc_service_retry(void)
```

K&R 版 C の形式

```
#include <dcrpc.h>
int  dc_rpc_service_retry()
```

機能

実行中のサービス関数の処理をリトライします。リトライする場合は，サービス関数で dc_rpc_service_retry 関数を呼び出したあとで，リトライするサービス関数をリターンしてください。リターンしたあと，同じプロセスで同じサービス関数が再起動されます。

応答型 RPC で呼ばれたサービス関数がリトライされた場合は，リトライ前のサービス関数が設定した値（応答を格納する領域と応答の長さ）は無効になります。

ユーザサービス定義の rpc_service_retry_count オペランドに指定した回数を超えた（rpc_service_retry_count オペランドに 0 を指定した場合も含む）あとで dc_rpc_service_retry 関数を呼び出した場合，関数は DCRPCER_RETRY_COUNT_OVER でエラーリターンします。このとき，サービス関数はリトライされません。応答型 RPC で呼ばれたサービス関数の場合は，応答を格納する領域の内容をクライアント UAP に返します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCRPCER_PROTO	-302	dc_rpc_service_retry 関数を呼び出す条件が間違っています。次に示すことが考えられます。 • サービス関数の中で呼び出していません。 • グローバルトランザクションの範囲の中で呼び出しています。
DCRPCER_RETRY_COUNT_OVER	-377	ユーザサービス定義の rpc_service_retry_count オペランドに指定したサービスリトライ回数最大値を超えて，

リターン値	リターン値（数値）	意味
DCRPCER_RETRY_COUNT_OVER	-377	dc_rpc_service_retry 関数を呼び出しています。これ以上サービス関数をリトライできません。

注意事項

1. dc_rpc_service_retry 関数を呼び出す場合は，次に示す条件を満たしてください。この条件を満たしていない場合，dc_rpc_service_retry 関数はエラーリターンします。
   - サービス関数の中で dc_rpc_service_retry 関数を呼び出していること。
   - 実行中のサービス関数が，グローバルトランザクションの範囲でないこと。
2. dc_rpc_service_retry 関数を呼び出すサービス関数では，クライアント UAP から渡されたデータは参照できますが，変更できません。入力データ領域の内容を変更した場合，システムの動作は保証しません。
3. dc_rpc_service_retry 関数は，OpenTP1 独自のリモートプロシジャコール（dc_rpc_call 関数）でサービスを要求されたサービス関数でだけ呼び出せます。それ以外のサービス関数の処理は，dc_rpc_service_retry 関数でリトライできません。

dc_rpc_set_service_prio

名称

サービス要求のスケジュールプライオリティの設定

形式

ANSI C，C++の形式

```
#include <dcrpc.h>
void  dc_rpc_set_service_prio(DCLONG prio)
```

K&R 版 C の形式

```
#include <dcrpc.h>
void  dc_rpc_set_service_prio(prio)
DCLONG      prio;
```

機能

サービス要求のプライオリティを設定します。サービス要求単位でスケジュールプライオリティを制御する場合に呼び出します。この関数で設定したプライオリティは，この関数を再び呼び出すまで更新されません。したがって，同じプライオリティでまとめてサービス要求する場合は，この関数を 1 回だけ呼び出します。

この関数で指定したプライオリティは，直後に呼び出す dc_rpc_call 関数，および dc_rpc_call_to 関数で，スケジュールキューを経由してサーバに通知されます。

この関数を一度も呼び出さない場合の処理は，スケジュールサービスの省略時解釈値である 4 が，サービス要求のプライオリティとして指定されます。

UAP で値を設定する引数

●prio

サービス要求のスケジュールプライオリティを，0 または 1 から 8 の範囲で設定します。prio の設定は省略できません。

最も高いプライオリティの値は 1 で，最も低いプライオリティの値は 8 です。

0 を設定した場合は，スケジュールサービスの省略時解釈となります。

上記以外の値を設定した場合は，dc_rpc_set_service_prio 関数は無視されます。

リターン値

dc_rpc_set_service_prio 関数のリターン値はありません。

注意事項

1. キュー受信型サーバでは，設定したサービス要求のプライオリティは，サーバ UAP のユーザサービス定義に，service_priority_control=Y（プライオリティを制御する）を指定している場合だけ有効です。サービス要求する相手のサーバ UAP でプライオリティを制御していない場合は，この関数を呼び出しても無効になります。
2. 2 回目以降の連鎖 RPC での dc_rpc_call 関数，および dc_rpc_call_to 関数と，連鎖 RPC を終了させるために呼び出す同期応答型 RPC の dc_rpc_call 関数（flags に DCNOFLAGS を設定），および dc_rpc_call_to 関数（flags に DCNOFLAGS を設定）のサービス要求に対して dc_rpc_set_service_prio 関数を呼び出しても無効となります。
3. dc_rpc_call 関数，および dc_rpc_call_to 関数は，サービス要求のプライオリティを省略値にリセットしません。サービス要求のプライオリティをリセットする場合は，引数 prio に 0 を設定した dc_rpc_set_service_prio 関数を呼び出し直してください。
4. この関数は，リモート API 機能では使用できません。rap クライアントで発行する dc_rpc_call 関数，および dc_rpc_call_to 関数のサービス要求に対して，dc_rpc_set_service_prio 関数を呼び出しても無効になります。

使用例

```
int     rc;
DCULONG in_len,len;
char    *buf;

/* サービス要求1回目:
 * プライオリティの指定はなし(スケジュールサービスの省略時解釈)
 */
rc = dc_rpc_call("SPPG","ECHO","ex1",&in_len,buf,&len,DCNOFLAGS);

/* サービス要求2回目:プライオリティ = 8
 */
dc_rpc_set_service_prio(8);
rc = dc_rpc_call("SPPG","ECHO","ex2",&in_len,buf,&len,DCNOFLAGS);

/* サービス要求3回目(連鎖RPC):プライオリティ = 1
 */
dc_rpc_set_service_prio(1);
rc = dc_rpc_call("SPPG","ECHO","ex3",&in_len,buf,&len,DCRPC_CHAINED);
   :
(連鎖RPC dc_rpc_call(DCRPC_CHAINED) n回繰り返し)
   :
rc = dc_rpc_call("SPPG","ECHO","ex3",&in_len,buf,&len,DCNOFLAGS);

/* サービス要求(4+n+1)回目(以降):
 * プライオリティをリセット(スケジュールサービスの省略時解釈)
 */
dc_rpc_set_service_prio(0);
rc = dc_rpc_call("SPPG","ECHO","ex4",&in_len,buf,&len,DCRPC_NOREPLY);
```

dc_rpc_set_watch_time

名称

サービスの応答待ち時間の更新

形式

ANSI C，C++の形式

```
#include <dcrpc.h>
int  dc_rpc_set_watch_time(int var)
```

K&R 版 C の形式

```
#include <dcrpc.h>
int  dc_rpc_set_watch_time(var)
int       var;
```

機能

サービス要求の応答待ち時間を変更します。この関数で変更した値は，dc_rpc_close 関数を呼び出すまで有効です。

サービス要求の応答待ち時間をこの関数を呼び出す前の値に戻すときは，dc_rpc_get_watch_time 関数で返された元の値を，この関数で再設定してください。

この関数を呼び出しても，システム共通定義の watch_time オペランドに指定した値は変更しません。この関数で設定する値は，あとから呼び出す dc_rpc_call 関数にだけ影響します。

UAP で値を設定する引数

●var

変更後のサービス応答待ち時間を設定します。1～65535 の範囲で設定します。無制限に待ち続ける場合は，0 を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCRPCER_INVALID_ARGS	-301	var に設定した値が間違っています。
上記以外		プログラムの破壊などによる，予期しないエラーが起こりました。

リアルタイム統計情報サービス（dc_rts_～）

リアルタイム統計情報サービスの関数について説明します。リアルタイム統計情報サービスの関数を次に示します。

- dc_rts_utrace_put － 任意区間でのリアルタイム統計情報の取得

dc_rts_utrace_put

名称

任意区間でのリアルタイム統計情報の取得

形式

K&R 版 C の形式

```
#include <dcrts.h>
int dc_rts_utrace_put(event_id, flags);
DCLONG  event_id;
DCLONG  flags;
```

機能

UAP 内の任意の区間で，event_id に設定した項目の実行時間および実行回数を，リアルタイム統計情報として取得します。

UAP で値を設定する引数

●event_id

取得するリアルタイム統計情報の項目 ID を設定します。

ID に使用できる値は，1000000～2147483647 です。

●flags

dc_rts_utrace_put 関数で実行する処理を設定します。

DCRTS_START

event_id に設定した項目 ID の実行時間の計測を開始します。

このフラグを設定して dc_rts_utrace_put 関数を呼び出した時点では，リアルタイム統計情報を取得しません。

DCRTS_END

event_id に設定した項目 ID の実行時間を取得して，計測を終了します。

DCNOFLAGS

event_id に設定した項目 ID の実行回数だけを取得します。実行時間は 0 秒となります。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCRTSER_PARAM	-7802	引数に設定した値に誤りがあります。
DCRTSER_PROTO	-7803	dc_rpc_open 関数を呼び出していません。
		すでに実行時間の計測を開始している項目 ID を event_id に設定して，flags に DCRTS_START を設定した dc_rts_utrace_put 関数を呼び出しました。
		実行時間の計測を開始していない項目 ID を event_id に設定して，flags に DCRTS_END を設定した dc_rts_utrace_put 関数を呼び出しました。
DCRTSER_ITEM_OVER	-7804	取得項目の数がリアルタイム統計情報サービス定義の rts_item_max オペランドの指定値を超えるため，情報を取得できません。
DCRTSER_ITEM_OVER_SRV	-7805	サーバ単位の取得項目の数が，リアルタイム統計情報サービス定義の rts_item_max オペランドの指定値を超えるため，情報を取得できません。このリターン値が返った場合，サービス単位またはサービス以外の処理の統計情報は取得しています。
DCRTSER_ITEM_OVER_SVC	-7806	サービス単位またはサービス以外の処理での取得項目の数が，リアルタイム統計情報サービス定義の rts_item_max オペランドの指定値を超えるため，情報を取得できません。このリターン値が返った場合，サーバ単位の統計情報は取得しています。
DCRTSER_NOMEM	-7807	プロセスメモリが不足したため，処理を実行できません。
DCRTSER_RTS_NOT_START	-7808	リアルタイム統計情報サービスが開始していません。
DCRTSER_NOENTRY	-7809	dc_rts_utrace_put 関数の呼び出し元が，サーバ単位およびサービス単位でリアルタイム統計情報の取得対象に登録されていません。
DCRTSER_VERSION	-7810	UAP が，現在稼働しているリアルタイム統計情報サービスでは稼働できないバージョンのライブラリと結合しています。

注意事項

1. dc_rts_utrace_put 関数では，システム全体のリアルタイム統計情報は取得できません。
2. マルチサーバを使用している UAP では，同じ呼び出し元サービスおよび同じ event_id を設定した dc_rts_utrace_put 関数を複数プロセスから同時に呼び出した場合，プロセスによっては統計情報が取得されないことがあります。これは，統計情報の取得処理では排他制御がされないため，書き込み処理が同時に行われることが要因です。
3. XATMI インタフェースを使用している UAP では，サービス単位のリアルタイム統計情報は取得できません。すべてサービス以外の処理の統計情報として取得されます。

4. dc_rts_utrace_put 関数は，UAP トレースを取得しません。

5. flags に DCRTS_START を指定した dc_rts_utrace_put 関数が，DCRTSER_RTS_NOT_START または DCRTSER_NOENTRY でリターンしたあとで，同じ event_id で flags に DCRTS_END を指定した dc_rts_utrace_put 関数を呼び出すまでにリアルタイム統計情報サービスを開始して呼び出し元の UAP を取得対象に追加した場合，dc_rts_utrace_put 関数は，DCRTSER_PROTO でリターンします。

TAM ファイルサービス（dc_tam_～）

TAM ファイルサービスの関数について説明します。TAM ファイルサービスの関数を次に示します。

- dc_tam_close － TAM テーブルのクローズ
- dc_tam_delete － TAM テーブルのレコードの削除
- dc_tam_get_inf － TAM テーブルの状態の取得
- dc_tam_open － TAM テーブルのオープン
- dc_tam_read － TAM テーブルからレコードの入力
- dc_tam_read_cancel － TAM テーブルのレコードの入力取り消し
- dc_tam_rewrite － TAM テーブルのレコード入力を前提の更新
- dc_tam_status － TAM テーブルの情報の取得
- dc_tam_write － TAM テーブルのレコードの更新／追加

TAM ファイルサービスの関数（dc_tam_～）は，TP1/Server Base の UAP でだけ使えます。TP1/LiNK の UAP では，TAM ファイルサービスの関数は使えません。

dc_tam_close

名称

TAM テーブルのクローズ

形式

ANSI C，C++の形式

```
#include <dctam.h>
int  dc_tam_close(DCLONG tblid,DCLONG flags)
```

K&R 版 C の形式

```
#include <dctam.h>
int dc_tam_close(tblid,flags)
DCLONG   tblid;
DCLONG   flags;
```

機能

TAM テーブルをクローズします。dc_tam_close 関数を呼び出したあとは，tblid に設定したテーブル記述子は使えません。

dc_tam_close 関数がエラーリターンした場合は，この関数内で確保した資源はすべて解放して，関数を呼び出す前の状態に戻ります。

トランザクション外で dc_tam_open 関数を呼び出した場合，dc_tam_close 関数はトランザクション外で呼び出してください。

トランザクション内で dc_tam_open 関数を呼び出した場合は，dc_tam_close 関数はトランザクション内で呼び出してください。また，トランザクション終了時までに dc_tam_close 関数を呼び出さなかった場合は，同期点で TAM テーブルがクローズされます。

サービス関数の中で，トランザクション外のオープンに対する dc_tam_close 関数を呼び出す場合，クローズさせる TAM テーブルにアクセスしている同一プロセス上のトランザクションは，すべて終了させてください。このことに関するエラーチェックはしないので，終了しないでこの関数を呼び出した場合の動作については保証しません。

UAP で値を設定する引数

●tblid

クローズする TAM テーブルの，テーブル記述子を設定します。テーブル記述子は，dc_tam_open 関数で返された値です。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	TAM テーブルを正常にクローズしました。
DCTAMER_PARAM_TID	-1700	tblid に設定したテーブル記述子が間違っています。
DCTAMER_PARAM_FLG	-1708	flags に設定した値が間違っています。
DCTAMER_TAMEND	-1720	TAM サービスが終了中です。
DCTAMER_PROTO	-1721	TAM テーブルへアクセスする順序が間違っています。
		UAP にリンケージしているトランザクション制御用オブジェクトファイルのリソースマネジャ登録が間違っています。 または，UAP にトランザクション制御用オブジェクトファイルをリンケージしていません。
		関数を呼び出した UAP のユーザサービス定義に，トランザクション属性なし（atomic_update=N）を指定しています。
DCTAMER_TRNOPN	-1722	dc_tam_open 関数はトランザクション外で呼び出しています。
DCTAMER_NOOPEN	-1726	TAM テーブルがオープン状態ではありません。
DCTAMER_MEMORY	-1769	メモリが不足しました。
DCTAMER_IO	-1770	入出力エラーが起こりました。

dc_tam_delete

名称

TAM テーブルのレコードの削除

形式

ANSI C，C++の形式

```
#include <dctam.h>
int  dc_tam_delete(DCLONG tblid,struct DC_TAMKEY *keyadr,
                   int keyno,
                   char *bufadr,int bufsize,DCLONG flags)
```

K&R 版 C の形式

```
#include <dctam.h>
int dc_tam_delete(tblid,keyadr,keyno,bufadr,bufsize,flags)
DCLONG           tblid;
struct DC_TAMKEY *keyadr;
int              keyno;
char             *bufadr;
int              bufsize;
DCLONG           flags;
```

機能

キー値に示すレコードを，TAM テーブルから削除します。削除するレコードをバッファに退避することもできます。ただし，この関数がエラーリターンした場合には，バッファの内容は保証できません。

レコード排他で TAM テーブルがオープンしている場合，更新排他でテーブル排他を確保します。

dc_tam_delete 関数がエラーリターンした場合は，この関数内で設定した資源はすべて解放して，関数を呼び出す前の状態に戻ります。ただし，関数を呼び出す前に参照排他で確保されていた TAM テーブルを削除しようとした場合は更新排他となり，参照排他には戻りません。

複数のレコードを設定して削除する場合，それらのレコードのうち一つでもエラーが起こったときは，この関数で設定した全レコードの処理をエラーとして，関数を呼び出す前の状態に戻ります。

UAP で値を設定する引数

●tblid

レコードを削除する TAM テーブルの，テーブル記述子を設定します。テーブル記述子は，dc_tam_open 関数で返された値です。

●keyadr

削除するレコードのキー値のアドレスを持つ，構造体のアドレスを設定します。構造体の形式は次のとおりです。

```
struct DC_TAMKEY {
                   char  *keyname;
                  };
```

- keyname

  キー値のアドレスを設定します。キー値は，削除するレコードのキー領域の長さで設定してください。

●keyno

要求レコード数（keyadr で設定する構造体の数）を設定します。

●bufadr

削除するレコードをバッファに退避する場合に，そのバッファのアドレスを設定します。flags に DCTAM_NOOUTREC（削除するレコードを退避しない）を設定した場合は，この設定は無効です。

●bufsize

削除するレコードをバッファに退避する場合に，そのバッファ長を設定します。返却バッファ長は，（レコード長×要求レコード数）以上にします。flags に DCTAM_NOOUTREC（削除するレコードを退避しない）を設定した場合は，この設定は無効です。

●flags

レコードのアクセス種別，資源の競合が起こった場合の排他解除待ち種別を，次の形式で設定します。

```
{DCTAM_NOOUTREC|DCTAM_OUTREC} [|{DCTAM_WAIT|DCTAM_NOWAIT}]
```

- フラグ 1

  レコードのアクセス種別の設定は省略できません。アクセス種別は重複して設定できません。

  DCTAM_NOOUTREC … 削除するレコードを退避しない。

  DCTAM_OUTREC … 削除するレコードを退避する。

- フラグ 2

  排他解除待ち種別を省略した場合は，排他解除待ちをしないでエラーリターンします。排他解除待ち種別は重複して設定できません。

  DCTAM_WAIT… 排他解除待ちをします。

  DCTAM_NOWAIT… 待たないで，エラーリターンします。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	TAM テーブルのレコードを正常に削除しました。
DCTAMER_PARAM_TID	-1700	tblid に設定したテーブル記述子が間違っています。
DCTAMER_PARAM_KEY	-1702	keyadr に設定したキー値が間違っています。
DCTAMER_PARAM_KNO	-1703	keyno に設定した値が間違っています。
DCTAMER_PARAM_BFA	-1704	bufadr に設定した値が間違っています。
DCTAMER_PARAM_BFS	-1705	bufsize に設定したバッファ長が短過ぎます。
DCTAMER_PARAM_FLG	-1708	flags に設定した値が間違っています。
DCTAMER_NOTTAM	-1709	tblid に設定したテーブルは TAM テーブルではありません。
DCTAMER_TAMEND	-1720	TAM サービスが終了中です。
DCTAMER_PROTO	-1721	TAM テーブルへアクセスする順序が間違っています。
		UAP にリンケージしているトランザクション制御用オブジェクトファイルのリソースマネジャ登録が間違っています。 または，UAP にトランザクション制御用オブジェクトファイルをリンケージしていません。
		関数を呼び出した UAP のユーザサービス定義に，トランザクション属性なし（atomic_update=N）を指定しています。
DCTAMER_RMTBL	-1723	TAM テーブルが削除されています。
DCTAMER_NOLOAD	-1724	TAM テーブルがロードされていません。
DCTAMER_NOOPEN	-1726	TAM テーブルがオープン状態ではありません。
DCTAMER_LOGHLD	-1727	TAM テーブルが論理閉塞状態です。
DCTAMER_OBSHLD	-1728	TAM テーブルが障害閉塞状態です。
DCTAMER_ACSATL	-1730	TAM サービス定義で指定した TAM テーブルのアクセス形態では実行できません。
DCTAMER_NOREC	-1731	指定されたレコードは存在しません。
DCTAMER_LOCK	-1736	排他エラーが起こりました。flags に DCTAM_WAIT を設定した場合，ロックサービス定義で指定した待ち時間のタイムアウトのため，資源を確保できませんでした。
DCTAMER_DLOCK	-1737	デッドロックが起こりました。
DCTAMER_TBLVR	-1760	UAP が，現在稼働している TAM テーブルでは動作できないバージョンの TAM ライブラリと結合されています。

リターン値	リターン値（数値）	意味
DCTAMER_FLSVR	-1761	UAP が，現在稼働している OpenTP1 ファイルサービスでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_RECOBS	-1764	レコードが破壊されています。
DCTAMER_TRNNUM	-1765	TAM サービスで管理できるトランザクション数を超えました。
DCTAMER_OPENNUM	-1766	キャラクタ型スペシャルファイルのオープン数の制限値を超えました。
DCTAMER_ACCESSS	-1767	スペシャルファイルに対するアクセス権がありません。
DCTAMER_ACCESSF	-1768	TAM ファイルに対するアクセス権がありません。
DCTAMER_MEMORY	-1769	メモリが不足しました。
DCTAMER_IO	-1770	入出力エラーが起こりました。
DCTAMER_TMERR	-1771	トランザクションサービスでエラーが起こりました。
DCTAMER_ACCESS	-1773	アクセスしようとした TAM ファイルは，セキュリティ機能で保護されています。dc_tam_delete 関数を呼び出した UAP には，アクセス権限がありません。

注意事項

ハッシュ形式の TAM テーブルに格納されているレコードを全件削除する場合，次の手順で行ってください。

1. 先頭検索して見つかったレコードのキー値を変数 1 に退避する
2. 変数 1 のキー値を使い，NEXT 検索する
3. 手順 2.で見つかったレコードのキー値を変数 2 に退避する
4. 退避しておいた変数 1 のキー値のレコードを削除する
5. 変数 2 のキー値を変数 1 に退避する
6. 手順 2.がエラーになるまで手順 2.から手順 5.を繰り返す（NEXT 検索する）
7. 手順 2.がエラーになったあとで，最後に変数 1 に退避したキー値のレコードを削除する

レコードを全件削除する場合，次に示す手順を実行すると CPU が高負荷になることがありますので注意してください。

1. レコードを先頭検索する
2. 手順 1.で見つかったレコードを削除する
3. 手順 1.と手順 2.すなわちレコードの先頭検索とレコードの削除を繰り返す

dc_tam_get_inf

名称

TAM テーブルの状態の取得

形式

ANSI C，C++の形式

```
#include <dctam.h>
int  dc_tam_get_inf(char *tblname,DCLONG flags)
```

K&R 版 C の形式

```
#include <dctam.h>
int dc_tam_get_inf(tblname,flags)
char      *tblname;
DCLONG    flags;
```

機能

TAM テーブルの状態を取得します。取得する TAM テーブルの状態を次に示します。

- オープン状態
- クローズ状態
- 論理閉塞状態
- 障害閉塞状態

dc_tam_get_inf 関数は，トランザクション内でもトランザクション外でも呼び出せます。

dc_tam_get_inf 関数を呼び出したプロセスで dc_tam_open 関数を呼び出していなくても，設定した TAM テーブルにほかのプロセスで dc_tam_open 関数を呼び出している場合は，TAM テーブルはオープン状態としてリターンします。

UAP で値を設定する引数

●tblname

状態を取得する TAM テーブル名のアドレスを設定します。TAM テーブル名は 32 文字以内で設定して，文字列の最後にはヌル文字を付けてください。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値が正の値のとき（TAM テーブルの状態を示します）

リターン値	リターン値（数値）	意味
DCTAM_STS_OPN	1	オープン状態であることを示します。
DCTAM_STS_CLS	2	クローズ状態であることを示します。
DCTAM_STS_LHLD	3	論理閉塞状態であることを示します。
DCTAM_STS_OHLD	4	障害閉塞状態であることを示します。

リターン値が負の値のとき（エラーが起こったことを示します）

リターン値	リターン値（数値）	意味
DCTAMER_PARAM_TBL	-1701	tblname に設定した値が間違っています。
DCTAMER_PARAM_FLG	-1708	flags に設定した値が間違っています。
DCTAMER_UNDEF	-1710	TAM テーブルが定義されていません。
DCTAMER_TAMEND	-1720	TAM サービスが終了中です。
DCTAMER_PROTO	-1721	TAM テーブルへアクセスする順序が間違っています。
		UAP にリンケージしているトランザクション制御用オブジェクトファイルのリソースマネジャ登録が間違っています。 または，UAP にトランザクション制御用オブジェクトファイルをリンケージしていません。
		関数を呼び出した UAP のユーザサービス定義に，トランザクション属性なし（atomic_update=N）を指定しています。
DOTAMER_TAMVR	-1762	UAP が，現在稼働している TAM サービスでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_NO_ACL	-1772	アクセスしようとした TAM ファイルは，セキュリティ機能で保護されています。該当するファイルに対する ACL がありません。
DCTAMER_ACCESS	-1773	アクセスしようとした TAM ファイルは，セキュリティ機能で保護されています。dc_tam_get_inf 関数を呼び出した UAP には，アクセス権限がありません。

dc_tam_open

名称

TAM テーブルのオープン

形式

ANSI C，C++の形式

```
#include <dctam.h>
DCLONG  dc_tam_open(char *tblname,DCLONG flags)
```

K&R 版 C の形式

```
#include <dctam.h>
DCLONG  dc_tam_open(tblname,flags)
char        *tblname;
DCLONG      flags;
```

機能

TAM テーブルをオープンします。dc_tam_open 関数は，トランザクション内でもトランザクション外でも呼び出せます。

トランザクション内で呼び出して，排他種別にテーブル排他を設定した場合，更新排他でテーブル排他を確保します。

dc_tam_open 関数がエラーリターンした場合は，この関数内で確保した資源はすべて解放して，関数を呼び出す前の状態に戻ります。

UAP で値を設定する引数

●tblname

オープンする TAM テーブル名を設定します。TAM テーブル名は 32 文字以内で設定して，文字列の最後にはヌル文字を付けてください。

●flags

テーブル排他を掛けるかレコード排他を掛けるかを，次の形式で設定します。

```
〔{DCTAM_TBL_EXCLUSIVE〔|{DCTAM_WAIT|DCTAM_NOWAIT}〕|DCTAM_REC_EXCLUSIVE}〕
```

- フラグ 1

  テーブル排他の場合は，更新排他で確保します。レコード排他の場合は，レコードのアクセス関数内で排他を確保します。

排他解除待ち種別は重複して設定できません。dc_tam_open 関数をトランザクション外で呼び出す場合は，テーブル排他は設定できません。

このフラグの設定を省略した場合は，DCTAM_REC_EXCLUSIVE が仮定されます。

DCTAM_TBL_EXCLUSIVE…テーブル排他

DCTAM_REC_EXCLUSIVE…レコード排他

- フラグ 2

  テーブル排他の場合は，資源の競合が起こったときの排他解除待ち種別を設定します。排他解除種別は重複して設定できません。

  このフラグの設定を省略した場合は，DCTAM_NOWAIT が仮定されます。

  DCTAM_WAIT…排他解除待ちをします。

  DCTAM_NOWAIT…待たないで，エラーリターンします。

flags に設定する値と，設定内容の関係について，次に示します。

フラグ 1※1	フラグ 2※2	flags の設定内容
TBL_EXCLUSIVE	WAIT	テーブル排他，排他エラー時は解除待ち
	NOWAIT	テーブル排他，排他エラー時はエラーリターン
REC_EXCLUSIVE	×	レコード排他

（凡例）

×：設定できません。

注※1

省略した場合は，REC_EXCLUSIVE が仮定されます。

注※2

省略した場合は，NOWAIT が仮定されます。

リターン値

リターン値	リターン値（数値）	意味
正の整数		テーブル記述子を示します。
DCTAMER_PARAM_TBL	-1701	tblname に設定した値が間違っています。
DCTAMER_PARAM_FLG	-1708	flags に設定した値が間違っています。
DCTAMER_NOTTAM	-1709	tblname に設定したテーブルは TAM テーブルではありません。
DCTAMER_UNDEF	-1710	TAM テーブルが定義されていません。
DCTAMER_TAMEND	-1720	TAM サービスが終了中です。
DCTAMER_PROTO	-1721	TAM テーブルへアクセスする順序が間違っています。

リターン値	リターン値（数値）	意味
DCTAMER_PROTO	-1721	UAP にリンケージしているトランザクション制御用オブジェクトファイルのリソースマネジャ登録が間違っています。 または，UAP にトランザクション制御用オブジェクトファイルをリンケージしていません。
		関数を呼び出した UAP のユーザサービス定義に，トランザクション属性なし（atomic_update=N）を指定しています。
DCTAMER_NOLOAD	-1724	TAM テーブルがロードされていません。
DCTAMER_OPENED	-1725	TAM テーブルがオープン済みです。
DCTAMER_LOGHLD	-1727	TAM テーブルが論理閉塞状態です。
DCTAMER_OBSHLD	-1728	TAM テーブルが障害閉塞状態です。
DCTAMER_LOCK	-1736	排他エラーが起こりました。flags に DCTAM_WAIT を設定した場合，ロックサービス定義で指定した待ち時間のタイムアウトのため，資源を確保できませんでした。
DCTAMER_DLOCK	-1737	デッドロックが起こりました。
DCTAMER_TBLVR	-1760	UAP が，現在稼働している TAM テーブルでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_FLSVR	-1761	UAP が，現在稼働している OpenTP1 ファイルサービスでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_TAMVR	-1762	UAP が，現在稼働している TAM サービスでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_RECOBS	-1764	レコードが破壊されています。
DCTAMER_TRNNUM	-1765	TAM サービスで管理できるトランザクション数を超えています。
DCTAMER_OPENNUM	-1766	キャラクタ型スペシャルファイルのオープン数の制限値を超えています。
DCTAMER_ACCESSS	-1767	スペシャルファイルに対するアクセス権がありません。
DCTAMER_ACCESSF	-1768	TAM ファイルに対するアクセス権がありません。
DCTAMER_MEMORY	-1769	メモリが不足しました。
DCTAMER_IO	-1770	入出力エラーが起こりました。
DCTAMER_TMERR	-1771	トランザクションサービスでエラーが起こりました。
DCTAMER_NO_ACL	-1772	オープンしようとした TAM ファイルは，セキュリティ機能で保護されています。該当するファイルに対する ACL がありません。

dc_tam_read

名称

TAM テーブルからレコードの入力

形式

ANSI C，C++の形式

```
#include <dctam.h>
int  dc_tam_read(DCLONG tblid,struct DC_TAMKEY *keyadr,
                 int keyno,
                 char *bufadr,int bufsize,DCLONG flags)
```

K&R 版 C の形式

```
#include <dctam.h>
int dc_tam_read(tblid,keyadr,keyno,bufadr,bufsize,flags)
DCLONG           tblid;
struct DC_TAMKEY *keyadr;
int              keyno;
char             *bufadr;
int              bufsize;
DCLONG           flags;
```

機能

flags に設定した検索種別に従って，TAM テーブル上のレコードを参照または更新の目的で入力します。検索種別とインデクス種別の関係を次の表に示します。

表 2-3　検索種別とインデクス種別の関係

検索種別	検索処理の概要	
	インデクス種別：ハッシュ	インデクス種別：ツリー
'キー値＝'検索	設定したキー値を持つレコードを検索します。 設定したキー値を持つレコードがない場合はエラーリターンします。	設定したキー値を持つレコードを検索します。 設定したキー値を持つレコードがない場合はエラーリターンします。
'キー値＜＝'検索	エラーリターンします。	設定したキー値以上のキー値を持つレコードを検索します。
'キー値＜'検索	エラーリターンします。	設定したキー値より大きいキー値を持つレコードを検索します。
'キー値＞＝'検索	エラーリターンします。	設定したキー値以下のキー値を持つレコードを検索します。
'キー値＞'検索	エラーリターンします。	設定したキー値より小さいキー値を持つレコードを検索します。

検索種別	検索処理の概要	
	インデクス種別：ハッシュ	インデクス種別：ツリー
先頭検索※	キー値に対応してハッシングをした先頭レコードを検索します。keyadr に設定したキー値は無視します。	エラーリターンします。
NEXT 検索※	キー値に対応してハッシングをした，次のレコードを検索します。	エラーリターンします。

注※

インデクス種別がハッシュで，TAM テーブルファイルの初期作成時に，データ部にキー値を付けている（tamcre コマンドに-s オプションを設定していない）場合，先頭検索と NEXT 検索を使って，TAM テーブル上の全レコードを検索できます。

参照目的の入力で排他を掛ける場合，参照排他でテーブル排他とレコード排他を確保します。レコード排他でオープンした TAM テーブルを更新目的で入力する場合は，参照排他でテーブル排他を確保し，更新排他でレコード排他を確保します。

dc_tam_read 関数がエラーリターンした場合は，この関数内で設定した資源はすべて解放して，関数を呼び出す前の状態に戻ります。ただし，関数を呼び出す前に参照排他で確保されていたレコードを更新目的で入力した場合は，更新排他となり参照排他には戻りません。また，エラーリターンした場合には，バッファの内容は保証できません。

複数のレコードを設定して入力する場合，それらのレコードのうち一つでもエラーが発生したときは，この関数で設定した全レコードの処理をエラーとします。

UAP で値を設定する引数

●tblid

レコードを入力する TAM テーブルの，テーブル記述子を設定します。テーブル記述子は dc_tam_open 関数で返された値です。

●keyadr

レコードを検索するための，キー値のアドレスを持つ構造体のアドレスを設定します。構造体の形式は次のとおりです。

```
struct DC_TAMKEY {
                  char  *keyname;
                  };
```

- keyname

  キー値のアドレスを設定します。キー値は入力するレコードのキー領域の長さで設定します。

●keyno

要求レコード数（keyadr で設定する構造体の数）を設定します。

●bufadr

レコードを入力するバッファのアドレスを設定します。

●bufsize

レコードを入力するバッファの長さを設定します。バッファ長は（レコード長×要求レコード数）以上にします。

●flags

レコードの検索種別，アクセス種別，参照目的の排他のときの排他要否種別を設定します。また，排他を掛ける場合に，資源の競合が起こったときの排他解除待ち種別を設定します。

```
{フラグ1}|{DCTAM_REFERENCE〔|{DCTAM_EXCLUSIVE|DCTAM_NOEXCLUSIVE}〕|DCTAM_MODIFY}〔|{DCTAM_WA
IT|DCTAM_NOWAIT}〕
```

- フラグ 1

  フラグ 1 には，次のレコードの検索種別のうちどれか一つを指定してください。

  レコードの検索種別の設定は省略できません。また，検索種別は重複して設定できません。

  DCTAM_EQLSRC…'キー値＝'を検索する（ハッシュ，ツリー）。

  DCTAM_GRTEQLSRC…'キー値＜＝'を検索する（ツリー）。

  DCTAM_GRTSRC…'キー値＜'を検索する（ツリー）。

  DCTAM_LSSEQLSRC…'キー値＞＝'を検索する（ツリー）。

  DCTAM_LSSSRC…'キー値＞'を検索する（ツリー）。

  DCTAM_FIRSTSRC…先頭から検索する（ハッシュ）。

  DCTAM_NEXTSRC…設定したキー値の，次のレコードから検索する（ハッシュ）。

- フラグ 2

  レコードのアクセス種別の設定も省略できません。また，アクセス種別も重複して設定できません。

  DCTAM_REFERENCE…参照目的の排他

  DCTAM_MODIFY…更新目的の排他

- フラグ 3

  参照目的の排他の場合は，排他するかどうかを設定します。排他要否種別は重複して設定できません。このフラグの設定を省略した場合は，DCTAM_NOEXCLUSIVE が仮定されます。

  DCTAM_EXCLUSIVE…排他します。

  DCTAM_NOEXCLUSIVE…排他をしません。

  排他しないで dc_tam_read 関数を呼び出した場合，dc_tam_read 関数の処理中にほかの UAP から該当する TAM レコードが更新されることがあります。この場合，dc_tam_read 関数で入力するレコードの内容は，ほかの UAP での更新処理の状態（コミット処理が完了するタイミング）に依存します。したがって，UAP 間でデータの整合性をとるためには，必ず DCTAM_EXCLUSIVE を指定してください。DCTAM_NOEXCLUSIVE は，参照型の TAM テーブルの場合や，同一の TAM レコードに対して業務処理全体として参照と更新が競合する可能性がない場合に使用します。

- フラグ 4

  排他解除待ち種別も重複して設定できません。このフラグの設定を省略した場合は，DCTAM_NOWAIT が仮定されます。

  DCTAM_WAIT…排他解除待ちをします。

  DCTAM_NOWAIT…待たないで，エラーリターンします。

flags に設定する値と，設定内容について次に示します。

フラグ 1	フラグ 2	フラグ 3※1	フラグ 4※2	flags の設定内容
EQLSRC GRTEQLSRC GRTSRC LSSEQLSRC LSSSRC FIRSTSRC NEXTSRC	REFERENCE	EXCLUSIVE	WAIT	参照目的，排他あり， 排他エラー時は解除待ち
			NOWAIT	参照目的，排他あり， 排他エラー時はエラーリターン
		NOEXCLUSIVE	×	参照目的，排他なし
	MODIFY	－	WAIT	更新目的， 排他エラー時は解除待ち
			NOWAIT	更新目的， 排他エラー時はエラーリターン

（凡例）

×：設定できません。

－：常に EXCLUSIVE を設定します。NOEXCLUSIVE は設定できません。

注※1

省略した場合は，NOEXCLUSIVE が仮定されます。

注※2

省略した場合は，NOWAIT が仮定されます。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	TAM テーブルのレコードを正常に入力しました。
DCTAMER_PARAM_TID	-1700	tblid に設定したテーブル記述子が間違っています。
DCTAMER_PARAM_KEY	-1702	keyadr に設定したキー値が間違っています。
DCTAMER_PARAM_KNO	-1703	keyno に設定した値が間違っています。
DCTAMER_PARAM_BFA	-1704	bufadr に設定した値が間違っています。
DCTAMER_PARAM_BFS	-1705	bufsize に設定したバッファ長が短過ぎます。
DCTAMER_PARAM_FLG	-1708	flags に設定した値が間違っています。

リターン値	リターン値（数値）	意味
DCTAMER_NOTTAM	-1709	tblid に設定したテーブルは TAM テーブルではありません。
DCTAMER_TAMEND	-1720	TAM サービスが終了中です。
DCTAMER_PROTO	-1721	TAM テーブルへアクセスする順序が間違っています。
		UAP にリンケージしているトランザクション制御用オブジェクトファイルのリソースマネジャ登録が間違っています。 または，UAP にトランザクション制御用オブジェクトファイルをリンケージしていません。
		関数を呼び出した UAP のユーザサービス定義に，トランザクション属性なし（atomic_update=N）を指定しています。
DCTAMER_RMTBL	-1723	TAM テーブルが削除されています。
DCTAMER_NOLOAD	-1724	TAM テーブルがロードされていません。
DCTAMER_NOOPEN	-1726	TAM テーブルがオープン状態ではありません。
DCTAMER_LOGHLD	-1727	TAM テーブルが論理閉塞状態です。
DCTAMER_OBSHLD	-1728	TAM テーブルが障害閉塞状態です。
DCTAMER_IDXTYP	-1729	TAM テーブルファイルの初期作成で設定した TAM テーブルのインデクス種別では実行できません。
DCTAMER_ACSATL	-1730	TAM サービス定義で設定した TAM テーブルのアクセス形態では実行できません。
DCTAMER_NOREC	-1731	flags に設定した検索条件を満たすレコードがありません。
DCTAMER_LOCK	-1736	排他エラーが起こりました。flags に DCTAM_WAIT を設定した場合，ロックサービス定義で指定した待ち時間のタイムアウトのため，資源を確保できませんでした。
DCTAMER_DLOCK	-1737	デッドロックが起こりました。
DCTAMER_TBLVR	-1760	UAP が，現在稼働している TAM テーブルでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_FLSVR	-1761	UAP が，現在稼働している OpenTP1 ファイルサービスでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_RECOBS	-1764	レコードが破壊されています。
DCTAMER_TRNNUM	-1765	TAM サービスで管理できるトランザクション数を超えました。
DCTAMER_OPENNUM	-1766	キャラクタ型スペシャルファイルのオープン数の制限値を超えました。
DCTAMER_ACCESSS	-1767	スペシャルファイルに対するアクセス権がありません。
DCTAMER_ACCESSF	-1768	TAM ファイルに対するアクセス権がありません。

リターン値	リターン値（数値）	意味
DCTAMER_MEMORY	-1769	メモリが不足しました。
DCTAMER_IO	-1770	入出力エラーが起こりました。
DCTAMER_TMERR	-1771	トランザクションサービスでエラーが起こりました。
DCTAMER_ACCESS	-1773	アクセスしようとした TAM ファイルは，セキュリティ機能で保護されています。dc_tam_read 関数を呼び出した UAP には，アクセス権限がありません。

dc_tam_read_cancel

名称

TAM テーブルのレコードの入力取り消し

形式

ANSI C，C++の形式

```
#include <dctam.h>
int  dc_tam_read_cancel(DCLONG tblid,struct DC_TAMKEY *keyadr,
                        int keyno,DCLONG flags)
```

K&R 版 C の形式

```
#include <dctam.h>
int dc_tam_read_cancel(tblid,keyadr,keyno,flags)
DCLONG           tblid;
struct DC_TAMKEY *keyadr;
int              keyno;
DCLONG           flags;
```

機能

トランザクション内で，dc_tam_read 関数を使って排他を掛けた参照目的の入力，および更新目的の入力を取り消し，レコード排他を解除します。

更新，または追加済みのレコードには，排他を掛けた参照目的の入力を取り消すことはできません。また，dc_tam_rewrite 関数で更新したレコードの，更新目的の入力の取り消しはできません。

更新，または追加済みのレコード，およびテーブル排他でオープンした，TAM テーブル上のレコードに対する更新目的の入力の取り消しでは，排他を解除しません。

dc_tam_read_cancel 関数で入力を取り消したあとも，トランザクションが終了するまでは，入力した TAM テーブルに対して，ほかのトランザクションからレコードの追加，削除はできません。

dc_tam_read_cancel 関数がエラーリターンした場合，この関数で解放した資源は再確保しないで，関数を呼び出す前の状態には戻しません。また，複数のレコードを設定してアクセスを要求した場合，それらのレコードのうち，一つでもエラーが発生したら処理を中断して，エラーリターンします。

UAP で値を設定する引数

●tblid

レコードの入力を取り消す TAM テーブルの，テーブル記述子を設定します。テーブル記述子は dc_tam_open 関数で返された値です。

●keyadr

入力を取り消すレコードの，キー値のアドレスを持つ構造体のアドレスを設定します。構造体の形式は次のとおりです。

```
struct DC_TAMKEY {
                  char *keyname;
                  };
```

- keyname
  キー値のアドレスを設定します。キー値は，入力を取り消すレコードのキー領域の長さで設定します。

●keyno

要求レコード数（keyadr で設定する構造体の数）を設定します。

●flag

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	TAM テーブルのレコードの検索を取り消して，レコード排他を正常に解除しました。
DCTAMER_PARAM_TID	-1700	tblid に設定したテーブル記述子が間違っています。
DCTAMER_PARAM_KEY	-1702	keyadr に設定したキー値が間違っています。
DCTAMER_PARAM_KNO	-1703	keyno に設定した値が間違っています。
DCTAMER_PARAM_FLG	-1708	flags に設定した値が間違っています。
DCTAMER_NOTTAM	-1709	tblid に設定したテーブルは TAM テーブルではありません。
DCTAMER_TAMEND	-1720	TAM サービスが終了中です。
DCTAMER_PROTO	-1721	TAM テーブルへアクセスする順序が間違っています。
		UAP にリンケージしているトランザクション制御用オブジェクトファイルのリソースマネジャ登録が間違っています。 または，UAP にトランザクション制御用オブジェクトファイルをリンケージしていません。
		関数を呼び出した UAP のユーザサービス定義に，トランザクション属性なし（atomic_update=N）を指定しています。
DCTAMER_RMTBL	-1723	TAM テーブルが削除されています。
DCTAMER_NOLOAD	-1724	TAM テーブルがロードされていません。
DCTAMER_NOOPEN	-1726	TAM テーブルがオープン状態ではありません。

リターン値	リターン値（数値）	意味
DCTAMER_LOGHLD	-1727	TAM テーブルが論理閉塞状態です。
DCTAMER_OBSHLD	-1728	TAM テーブルが障害閉塞状態です。
DCTAMER_NOREC	-1731	指定されたレコードは存在しません。
DCTAMER_SEQENCE	-1732	dc_tam_read 関数を呼び出していません。
DCTAMER_EXWRITE	-1733	tblid に設定したテーブル記述子は dc_tam_write 関数で更新または追加したレコードです。
DCTAMER_EXREWRT	-1734	tblid に設定したテーブル記述子は，すでに dc_tam_rewrite 関数で更新済みです。
DCTAMER_TBLVR	-1760	UAP が，現在稼働している TAM テーブルでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_FLSVR	-1761	UAP が，現在稼働している OpenTP1 ファイルサービスでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_TRNNUM	-1765	TAM サービスで管理できるトランザクション数を超えました。
DCTAMER_OPENNUM	-1766	キャラクタ型スペシャルファイルのオープン数の制限値を超えました。
DCTAMER_ACCESSS	-1767	スペシャルファイルに対するアクセス権がありません。
DCTAMER_ACCESSF	-1768	TAM テーブルファイルに対するアクセス権がありません。
DCTAMER_MEMORY	-1769	メモリが不足しました。
DCTAMER_IO	-1770	入出力エラーが起こりました。
DCTAMER_TMERR	-1771	トランザクションサービスでエラーが起こりました。

dc_tam_rewrite

名称

TAM テーブルのレコード入力を前提の更新

形式

ANSI C，C++の形式

```
#include <dctam.h>
int  dc_tam_rewrite(DCLONG tblid,struct DC_TAMKEY *keyadr,
                    int keyno,
                    char *datadr,int datsize,DCLONG flags)
```

K&R 版 C の形式

```
#include <dctam.h>
int  dc_tam_rewrite(tblid,keyadr,keyno,datadr,datsize,flags)
DCLONG            tblid;
struct DC_TAMKEY  *keyadr;
int               keyno;
char              *datadr;
int               datsize;
DCLONG            flags;
```

機能

dc_tam_read 関数で入力したレコードを，更新して出力します。

更新目的の入力の dc_tam_read 関数を 1 回呼び出せば，その後トランザクションの同期点まで，何度でも dc_tam_rewrite 関数を使えます。ただし，dc_tam_delete 関数，dc_tam_read_cancel 関数を呼び出したあとには，dc_tam_rewrite 関数を使えません。

dc_tam_rewrite 関数がエラーリターンした場合は，この関数内で設定した資源はすべて解放して，関数を呼び出す前の状態に戻ります。

複数のレコードを設定して更新を要求する場合，それらのレコードのうち一つでもエラーが発生したときは，この関数で設定した全レコードの処理をエラーとします。

更新データ内のキー値の格納位置，およびキー領域長は，TAM テーブルファイルの初期作成時の tamcre コマンドに設定した値です。

TAM テーブルファイルの初期作成時，データ部にキー値を付けている（tamcre コマンドに-s オプションを指定していない）場合は，データ部にキー値があります。そのため，dc_tam_rewrite 関数に設定したキー値が更新データ内になければ，エラーリターンします。また，データ部にキー値を付けていない（tamcre コマンドに-s オプションを指定）場合は，データ部にキー値はありません。この場合は，更新データの内容をチェックしません。

UAP で値を設定する引数

●tblid

レコードを更新する TAM テーブルの，テーブル記述子を設定します。テーブル記述子は dc_tam_open 関数で返された値です。

●keyadr

更新するレコードの，キー値のアドレスを持つ構造体のアドレスを設定します。

構造体の形式は次のとおりです。

```
struct DC_TAMKEY {
                  char *keyname;
                  };
```

- keyname
  キー値のアドレスを設定します。キー値は，更新するレコードのキー領域長さで設定します。

●keyno

要求レコード数（keyadr で設定する構造体の数）を設定します。

●datadr

更新データのアドレスを設定します。

●datsize

更新データ長を設定します。更新データ長は，（レコード長×要求レコード数）以上にしてください。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	TAM テーブルのレコードを正常に更新しました。
DCTAMER_PARAM_TID	-1700	tblid に設定したテーブル記述子が間違っています。
DCTAMER_PARAM_KEY	-1702	keyadr に設定したキー値が間違っています。
DCTAMER_PARAM_KNO	-1703	keyno に設定した値が間違っています。
DCTAMER_PARAM_DTA	-1706	datadr に設定したデータ領域内のキー値が，keyname で設定したキー値と一致しません。
DCTAMER_PARAM_DTS	-1707	datsize に設定したデータ長が短過ぎます。

リターン値	リターン値（数値）	意味
DCTAMER_PARAM_FLG	-1708	flags に設定した値が間違っています。
DCTAMER_NOTTAM	-1709	tblid に設定したテーブルは TAM テーブルではありません。
DCTAMER_TAMEND	-1720	TAM サービスが終了中です。
DCTAMER_PROTO	-1721	TAM テーブルへアクセスする順序が間違っています。
		UAP にリンケージしているトランザクション制御用オブジェクトファイルのリソースマネジャ登録が間違っています。 または，UAP にトランザクション制御用オブジェクトファイルをリンケージしていません。
		関数を呼び出した UAP のユーザサービス定義に，トランザクション属性なし（atomic_update=N）を指定しています。
DCTAMER_RMTBL	-1723	TAM テーブルが削除されています。
DCTAMER_NOLOAD	-1724	TAM テーブルがロードされていません。
DCTAMER_NOOPEN	-1726	TAM テーブルがオープン状態ではありません。
DCTAMER_LOGHLD	-1727	TAM テーブルが論理閉塞状態です。
DCTAMER_OBSHLD	-1728	TAM テーブルが障害閉塞状態です。
DCTAMER_NOREC	-1731	指定されたレコードは存在しません。
DCTAMER_SEQENCE	-1732	dc_tam_read 関数を呼び出していません。
DCTAMER_TBLVR	-1760	UAP が，現在稼働している TAM テーブルでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_FLSVR	-1761	UAP が，現在稼働している OpenTP1 ファイルサービスでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_RECOBS	-1764	レコードが破壊されています。
DCTAMER_TRNNUM	-1765	TAM サービスで管理できるトランザクション数を超えました。
DCTAMER_OPENNUM	-1766	キャラクタ型スペシャルファイルのオープン数の制限値を超えました。
DCTAMER_ACCESSS	-1767	スペシャルファイルに対するアクセス権がありません。
DCTAMER_ACCESSF	-1768	TAM テーブルファイルに対するアクセス権がありません。
DCTAMER_MEMORY	-1769	メモリが不足しました。
DCTAMER_IO	-1770	入出力エラーが起こりました。
DCTAMER_TMERR	-1771	トランザクションサービスでエラーが起こりました。

dc_tam_status

名称

TAM テーブルの情報の取得

形式

ANSI C，C++の形式

```
#include <dctam.h>
int  dc_tam_status(char *tblname,struct DC_TAMSTAT *stbuf,
                   DCLONG  flags)
```

K&R 版 C の形式

```
#include <dctam.h>
int  dc_tam_status(tblname,stbuf,flags)
char       *tblname;
struct  DC_TAMSTAT  *stbuf;
DCLONG     flags;
```

機能

TAM テーブルの情報を，DC_TAMSTAT 構造体に返します。リターンする値を次に示します。

- TAM ファイル名
- TAM テーブルの状態
- 使用中のレコード数
- 最大レコード数
- インデクス種別
- アクセス形態
- ローディング契機
- TAM レコード長
- キー長
- キー開始位置
- セキュリティ属性

UAP で値を設定する引数

●tblname

情報を取得する TAM テーブル名を設定します。TAM テーブル名は，32 文字以内で設定します。文字列の最後にはヌル文字を付けてください。

●stbuf

TAM テーブルの情報を受け取る構造体 DC_TAMSTAT のアドレスを設定します。構造体には，dc_tam_status 関数に設定した TAM テーブルの状態が返されます。

●flags

DCNOFLAGS を設定します。

OpenTP1 から値が返される引数

●stbuf

TAM テーブルの情報が，構造体 DC_TAMSTAT で返されます。構造体の形式は次のとおりです。

```
struct  DC_TAMSTAT {
                    char    st_file_name[64];
                    DCLONG  st_tbl_stat;
                    DCLONG  st_rec_usenum;
                    DCLONG  st_rec_maxnum;
                    char    st_idx_type;
                    char    st_acs_type;
                    char    st_lod_type;
                    char    reserve1;
                    DCLONG  st_rec_len;
                    DCLONG  st_key_len;
                    DCLONG  st_key_pos;
                    DCLONG  st_tbl_sec;
                    DCLONG  reserve2[8];
                    };
```

- st_file_name

  TAM ファイル名が返されます。

- st_tbl_stat

  TAM テーブルの状態が，次に示す値のどれかで返されます。

  DCTAM_STS_OPN…オープン状態であることを示します。

  DCTAM_STS_CLS…クローズ状態であることを示します。

  DCTAM_STS_LHLD…論理閉塞状態であることを示します。

  DCTAM_STS_OHLD…障害閉塞状態であることを示します。

- st_rec_usenum

  TAM テーブルで現在使っているレコードの数が返されます。ただし，dc_tam_status 関数を呼び出したあとでレコードの追加や削除があった場合は，値を保証しません。

- st_rec_maxnum

  TAM テーブルの最大レコード数が返されます。

- st_idx_type

  TAM テーブルのインデクス種別が，次に示す値のどれかで返されます。

DCTAM_STS_HASH…ハッシュ形式であることを示します。
DCTAM_STS_TREE…ツリー形式であることを示します。

- st_acs_type
  TAM テーブルのアクセス形態が，次に示す値のどれかで返されます。
  DCTAM_STS_READ…参照型であることを示します。
  DCTAM_STS_REWRITE…追加・削除できない更新型であることを示します。
  DCTAM_STS_WRITE…追加・削除できる更新型であることを示します。
  DCTAM_STS_RECLCK…テーブル排他を確保しない，追加・削除できる更新型であることを示します。
- st_lod_type
  TAM テーブルのローディング契機が，次に示す値のどれかで返されます。
  DCTAM_STS_START…TAM サービスの開始時であることを示します。
  DCTAM_STS_LIB…dc_tam_open 関数で TAM テーブルをオープンしたときであることを示します。
  DCTAM_STS_CMD…tamload コマンドを実行したときであることを示します。
- reserve1
  予備の領域です。
- st_rec_len
  TAM テーブルのレコード長が返されます。
- st_key_len
  TAM テーブルのキー長が返されます。
- st_key_pos
  TAM テーブルのデータ内のキー開始位置が返されます。
- st_tbl_sec
  TAM サービス定義で指定した TAM テーブルのセキュリティ属性が，次に示す値のどれかで返されます。
  DCTAM_STS_NOSEC…セキュリティの指定がないことを示します。
  DCTAM_STS_SEC…セキュリティの指定があることを示します。
- reserve2
  予備の領域です。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	TAM テーブルの情報を正常に取得しました。
DCTAMER_PARAM_TBL	-1701	tblname に設定した値が間違っています。

リターン値	リターン値（数値）	意味
DCTAMER_PARAM_FLG	-1708	flags に設定した値が間違っています。
DCTAMER_NOTTAM	-1709	tblname に設定した名称は，TAM テーブルではありません。
DCTAMER_UNDEF	-1710	TAM テーブルが定義されていません。
DCTAMER_TAMEND	-1720	TAM サービスが終了中です。
DCTAMER_PROTO	-1721	TAM テーブルへアクセスする順序が間違っています。
		UAP にリンケージしているトランザクション制御用オブジェクトファイルのリソースマネジャ登録が間違っています。または，UAP にトランザクション制御用オブジェクトファイルをリンケージしていません。
		関数を呼び出した UAP のユーザサービス定義に，トランザクション属性なし（atomic_update=N）を指定しています。
DCTAMER_TBLVR	-1760	UAP が，現在稼働している TAM テーブルでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_TAMVR	-1762	UAP が，現在稼働している TAM サービスでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_OPENNUM	-1766	キャラクタ型スペシャルファイルのオープン数の制限値を超えました。
DCTAMER_ACCESSS	-1767	スペシャルファイルに対するアクセス権がありません。
DCTAMER_MEMORY	-1769	メモリが不足しました。
DCTAMER_IO	-1770	入出力エラーが起こりました。
DCTAMER_NO_ACL	-1772	情報を取得しようとした TAM テーブルは，セキュリティ機能で保護されています。該当する TAM テーブルに対する ACL がありません。
DCTAMER_ACCESS	-1773	情報を取得しようとした TAM テーブルは，セキュリティ機能で保護されています。dc_tam_status 関数を呼び出した UAP には，アクセス権限がありません。

dc_tam_write

名称

TAM テーブルのレコードの更新/追加

形式

ANSI C，C++の形式

```
#include <dctam.h>
int  dc_tam_write(DCLONG tblid,struct DC_TAMKEY *keyadr,
                  int keyno,
                  char *datadr,int datsize,DCLONG flags)
```

K&R 版 C の形式

```
#include <dctam.h>
int  dc_tam_write(tblid,keyadr,keyno,datadr,datsize,flags)
DCLONG           tblid;
struct DC_TAMKEY *keyadr;
int              keyno;
char             *datadr;
int              datsize;
DCLONG           flags;
```

機能

キー値に示すレコードを，TAM テーブル上に更新または追加します。

レコード排他で TAM テーブルがオープンしている場合，次のように排他を確保します。

- アクセス種別が「更新」の場合（flags に DCTAM_WRITE を設定）
  参照排他でテーブル排他を確保して，更新排他でレコード排他を確保します。ただし，TAM サービス定義の「アクセス時のテーブル排他モード」に，「テーブル排他なしモード」を指定している場合は，アクセス形態が「参照型」「追加・削除できない更新型」のテーブルには，テーブル排他を確保しません。
- アクセス種別が「更新または追加」，「追加」の場合
  （flags に DCTAM_WRTADD，または DCTAM_ADD を設定）
  更新排他でテーブル排他を確保します。

dc_tam_write 関数がエラーリターンした場合，この関数内で確保した資源はすべて解放して，関数を呼び出す前の状態に戻します。ただし，この関数を呼び出す前に，参照排他で確保されていた TAM テーブルを更新または追加した場合は，更新排他となり参照排他には戻りません。

複数のレコードを設定して更新または追加する場合，それらのレコードのうち一つでもエラーが発生したら，この関数で設定した全レコードの処理をエラーとします。

更新または追加するデータ内のキー値の格納位置，およびキー領域長は，TAM テーブルファイルの初期作成時の tamcre コマンドに設定した値です。

TAM テーブルファイルの初期作成時，データ部にキー値を付けている（tamcre コマンドに-s オプションを指定していない）場合は，データ部にキー値があります。そのため，dc_tam_write 関数に設定したキー値が更新または追加するデータ内になければ，エラーリターンします。また，データ部にキー値を付けていない（tamcre コマンドに-s オプションを指定）場合は，データ部にキー値はありません。この場合は，更新または追加するデータの内容をチェックしません。

UAP で値を設定する引数

●tblid

レコードを更新または追加する TAM テーブルの，テーブル記述子を設定します。テーブル記述子は dc_tam_open 関数で返された値です。

●keyadr

更新，または追加するレコードのキー値のアドレスを持つ，構造体のアドレスを設定します。構造体の形式は次のとおりです。

```
struct DC_TAMKEY {
                  char *keyname;
                  };
```

- keyname

  キー値のアドレスを設定します。キー値は，更新または追加するレコードのキー領域の長さで設定します。

●keyno

要求レコード数（keyadr で設定する構造体の数）を設定します。

●datadr

更新または追加するデータのアドレスを設定します。

●datsize

更新または追加するデータ長を設定します。更新または追加するデータ長は，（レコード長×要求レコード数）以上にしてください。

●flags

レコードのアクセス種別，資源が競合した場合の排他解除待ち種別を，次の形式で設定します。

```
{DCTAM_WRITE|DCTAM_WRTADD|DCTAM_ADD} 〔|{DCTAM_WAIT|DCTAM_NOWAIT}〕
```

- フラグ 1

レコードのアクセス種別の設定は省略できません。また，アクセス種別は重複して設定できません。

DCTAM_WRITE…更新

DCTAM_WRTADD…更新または追加

DCTAM_ADD…追加

- フラグ 2

  排他解除待ち種別は，重複して設定できません。このフラグの設定を省略した場合は，DCTAM_NOWAIT が仮定されます。

  DCTAM_WAIT…排他解除待ちをします。

  DCTAM_NOWAIT…排他解除を待たないで，エラーリターンします。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	TAM テーブルのレコードを正常に更新，または追加しました。
DCTAMER_PARAM_TID	-1700	tblid に設定したテーブル記述子が間違っています。
DCTAMER_PARAM_KEY	-1702	keyadr に設定したキー値が間違っています。
DCTAMER_PARAM_KNO	-1703	keyno に設定した値が間違っています。
DCTAMER_PARAM_DTA	-1706	datadr に設定したデータ領域内のキー値が，keyname で設定したキー値と一致しません。
DCTAMER_PARAM_DTS	-1707	datsize に設定したデータ長が短過ぎます。
DCTAMER_PARAM_FLG	-1708	flags に設定した値が間違っています。
DCTAMER_NOTTAM	-1709	tblid に設定したテーブルは TAM テーブルではありません。
DCTAMER_TAMEND	-1720	TAM サービスが終了中です。
DCTAMER_PROTO	-1721	TAM テーブルへアクセスする順序が間違っています。
		UAP にリンケージしているトランザクション制御用オブジェクトファイルのリソースマネジャ登録が間違っています。 または，UAP にトランザクション制御用オブジェクトファイルをリンケージしていません。
		関数を呼び出した UAP のユーザサービス定義に，トランザクション属性なし（atomic_update=N）を指定しています。
DCTAMER_RMTBL	-1723	TAM テーブルが削除されています。
DCTAMER_NOLOAD	-1724	TAM テーブルがロードされていません。
DCTAMER_NOOPEN	-1726	TAM テーブルがオープン状態ではありません。
DCTAMER_LOGHLD	-1727	TAM テーブルが論理閉塞状態です。
DCTAMER_OBSHLD	-1728	TAM テーブルが障害閉塞状態です。

リターン値	リターン値（数値）	意味
DCTAMER_ACSATL	-1730	TAM サービス定義で設定した TAM テーブルのアクセス形態では実行できません。
DCTAMER_NOREC	-1731	指定されたレコードは存在しません。
DCTAMER_EXKEY	-1735	keyadr に設定したキー値が TAM テーブルに存在するので，レコードの追加はできません。
DCTAMER_LOCK	-1736	排他エラーが起こりました。flags に DCTAM_WAIT を設定した場合，ロックサービス定義で指定した待ち時間のタイムアウトのため，資源を確保できませんでした。
DCTAMER_DLOCK	-1737	デッドロックが起こりました。
DCTAMER_TBLVR	-1760	UAP が，現在稼働している TAM テーブルでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_FLSVR	-1761	UAP が，現在稼働している OpenTP1 ファイルサービスでは動作できないバージョンの TAM ライブラリと結合されています。
DCTAMER_NOAREA	-1763	TAM テーブルに空きレコードがありません。
DCTAMER_RECOBS	-1764	レコードが破壊されています。
DCTAMER_TRNNUM	-1765	TAM サービスで管理できるトランザクション数を超えました。
DCTAMER_OPENNUM	-1766	キャラクタ型スペシャルファイルのオープン数の制限値を超えました。
DCTAMER_ACCESSS	-1767	スペシャルファイルに対するアクセス権がありません。
DCTAMER_ACCESSF	-1768	TAM テーブルファイルに対するアクセス権がありません。
DCTAMER_MEMORY	-1769	メモリが不足しました。
DCTAMER_IO	-1770	入出力エラーが起こりました。
DCTAMER_TMERR	-1771	トランザクションサービスでエラーが起こりました。
DCTAMER_ACCESS	-1773	アクセスしようとした TAM ファイルは，セキュリティ機能で保護されています。dc_tam_write 関数を呼び出した UAP には，アクセス権限がありません。

トランザクション制御（dc_trn_～）

OpenTP1 独自のトランザクション制御をする関数について説明します。トランザクション制御の関数を次に示します。

- dc_trn_begin－トランザクションの開始
- dc_trn_chained_commit－連鎖モードのコミット
- dc_trn_chained_rollback－連鎖モードのロールバック
- dc_trn_info－現在のトランザクションに関する情報の報告
- dc_trn_rm_select－リソースマネジャ接続先選択
- dc_trn_unchained_commit－非連鎖モードのコミット
- dc_trn_unchained_rollback－非連鎖モードのロールバック

トランザクション制御の関数（dc_trn_～）は，TP1/Server Base と TP1/LiNK のどちらの UAP でも使えます。

dc_trn_begin

名称

トランザクションの開始

形式

ANSI C，C++ の形式

```
#include <dctrn.h>
int  dc_trn_begin(void)
```

K&R 版 C の形式

```
#include <dctrn.h>
int dc_trn_begin()
```

機能

グローバルトランザクションを，この関数を呼び出したプロセスから開始します。dc_trn_begin 関数を呼び出したプロセスは，グローバルトランザクションのルートトランザクションブランチになります。

dc_trn_begin 関数を呼び出した UAP は，実行環境の設定でトランザクション属性を指定しておいてください。また，すでに dc_trn_begin 関数を呼び出しているグローバルトランザクションの中では，どのトランザクションブランチからも再び dc_trn_begin 関数を呼び出せません。一つのグローバルトランザクション中で重複して呼び出した場合はエラーリターンします。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。グローバルトランザクションが開始して，dc_trn_begin 関数を呼び出したプロセスはその範囲に含まれています。
DCTRNER_PROTO	-905	dc_trn_begin 関数を正しくないコンテクスト（例えば，すでにトランザクション内にいる）から呼び出しています。 または，実行環境がジャーナルファイルレスモードのため，トランザクションは開始できませんでした。
DCTRNER_RM	-906	リソースマネジャ（RM）でエラーが起こりました。トランザクションは開始できませんでした。
DCTRNER_TM	-907	トランザクションサービスでエラーが起こったので，トランザクションは開始できませんでした。トランザクションサービス定義の trn_tran_process_count オペランドの指定値が不足している可能性があります。

リターン値	リターン値（数値）	意味
DCTRNER_TM	-907	このリターン値が戻った場合は，再び実行すれば成功する可能性が高いので，再実行してください。

指定例

```
if(!dc_trn_info(NULL) && dc_trn_begin() < 0)
   fputs("cannot begin transaction¥n", stderr);
```

dc_trn_chained_commit

名称

連鎖モードのコミット

形式

ANSI C，C++の形式

```
#include <dctrn.h>
int  dc_trn_chained_commit(void)
```

K&R 版 C の形式

```
#include <dctrn.h>
int dc_trn_chained_commit()
```

機能

トランザクションの同期点を取得します。グローバルトランザクションのルートトランザクションブランチとして，処理が正常に終了したこと（コミット）を，トランザクションを構成するトランザクションブランチの UAP，トランザクションサービス，およびリソースマネジャに知らせます。

dc_trn_chained_commit 関数が正常に終了すると，新しいグローバルトランザクションが開始して，関数を呼び出したプロセスはこのトランザクションの範囲内です。しかし，この関数を呼び出した UAP 以外のトランザクションモードに対しての指定を意味しません。

グローバルトランザクションが複数のトランザクションブランチから構成されるとき（関数を呼び出した UAP だけでないとき）は，それぞれのトランザクションブランチの処理結果がコミットとならないかぎりコミットされません。

dc_trn_chained_commit 関数を呼び出せるのはルートトランザクションブランチ（dc_trn_begin 関数を呼び出した UAP）だけです。それ以外の UAP から呼び出した場合は，リターン値 DCTRNER_PROTO でエラーリターンします。

dc_trn_chained_commit 関数を呼び出すプロセスは，このマニュアルの記述に従って正しく作成された UAP を稼働させたものでなければなりません。

dc_trn_chained_commit 関数は，同期点処理が完了したときに正常に終了，またはエラーリターンのどちらかで返ります。dc_trn_chained_commit 関数が正常に終了するためには，UAP の実行環境を設定するときに，トランザクション属性を指定していることが必要です。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。dc_trn_chained_commit 関数が終了したあとも，このプロセスはトランザクション下にあって，グローバルトランザクションの範囲内です。
DCTRNER_ROLLBACK	-902	現在のトランザクションは，コミットできないでロールバックしました。 このリターン値が返ったあとも，このプロセスはトランザクション下にあって，グローバルトランザクションの範囲内です。
DCTRNER_HEURISTIC	-903	dc_trn_chained_commit 関数を呼び出したグローバルトランザクションは，ヒューリスティック決定のため，あるトランザクションブランチはコミットとなり，あるトランザクションブランチはロールバックとなりました。 このリターン値は，ヒューリスティック決定の結果が，グローバルトランザクションの同期点の結果と一致しなかった場合にリターンします。 このリターン値が返る原因になった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルを参照してください。 このリターン値が返ったあとも，このプロセスはトランザクション下にあって，グローバルトランザクションの範囲内です。
DCTRNER_HAZARD	-904	グローバルトランザクションのトランザクションブランチがヒューリスティックに完了しました。しかし，障害のため，ヒューリスティックに完了したトランザクションブランチの同期点の結果が判明しません。 このリターン値が返る原因になった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルを参照してください。 このリターン値が返ったあとも，このプロセスはトランザクション下にあって，グローバルトランザクションの範囲内です。 トランザクションサービス定義で，trn_extend_function オペランドに 00000001 を指定し，1 相コミット時にリソースマネジャからのリターン値が XAER_NOTA の場合も，DCTRNER_HAZARD を返します。
DCTRNER_PROTO	-905	dc_trn_chained_commit 関数を正しくないコンテクスト（例えば，すでにトランザクション中にいない）で呼び出しています。トランザクションモードに対する影響はありません。
DCTRNER_NO_BEGIN	-924	コミット処理は正常に終了しましたが，新しいトランザクションは開始できませんでした。このリターン値が返ったあと，このプロセスはトランザクション下にはありません。
DCTRNER_ROLLBACK_NO_BEGIN	-925	コミットしようとしたトランザクションは，コミットできないでロールバックしました。新しいトランザクションは開始でき

リターン値	リターン値（数値）	意味
DCTRNER_ROLLBACK_NO_BEGIN	-925	ませんでした。このリターン値が返ったあと，このプロセスはトランザクション下にはありません。
DCTRNER_HEURISTIC_NO_BEGIN	-926	dc_trn_chained_commit 関数を呼び出したグローバルトランザクションは，ヒューリスティック決定のため，あるトランザクションブランチはコミットとなって，あるトランザクションブランチはロールバックとなりました。 このリターン値は，ヒューリスティック決定の結果が，グローバルトランザクションの同期点の結果と一致しなかった場合にリターンされます。 このリターン値が返される原因となった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルの内容を参照してください。新しいトランザクションは開始できませんでした。このリターン値が返ったあと，このプロセスはトランザクション下にはありません。
DCTRNER_HAZARD_NO_BEGIN	-927	グローバルトランザクションのトランザクションブランチがヒューリスティックに完了しました。しかし，障害のため，ヒューリスティックに完了したトランザクションブランチの同期点の結果がわかりません。 このリターン値が返される原因となった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルの内容を参照してください。 新しいトランザクションは開始できませんでした。このリターン値が返ったあと，このプロセスはトランザクション下にはありません。 トランザクションサービス定義で，trn_extend_function オペランドに 00000001 を指定し，1 相コミット時にリソースマネジャからのリターン値が XAER_NOTA の場合も，DCTRNER_HAZARD_NO_BEGIN を返します。

指定例

```
if(dc_trn_info(NULL) && dc_trn_chained_commit() < 0)
  fputs(”cannot commit transaction¥n”, stderr);
```

dc_trn_chained_rollback

名称

連鎖モードのロールバック

形式

ANSI C，C++の形式

```
#include <dctrn.h>
int  dc_trn_chained_rollback(void)
```

K&R 版 C の形式

```
#include <dctrn.h>
int  dc_trn_chained_rollback()
```

機能

トランザクションをロールバックさせます。dc_trn_chained_rollback 関数を呼び出したあとには，続けてトランザクションが開始します。

dc_trn_chained_rollback 関数を呼び出すことで，ルートトランザクションブランチから，トランザクションブランチ，トランザクションサービス，およびリソースマネジャにロールバックを知らせます。

dc_trn_chained_rollback 関数が正常に終了すると，関数を呼び出したプロセスはロールバックしてリターンします。そのあとで新しいグローバルトランザクションが開始します。関数を呼び出したプロセスはこのトランザクションの範囲内です。ただし，この関数を呼び出した UAP 以外のトランザクションモードに対しての指定を意味しません。

dc_trn_chained_rollback 関数を呼び出せるのは，ルートトランザクションブランチ（dc_trn_begin 関数を呼び出した UAP）からだけです。それ以外の UAP から呼び出した場合は，リターン値 DCTRNER_PROTO でエラーリターンします。

dc_trn_chained_rollback 関数を呼び出すプロセスは，このマニュアルの記述に従って正しく作成された UAP を稼働させたものでなければなりません。

dc_trn_chained_rollback 関数は，同期点処理が完了したときに，正常に終了，またはエラーリターンのどちらかで返ります。dc_trn_chained_rollback 関数を呼び出すサービスが正常に終了するためには，UAP の実行環境を設定するときに，トランザクション属性を指定していることが前提です。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。dc_trn_chained_rollback 関数が終了したあとも，このプロセスはトランザクション下にあって，グローバルトランザクションの範囲内です。
DCTRNER_HEURISTIC	-903	dc_trn_chained_rollback 関数を呼び出したグローバルトランザクションは，ヒューリスティック決定のため，あるトランザクションブランチはコミットとなり，あるトランザクションブランチはロールバックとなりました。 このリターン値は，ヒューリスティック決定の結果が，グローバルトランザクションの同期点の結果と一致しなかった場合にリターンします。 このリターン値が返る原因になった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルを参照してください。 このリターン値が返ったあとも，このプロセスはトランザクション下にあって，グローバルトランザクションの範囲内です。
DCTRNER_HAZARD	-904	グローバルトランザクションのトランザクションブランチがヒューリスティックに完了しました。しかし，障害のため，ヒューリスティックに完了したトランザクションブランチの同期点の結果がわかりません。 このリターン値が返る原因になった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルを参照してください。 このリターン値が返ったあとも，このプロセスはトランザクション下にあって，グローバルトランザクションの範囲内です。
DCTRNER_PROTO	-905	dc_trn_chained_rollback 関数を正しくないコンテクスト（例えば，すでにトランザクション中にいない）で呼び出しています。トランザクションモードに対する影響はありません。
DCTRNER_NO_BEGIN	-924	ロールバック処理は正常に終了しましたが，新しいトランザクションは開始できませんでした。このリターン値が返ったあと，このプロセスはトランザクション下にはありません。
DCTRNER_HEURISTIC_NO_BEGIN	-926	dc_trn_chained_rollback 関数を呼び出したグローバルトランザクションは，ヒューリスティック決定のため，あるトランザクションブランチはコミットとなって，あるトランザクションブランチはロールバックとなりました。このリターン値は，ヒューリスティック決定の結果が，グローバルトランザクションの同期点の結果と一致しなかった場合にリターンされます。 このリターン値が返される原因となった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルの内容を参照してください。新しいトランザクションは開始できませんでした。このリターン値が返ったあと，このプロセスはトランザクション下にはありません。

リターン値	リターン値（数値）	意味
DCTRNER_HAZARD_NO_BEGIN	-927	グローバルトランザクションのトランザクションブランチがヒューリスティックに完了しました。しかし，障害のため，ヒューリスティックに完了したトランザクションブランチの同期点の結果がわかりません。 このリターン値が返される原因となったUAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルの内容を参照してください。 新しいトランザクションは開始できませんでした。このリターン値が返ったあと，このプロセスはトランザクション下にはありません。

指定例

```
if (dc_trn_info(NULL) && dc_trn_chained_rollback() <0)
  fputs(”cannot rollback transaction¥n”, stderr);
```

dc_trn_info

名称

現在のトランザクションに関する情報の報告

形式

ANSI C，C++の形式

```
#include <dctrn.h>
int  dc_trn_info(char *flags)
```

K&R 版 C の形式

```
#include <dctrn.h>
int  dc_trn_info(flags)
char      *flags;
```

機能

dc_trn_info 関数を呼び出した UAP が，現在トランザクションとして起動しているかどうかをリターンします。

dc_trn_info 関数を呼び出すプロセスは，このマニュアルの記述に従って正しく作成された UAP を稼働させたものでなければなりません。dc_trn_info 関数を呼び出すサービスが正常に終了するためには，UAP の実行環境を設定するときに，トランザクション属性を指定していることが前提です。

UAP で値を設定する引数

●flags

NULL を設定します。

リターン値

リターン値	意味
1	dc_trn_info 関数を呼び出したプロセスは，トランザクションとして起動しています。
0	dc_trn_info 関数を呼び出したプロセスは，トランザクションとして起動していません。

指定例

```
if(!dc_trn_info(NULL) && dc_trn_begin() < 0)
  fputs("cannot begin transaction¥n", stderr);
```

注意事項

この API は UAP トレースを取得していません。

dc_trn_rm_select

名称

リソースマネジャ接続先選択

形式

ANSI C，C++の形式

```
#include <dctrn.h>
int dc_trn_rm_select(char *rmname, char *id, DCLONG flags)
```

K&R 版 C の形式

```
#include <dctrn.h>
int dc_trn_rm_select(rmname, id, flags)
char *rmname;
char *id;
DCLONG flags;
```

機能

ユーザサービス定義の trnrmid 定義コマンドに-k オプションを指定したリソースマネジャについて，該当するプロセスまたはトランザクションで接続先となるリソースマネジャ（リソースマネジャ名＋リソースマネジャ拡張子）を指定します。

dc_trn_rm_select 関数は，グローバルトランザクション外でだけ呼び出せます。トランザクションブランチ内で dc_trn_rm_select 関数を呼び出した場合は，エラーリターンします。

dc_trn_rm_select 関数が正常に終了したあとは，指定されたリソースマネジャにだけ接続するようになり，trnrmid 定義コマンドに-k オプションの指定があるほかの同一名称のリソースマネジャには接続しません。同一 UAP プロセス内で，先に指定したリソースマネジャから別のリソースマネジャに変更する場合は，実行中のトランザクションが完了し新たなトランザクションを開始する前に変更対象となるリソースマネジャ（リソースマネジャ名とリソースマネジャ拡張子）を指定した dc_trn_rm_select 関数を呼び出してください。

UAP で値を設定する引数

●rmname

接続先となるリソースマネジャ名（trnrmid 定義コマンドの-n オプション指定値）を 31 文字以内の識別子で指定します。

●id

rmname でリソースマネジャに指定した拡張子（trnrmid 定義コマンドの-i オプション指定値）のうち，接続先となる拡張子を 2 文字以内の識別子で指定します。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。 指定されたリソースマネジャは，dc_trn_rm_select 関数を呼び出したプロセスで実行されるトランザクションで接続先として設定されました。
DCTRNER_PROTO	-905	dc_trn_rm_select 関数を正しくないコンテクスト（例えば，すでにトランザクション内にいる）で呼び出しています。または，実行環境がジャーナルファイルレスモードの場合に関数が呼び出されました。
DCTRNER_RM	-906	リソースマネジャでエラーが起こりました。指定されたリソースマネジャの XA 関数の呼び出し処理でエラーが発生しました。
DCTRNER_INVAL	-908	引数の指定値に次の誤りがあります。 • トランザクションサービス定義の trnstring 定義コマンドに指定されたリソースマネジャが存在しない。 • trnrmid 定義コマンドの-k オプションが指定されていない。 • flags に無効な値が指定されている。

指定例

```
if(dc_trn_rm_select(”RM_A”, ”Z1”, DCNOFLAGS))
  fputs(”cannot rm select¥n”,stderr);
```

dc_trn_unchained_commit

名称

非連鎖モードのコミット

形式

ANSI C，C++の形式

```
#include <dctrn.h>
int  dc_trn_unchained_commit(void)
```

K&R 版 C の形式

```
#include <dctrn.h>
int dc_trn_unchained_commit()
```

機能

グローバルトランザクションの正常終了（コミット）を，トランザクションを構成するトランザクションブランチ，トランザクションサービス，およびリソースマネジャに知らせます。

dc_trn_unchained_commit 関数が正常に終了したあとで，新しいグローバルトランザクションは開始しません。

グローバルトランザクションが複数のトランザクションブランチから構成されるとき（関数を呼び出した UAP だけでないとき）は，それぞれのトランザクションブランチの処理結果がコミットとならないかぎり，コミットされません。

dc_trn_unchained_commit 関数を呼び出せるのは，ルートトランザクションブランチ（トランザクションを開始した UAP）だけです。それ以外の UAP から呼び出した場合は，リターン値 DCTRNER_PROTO でエラーリターンします。

dc_trn_unchained_commit 関数を呼び出すプロセスは，このマニュアルの記述に従って正しく作成された UAP を稼働させたものでなければなりません。

dc_trn_unchained_commit 関数は，同期点処理が完了したときに，正常に終了，またはエラーリターンのどちらかで返ります。dc_trn_unchained_commit 関数を呼び出すサービスが正常に終了するためには，UAP の実行環境を設定するときに，トランザクション属性を指定していることが前提です。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。このプロセスはトランザクション下にはありません。プロセスはグローバルトランザクションの範囲の外です。
DCTRNER_ROLLBACK	-902	現在のトランザクションは，コミットできないでロールバックしました。プロセスはトランザクションの範囲の外です。
DCTRNER_HEURISTIC	-903	dc_trn_unchained_commit 関数を呼び出したグローバルトランザクションは，ヒューリスティック決定のため，あるトランザクションブランチはコミットとなり，あるトランザクションブランチはロールバックとなりました。 このリターン値は，ヒューリスティック決定の結果が，グローバルトランザクションの同期点の結果と一致しなかった場合にリターンします。 このリターン値が返る原因になった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルを参照してください。 このリターン値が返ったあと，このプロセスはトランザクション下にはありません。プロセスはグローバルトランザクションの範囲の外です。
DCTRNER_HAZARD	-904	グローバルトランザクションのトランザクションブランチがヒューリスティックに完了しました。しかし，障害のため，ヒューリスティックに完了したトランザクションブランチの同期点の結果がわかりません。 このリターン値が返る原因になった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルを参照してください。 このリターン値が返ったあと，このプロセスはトランザクション下にはありません。プロセスはグローバルトランザクションの範囲の外です。 トランザクションサービス定義で，trn_extend_function オペランドに 00000001 を指定し，1 相コミット時にリソースマネジャからのリターン値が XAER_NOTA の場合も，DCTRNER_HAZARD を返します。
DCTRNER_PROTO	-905	dc_trn_unchained_commit 関数を，正しくないコンテクスト（例えば，すでにトランザクション中にいない）で呼び出しています。トランザクションモードに対する影響はありません。

指定例

```
if(dc_trn_info(NULL) && dc_trn_unchained_commit() < 0)
  fputs("cannot commit transaction¥n",stderr);
```

dc_trn_unchained_rollback

名称

非連鎖モードのロールバック

形式

ANSI C，C++の形式

```
#include <dctrn.h>
int  dc_trn_unchained_rollback(void)
```

K&R 版 C の形式

```
#include <dctrn.h>
int  dc_trn_unchained_rollback()
```

機能

トランザクションをロールバックします。非連鎖モードでロールバックしたあとには，トランザクションは続けて開始しません。

dc_trn_unchained_rollback 関数を呼び出すことで，トランザクションブランチ，トランザクションサービス，およびリソースマネジャにロールバックを知らせます。

dc_trn_unchained_rollback 関数は，グローバルトランザクションのどのトランザクションブランチからでも呼び出せます。ルートトランザクションブランチから呼び出した場合，dc_trn_unchained_rollback 関数が正常に終了したあとには，新しいトランザクションは開始しません。ルートトランザクションブランチ以外から呼び出した場合は，そのトランザクションブランチを rollback_only 状態にします。この場合，ルートトランザクションブランチの同期点処理が完了するまで，dc_trn_unchained_rollback 関数を呼び出したトランザクションブランチはトランザクションの範囲内です。

dc_trn_unchained_rollback 関数を呼び出すプロセスは，このマニュアルの記述に従って正しく作成された UAP を稼働させたものでなければなりません。dc_trn_unchained_rollback 関数を呼び出すサービスが正常に終了するためには，UAP の実行環境を設定するときに，トランザクション属性を指定していることが前提です。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。dc_trn_unchained_rollback 関数をルートトランザクションブランチから呼び出している場合は，このプロセスはトランザクション下にありません。プロセスはグローバルトランザクションの範囲の外です。ルートトランザク

リターン値	リターン値（数値）	意味
DC_OK	0	ションブランチ以外から呼び出している場合は，このプロセスを rollback_only 状態とします。
DCTRNER_HEURISTIC	-903	dc_trn_unchained_rollback 関数を呼び出したグローバルトランザクションは，ヒューリスティック決定のため，あるトランザクションブランチはコミットとなり，あるトランザクションブランチはロールバックとなりました。 このリターン値は，ヒューリスティック決定の結果が，グローバルトランザクションの同期点の結果と一致しなかった場合にリターンします。 このリターン値が返る原因になった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルを参照してください。 このリターン値が返ったあと，このプロセスはトランザクション下になく，グローバルトランザクションの範囲の外です。
DCTRNER_HAZARD	-904	グローバルトランザクションのトランザクションブランチがヒューリスティックに完了しました。しかし，障害のため，ヒューリスティックに完了したトランザクションブランチの同期点の結果がわかりません。 このリターン値が返る原因になった UAP，リソースマネジャ，およびグローバルトランザクションの同期点の結果は，メッセージログファイルを参照してください。 このリターン値が返ったあと，このプロセスはトランザクション下になく，グローバルトランザクションの範囲の外です。
DCTRNER_PROTO	-905	dc_trn_unchained_rollback 関数を，正しくないコンテクスト（例えば，すでにトランザクション中にいない）で呼び出しています。トランザクションモードに対する影響はありません。

指定例

```
if (dc_trn_info(NULL) && dc_trn_unchained_rollback() < 0)
  fputs(”cannot rollback transaction¥n”,stderr);
```

オンラインテスタの管理（dc_uto_～）

OpenTP1 でオンラインテスタ（TP1/Online Tester）を使っている場合に，ユーザサーバから状態を保守する関数について説明します。オンラインテスタの管理の関数を次に示します。

- dc_uto_test_status － ユーザサーバのテスト状態の報告

  オンラインテスタの関数（dc_uto_～）は，TP1/Server Base の UAP でだけ使えます。TP1/LiNK の UAP では，オンラインテスタの関数は使えません。

dc_uto_test_status

名称

ユーザサーバのテスト状態の報告

形式

ANSI C，C++の形式

```
#include <dcuto.h>
int  dc_uto_test_status(struct DC_UTOSTAT *test_stat,
                        DCLONG flags)
```

K&R 版 C の形式

```
#include <dcuto.h>
int  dc_uto_test_status(test_stat,flags)
struct  DC_UTOSTAT  *test_stat;
DCLONG     flags;
```

機能

この関数を呼び出したユーザサーバのテスト状態を報告します。テスト状態は，dc_uto_test_status 関数が正常に終了したあと，引数に格納されます。

dc_uto_test_status 関数がエラーリターンした場合は，引数に格納されたテスト状態を示す情報は保証しません。

UAP で値を設定する引数

●test_stat

ユーザサーバのテスト状態を示す構造体のアドレスを設定します。

●flags

DCNOFLAGS を設定します。

OpenTP1 から値が返される引数

●test_stat

ユーザサーバのテスト状態を示す情報が，構造体で返されます。構造体の形式は次のとおりです。

```
struct  DC_UTOSTAT {
                    char  testID[5];
                    char  mode;
                    char  gbl_tran;
                    char  type;
```

```
char  svr_tran;
char  comd;
char  _res[22];
};
```

- testID

  テストユーザ ID（環境変数 DCUTOKEY に設定した値）が設定されます。

- mode

  ユーザサーバがテストモードで稼働しているかどうかが設定されます。

  DCUTO_TEST

  テストモードで稼働しています。

  DCUTO_NOTEST

  テストモードで稼働していません。

- gbl_tran

  グローバルトランザクションの処理状態が設定されます。

  DCUTO_TRN_COMMIT

  同期点処理でコミットします。

  DCUTO_TRN_ROLLBACK

  同期点処理でロールバックします。

  DCUTO_TRN_NOTRN

  非トランザクションの状態です。

  NULL（ヌル文字）

  非テストモードです。または，MCF のライブラリを結合した MHP です。

- type

  ユーザサービス定義の test_mode オペランドに指定した，テスト種別が設定されます。

  DCUTO_TEST_MODE_TARGET

  テスト専用 UAP（target）としてテストしています。

  DCUTO_TEST_MODE_USABLE

  使用可能 UAP（usable）としてテストしています。

  DCUTO_TEST_MODE_SIMMHP

  シミュレート MHP（simmhp）としてテストしています。

  DCUTO_TEST_MODE_NO

  テスト対象外 UAP（no）です。

- svr_tran

  ユーザサービス定義の test_transaction_commit オペランドで指定した，トランザクションの同期点の扱いが設定されます。

DCUTO_TRN_COMMIT
同期点でコミット（Y）します。

DCUTO_TRN_ROLLBACK
同期点でロールバック（N）します。

NULL（ヌル文字）
非テストモードです。または，MCF のライブラリを結合した MHP です。

- comd

ユーザサービス定義の test_adm_call_command オペランドで指定した，コマンドの実行結果の扱いが設定されます。

DCUTO_COMMAND_DO
コマンドを実行（do）します。

DCUTO_COMMAND_SKIP
実行結果に仮定値を設定（skip）します。

DCUTO_COMMAND_FILE
運用コマンド結果データファイルのデータを使用（file）します。

NULL（ヌル文字）
非テストモードです。または，MCF のライブラリを結合した MHP です。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。構造体 DC_UTOSTAT に示す領域にテストの状態が設定されました。
DCUTOER_PROTO	-2701	dc_rpc_open 関数を呼び出していません。
DCUTOER_TRAN	-2734	UAP が，現在稼働しているトランザクションサービスでは動作できないバージョンの OpenTP1 ライブラリと結合されています。
DCUTOER_PARAM_FLAGS	-2757	flags に設定した値が間違っています。
DCUTOER_PARAM_ADDS	-2758	test_stat に設定した値が間違っています。

注意事項

MHP から dc_uto_test_status 関数を呼び出した場合は，構造体 DC_UTOSTAT には次に示す値が設定されます。

- testID：テストユーザ ID
- mode：現在動作しているサービスのモード

- gbl_tran：ヌル文字
- type：DCUTO_TEST_MODE_NO
- svr_tran：ヌル文字
- comd：ヌル文字

3 OpenTP1 のライブラリ関数の文法（メッセージログの通知）

OpenTP1 の状態を OpenTP1 以外の製品に通知するため，メッセージログを通知できます。この章では，OpenTP1 の状態を知るためにメッセージログの通知を受信する場合に使う，OpenTP1 のライブラリ関数の文法について説明します。

メッセージログの通知（dc_log_～）

OpenTP1 の状態を知るためにメッセージログの通知を受信する場合に使う，OpenTP1 のライブラリ関数の文法について説明します。メッセージログの通知の関数を次に示します。

- dc_log_notify_close－ メッセージログ通知の受信の終了
- dc_log_notify_open－ メッセージログ通知の受信の開始
- dc_log_notify_receive－ メッセージログ通知の受信
- dc_log_notify_send－ ユーザ作成メッセージログの送信

メッセージログの通知の関数（dc_log_～）は，TP1/Server Base の場合にだけ使えます。TP1/LiNK では，メッセージログの通知の関数は使えません。

メッセージログの通知を受信できるのは，**受信用に作成したアプリケーションプログラムだけ**です。OpenTP1 の UAP（SUP，SPP，MHP）では，メッセージログを受信できません。

メッセージログの通知を受信するときの注意

メッセージログの通知を受信するときの注意を，次に示します。

1. dc_log_notify_open 関数，dc_log_notify_receive 関数，dc_log_notify_close 関数は，割り込みルーチンからは実行できません。
2. dc_log_notify_receive 関数を呼び出すタイミングによっては，受信できないメッセージログがあります。受信できないメッセージログを，次に示します。
   - アプリケーションプログラムが停止している間，またはアプリケーションプログラムが dc_log_notify_open 関数を呼び出す前か dc_log_notify_close 関数を呼び出したあとに OpenTP1 が出力したメッセージログ。
   - OpenTP1 がメッセージログを通知していても，dc_log_notify_receive 関数を呼び出していない場合に，通知されるメッセージログを退避しておく領域に空きがなくなった以降のメッセージログ。

dc_log_notify_close

名称

メッセージログ通知の受信の終了

形式

ANSI C，C++ の形式

```
#include <dclog.h>
DCLONG  dc_log_notify_close(DCLONG  flags)
```

K&R 版 C の形式

```
#include <dclog.h>
DCLONG  dc_log_notify_close(flags)
DCLONG     flags;
```

機能

OpenTP1 から通知されるメッセージログの受信を終了します。dc_log_notify_open 関数をもう一度呼び出せば，再びメッセージログの受信を開始できます。

UAP で値を設定する引数

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCLOGER_PARAM_ARGS	-1900	引数に設定した値が間違っています。
DCLOGER_PROTO	-1999	dc_log_notify_open 関数を呼び出していません。

dc_log_notify_open

名称

メッセージログ通知の受信の開始

形式

ANSI C，C++ の形式

```
#include <dclog.h>
DCLONG  dc_log_notify_open(DCLONG id,DCLONG flags)
```

K&R 版 C の形式

```
#include <dclog.h>
DCLONG  dc_log_notify_open(id,flags)
DCLONG     id;
DCLONG     flags;
```

機能

OpenTP1 から通知されるメッセージログの受信を開始します。

UAP で値を設定する引数

●id

0 を設定します。

●flags

DCNOFLAGS

ログサービス定義で，メッセージログ通知機能オペランドを使用すると設定されているかどうかをチェックしない場合に設定します。

DCLOG_CHKRTN

ログサービス定義で，メッセージログ通知機能オペランドを使用すると設定されているかどうかをチェックする場合に設定します。使用しないと設定されているときは，DCLOGER_PROTO を返します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCLOGER_PARAM_ARGS	-1900	引数に設定した値が間違っています。

リターン値	リターン値（数値）	意味
DCLOGER_COMM	-1901	通信路の初期化に失敗しました。
DCLOGER_MEMORY	-1902	メモリが不足しました。
DCLOGER_DEFFILE	-1904	システムの環境設定が間違っています。
DCLOGER_PROTO	-1999	dc_log_notify_open 関数は，すでに呼び出しています。 または，flags に DCLOG_CHKRTN が指定されている場合は，ログサービス定義がメッセージログ通知機能を使用しない指定となっています。

dc_log_notify_receive

名称

メッセージログ通知の受信

形式

ANSI C，C++ の形式

```
#include <dclog.h>
DCLONG  dc_log_notify_receive(char *msg,DCLONG msglen,
                              DCLONG timeout,DCLONG flags)
```

K&R 版 C の形式

```
#include <dclog.h>
DCLONG  dc_log_notify_receive(msg,msglen,timeout,flags)
char       *msg;
DCLONG     msglen;
DCLONG     timeout;
DCLONG     flags;
```

機能

OpenTP1 から通知されるメッセージログを受信します。dc_log_notify_receive 関数を 1 回呼び出すと，メッセージログを一つ取り出せます。

UAP で値を設定する引数

●msg

受信したメッセージログを格納する領域を設定します。ここに設定する長さは，
DCLOG_NOTIFY_MSG_LEN に設定した長さ以上の値を設定してください。

●msglen

msg に設定した領域の長さを設定します。

●timeout

メッセージログが到着しない場合の，待ち時間を秒単位で設定します。秒数は，-1～65535 の範囲で設定します。0 を設定した場合は，到着を待たないで dc_log_notify_receive 関数はリターンします。-1 を設定した場合は，メッセージログが到着するまで待ち続けます。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
0 以上の整数		msg に設定した領域に，メッセージログが正常に設定されました。0 以上の整数は，受信したメッセージログの長さを示します。
DCLOGER_PARAM_ARGS	-1900	引数に設定した値が間違っています。
DCLOGER_COMM	-1901	通信路の初期化に失敗しました。
DCLOGER_TIMEOUT	-1907	timeout に設定した秒数を超えたのに，メッセージログが通知されません。
DCLOGER_PROTO	-1999	dc_log_notify_open 関数を呼び出していません。

dc_log_notify_send

名称

ユーザ作成メッセージログの送信

形式

ANSI C，C++ の形式

```
#include <dclog.h>
DCLONG  dc_log_notify_send(char *msg,DCLONG msglen,DCLONG flags)
```

K&R 版 C の形式

```
#include <dclog.h>
DCLONG  dc_log_notify_send(msg,msglen,flags)
char       *msg;
DCLONG     msglen;
DCLONG     flags;
```

機能

OpenTP1 からのメッセージログの通知を待っているアプリケーションプログラムへ，ユーザが任意で作成したメッセージログを送信します。dc_log_notify_send 関数は，メッセージログの通知を待っているアプリケーションプログラムの終了を要求するときに使います。

UAP で値を設定する引数

●msg

送信するメッセージログを格納した領域を設定します。

●msglen

msg に設定した領域の長さを設定します。ここに設定する長さは，DCLOG_NOTIFY_MSG_LEN に設定した長さ以下の値を設定してください。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。

リターン値	リターン値（数値）	意味
DCLOGER_PARAM_ARGS	-1900	引数に設定した値が間違っています。
DCLOGER_COMM	-1901	通信路の初期化に失敗しました。
DCLOGER_PROTO	-1999	dc_log_notify_open 関数をすでに呼び出しているため，該当するアプリケーションプログラムでは dc_log_notify_send 関数を呼び出せません。

4 X/Open に準拠したアプリケーションプログラミングインタフェース

この章では，X/Open で規定するアプリケーションプログラミングインタフェースのライブラリ関数について説明します。

X/Open に準拠した関数

X/Open に準拠した関数（XATMI インタフェース，TX インタフェース）と機能の対応を表 4-1 に，OpenTP1 の UAP との関係を表 4-2 に示します。

表 4-1　X/Open に準拠した関数と機能の対応

機能	X/Open に準拠した関数名と機能	
XATMI インタフェース	tpacall()	リクエスト/レスポンス型サービスの呼び出し
	tpalloc()	型付きバッファの割り当て
	tpadvertise()	サービス名の広告
	tpcall()	リクエスト/レスポンス型サービスの呼び出しと応答の受信
	tpcancel()	リクエスト/レスポンス型サービスのキャンセル
	tpconnect()	会話型サービスとのコネクションの確立
	tpdiscon()	会話型サービスとのコネクションの切断
	tpfree()	型付きバッファの解放
	tpgetrply()	リクエスト/レスポンス型サービスからの非同期応答の受信
	tprealloc()	型付きバッファのサイズの変更
	tprecv()	会話型サービスからのメッセージの受信
	tpreturn()	サービス関数からのリターン
	tpsend()	会話型サービスへのメッセージの送信
	tpservice()	サービス関数のテンプレート
	tptypes()	型付きバッファの情報の取得
	tpunadvertise()	サービス名の広告の取り消し
TX インタフェース	tx_begin()	トランザクションの開始
	tx_close()	リソースマネジャ集合のクローズ
	tx_commit()	トランザクションのコミット
	tx_info()	現在のトランザクションに関する情報の返却
	tx_open()	リソースマネジャ集合のオープン
	tx_rollback()	トランザクションのロールバック
	tx_set_commit_return()	commit_return 特性の設定
	tx_set_transaction_control()	transaction_control 特性の設定
	tx_set_transaction_timeout()	transaction_timeout 特性の設定

表 4–2　X/Open に準拠した関数と OpenTP1 の UAP との関係

X/Open に準拠した関数	SUP		SPP			MHP		オフラインの業務をする UAP
	Trn の処理の範囲でない	Trn の処理範囲 (root)	Trn の処理の範囲でない	Trn の範囲		Trn の処理の範囲でない	Trn の処理範囲 (root)	
				root	root 以外			
tpacall	○	○	○	○	○	−	−	−
tpadvertise	−	−	○※1	○※1	○※1	−	−	−
tpalloc	○	○	○	○	○	−	−	−
tpcall	○	○	○	○	○	−	−	−
tpcancel	○	○	○	○	○	−	−	−
tpconnect	○	○	○	○	○	−	−	−
tpdiscon	○	○	○	○	○	−	−	−
tpgetrply	○	○	○	○	○	−	−	−
tpfree	○	○	○	○	○	−	−	−
tprecv	○	○	○	○	○	−	−	−
tprealloc	○	○	○	○	○	−	−	−
tpreturn	−	−	○※2	○※2	○※2	−	−	−
tpsend	○	○	○	○	○	−	−	−
tpservice※3	_※3	_※3	_※3	_※3	_※3	−	−	−
tptypes	○	○	○	○	○	−	−	−
tpunadvertise	−	−	○※1	○※1	○※1	−	−	−
tx_begin※4	○	−	○	−	−	○	−	−
tx_close	○	−	○	−	−	−	−	−
tx_commit TX_CHAINED 指定※4	−	○	○	−	−	−	−	−
tx_commit TX_UNCHAINED 指定※4	−	○	○	−	−	−	−	−
tx_info	○	○	○	○	○	−	−	−
tx_open	○	−	○	−	−	−	−	−
tx_rollback TX_CHAINED 指定※4	−	○	−	○	−	−	−	−
tx_rollback TX_UNCHAINED 指定※4	−	○	−	○	−	−	−	−

X/Open に準拠した関数	SUP		SPP			MHP		オフラインの業務をするUAP
	Trn の処理の範囲でない	Trn の処理範囲(root)	Trn の処理の範囲でない	Trn の範囲		Trn の処理の範囲でない	Trn の処理範囲(root)	
				root	root 以外			
tx_set_commit_return※4	○	○	○	○	○	－	－	－
tx_set_transaction_control※4	○	○	○	○	○	－	－	－
tx_set_transaction_timeout※4	○	○	○	○	○	－	－	－

（凡例）

Trn：トランザクション

root：ルート

○：該当する条件で呼び出せます。

－：該当する条件では呼び出せません。

注

MHP の「Trn（トランザクション）処理の範囲でない」とは，非トランザクション属性の MHP，または MHP のメイン関数の範囲を示します。

注※1

※1 で示す関数は，サービス関数の中でだけ，呼び出せます。

注※2

※2 で示す関数は，XATMI インタフェースのサービス関数をリターンするためだけに使います。

注※3

tpservice は，サービス関数の実体です。

注※4

※4 で示す関数を呼び出す UAP は，ユーザサービス定義で atomic_update=Y を指定してください。

XATMI インタフェースのアプリケーションプログラミングインタフェース（tp～）

XATMI インタフェースの API（関数）の文法について説明します。この節の記述は，X/Open 発行の「X/Open CAE Specification Distributed TP：The XATMI Specification」の文法部である「Chapter 5 C Reference Manual Pages」の記述を，日本語訳したものです。

なお，OpenTP1 の UAP で XATMI インタフェースの関数を使うときに注意する項目として追加した文章は，『 』で示します。

XATMI インタフェースの関数を次に示します。

- tpacall－ リクエスト/レスポンス型サービスの呼び出し
- tpadvertise－ サービス名の広告
- tpalloc－ 型付きバッファの割り当て
- tpcall－ リクエスト/レスポンス型サービスの呼び出しと応答の受信
- tpcancel－ リクエスト/レスポンス型サービスのキャンセル
- tpconnect－ 会話型サービスとのコネクションの確立
- tpdiscon－ 会話型サービスとのコネクションの切断
- tpfree－ 型付きバッファの解放
- tpgetrply－ リクエスト/レスポンス型サービスからの非同期応答の受信
- tprealloc－ 型付きバッファのサイズの変更
- tprecv－ 会話型サービスからのメッセージの受信
- tpreturn－ サービス関数からのリターン
- tpsend－ 会話型サービスへのメッセージの送信
- tpservice－ サービス関数のテンプレート
- tptypes－ 型付きバッファの情報の取得
- tpunadvertise－ サービス名の広告の取り消し

XATMI インタフェースの関数（tp～）は，TP1/Server Base の場合にだけ使えます。TP1/LiNK では，XATMI インタフェースの関数は使えません。

tpacall

名称

リクエスト/レスポンス型サービスの呼び出し

形式

ANSI C，C++の形式

```
#include <xatmi.h>
int  tpacall(char *svc,char *data,long len,long flags)
```

K&R 版 C の形式

```
#include <xatmi.h>
int   tpacall(svc,data,len,flags)
char  *svc;
char  *data;
long  len;
long  flags;
```

機能

関数 tpacall()は，svc で示すサービスにサービス要求のメッセージを送信します。data が NULL でない場合は，data はあらかじめ tpalloc()で割り当てたバッファを指して，len には送信するデータの長さを設定します。

Note：長さを指定する必要がないバッファ型を data が指す場合，len は無視されます（0 にしてください)。長さが必要なバッファ型を data が指す場合は，len には 0 を設定しないでください。data が NULL の場合，len は無視されて，サービス要求はデータ部がないデータで送信されます。data の type と subtype は，svc で指すサービスで認識している type と subtype のうちのどれか一つに必ず一致させてください。

Note：トランザクションモードから送信した，それぞれのサービス要求に対する応答は，最終的には必ず受信するようにしてください。

『引数』

『●svc

要求するサービスのサービス名を設定します。』

『●data

送信データ格納領域へのポインタを設定します。』

『●len

送信データの長さを設定します。』

『●flags

flags には，次に示す値を設定します。』

TPNOTRAN

呼び出し側がトランザクションモードで，かつ このフラグを設定している場合，起動された svc は，呼び出し側のトランザクションの一部にはなりません。呼び出し側がトランザクションモードで，かつ svc がトランザクション処理ができない場合は，このフラグを必ず設定してください。呼び出し側がトランザクションモードであれば，このフラグを設定していても，トランザクションタイムアウトは起こります（トランザクションモードでなければ起こりません）。このフラグで起動されたサービスが失敗しても，呼び出し側のトランザクションには影響しません。

TPNOREPLY

サービス要求に対する応答が不要であることを，tpacall()に設定します。TPNOREPLY を設定したとき，tpacall()は，正常に 0 をリターンしますが，その 0 は記述子としては意味を持ちません。呼び出し側がトランザクションモードである場合は，TPNOTRAN と一緒に設定しなければ，このフラグは使えません。

TPNOBLOCK

ブロッキング状態の場合（例えば，送信しようとするメッセージで内部バッファが満杯），サービス要求は送信されません。TPNOBLOCK が設定されていないで，ブロッキング状態のときは，呼び出し側は状況が収まるか，タイムアウト（トランザクション，またはブロッキングタイムアウト）が起こるまで，ブロッキングしています。

TPNOTIME

呼び出し側を無期限にブロックして，ブロッキングタイムアウトが起こらないことを意味します。トランザクションタイムアウトは起こります。

TPSIGRSTRT

シグナルが実行中のシステムコールを中断したら，中断したシステムコールを再び呼びます。

リターン値

成功した場合，tpacall()は送信したサービス要求の応答を受信するために使う記述子を返します。エラー時には，-1 をリターンして，tperrno にエラーの状態を示す値を設定します。

エラー

次のような場合，tpacall()はエラーリターンして，次のうちどれか一つの値を tperrno に設定します。特に示さないかぎり，失敗は呼び出し側のトランザクションに影響を与えません。

リターン値	リターン値（数値）	意味
TPEBLOCK	3	TPNOBLOCK を設定した tpacall()で送信したときに，ブロッキング状態になりました。

リターン値	リターン値（数値）	意味
TPEINVAL	4	間違った引数が与えられました（例えば，svc が NULL，data が tpalloc()に割り当てられた領域を指していない，flags に設定した値が間違っている）。
TPELIMIT	5	解決していない非同期の送信要求が最大数に達したので，呼び出し側の要求は送信されません。
TPENOENT	6	svc で示すサービスが存在しないので，送信できません。
TPEOS	7	オペレーティングシステムにエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPEPROTO	9	tpacall()が間違った状況で呼ばれました。
TPESYSTEM	12	コミュニケーションリソースマネジャシステムでエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPETIME	13	タイムアウトが起こりました。呼び出し側がトランザクションモードの場合は，これはトランザクションタイムアウトで，トランザクションは rollback_only 状態となります。そうでない場合は，TPNOBLOCK も TPNOTIME も設定されていない状態でブロッキングタイムアウトが起こったことを意味します。トランザクションタイムアウトが起こった場合は，トランザクションをロールバックするまでは，新しく送信しようとしていた送信や，解決していない応答は，TPETIME でエラーリターンします。
TPETRAN	14	svc がトランザクション処理をできないにもかかわらず，TPNOTRAN が設定されていません。
TPEGOTSIG	15	シグナルは受信されましたが，TPSIGRSTRT が設定されていません。
TPEITYPE	17	data の type と subtype が，svc で使える形式ではありません。

関連項目

tpalloc()，tpcall()，tpcancel()，tpgetrply()

『OpenTP1 で使う場合の注意事項』

1. 『該当バージョンの OpenTP1 では，TPNOBLOCK フラグは無効となります。そのため，TPEBLOCK が tperrno に返ることはありません。OpenTP1 では，ブロッキング状態のため通信ができない場合は，ネットワークダウンが原因で通信ができない場合と同様に，TPESYSTEM をリターンする仕様となっています。』
2. 『該当バージョンの OpenTP1 では，TPNOTIME フラグは無効となります。』

3.『TPSIGRSTRT フラグは無効となります。このフラグの有無に関係なく，シグナル受信時には，中断したシステムコールを再び呼びます。TPEGOTSIG がリターンすることはありません。』

4.『該当バージョンの OpenTP1 では，TPEITYPE はリターンされません。svc で使えないタイプの data を渡した場合，tpacall()は正常に終了しますが，tpgetrply()を呼び出した時点で TPESYSTEM がリターンされて，エラーであることがわかります。もし呼び出し側がトランザクションモードであれば，rollback_only 状態となります。』

5.『OpenTP1 では，トランザクションタイムアウトが起こったときには，そのプロセスは異常終了します。そのため，TPETIME がリターンされるのは，ブロッキングタイムアウトの場合だけとなります。』

6.『該当バージョンの OpenTP1 では，ロールバックする必要があるデータは，X/Open で特に指定がないかぎり，TPESYSTEM としています。ただし，TPESYSTEM がリターンしても，rollback_only 状態とならない場合もあります。』

7.『該当バージョンの OpenTP1 では，TPELIMIT はリターンされません。』

8.『TP1/NET/OSI-TP-Extended を使った OSI TP 通信をする場合，送信するデータの長さは NET/Library 共通定義の NET バッファグループ定義 nettbuf にある length オペランドに指定した値を超えないようにしてください。』

9.『次の場合，OSI TP 通信では，tpcall，または tpgetrply で TPESVCERR のエラーになり，TCP/IP 通信では TPENOENT，または TPESYSTEM のエラーになります。

- 指定したサービスが要求先に存在しない
- タイプトバッファがサーバに認識されない
- サービス起動中にエラーになる』

10.『OSI TP 通信で，システムのアソシエーション本数が足りない場合，ログメッセージを出力し，TPESYSTEM でリターンします。』

11.『OSI TP 通信では，TPNOTIME を指定しても，ブロッキングタイムアウトします。TCP/IP 通信では，トランザクションでないとき，ブロッキングタイムアウトします。』

12.『OSI TP 通信では，ユーザサービス定義の message_store_buflen は，nettbuf -g で指定したサイズ以上を定義してください。TCP/IP 通信では，dc_rpc_call と同様です。』

13.『OSI TP 通信をする XATMI のエラーは，従来の TCP/IP とエラー動作が異なる場合があります。』

tpadvertise

名称

サービス名の広告

形式

ANSI C，C++の形式

```
#include <xatmi.h>
int  tpadvertise(char *svcname,void(*func)(TPSVCINFO *))
```

K&R 版 C の形式

```
#include <xatmi.h>
int   tpadvertise(svcname,func)
char  *svcname;
void  (*func)();
```

機能

関数 tpadvertise()は，サーバで提供するサービスを広告します。この関数を呼び出さない場合，サーバのサービスは，ブート時に広告されて，シャットダウンされたときに広告を取り消されます。

関数 tpadvertise()は，svcname のサーバを広告します。svcname は 15 文字以下にしてください。ただし，NULL や NULL 文字列（""）は使えません。長過ぎる名称でも有効となりますが，15 文字に切り詰められます。ユーザは，切り詰められた名称がほかのサービス名と一致しないようにしてください。

引数の func は，サービス関数のアドレスです。この関数は，svcname の要求がサーバに受信されたら，いつでも起動されます。引数の func には，NULL を設定しないでください。

svcname がすでにそのサーバで広告されていて，func が広告されている関数と一致した場合は，tpadvertise()は正常にリターンします（切り詰められた名称が，すでに広告されていた名称と一致していた場合も含みます）。しかし，もしその svcname がすでにそのサーバで広告されていても，func が広告されている関数と一致しない場合は，エラーリターンします（切り詰められた名称が，すでに広告されている名称と一致していた場合にも起こります）。

『引数』

「●svcname

要求するサービスのサービス名を設定します。』

「● (*func) ()

サービス関数のアドレスです。』

リターン値

エラー時には，-1 をリターンして，tperrno にエラーの状態を示す値を設定します。

エラー

次のような場合，tpadvertise()はエラーリターンして，次のうちどれか一つの値を tperrno に設定します。

リターン値	リターン値（数値）	意味
TPEINVAL	4	引数 svcname が NULL か NULL 文字（""），または func に NULL が設定されています。
TPELIMIT	5	領域の制限で，引数 svcname を広告できません。
TPEOS	7	オペレーティングシステムにエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPEPROTO	9	tpadvertise()が間違った状況で呼ばれました。
TPESYSTEM	12	コミュニケーションリソースマネジャシステムでエラーが起こりました。 厳密なエラーの性質は，product-specific な方法で定義されます。
TPEMATCH	23	引数 svcname はすでにそのサーバで広告されていますが，関数は func で示されているものではありません。関数が失敗しても，svcname は，現在広告している関数を広告したままです（つまり，func は現在広告されている関数と置き換わりません）。

関連項目

tpservice(),tpunadvertise().

『OpenTP1 で使う場合の注意事項』

1. 『tpadvertise()は SPP でだけ呼び出せます。サーバの起動時には，ユーザサービス定義で指定した，すべてのサービスが自動的に広告されます。この関数のユーザサービス定義で指定してあるサービス名と関数の組み合わせだけ広告できます。』
2. 『OpenTP1 では，tpadvertise()を呼び出す UAP のサービスグループと，広告されているサービスを広告した UAP のサービスグループが同じである場合は，広告済みと見なして，関数は正常に終了します。サービスグループが一致していない場合は，関数はエラーリターンします。』
3. 『OSI TP 通信をする XATMI のエラーは，従来の TCP/IP とエラー動作が異なる場合があります。』

tpalloc

名称

型付きバッファの割り当て

形式

ANSI C，C++の形式

```
#include <xatmi.h>
char  *tpalloc(char *type,char *subtype,long size)
```

K&R 版 C の形式

```
#include <xatmi.h>
char  *tpalloc(type,subtype,size)
char  *type;
char  *subtype;
long  size;
```

機能

関数 tpalloc()は，引数 type で示す型で割り当てられたバッファへのポインタをリターンします。subtype と size は，バッファ型の範囲で任意に設定します。

使用できる subtype が何種類かあるバッファ型の場合は，tpalloc()を呼ぶときに，subtype を必ず設定してください。指定した type が subtype を持たない場合は，*subtype は無視されます（null を指定してください)。割り当てられたバッファは，少なくとも size 分の大きさになります。

Note：type の先頭 8 バイトと subtype の先頭 16 バイトだけが有効になります。

バッファ型によっては，使用する前に初期化が必要なものがあるので，tpalloc()は，バッファが割り当てられてからリターンするまでの間に，バッファを初期化します（その方法は，コミュニケーションリソースマネジャで規定)。

Note：このようにして，使える状態になってから，バッファは呼び出し側にリターンされます。バッファの初期化処理が規定されていない場合，tpalloc()はバッファを 0 に初期化しません。

『引数』

『●type

type 名を設定します。』

『●subtype

subtype 名を設定します。』

『●size

割り当てるバッファのサイズを設定します。』

リターン値

成功した場合，tpalloc()は long ワードに位置合わせした，適切な形式のバッファへのポインタをリターンします。エラー時には，NULL をリターンして，tperrno にエラーの状態を示す値を設定します。

エラー

次のような場合，tpalloc()はエラーリターンして，次のうちどれか一つの値を tperrno に設定します。

リターン値	リターン値（数値）	意味
TPEINVAL	4	間違った引数が与えられました（例えば，type が NULL）。
TPENOENT	6	type または subtype は，存在しない値です。
TPEOS	7	オペレーティングシステムにエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPEPROTO	9	tpalloc()が間違った状況で呼ばれました。
TPESYSTEM	12	コミュニケーションリソースマネジャシステムでエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。

アプリケーションの使い方

バッファの初期化処理が失敗すると，割り当てられたバッファは解放されて，tpalloc()は失敗して NULL をリターンします。

この関数は，C ライブラリの malloc()，realloc()，free()と一緒には使わないでください（例えば，tpalloc()で割り当てたバッファは，free()で解放しないでください）。

関連項目

tpfree(),tprealloc(),tptypes().

『OpenTP1 で使う場合の注意事項』

1. 『OpenTP1 では，tpalloc()がリターンしたバッファは，0 で初期化されています。』
2. 『OSI TP 通信をする XATMI のエラーは，従来の TCP/IP とエラー動作が異なる場合があります。』

tpcall

名称

リクエスト/レスポンス型サービスの呼び出しと応答の受信

形式

ANSI C，C++の形式

```
#include <xatmi.h>
int  tpcall(char *svc,char *idata,long ilen,char **odata,
            long *olen,long flags)
```

K&R 版 C の形式

```
#include <xatmi.h>
int  tpcall(svc,idata,ilen,odata,olen,flags)
char    *svc;
char    *idata;
long    ilen;
char    **odata;
long    *olen;
long    flags;
```

機能

関数 tpcall()は，サービス要求を送信して，同期的に応答を待ちます。この関数を呼ぶことは，tpacall()を呼び出して，すぐに tpgetrply()を呼び出すのと同じことです。関数 tpcall()は svc で示されたサービスにサービス要求を送信します。サービス要求のデータ部は，idata で指しています。バッファは，事前に tpalloc()で割り当てておきます。引数 ilen には，idata をどれだけ送信するかを設定します。

Note：idata が，長さを指定する必要がないバッファ型を指す場合は，ilen は無視されます（0 にしてください)。長さが必要なバッファ型を idata が指す場合は，ilen には 0 を設定しないでください。idata が NULL の場合も，ilen は無視されます。idata の type と subtype は，svc で認識できる type と subtype のうちのどれか一つに必ず一致させてください。

odata は応答が読み込まれるバッファへのポインタのアドレスであり，その応答の長さは*olen にリターンされます。*odata は，tpalloc()であらかじめ割り当てたバッファを必ず指すようにしてください。送信と受信に同じバッファを使用する場合は，odata に idata のアドレスを入れておいてください。

応答のバッファのサイズを変更したかどうかの判断は，tpcall()を呼び出す前のバッファの合計サイズと*olen の長さを比較します。*olen の方が長い場合は，バッファは大きくなっています。長くない場合は，バッファのサイズは変わりません。

また，tpcall()が呼び出されたときに，idata と*odata が等しくても，*odata が変更されたときは，idata が指しているアドレスは無効です。

Note：*odata はバッファのサイズを増やす以外の理由でも変わるかも知れません。*olen に 0 が返ってきたら，応答にはデータ部がなくて，*odata とそれが指すバッファは変更されません。*odata や olen が NULL の場合はエラーになります。

『引数』

『●svc

要求するサービスのサービス名を設定します。』

『●idata

送信バッファへのポインタを設定します。』

『●ilen

送信バッファの長さを設定します。』

『●odata

応答データを格納するバッファへのポインタのアドレスを設定します。』

『●olen

応答バッファの長さを示す long 型へのポインタを示します。』

『●flags』

flags には，次に示す値を設定します。

TPNOTRAN

呼び出し側がトランザクションモードで，かつこのフラグを設定している場合，起動された svc は，呼び出し側のトランザクションの一部にはなりません。呼び出し側がトランザクションモードで，かつ svc がトランザクション処理ができない場合は，このフラグを必ず設定してください。呼び出し側がトランザクションモードであれば，このフラグを設定していても，トランザクションタイムアウトは起こります（トランザクションモードでなければ起こりません）。このフラグで起動されたサービスが失敗しても，呼び出し側のトランザクションには影響しません。

TPNOCHANGE

バッファが*odata で指しているバッファ型と異なる型のデータを受信したとき，このフラグを設定していなければ，*odata のバッファ型は受け取り側がそのバッファ型を認識しているかぎり，受信したデータのバッファ型に変換されます。このフラグを設定した場合は，*odata で指しているバッファ型は変換されません。つまり，受信したバッファの type，subtype，および*odata で指しているバッファの type と subtype は必ず一致させてください。

TPNOBLOCK

ブロッキング状態の場合（例えば，送信されたメッセージで内部バッファが満杯），サービス要求は送信されません。

Note：このフラグは，tpcall()の送信時だけ有効となります。この関数は，応答を待っているときはブロックします。TPNOBLOCK が設定されていないで，ブロッキング状態のときは，呼び出し側は状況が収まるか，タイムアウト（トランザクション，またはブロッキングタイムアウト）が起こるまで，ブロッキングしています。

TPNOTIME

呼び出し側を無期限にブロックして，ブロッキングタイムアウトが起こらないことを意味します。トランザクションタイムアウトは起こります。

TPSIGRSTRT

シグナルが実行中のシステムコールを中断したら，中断したシステムコールを再び呼びます。

リターン値

tpcall()が正常リターンするか，または TPESVCFAIL が tperrno に設定されてリターンしたとき，アプリケーションが tpreturn()の引数として渡した定義値は，グローバル変数 tpurcode として参照できます。エラー時には，-1 をリターンして，tperrno にエラーの状態を示す値を設定します。

エラー

次のような場合，tpcall()はエラーリターンして，次のうちどれか一つの値を tperrno に設定します。特に示さないかぎり，呼び出し側のトランザクションがあった場合でも，失敗は呼び出し側のトランザクションに影響を与えません。

リターン値	リターン値（数値）	意味
TPEBLOCK	3	TPNOBLOCK を設定した tpcall()で送信したときに，ブロッキング状態になりました。
TPEINVAL	4	間違った引数が与えられました（例えば，svc が NULL か，または flags に設定した値が間違っている）。
TPENOENT	6	svc で示すサービスが存在しないので，送信できません。
TPEOS	7	オペレーティングシステムにエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPEPROTO	9	tpcall()が間違った状況で呼ばれました。
TPESVCERR	10	このエラーは，サービス関数を呼び出しているときか，tpreturn()で完了するときに起こります（例えば，適切でない引数を設定した）。このエラーが起こったときは，応答のデータはリターンされません（つまり，*odata も*olen も変わっていません）。サービスに応答する処理が呼び出し側のトランザクションとして実行された場合は，トランザクションは rollback_only 状態になります。 注 トランザクションタイムアウトにならないかぎり，これ以降の通信は，ロールバックするまでの処理が実行されます。

リターン値	リターン値（数値）	意味
TPESVCERR	10	この処理は正常になるか，または無効になります（エラーリターンまたはイベントが通知されます）。継続する処理を有効にする通信には，TPNOTRAN を設定してください。トランザクション機能によって，呼び出し側のトランザクションをロールバックする何らかの処理が実行されます。
TPESVCFAIL	11	呼び出し側の応答を送信するサービス関数が，TPFAIL を設定した tpreturn()を呼び出しました。これは，アプリケーションレベルの失敗です。送信されたサービスの応答の内容は，*odata で指すバッファとして有効になります。サービス要求が呼び出し側の現在のトランザクションとして実行されている場合は，トランザクションは rollback_only 状態となります。 注 トランザクションがタイムアウトしないかぎり，それ以降の通信は，ロールバックする前までは実行されて，呼び出し側のトランザクションとして実行されたこれらの処理は，トランザクションの完了時にロールバックします。継続する処理を有効にする通信には，TPNOTRAN を設定してください。トランザクション機能によって，呼び出し側のトランザクションをロールバックする何らかの処理が実行されます。
TPESYSTEM	12	コミュニケーションリソースマネジャシステムでエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPETIME	13	タイムアウトが起こりました。呼び出し側がトランザクションモードの場合，これはトランザクションタイムアウトで，トランザクションは rollback_only 状態となります。そうでない場合は，TPNOBLOCK も TPNOTIME も設定されていない状態で，ブロッキングタイムアウトが起こったことを意味します。どちらの場合も，*odata も*olen も変わっていません。トランザクションタイムアウトが起こった場合は，トランザクションをロールバックするまでは，新しく送信しようとしていた送信や，解決していない応答は，TPETIME でエラーリターンします。
TPETRAN	14	svc がトランザクション処理をできないにもかかわらず，TPNOTRAN が設定されていません。
TPEGOTSIG	15	シグナルは受信されましたが，TPSIGRSTRT が設定されていません。
TPEITYPE	17	idata の type と subtype が，svc で使える形式ではありません。
TPEOTYPE	18	応答の type と subtype を，呼び出し側で認識できません。または，TPNOCHANGE のフラグが設定されているのに，*odata で指しているバッファの type と subtype が，サービスから送信されてきた応答のバッファの type と subtype に一

リターン値	リターン値（数値）	意味
TPEOTYPE	18	致していません。このエラーが起こった場合，*odata と*olen は変更されません。 呼び出し側のトランザクションとしてサービス要求が実行されている場合は，応答を捨てるまで，トランザクションは rollback_only 状態となります。

関連項目

tpalloc()，tpacall()，tpgetrply()，tpreturn().

『OpenTP1 で使う場合の注意事項』

1. 『該当バージョンの OpenTP1 では，TPNOBLOCK フラグは無効となります。そのため，TPEBLOCK が tperrno にリターンすることはありません。OpenTP1 では，ブロッキング状態のため通信ができない場合は，ネットワークダウンが原因で通信ができない場合と同様に，TPESYSTEM をリターンする仕様となっています。』
2. 『該当バージョンの OpenTP1 では，TPNOTIME フラグは応答受信時にだけ有効です。サービス要求の送信時にブロッキング状態が起こった場合は，TPNOTIME フラグは無効です。』
3. 『TPSIGRSTRT フラグは無効となります。このフラグの有無に関係なく，シグナル受信時には，中断したシステムコールを再び呼びます。TPEGOTSIG がリターンすることはありません。』
4. 『該当バージョンの OpenTP1 では，TPEITYPE はリターンされません。svc で使えないタイプの data を渡した場合，TPESYSTEM がリターンされます。もし呼び出し側がトランザクションモードの場合は，rollback_only 状態となります。』
5. 『OpenTP1 では，トランザクションタイムアウトが起こったときには，そのプロセスは異常終了します。そのため，TPETIME がリターンされるのは，ブロッキングタイムアウトの場合だけとなります。』
6. 『該当バージョンの OpenTP1 では，ロールバックする必要があるエラーは，X/Open で特に指定がないかぎり，TPESYSTEM としています。ただし，TPESYSTEM がリターンしても，rollback_only 状態とならない場合もあります。
7. 『サービス要求先の SPP が異常終了した場合，定義の watch_time に指定した時間よりも早く，TPETIME でエラーリターンするときがあります。また，watch_time に 0（応答を受信するまで無限に待つ）を指定している場合，TPEPROTO でエラーリターンするときがあります。』
8. 『OpenTP1 のセキュリティ機能を使っている場合で，サービス要求が認証されなかったときは，TPEPROTO でエラーリターンします。サービス要求が認証されなかったことが原因かどうかは，UAP トレースの詳細エラーコードで確認してください。』
9. 『TP1/NET/OSI-TP-Extended を使った OSI TP 通信では，回線障害は TPESVCERR でエラーリターンします。』

10.『TP1/NET/OSI-TP-Extended を使った OSI TP 通信をする場合，送受信するデータの長さは NET/Library 共通定義の NET バッファグループ定義 nettbuf にある length オペランドに指定した値を超えないようにしてください。』

11.『次の場合，OSI TP 通信では，tpcall，または tpgetrply で TPESVCERR のエラーになり，TCP/IP 通信では TPENOENT，または TPESYSTEM のエラーになります。

- 指定したサービスが要求先に存在しない
- タイプトバッファがサーバに認識されない
- サービス起動中にエラーになる』

12.『OSI TP 通信で，システムのアソシエーション本数が足りない場合，ログメッセージを出力し，TPESYSTEM でリターンします。』

13.『OSI TP 通信では，TPNOTIME を指定しても，ブロッキングタイムアウトします。TCP/IP 通信では，トランザクションでないとき，ブロッキングタイムアウトします。』

14.『OSI TP 通信では，ユーザサービス定義の message_store_buflen は，nettbuf -g で指定したサイズ以上を定義してください。TCP/IP 通信では，dc_rpc_call と同様です。』

15.『TP1 間で OSI TP 通信を行う場合で，atomic_update オペランドが'N'のサービスをトランザクションとして呼ばれる場合，サービス要求元には，TPESVCERR が返ります。』

16.『TCP/IP 通信で呼び出されたサービスから，さらに OSI TP 通信でサービスを呼び出し，応答を受け取らないでサービス関数を終了した場合，TCP/IP 通信でサービスを呼び出した UAP には正常な応答電文が返り，tpcall()，tpgetrply()はエラーリターンしません。』

17.『OSI TP 通信をする XATMI のエラーは，従来の TCP/IP とエラー動作が異なる場合があります。』

tpcancel

名称

リクエスト/レスポンス型サービスのキャンセル

形式

ANSI C，C++の形式

```
#include <xatmi.h>
int  tpcancel(int cd)
```

K&R 版 C の形式

```
#include <xatmi.h>
int  tpcancel(cd)
int     cd;
```

機能

関数 tpcancel()は，関数 tpacall()のリターン値である記述子 cd をキャンセルします。グローバルトランザクションに属している記述子をキャンセルしようとすると，エラーになります。

成功した場合，cd は無効となって，cd として受信した応答は，暗黙的に捨てられます（コミュニケーションリソースマネジャによって）。

『引数』

『●cd

記述子を設定します。』

リターン値

エラー時には，-1 をリターンして，tperrno にエラーの状態を示す値を設定します。

エラー

次のような場合，tpcancel()はエラーリターンして，次のうちどれか一つの値を tperrno に設定します。

リターン値	リターン値（数値）	意味
TPEBADDESC	2	引数 cd は，間違った記述子です。
TPEOS	7	オペレーティングシステムにエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。

リターン値	リターン値（数値）	意味
TPEPROTO	9	tpcancel()が間違った状況で呼ばれました。
TPESYSTEM	12	コミュニケーションリソースマネジャシステムでエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPETRAN	14	引数 cd が，呼び出し側のグローバルトランザクションに参加しています。エラーのあとでも，引き続き記述子 cd は有効で，呼び出し側の現在のトランザクションには影響しません。

関連項目

tpacall().

『OpenTP1 で使う場合の注意事項』

1.『OSI TP 通信をする XATMI のエラーは，従来の TCP/IP とエラー動作が異なる場合があります。』

tpconnect

名称

会話型サービスとのコネクションの確立

形式

ANSI C，C++の形式

```
#include <xatmi.h>
int  tpconnect(char *svc,char *data,long len,long flags)
```

K&R 版 C の形式

```
#include <xatmi.h>
int  tpconnect(svc,data,len,flags)
char    *svc;
char    *data;
long    len;
long    flags;
```

機能

関数 tpconnect()は，会話型サービスである svc に，半二重型の通信路を設定します。コネクションを確立する処理の一部として，呼び出し側はデータを受け取り側のサービス関数へ渡せます。呼び出し側がデータを渡す場合は，data は tpalloc()で割り当てたバッファを指しておいてください。len にはどれぐらいのバッファを送信するかを設定します。

Note：data が，長さを指定する必要がないバッファ型を指す場合は，len は無視されます（0 にしてください）。長さが必要なバッファ型を data が指す場合は，len には 0 を設定しないでください。また，data は NULL を設定することもできます。その場合も len は無視されます（会話型サービスにはアプリケーションデータは送信されません）。data の type と subtype は，svc で認識する type と subtype のうちのどれか一つに必ず一致させてください。

会話型サービスが正常に開始すると，TPSVCINFO を経由して data と len を受信します。そのため，会話型サービスは，tpconnect()から送信された data を受信するために tprecv()を呼ぶ必要はありません。

『引数』

「●svc

要求するサービスのサービス名を設定します。』

「●data

送信データ格納領域へのポインタを設定します。』

『●len

送信データの長さを設定します。』

『●flags』

flags には，次に示す値を設定します。

TPNOTRAN

呼び出し側がトランザクションモードで，かつこのフラグを設定している場合，起動された svc は，呼び出し側のトランザクションの一部にはなりません。呼び出し側がトランザクションモードで，かつ svc がトランザクション処理ができないサーバに属している場合は，このフラグを必ず設定してください。呼び出し側がトランザクションモードであれば，このフラグが設定されていても，トランザクションタイムアウトは起こります（トランザクションモードでなければ起こりません）。このフラグで起動されたサービスが失敗しても，呼び出し側のトランザクションには影響しません。

TPSENDONLY

呼び出し側がデータを送信して，呼ばれたサービスがデータの受信だけできるように，コネクションを最初に確立します（つまり，呼び出し側が最初にコネクションの制御権を得ます）。TPSENDONLY または TPRECVONLY は，どちらかを必ず設定してください。

TPRECVONLY

呼び出し側はデータを受信するだけで，呼ばれたサービスがデータの送信だけできるように，コネクションを最初に確立します（つまり，呼び出されたサービス側が最初にコネクションの制御権を得ます）。TPSENDONLY または TPRECVONLY は，どちらかを必ず設定してください。

TPNOBLOCK

ブロッキング状態の場合（例えば，送信されたメッセージで内部バッファが満杯），コネクションは確立されないで，データは送信されません。TPNOBLOCK が設定されていないで，ブロッキング状態のときは，呼び出し側は状況が収まるか，タイムアウト（トランザクション，またはブロッキングタイムアウト）が起こるまで，ブロッキングしています。

TPNOTIME

呼び出し側を無期限にブロックして，ブロッキングタイムアウトが起こらないことを意味します。トランザクションタイムアウトは起こります。

TPSIGRSTRT

シグナルが実行中のシステムコールを中断したら，中断したシステムコールを再び呼びます。

リターン値

成功した場合，tpconnect()は，そのあとでこのコネクションを指定するために使われる記述子をリターンします。エラー時には，-1 をリターンして，tperrno にエラーの状態を示す値を設定します。

エラー

次のような場合，tpconnect()はエラーリターンして，次のうちどれか一つの値を tperrno に設定します。特に示さないかぎり，呼び出し側のトランザクションがあった場合でも，失敗は呼び出し側のトランザクションに影響を与えません。

リターン値	リターン値（数値）	意味
TPEBLOCK	3	TPNOBLOCK を設定した tpconnect()を呼び出したときに，ブロッキング状態になりました。
TPEINVAL	4	間違った引数が与えられました（例えば，svc が NULL，data が NULL でなく tpalloc()で確保したバッファへのポインタでない，TPSENDONLY または TPRECVONLY が flags に設定されていない，flags に設定した値が間違っている）。
TPELIMIT	5	未解決のコネクションが最大数に達したので，呼び出し側の要求は送信されません。
TPENOENT	6	svc で示すサービスが存在しないので，コネクションを確立できません。
TPEOS	7	オペレーティングシステムにエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPEPROTO	9	tpconnect()が間違った状況で呼ばれました。
TPESYSTEM	12	コミュニケーションリソースマネジャシステムでエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPETIME	13	タイムアウトが起こりました。呼び出し側がトランザクションモードの場合，これはトランザクションタイムアウトで，トランザクションは rollback_only 状態となります。そうでない場合は，TPNOBLOCK も TPNOTIME も設定されていない状態で，ブロッキングタイムアウトが起こったことを意味します。トランザクションタイムアウトが起こった場合は，トランザクションをロールバックするまでは，どのコネクションでのどのような送信，受信の試みも，TPETIME でエラーリターンします。
TPETRAN	14	svc がトランザクション処理をできないにもかかわらず，TPNOTRAN が設定されていません。
TPEGOTSIG	15	シグナルは受信されましたが，TPSIGRSTRT が設定されていません。
TPEITYPE	17	data の type と subtype が，svc で使える形式ではありません。

関連項目

tpcall()，tpdiscon()，tprecv()，tpsend()，tpservice()

『OpenTP1 で使う場合の注意事項』

1. 『該当バージョンの OpenTP1 では，TPNOBLOCK フラグは無効となります。そのため，TPEBLOCK が tperrno にリターンすることはありません。OpenTP1 では，ブロッキング状態のため通信ができない場合は，ネットワークダウンが原因で通信ができない場合と同様に，TPESYSTEM をリターンする仕様となっています。』
2. 『該当バージョンの OpenTP1 では，TPNOTIME フラグは無効となります。』
3. 『TPSIGRSTRT フラグは無効となります。このフラグの有無に関係なく，シグナル受信時には，中断したシステムコールを再び呼びます。TPEGOTSIG がリターンすることはありません。』
4. 『該当バージョンの OpenTP1 では，TPEITYPE はリターンされません。svc で使えないタイプの data を渡した場合，TPESYSTEM がリターンされます。もし呼び出し側がトランザクションモードの場合は，rollback_only 状態となります。』
5. 『OpenTP1 では，トランザクションタイムアウトが起こったときには，そのプロセスは異常終了します。そのため，TPETIME がリターンされるのは，ブロッキングタイムアウトの場合だけとなります。』
6. 『該当バージョンの OpenTP1 では，ロールバックする必要があるエラーは，X/Open で特に指定がないかぎり，TPESYSTEM としています。ただし，TPESYSTEM がリターンしても，rollback_only 状態とならない場合もあります。』
7. 『OpenTP1 のセキュリティ機能を使っている場合で，サービス要求が認証されなかったときは，TPEPROTO でエラーリターンします。サービス要求が認証されなかったことが原因かどうかは，UAP トレースの詳細エラーコードで確認してください。』
8. 『TP1/NET/OSI-TP-Extended を使った OSI TP 通信をする場合は，会話型サービスの通信はできません。OSI TP 通信で会話型サービスの通信を使った場合，システムの動作は保証しません。』
9. 『サーバ AP が閉塞状態の場合は，ローカルノードの要求先 SPP が閉塞しているか，リモートノードの要求先 SPP が閉塞しているかによって，次のように動作します。

   ローカルノードの要求先 SPP が閉塞しているとき

   tpconnect()は-1 を返し，tperrno に TPEPROTO が設定されます。

   リモートノードの要求先 SPP が閉塞しているとき

   トランザクションモードであれば，トランザクションタイムアウトでサーバ AP が異常終了します。

   非トランザクションモードであれば，tpconnect()は-1 を返し，tperrno に TPETIME が設定されます。』

tpdiscon

名称

会話型サービスとのコネクションの切断

形式

ANSI C，C++形式

```
#include <xatmi.h>
int  tpdiscon(int cd)
```

K&R 版 C の形式

```
#include <xatmi.h>
int  tpdiscon(cd)
int     cd;
```

機能

関数 tpdiscon()は，記述子 cd で示すコネクションをすぐに切断して，他端点のコネクションに TPEV_DISCONIMM のイベントを通知します。

関数 tpdiscon()は，会話を始めた側（オリジネータ）からだけしか呼び出せません。会話型サービス内でこのサービスを呼び出した記述子では，tpdiscon()は呼び出せません。むしろ，会話型サービスは，会話の一方が完了したことを意味するために tpreturn()を使わなければなりません。同様に，会話型サービスと会話しているプログラムは tpdiscon()を使えますが，結果を確実にするために望ましい方法は，サービスに tpreturn()でコネクションを終わらせてもらうことです。

関数 tpdiscon()は，コネクションをすぐに切断します（正常終了ではなく異常終了として）。送信先に届いていないデータは捨てられます。コネクションの他端点のプログラムが，呼び出し側のトランザクションに参加していても，tpdiscon()は使えます。この場合，トランザクションはロールバックします。また，tpdiscon()が呼ばれたとき，呼び出し側は，コネクションの制御権を持っている必要はありません。

『引数』

『●cd

記述子を設定します。』

リターン値

エラー時には，tpdiscon()は-1 をリターンして，tperrno にエラーの状態を示す値を設定します。

エラー

次のような場合，tpdiscon()はエラーリターンして，次のうちどれか一つの値を tperrno に設定します。

リターン値	リターン値（数値）	意味
TPEBADDESC	2	引数 cd が間違っているか，または，呼び出された会話型サービスの記述子です。
TPEOS	7	オペレーティングシステムにエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPEPROTO	9	tpdiscon()が間違った状況で呼ばれました。
TPESYSTEM	12	コミュニケーションリソースマネジャシステムでエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPETIME	13	タイムアウトが起こりました。この記述子は無効になります。

関連項目

tpconnect()，tprecv()，tpreturn()，tpsend()

『OpenTP1 で使う場合の注意事項』

1. 『該当バージョンの OpenTP1 では，TPETIME が tperrno にリターンすることはありません。』
2. 『TP1/NET/OSI-TP-Extended を使った OSI TP 通信をする場合は，会話型サービスの通信はできません。OSI TP 通信で会話型サービスの通信を使った場合，システムの動作は保証しません。』

tpfree

名称

型付きバッファの解放

形式

ANSI C，C++の形式

```
#include <xatmi.h>
void  tpfree(char *ptr)
```

K&R 版 C の形式

```
#include <xatmi.h>
void  tpfree(ptr)
char    *ptr;
```

機能

関数 tpfree()の引数は，以前に tpalloc()や tprealloc()で取得したバッファへのポインタです。ptr が NULL の場合は，何もしません。ptr が型付きバッファへのポインタでない場合（または，tpfree()で，事前に解放されていた場合），関数を呼び出した結果は不確定です。サービス関数内では，ptr がサービス関数に渡されたバッファを指す場合，tpfree()はバッファを解放しないでリターンします。

情報を要求していたり，データと結び付いているバッファ型は，バッファの解放と同時にそれらも削除されます。tpfree()は，バッファが解放される前にこれらの結び付きを削除します（その方法は，コミュニケーションリソースマネジャで規定）。

いったん tpfree()がリターンしたあとで，ptr を引数として XATMI 関数に渡したり，ほかの定義に使ったりしないでください。

『引数』

『●ptr

tpalloc()や tprealloc()で割り当てたバッファへのポインタを設定します。』

リターン値

tpfree()は，呼び出し元には何も値を返しません。そのため，void 型の宣言となっています。

アプリケーションの使い方

この関数は，C ライブラリの malloc()，realloc()，free()と一緒には使わないでください（例えば，tpalloc()で割り当てたバッファは，free()で解放しないでください）。

関連項目

tpalloc()，tprealloc()

『OpenTP1 で使う場合の注意事項』

1.『OSI TP 通信をする XATMI のエラーは，従来の TCP/IP とエラー動作が異なる場合があります。』

tpgetrply

名称

リクエスト/レスポンス型サービスからの非同期応答の受信

形式

ANSI C，C++の形式

```
#include <xatmi.h>
int  tpgetrply(int *cd,char **data,long *len,long flags)
```

K&R 版 C の形式

```
#include <xatmi.h>
int  tpgetrply(cd,data,len,flags)
int     *cd;
char    **data;
long    *len;
long    flags;
```

機能

関数 tpgetrply()は，先に送信したサービス要求の応答をリターンします。この関数の第 1 引数の cd は，tpacall()がリターンした記述子へのポインタです。フラグに指定がなければ，この関数は応答が*cd と一致するか，タイムアウトが起こるまで待ちます。

data は，tpalloc()で事前に割り当てたバッファへのポインタのアドレスを設定してください。len は必ず long 型を指定してください。これには，tpgetrply()が成功したときに受信したデータ長が設定されます。tpgetrply()は，必要に応じてバッファを確保し直します。成功した場合は，*data は応答を含むバッファのポインタで，*len にはデータのサイズが入っています。

Note：*data は，バッファのサイズが大きくなったことでリターン時に変わることがあります。*len が以前に呼んだバッファの合計サイズよりも大きかった場合は，新しいバッファのサイズは*len です。*len が 0 の場合は，受信する必要があるデータはなく，*data やバッファへのポインタは変更されません。*data や len が NULL である場合は，エラーになります。

『引数』

『●cd

記述子を設定します。』

『●data

受信したデータを格納するバッファへのポインタのアドレスを設定します。』

『●len

受信したデータの長さを格納する領域のアドレスを設定します。』

『●flags』

flags には，次に示す値を設定します。

TPGETANY

このフラグは，tpgetrply()が cd で指した記述子を無視して，受信できる何らかの応答をリターンして，返ってきた応答の記述子を cd に設定することを意味します。応答がなくて，このフラグを設定していない場合は，tpgetrply()は応答の到着を待ちます。

TPNOCHANGE

*data で示されたバッファとは異なるタイプをバッファに受信したとき，このフラグを設定していなければ，*data のバッファ型は，受け取り側が到着したバッファ型を認識しているかぎり，受信したデータのバッファ型に変換されます。このフラグを設定した場合は，*data で指しているバッファのタイプは変換されません。つまり，受信したバッファの type と subtype と，*data で指しているバッファの type と subtype は必ず一致させてください。

TPNOBLOCK

tpgetrply()は応答の到着を待ちません。応答がすぐに受信できる状態の場合は，tpgetrply()は応答とリターン値を得ます。このフラグを設定していない場合，tpgetrply()発行時に応答が到着していなければ，呼び出し側は応答が到着するか，またはタイムアウト（トランザクション，またはブロッキングタイムアウト）が起こるまで，ブロッキングしています。

TPNOTIME

呼び出し側を無期限にブロックして，ブロッキングタイムアウトが起こらないことを意味します。トランザクションタイムアウトは起こります。

TPSIGRSTRT

シグナルが実行中のシステムコールを中断したら，中断したシステムコールを再び呼びます。

以下，特に記述しないかぎり，応答を受信したあとは，*cd は無効になります。

リターン値

tpgetrply()が正常リターンするか，または tperrno に TPESVCFAIL が設定されてリターンしたとき，アプリケーションが tpreturn()の引数として渡した定義値は，グローバル変数 tpurcode として参照できます。エラー時には，-1 をリターンして，tperrno にエラーの状態を示す値を設定します。

エラー

次のような場合，tpgetrply()はエラーリターンして，次のうちどれか一つの値を tperrno に設定します。TPGETANY を flags に設定していない場合，特に述べなければ，*cd は無効になります。TPGETANY を flags に設定している場合は，*cd はエラーになった応答の記述子を示します。応答が戻ってくる前にエラーが起こった場合は，*cd は 0 を指します。特に示さないかぎり，呼び出し側がトランザクション下にあった場合でも，失敗は呼び出し側のトランザクションに影響を与えません。

リターン値	リターン値（数値）	意味
TPEBADDESC	2	引数 cd が間違った記述子を指しています。
TPEBLOCK	3	TPNOBLOCK を設定していたときに，ブロッキング状態になりました。*cd は有効なままです。
TPEINVAL	4	間違った引数が与えられました（例えば，cd，data，*data，len が NULL，flags に設定した値が間違っている）。cd が NULL でない場合は，このエラーのあとも引き続き有効で，応答は解決していない状態のままです。
TPEOS	7	オペレーティングシステムにエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPEPROTO	9	tpgetrply()が間違った状況で呼ばれました。
TPESVCERR	10	このエラーは，サービス関数を呼び出しているときか，tpreturn()で完了したときに起こります（例えば，適切でない引数を設定した）。このエラーが起こったときは，応答のデータはリターンされません（つまり，*data も*len も変更されません）。サービスの応答が呼び出し側のトランザクションとして実行された場合は，トランザクションは rollback_only 状態になります。 注 トランザクションがタイムアウトしないかぎり，それ以降の通信は，ロールバックする前までは実行されます。この処理は正常になるか，または無効になります（エラーリターンまたはイベントが通知されます）。継続する処理を有効にする通信には，TPNOTRAN を設定してください。トランザクション機能によって呼び出し側のトランザクションをロールバックする何らかの処理が実行されます。
TPESVCFAIL	11	呼び出し側の応答を送信するサービス関数が，TPFAIL を設定した tpreturn()を呼び出しました。これは，アプリケーションレベルの失敗です。送信されたサービスの応答の内容は，*data で指すバッファとして有効になります。サービス要求が呼び出し側の現在のトランザクションとして実行されている場合は，トランザクションは rollback_only 状態となります。 注 トランザクションがタイムアウトしないかぎり，それ以降の通信は，ロールバックする前までは実行されて，呼び出し側のトランザクションとして実行されたこれらの処理は，トランザクションの完了時にロールバックします。継続する処理を有効にする通信には，TPNOTRAN を設定してください。トランザクション機能によって呼び出し側のトランザクションをロールバックする何らかの処理が実行されます。

リターン値	リターン値（数値）	意味
TPESYSTEM	12	コミュニケーションリソースマネジャシステムでエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPETIME	13	タイムアウトが起こりました。呼び出し側がトランザクションモードの場合，これはトランザクションタイムアウトで，トランザクションは rollback_only 状態となります。そうでない場合は，TPNOBLOCK も TPNOTIME も設定されていない状態で，ブロッキングタイムアウトが起こったことを意味します。呼び出し側がトランザクションモードでないかぎり（そして，TPGETANY が設定されていないかぎり），引数*cd は有効です。どちらの場合も，*data も*len も変わっていません。トランザクションタイムアウトが起こった場合は，トランザクションをロールバックするまでは，新しく送信しようとしていた送信や，解決していない応答は，TPETIME でエラーリターンします。
TPEGOTSIG	15	シグナルは受信されましたが，TPSIGRSTRT が設定されていません。
TPEOTYPE	18	応答の type と subtype を，呼び出し側で認識できません。または，TPNOCHANGE のフラグが設定されているのに，*data で指しているバッファの type と subtype が，サービスから送信されてきた応答のバッファの type と subtype に一致していません。このエラーが起こった場合，*data も*len も変わっていません。呼び出し側のトランザクションとしてサービス要求が実行されている場合，応答は捨てられて，トランザクションは rollback_only 状態となります。

関連項目

tpacall()，tpalloc()，tpreturn()

『OpenTP1 で使う場合の注意事項』

1. 『TPSIGRSTRT フラグは無効となります。このフラグの有無に関係なく，シグナル受信時には，中断したシステムコールを再び呼びます。TPEGOTSIG がリターンすることはありません。』
2. 『OpenTP1 では，トランザクションタイムアウトが起こったときには，そのプロセスは異常終了します。そのため，TPETIME がリターンされるのは，ブロッキングタイムアウトの場合だけとなります。』
3. 『該当バージョンの OpenTP1 では，ロールバックする必要があるデータは，X/Open で特に指定がないかぎり，TPESYSTEM としています。ただし，TPESYSTEM がリターンしても，rollback_only 状態とならない場合もあります。』
4. 『tpacall()で，呼び出したサービスが使えないタイプのデータをサービスに渡していた場合，tpacall()は正常に終了して，tpgetrply()でエラーとなります。tpgetrply()で TPESYSTEM のエラーが起こった場合は，tpacall()の実行結果も確認してください。』

5.『サービス要求先の SPP が異常終了した場合，定義の watch_time オペランドに指定した時間よりも早く，TPETIME でエラーリターンする場合があります。また，watch_time オペランドに 0（応答を受信するまで無限に待つ）を指定しているときは，TPEPROTO でエラーリターンする場合があります。』

6.『OpenTP1 のセキュリティ機能を使っている場合で，サービス要求が認証されなかったときは，TPEPROTO でエラーリターンします。サービス要求が認証されなかったことが原因かどうかは，UAP トレースの詳細エラーコードで確認してください。』

7.『TP1/NET/OSI-TP-Extended を使った OSI TP 通信をする場合，受信するデータの長さは NET/Library 共通定義の NET バッファグループ定義 nettbuf にある length オペランドに指定した値を超えないようにしてください。』

8.『TP1 間で OSI TP 通信を行う場合で，atomic_update オペランドが'N'のサービスをトランザクションとして呼ばれる場合，サービス要求元には，TPESVCERR が返ります。』

9.『TCP/IP 通信で呼び出されたサービスから，さらに OSI TP 通信でサービスを呼び出し，応答を受け取らないでサービス関数を終了した場合，TCP/IP 通信でサービスを呼び出した UAP には正常な応答電文が返り，tpcall()，tpgetrply()はエラーリターンしません。』

10.『OSI TP 通信をする XATMI のエラーは，従来の TCP/IP とエラー動作が異なる場合があります。』

tprealloc

名称

型付きバッファのサイズの変更

形式

ANSI C，C++形式

```
#include <xatmi.h>
char  *tprealloc(char *ptr,long size)
```

K&R 版 C の形式

```
#include <xatmi.h>
char  *tprealloc(ptr,size)
char    *ptr;
long    size;
```

機能

関数 tprealloc()は，引数 ptr で指すバッファのサイズを size で示すバイト長に変更して，新しい（移動したと思われる）バッファへのポインタをリターンします。割り当てられたバッファが少なくとも size 分の大きさであるのは，tpalloc()と同様です。バッファの type は，サイズ変更後も同じ type を変わらないで保持します。この関数が正常にリターンしたあとは，バッファはリターンされたポインタで参照してください。ptr は使わないでください。古いサイズと新しいサイズのうち，小さい方の部分まではバッファの内容が変化しません。

バッファ型によっては，使う前に初期化が必要なものがあります。そのため tprealloc()は，バッファが割り当てられてからリターンするまでの間に，バッファの初期化をします（その方法は，コミュニケーションリソースマネジャで規定）。

Note：このようにして，使える状態になってから，バッファは呼び出し側にリターンされます。バッファの初期化処理が規定されていない場合は，tprealloc()はバッファを 0 に初期化しません。

『引数』

「●ptr

バッファへのポインタを設定します。』

「●size

バッファの再割り当て後のサイズを設定します。』

リターン値

成功した場合，tprealloc()は long ワードに位置合わせした，適切な形式のバッファへのポインタをリターンします。エラー時には，NULL をリターンして，tperrno にエラーの状態を示す値を設定します。

エラー

次のような場合，tprealloc()はエラーリターンして，次のうちどれか一つの値を tperrno に設定します。

リターン値	リターン値（数値）	意味
TPEINVAL	4	間違った引数が与えられました（例えば，ptr が tpalloc()で割り当てたバッファへのポインタではない）。
TPEOS	7	オペレーティングシステムにエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPEPROTO	9	tprealloc()が間違った状況で呼ばれました。
TPESYSTEM	12	コミュニケーションリソースマネジャシステムでエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。

アプリケーションの使い方

バッファの再初期化処理が失敗すると，tprealloc()は失敗して NULL をリターンします。ptr で指すバッファの内容は有効な値ではなくなります。

この関数は，C ライブラリの malloc()，realloc()，free()と一緒には使わないでください（例えば，tprealloc()で割り当てたバッファは，free()で解放しないでください）。

関連項目

tpalloc()，tpfree()，tptypes()

『OpenTP1 で使う場合の注意事項』

1. 『OpenTP1 では，tprealloc()がリターンしたバッファは，0 で再初期化されています。』
2. 『OSI TP 通信をする XATMI のエラーは，従来の TCP/IP とエラー動作が異なる場合があります。』

tprecv

名称

会話型サービスからのメッセージの受信

形式

ANSI C，C++の形式

```
#include <xatmi.h>
int  tprecv(int cd,char **data,long *len,long flags,long *revent)
```

K&R 版 C の形式

```
#include <xatmi.h>
int  tprecv(cd,data,len,flags,revent)
int   cd;
char  **data;
long  *len;
long  flags;
long  *revent;
```

機能

関数 tprecv()は，ほかのプログラムからオープンコネクションをわたって送信されてきたデータを受信するために使います。tprecv()の第 1 引数の cd には，データを受信するためのオープンコネクションを設定します。cd は，tpconnect()のリターン値かサービスのパラメタ TPSVCINFO で示される記述子です。第 2 引数の data は，tpalloc()で事前に割り当てられたバッファへのポインタのアドレスです。

成功した場合，幾つかのイベントタイプでは，*data は，受信したデータへのポインタ，*len は受信データバッファのサイズを示します。

Note：*len が呼び出し前のバッファの合計サイズよりも大きかった場合は，新しいバッファのサイズが *len です。*len が 0 の場合は，受信したデータはなくて，*data やそれが指すバッファは変わりません。data，*data，および len が NULL である場合は，エラーとなります。

tprecv()は，コネクションの制御権を持たないプログラムでだけ呼び出せます。

『引数』

「●cd

記述子を設定します。』

「●data

受信したデータを格納するバッファへのポインタのアドレスを設定します。』

『●len

受信したデータの長さを格納する領域のアドレスを設定します。』

『●flags

フラグを示します。』

『●revent

イベントを示す long 型へのポインタを示します。』

flags には，次に示す値を設定します。

TPNOCHANGE

*data で指しているバッファタイプと異なる型のデータを受信したとき，このフラグを設定していなければ，*data のバッファ型は，受け取り側がそのバッファタイプを認識しているかぎり，受信したデータのバッファ型に変換されます。このフラグを設定した場合は，*data で指しているバッファのタイプは変換されません。つまり，受信したバッファの type と subtype と，*data で指されたバッファの type と subtype は必ず一致させてください。

TPNOBLOCK

tprecv()はデータの到着を待ちません。データがすでに受信できる状態の場合は，tprecv()はデータを受信してリターンします。このフラグを設定しないで，データが使えない状態の場合は，呼び出し側はデータが到着するまでブロックします。

TPNOTIME

呼び出し側を無期限にブロックして，ブロッキングタイムアウトが起こらないことを意味します。トランザクションタイムアウトは起こります。

TPSIGRSTRT

シグナルが実行中のシステムコールを中断したら，中断したシステムコールを再び呼びます。

イベント

記述子 cd にイベントが存在して，tprecv()がエラーにならなかった場合は，revent にイベントタイプがリターンされます。data はイベントの TPEV_SVCSUCC，TPEV_SVCFAIL，TPEV_SENDONLY と一緒に受信できます。tprecv()で有効なイベントは次のとおりです。

TPEV_DISCONIMM

会話のサブオーディネータに受信されるこのイベントは，会話のオリジネータが tpdiscon()を呼び出してコネクションをすぐに切断したか，コネクションをオープンしたままで tpreturn()，tx_commit()，または tx_rollback()を呼び出したことを表します。また，このイベントは，通信エラーのためにコネクションが切断された場合（例えば，サーバ，マシン，ネットワークがダウンしたとき），オリジネータ，またはサブオーディネータにリターンされます。コネクションの切断はすぐに通知されるので（正常終了ではなく異常終了として），転送中のデータは捨てられます。二つのプログラムが同じトランザクションに参加している場合は，トランザクションは rollback_only 状態になります。このコネクションに使われている記述子は，無効になります。

TPEV_SENDONLY

コネクションの他端点のプログラムは，コネクションの制御権を放棄しています。このイベントの受け取り側はデータの送信が許可されていますが，制御権を放棄するまではデータを受信できません。

TPEV_SVCERR

会話のオリジネータに受信されるこのイベントは，会話のサブオーディネータが tpreturn()を呼び出していることを示します。tpreturn()は，サービスが成功してリターンすることを妨げるエラーに遭遇しました。例えば，tpreturn()に間違った引数が渡されていたり，サービスがほかのサブオーディネータのためにコネクションをオープンしている間に，tpreturn()が呼ばれたりした場合です。このイベントの性質のために，アプリケーションで定義されたデータやリターン値は役に立ちません。このコネクションは切断されて，cd は無効になります。このイベントが受け取り側のトランザクションの一部で起こった場合は，トランザクションは rollback_only 状態となります。

TPEV_SVCFAIL

会話のオリジネータで受信されるこのイベントは，会話の他端点上のサブオーディネータのサービスがアプリケーションで指定されて，不成功に終了した（つまり，tpreturn()が TPFAIL で呼ばれた）ことを示します。tpreturn()が呼ばれたとき，サブオーディネータのサービスがこのコネクションを制御する場合は，サービスは会話のオリジネータへ型付きバッファを渡せます。サービス関数の終了の部分で，サーバはコネクションを切断します。よって，cd は無効になります。このイベントが受け取り側のトランザクションの一部で起こった場合は，トランザクションは rollback_only 状態となります。

TPEV_SVCSUCC

会話のオリジネータで受信されるこのイベントは，会話の他端点上のサブオーディネータのサービスがアプリケーションで指定されて，成功して終了した（つまり，tpreturn()が TPSUCCESS で呼ばれた）ことを示します。サービス関数の終了の部分で，サーバはコネクションを切断します。よって，cd は無効になります。このイベントが受け取り側のトランザクションの一部で起こった場合は，コミットまたはロールバックするために，サーバ（もまた，トランザクションモードの場合）によってなされた処理を引き起こしたトランザクションのコミット（受け取り側が（トランザクションを）開始した場合）や，ロールバックのどちらかを実行できます。

リターン値

tprecv()が，revent に TPEV_SVCSUCC か，TPEV_SVCFAIL を設定してリターンした場合，アプリケーションが tpreturn()の引数として渡した定義値は，グローバル変数 tpurcode として参照できます。エラー時には，tprecv()は-1 をリターンして，

tperrno にエラーの状態を示す値を設定します。イベントが通知されたときにエラーが起こっていなければ，tprecv()は-1 をリターンして，tperrno に TPEEVENT が設定されています。

エラー

次のような場合，tprecv()はエラーリターンして，次のうちどれか一つの値を tperrno に設定します。

リターン値	リターン値（数値）	意味
TPEBADDESC	2	引数 cd が間違った記述子を指しています。
TPEBLOCK	3	TPNOBLOCK を設定した tprecv()を呼び出したときに，ブロッキング状態になりました。
TPEINVAL	4	間違った引数が与えられました（例えば，data は tpalloc()で割り当てたバッファへのポインタではない，flags に設定した値が間違っている）。
TPEOS	7	オペレーティングシステムにエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPEPROTO	9	tprecv()が間違った状況で呼ばれました。
TPESYSTEM	12	コミュニケーションリソースマネジャシステムでエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPETIME	13	タイムアウトが起こりました。呼び出し側がトランザクションモードの場合，これはトランザクションタイムアウトで，トランザクションは rollback_only 状態となります。そうでない場合は，TPNOBLOCK も TPNOTIME も設定されていない状態で，ブロッキングタイムアウトが起こったことを意味します。どちらの場合も，*data とその内容は変わっていません。トランザクションタイムアウトが起こった場合は，トランザクションをロールバックするまでは，新しく送信しようとしていた送信や，解決していない応答は，TPETIME でエラーリターンします。
TPEGOTSIG	15	シグナルは受信されましたが，TPSIGRSTRT が設定されていません。
TPEOTYPE	18	到着したバッファの type と subtype を，呼び出し側で認識していません。または，flags に TPNOCHANGE を設定しているのに，*data の type と subtype が送信されてきたバッファの type と subtype に一致していません。どちらの場合も，*data も*len も変更されません。呼び出し側のトランザクションとして会話型サービスが実行されている場合は，到着したバッファを捨てるまで，トランザクションは rollback_only 状態となります。このエラーが起こった場合，cd に該当するイベントは捨てられて，会話型通信は未決着な状態になります。呼び出し側は，すぐに会話型通信を終了させてください。
TPEEVENT	22	あるイベントが起こって，そのタイプは revent にリターンされます。

関連項目

tpalloc()，tpconnect()，tpdiscon()，tpsend()

『OpenTP1 で使う場合の注意事項』

1. 『TPSIGRSTRT フラグは無効となります。このフラグの有無に関係なく，シグナル受信時には，中断したシステムコールを再び呼びます。TPEGOTSIG がリターンすることはありません。』
2. 『OpenTP1 では，トランザクションタイムアウトが起こったときには，そのプロセスは異常終了します。そのため，TPETIME がリターンされるのは，ブロッキングタイムアウトの場合だけとなります。』
3. 『該当バージョンの OpenTP1 では，ロールバックする必要があるエラーは，X/Open で特に指定がないかぎり，TPESYSTEM としています。ただし，TPESYSTEM がリターンしても，rollback_only 状態とならない場合もあります。』
4. 『TP1/NET/OSI-TP-Extended を使った OSI TP 通信をする場合は，会話型サービスの通信はできません。OSI TP 通信で会話型サービスの通信を使った場合，システムの動作は保証しません。』

tpreturn

名称

サービス関数からのリターン

形式

ANSI C，C++の形式

```
#include <xatmi.h>
void  tpreturn(int rval,long rcode,char *data,long  len,
               long flags)
```

K&R 版 C の形式

```
#include <xatmi.h>
void  tpreturn(rval,rcode,data,len,flags)
int     rval;
long    rcode;
char    *data;
long    len;
long    flags;
```

機能

関数 tpreturn()は，サービス関数が完了したことを示します。tpreturn()は，C 言語の return()に該当します（つまり，tpreturn()が呼び出されると，サービス関数は，コミュニケーションリソースマネジャにリターンします)。コミュニケーションリソースマネジャへ正しく制御が戻ることを保証するために，tpreturn()は，コミュニケーションリソースマネジャで割り当てられたサービス関数内から呼び出すことをお勧めします。

関数 tpreturn()は，サービスの応答メッセージを送信するために使用します。tpreturn()が成功すると，tpcall()，tpgetrply()，tprecv()で受信待ちしているプログラムの受信バッファに応答が返ります。

会話型サービスでは，tpreturn()はコネクションも終了させます。つまり，サービス関数は，tpdiscon()を直接呼べません。正しい結果を保証するために，会話型サービスと通信しているプログラムは，tpdiscon()を呼ばないでください。むしろ，会話型サービスが完了する通知を待ってください（つまり，tpreturn()から送信された TPEV_SVCSUCC か TPEV_SVCFAIL のようなイベントを待つ必要があります)。

サービス関数がトランザクションモードの場合は，tpreturn()は，トランザクションが完了したときに，そのサービスをコミットするかロールバックするかを決定します。サービスは，同じトランザクションの一部として複数回呼び出されるので，トランザクションの生成者から tx_commit()，tx_rollback()が呼ばれるまで，必ずしも完全にコミット，またはロールバックされるとはかぎりません。

関数 tpreturn()は，サービス関数から起動されたサービス要求のすべての応答を受信したあとで呼んでください。さもなければ，サービスの性質に従いますが，TPESVCERR やイベントである TPEV_SVCERR が，サービス関数との会話を開始したプログラムにリターンされます。受信されていない未解決の応答は，

コミュニケーションリソースマネジャで自動的に捨てられます。これらの応答のための記述子は，無効な記述子になります。

関数 tpreturn()は，サービスで開始されたすべてのコネクションがクローズされてから呼んでください。さもないと，サービスの性質に従って，TPESVCERR やイベントである TPEV_SVCERR が，サービス関数との会話を開始したプログラムにリターンされます。さらに，緊急のコネクション切断イベント（TPEV_DISCONIMM）が，すべてのサブオーディネータへのオープンコネクションに送信されます。

コネクション制御に関して，tpreturn()を呼び出したサービス関数が，このサービス関数を呼び出したコネクションの制御権を持たない場合は，二つの結果が起こることが考えられます。一つ目は，もしサービス関数が，rval に TPFAIL，data に NULL を設定した tpreturn()を呼ぶと，TPEV_SVCFAIL イベントが会話のオリジネータへ送信されます。二つ目は，ほかの値を設定した tpreturn()を呼ぶと，TPEV_SVCERR イベントがオリジネータへ送信されます。

会話型サービスは，開始していないオープンコネクションを一つだけしか持っていないので，コミュニケーションリソースマネジャは，送信された記述子のデータ（または，イベント）を認識しています。これらの理由で記述子は tpreturn()に渡されます。

引数

『●rval

TPSUCCESS，TPFAIL のどちらか一つを設定します。』

『●rcode

アプリケーションで定義するリターンコードを設定します。』

『●data

送信する応答データのバッファを指すポインタを設定します。』

『●len

送信されてきたデータのバッファの長さを設定します。』

『●flags

0 を設定します（将来の予約）。』

rval には，次のうちどれか一つの値が設定されます。

TPSUCCESS

サービスが成功して終了しました。データがある場合は，それも送信されます（リターン処理中で失敗がなければ）。呼び出し側がトランザクションモードである場合は，tpreturn()は，そのサービスが最終的にコミットされる場所に置きます。

Note：tpreturn()は，必ずしもトランザクション全体を決着する訳ではありません。また，呼び出し側が成功を示しているときでも，未解決の応答やオープンコネクションがあるときや，サービスの中でト

ランザクションが rollback_only 状態となるような何らかの処理をしていた場合，エラーメッセージが送信されます（つまり，応答の受け取り側は，TPESVCERR か，イベントである TPEV_SVCERR を受信します）。

Note：何らかの理由で，トランザクションがサービス関数内で rollback_only 状態になる場合は，rval には TPFAIL を設定してください。会話型サービスに TPSUCCESS が設定されていた場合は，イベントである TPEV_SVCSUCC が通知されます。

TPFAIL

サービスがアプリケーションの立場から見て不成功に終了しました。エラーは応答を受信するプログラムに送信されます。つまり，応答を受信する呼び出しは失敗して，受け取り側は TPESVCFAIL か，イベントである TPEV_SVCFAIL を受信します。呼び出し側がトランザクションモードである場合は，tpreturn()はトランザクションを rollback_only 状態にします（トランザクションはすでに rollback_only 状態になっているかもしれないことに注意）。リターンの処理中で失敗がなければ，呼び出し側のデータがある場合は，送信されます。呼び出し側のデータが送信されない理由の一つは，トランザクションタイムアウトが起こったときです。この場合，応答を待っているプログラムは，TPETIME のエラーを受信します。

rval にこれらのうちのどれか一つを設定していない場合は，デフォルトで TPFAIL が仮定されます。

アプリケーションで定義するリターンコードの rcode は，サービスの応答を受信するプログラムへ送信されます。このコードは，応答が正常に送信されているかぎり，rval に設定した内容に関係なく送信されます（つまり，呼び出しの受信が成功か，TPESVCFAIL をリターンするか，TPEV_SVCSUCC か TPEV_SVCFAIL のイベントのうちの一つを受信しているかぎり）。会話型サービスでは，このコードは，tpreturn()の呼び出し時にサービス関数がコネクションの制御権を持っているときだけ送信されます。rcode の値は変数 tpurcode として，受け取り側で有効となります。

data は，送信する応答のデータ部を指すポインタです。data が NULL でない場合は，data は事前に呼ばれた tpalloc()で得られたバッファを必ず指すようにしてください。このバッファが，呼び出し時にサービス関数に渡されたバッファと同じ場合は，この処理はコミュニケーションリソースマネジャに任せられて，サービス関数の作成者は，バッファが解放されているかどうかを意識する必要はありません。つまり，このバッファを解放しようとするユーザの試みは失敗します。しかし，tpreturn()に渡されたバッファが，サービスを呼び出したときと同じでないバッファの場合は，tpreturn()はこのバッファを解放できます。len は，送信されるデータバッファの合計サイズを設定します。data が長さを設定する必要がないバッファを指すポインタの場合は，len は無視されます（0 にしてください）。長さが必要なバッファ型を data が指す場合は，len には 0 を設定しないでください。

data が NULL の場合は，len は無視されます。この場合，応答がサービスを呼び出したプログラムから要求されている場合は，データ部がない応答が送信されます。応答が要求されていない場合は，tpreturn()は必要なときにデータを解放して，応答なしの送信をリターンします。

flags は将来の使用のために予約されているので，0 を設定しておきます。

サービスが会話型の場合，データ部が送信されない理由には，次の二つの場合があります。

- 呼び出しが完了したときに，すでにコネクションが切断されている場合（つまり，呼び出し側がコネクションに関する TPEV_DISCONIMM を受信した），この呼び出し側は，単にサービスルーチンを終わらせて，トランザクションがある場合は，ロールバックします。この場合，呼び出し側のデータは送信されません。
- 上記で説明したように，呼び出し側がコネクションを制御していない場合は，TPEV_SVCFAIL，または TPEV_SVCERR が，オリジネータ（コネクションの確立元）に送信されます。オリジネータがどのイベントを受け取るかどうかに関係なく，データは送信されません。しかし，オリジネータが TPEV_SVCFAIL を受信した場合は，オリジネータのリターンコードは tpurcode で有効になります。

リターン値

サービス関数は，呼び出し側には値をリターンしないで，コミュニケーションリソースマネジャが割り当てます。そのため，void 型の宣言となっています。ただし，サービス関数は，tpreturn()で終了することを期待されます。サービス関数が tpreturn()を使わないで終了した場合（つまり，C 言語の return()を使うか，または関数が終わったら），サーバはサービス要求者にサービスエラーをリターンします。すべてのサブオーディネータへのオープンコネクションはすぐに切断されて，未解決の非同期応答は捨てられます。サーバがトランザクションモードで失敗した場合は，トランザクションは rollback_only 状態になります。

注

tpreturn()がサービス関数の外で（例えば，サービスでないルーチンから）使われる場合は，何もしません。

エラー

tpreturn()は，サービス関数を終了させるので，引数をハンドリングしているときか，処理中のときに起こったエラーは，関数の呼び出し側に通知できません。このようなエラーは，tpcall()か tpgetrply()を経由してサービスの結果を受信するプログラムでは，TPESVCERR を tperrno に設定する要因となります。tpsend()や tprecv()を使っているプログラムでは，イベント TPEV_SVCERR が通知されます。

関連項目

tpalloc()，tpcall()，tpconnect()，tpdiscon()，tpgetrply()，tprecv()，tpsend()，tpservice()

『OpenTP1 で使う場合の注意事項』

1. 『該当バージョンの tpreturn()は，サービス関数を終了させません。tpreturn()を呼び出したあとには，すぐに return()でサービス関数を終了してください。tpreturn()を呼び出したあとに何らかの処理をした場合は，動作は保証しません。』
2. 『OSI TP 通信をする XATMI のエラーは，従来の TCP/IP とエラー動作が異なる場合があります。』

tpsend

名称

会話型サービスへのメッセージの送信

形式

ANSI C，C++の形式

```
#include <xatmi.h>
int  tpsend(int cd,char *data,long len,long flags,long *revent)
```

K&R 版 C の形式

```
#include <xatmi.h>
int  tpsend(cd,data,len,flags,revent)
int     cd;
char    *data;
long    len;
long    flags;
long    *revent;
```

機能

関数 tpsend()は，オープンコネクションをわたって，ほかのプログラムへデータを送信するために使います。呼び出し側は，コネクションの制御権がなくてはなりません。

この関数の第 1 引数の cd は，データを送信するオープンコネクションを示します。cd は，tpconnect()のリターン値か，会話型サービスに渡されたパラメタ TPSVCINFO で示される記述子です。第 2 引数の data は，tpalloc()で事前に割り当てられたバッファへのポインタです。len にはバッファをどれだけ送信したいかを示します。

注

data が大きさを指定する必要がないバッファへのポインタであれば，len は無視されます（0 にしてください)。長さが必要なバッファ型を data が指す場合は，len には 0 を設定しないでください。また，data は NULL にすることもできます。その場合も，len は無視されます（アプリケーションデータは送信されません。このような使い方をするのは，例えば，データを送信しないで，コネクションの制御を渡したい場合などです)。data の type と subtype は，コネクションの他端点に認識された type と subtype の一つに一致させてください。

『引数』

「●cd

記述子を設定します。』

『●data

送信するデータが格納されたバッファへのポインタを設定します。』

『●len

バッファの長さを設定します。』

『●flags

フラグを示します。』

『●revent

イベントを示す long 型へのポインタを示します。』

flags には，次に示す値を設定します。

TPRECVONLY

このフラグは，呼び出し側のデータが送信されたあと，呼び出し側は，コネクションの制御を放棄することを意味します（つまり，呼び出し側は，これ以上 tpsend()を呼び出せません）。コネクションの他端点である受信側が，tpsend()から送信されてきたこのデータを受信したとき，コネクションの制御を示すイベント（TPEV_SENDONLY）も受信します（これ以上，tprecv()の呼び出しができません）。

TPNOBLOCK

ブロッキング状態の場合（例えば，送信されたメッセージで内部バッファが満杯），データやイベントは送信されません。TPNOBLOCK が設定されていないで，ブロッキング状態のときは，呼び出し側は状況が収まるか，タイムアウト（トランザクション，またはブロッキングタイムアウト）が起こるまで，ブロッキングしています。

TPNOTIME

呼び出し側を無期限にブロックして，ブロッキングタイムアウトが起こらないことを意味します。トランザクションタイムアウトは起こります。

TPSIGRSTRT

シグナルが実行中のシステムコールを中断したら，中断したシステムコールを再び呼びます。

イベント

記述子 cd にイベントが存在した場合は，tpsend()は，呼び出し側のデータを送信しないで失敗します。revent にイベントタイプがリターンされます。tpsend()で有効なイベントは次のとおりです。

TPEV_DISCONIMM

会話のサブオーディネータに受信されるこのイベントは，会話のオリジネータが tpdiscon()を使ってコネクションをすぐに切断したか，コネクションをオープンしたままで tpreturn()，tx_commit()，または tx_rollback()を呼び出したことを表します。また，このイベントは，通信エラーのためにコネクションが切断された場合（例えば，サーバ，マシン，ネットワークがダウンしたとき），オリジネータ，またはサブオーディネータにリターンされます。

TPEV_SVCERR

会話のオリジネータに受信されるこのイベントは，会話のサブオーディネータが通信の制御権がない状態で tpreturn()を発行していることを表します。tpreturn()は，次に説明する TPEV_SVCFAIL とは異なった方法で使われます。

TPEV_SVCFAIL

会話のオリジネータで受信されるこのイベントは，会話の他端点上のサブオーディネータが通信の制御権がない状態で tpreturn()を呼び出したことを示します。さらに，tpreturn()は TPFAIL と data なしで呼び出されました（つまり，rval は TPFAIL を設定されて，data には NULL が入っています）。

これらのイベントは，コネクションがすぐに切断されたことを示します（正常終了ではなく異常終了として）。そのため，送信中のデータは失われます。このコネクションに使われている記述子は，無効になります。二つのプログラムが，同じトランザクションにある場合は，そのトランザクションは rollback_only 状態になります。

リターン値

エラー時には，tpsend()は-1 をリターンして，tperrno にエラーの状態を示す値を設定します。tpsend()がリターンして revent に TPEV_SVCFAIL が設定される場合，tpurcode 外部変数には，tpreturn()の一部でアプリケーションが定義した値を含みます。

エラー

次のような場合，tpsend()はエラーリターンして，次のうちどれか一つの値を tperrno に設定します。

リターン値	リターン値（数値）	意味
TPEBADDESC	2	引数 cd が間違った記述子を指しています。
TPEBLOCK	3	TPNOBLOCK を設定した tpsend()を呼び出したときに，ブロッキング状態になりました。
TPEINVAL	4	間違った引数が与えられました（例えば，data は tpalloc()で割り当てたバッファへのポインタではない，flags に設定した値が間違っている）。
TPEOS	7	オペレーティングシステムにエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPEPROTO	9	tpsend()が間違った状況で呼ばれました。
TPESYSTEM	12	コミュニケーションリソースマネジャシステムでエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPETIME	13	タイムアウトが起こりました。呼び出し側がトランザクションモードの場合，これはトランザクションタイムアウトで，トランザクションは rollback_only 状態となります。そうでない場合は，TPNOBLOCK も TPNOTIME も設定されていない状

リターン値	リターン値（数値）	意味
TPETIME	13	態で，ブロッキングタイムアウトが起こったことを意味します。どちらの場合も，*data とその内容は変わっていません。トランザクションタイムアウトが起こった場合は，トランザクションをロールバックするまでは，どのコネクションでのどのような送信，受信の試みも，また新しくコネクションを開始しようとする試みも，TPETIME でエラーリターンします。
TPEGOTSIG	15	シグナルは受信されましたが，TPSIGRSTRT が設定されていません。
TPEEVENT	22	イベントが起こりました。このエラーが起こったときは，data は送信されません。そのイベントタイプは revent にリターンされます。

関連項目

tpalloc()，tpconnect()，tpdiscon()，tprecv()，tpreturn()

『OpenTP1 で使う場合の注意事項』

1. 『該当バージョンの OpenTP1 では，TPNOBLOCK フラグは無効となります。そのため，TPEBLOCK がリターンすることはありません。OpenTP1 では，ブロッキング状態のため通信ができない場合は，ネットワークダウンが原因で通信ができない場合と同様に，TPESYSTEM をリターンする仕様となっています。』
2. 『該当バージョンの OpenTP1 では，TPNOTIME フラグは無効となります。』
3. 『TPSIGRSTRT フラグは無効となります。このフラグの有無に関係なく，シグナル受信時には，中断したシステムコールを再び呼びます。TPEGOTSIG がリターンすることはありません。』
4. 『OpenTP1 では，トランザクションタイムアウトが起こったときには，そのプロセスは異常終了します。そのため，TPETIME がリターンされるのは，ブロッキングタイムアウトの場合だけとなります。』
5. 『該当バージョンの OpenTP1 では，ロールバックする必要があるエラーは，X/Open で特に指定がないかぎり，TPESYSTEM としています。ただし，TPESYSTEM がリターンしても，rollback_only 状態とならない場合もあります。』
6. 『OpenTP1 では，会話の相手が tpdiscon()，tpreturn()を呼び出している場合でも，tpsend()を呼び出すプロセスでイベントを受信していない場合は，tpsend()でイベントを通知できません。』
7. 『TP1/NET/OSI-TP-Extended を使った OSI TP 通信をする場合は，会話型サービスの通信はできません。OSI TP 通信で会話型サービスの通信を使った場合，システムの動作は保証しません。』

tpservice

名称

サービス関数のテンプレート

形式

ANSI C，C++ の形式

```
#include <xatmi.h>
void  tpservice(TPSVCINFO  *svcinfo)
```

K&R 版 C の形式

```
#include <xatmi.h>
void  tpservice(svcinfo)
TPSVCINFO    *svcinfo;
```

機能

関数 tpservice()は，サービス関数を記述するためのテンプレートです。このテンプレートは，tpconnect()，tpsend()，tprecv()関数を経由して，会話するサービスはもちろん，tpcall()や tpacall()関数を経由してサービス要求を受信するサービスにも使われます。

tpcall()や tpacall()を経由して実行されたサービス要求を処理するサービス関数は，多くても一つの入力されたメッセージ（svcinfo の data 要素）を受信して，一つの応答（tpreturn()でサービス関数を出たこと）を送信します。

他方では，会話型サービスは，オープンコネクションを参照する手段のほかに，多くても一つの入力メッセージを伴った，コネクション要求で呼び出されます。会話型サービス関数が起動されると，コネクション要求を出したプログラムも，会話型サービスも，アプリケーションで定義されたデータを送信したり受信したりできるようになります。コネクションは半二重となっています。つまり，プログラムは，コネクションの制御権を明示的に放棄するまで，会話を制御（つまり，データを送信する）できます。

トランザクションに関しては，トランザクションモードで起動した場合，サービス関数は，少なくとも一つのトランザクションに参加できます。サービス関数の作成者は，トランザクションは，サービス関数から戻ることで終了すると考えてください。サービス関数をトランザクションモードで起動しない場合，サービス関数は，必要なだけのトランザクションを tx_begin()，tx_commit()，tx_rollback()を使うことで開始します。

Note：tpreturn()は，トランザクションの完了に使われません。したがって，サービス関数内で起こったトランザクションを解決しないままにして tpreturn()を呼ぶと，エラーになります。

『引数』

サービス関数は，一つの引数で起動します。それは，svcinfo で，サービス情報構造体へのポインタです。この構造体は，次の要素を含みます。

```
char     name[XATMI_SERVICE_NAME_LENGTH];
char     *data;
long     len;
long     flags;
int      cd;
```

name は，呼び出し側が，サービスを呼び出すときに使うサービスの名称です。サービス関数への入り口での flags の設定は，サービス関数が注意したい属性を指します。

flags には，次に示す値を設定します。

TPCONV

会話のためのコネクション要求が受信されて，会話のための記述子 cd が使えます。このフラグが設定されていない場合は，これはリクエスト/レスポンス型のサービスであり，cd は使えません。

TPTRAN

サービス関数は，トランザクションモード内にあります。

TPNOREPLY

呼び出し側は応答を期待しません。このオプションは，TPCONV が設定されている場合は，設定されません。

TPSENDONLY

このサービスは，コネクションをわたってデータを送信だけできるように起動されて，コネクションの他端点のプログラムは，データを受信だけできるようになります。このフラグは，TPRECVONLY とは互いに排他的であって，TPCONV が設定されているときだけ，設定できます。

TPRECVONLY

このサービスは，コネクションをわたってデータを受信だけできるように起動されて，コネクションの他端点のプログラムは，データを送信だけできるようになります。このフラグは，TPSENDONLY とは互いに排他的であって，TPCONV が設定されているときだけ，設定できます。

data は，要求メッセージのデータ部分へのポインタです。len は，data の長さです。data で指すバッファは，tpalloc()で事前にコミュニケーションリソースマネジャ内で割り当てられています。このバッファのサイズは，tprealloc()で大きくできます。ただし，ユーザで解放できません。サービスが終了したら，このバッファは tpreturn()に渡すことをお勧めします。これらのルーチンに異なるバッファが渡された場合は，そのバッファは tpreturn()によって解放されます。

Note：例え tpreturn()に渡されなくても，data で指されるバッファは次のサービスリクエストに上書きされることに注意してください。サービス要求に付随するデータがなければ，data は NULL です。この場合は，len は 0 です。

TPCONV が flags に設定されたとき，cd はコネクションの記述子です。これは，会話を開始したプログラムと通信するために tpsend()，tprecv()と一緒に使えます。

リターン値

サービス関数は，呼び出し側には値をリターンしないで，コミュニケーションリソースマネジャが割り当てます。そのため，void 型の宣言となっています。ただし，サービス関数は，tpreturn()で終了することを期待されます。サービス関数が tpreturn()を使わないで終了した場合（つまり，C 言語の return()を使うか，または関数が終わったら），サーバはサービス要求者にサービスエラーをリターンします。すべてのサブオーディネータへのオープンコネクションはすぐに切断されて，未解決の非同期応答は捨てられます。サーバがトランザクションモードで失敗した場合は，トランザクションは rollback_only 状態になります。

Note：tpreturn()がサービス関数の外で使われる場合（例えば，サービスでないルーチンから），何の効果もなくリターンします。

エラー

tpreturn()は，サービス関数を終了させるので，引数をハンドリングしているときか，処理中のときに起こったエラーは，関数の呼び出し側に通知できません。このようなエラーは，tpcall()か tpgetrply()でサービスの結果を受信するプログラムでは TPESVCERR が tperrno に設定する要因となります。tpsend()や tprecv()を使っているプログラムでは，イベントである TPEV_SVCERR となります。

関連項目

tpalloc()，tpcall()，tpconnect()，tpgetrply()，tprecv()，tpreturn()，tpsend()

『OpenTP1 で使う場合の注意事項』

1. 『OpenTP1 の UAP（サービス関数）の場合には，tpreturn()の直後に，必ず return を記述してください。これは，OpenTP1 の実行プロセスに制限があるためです。また，tpreturn()を呼び出したら，すぐに return を実行してください。tpreturn()と return の間で処理はしないでください。トランザクションの処理の範囲で tpreturn()を呼び出してから return を実行する前に資源を更新すると，その更新処理もトランザクションに含まれてしまいます。』
2. 『OSI TP 通信をする XATMI のエラーは，従来の TCP/IP とエラー動作が異なる場合があります。』

tptypes

名称

型付きバッファの情報の取得

形式

ANSI C，C++ の形式

```
#include <xatmi.h>
long tptypes(char *ptr,char *type,char *subtype)
```

K&R 版 C の形式

```
#include <xatmi.h>
long tptypes(ptr,type,subtype)
char    *ptr;
char    *type;
char    *subtype;
```

機能

関数 tptypes()は，引数 ptr にデータを格納したバッファへのポインタを設定して，引数 type，subtype にそれぞれバッファのタイプとサブタイプをリターンします。ptr は tpalloc()から得たバッファへのポインタです。type と subtype が NULL でない場合は，この関数はそれぞれが指しているタイプとサブタイプの名称を文字配列に格納します。名称が最大長（type は 8 バイト，subtype は 16 バイト）の場合，文字配列の終わりには NULL を入れません。サブタイプがない場合は，subtype が指す配列は NULL 文字列（""）です。

Note：type は先頭 8 バイト，subtype は先頭 16 バイトが格納されることに注意してください。

『引数』

『●ptr

バッファへのポインタを設定します。』

『●type

バッファの type へのポインタを設定します。』

『●subtype

バッファの subtype へのポインタを設定します。』

リターン値

成功した場合，tptypes()はバッファのサイズをリターンします。エラー時には，tptypes()は-1 をリターンして，tperrno にエラーの状態を示す値を設定します。

エラー

次のような場合，tptypes()はエラーリターンして，次のうちどれか一つの値を tperrno に設定します。

リターン値	リターン値（数値）	意味
TPEINVAL	4	間違った引数が与えられました（例えば，ptr が型付きバッファへのポインタではない）。
TPEOS	7	オペレーティングシステムにエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPEPROTO	9	tptypes()が間違った状況で呼ばれました。
TPESYSTEM	12	コミュニケーションリソースマネジャシステムでエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。

関連項目

tpalloc()，tpfree()，tprealloc()

『OpenTP1 で使う場合の注意事項』

1.『OSI TP 通信をする XATMI のエラーは，従来の TCP/IP とエラー動作が異なる場合があります。』

tpunadvertise

名称

サービス名の広告の取り消し

形式

ANSI C，C++ の形式

```
#include <xatmi.h>
int   tpunadvertise(char *svcname)
```

K&R 版 C の形式

```
#include <xatmi.h>
int   tpunadvertise(svcname)
char    *svcname;
```

機能

関数 tpunadvertise()は，あるサーバにそのサーバで提供する，あるサービスの広告を取り消すことを許可します。この関数を呼び出さない場合，サーバのサービスは，ブート時に広告されて，シャットダウンされたときに広告を取り消されます。

関数 tpunadvertise()は，サーバで宣伝していた svcname を削除します。svcname には，NULL や NULL 文字列（""）は使えません。また，svcname は 15 文字以下になるようにしてください。svcname が 15 文字よりも長い場合は，15 文字に切り詰められます。切り詰められた名称がほかのサービス名と一致しないようにしてください。

『引数』

『●svcname

サービスのサービス名を設定します。』

リターン値

エラー時には，-1 をリターンして，tperrno にエラーの状態を示す値を設定します。

エラー

次のような場合，tpunadvertise()はエラーリターンして，次のうちどれか一つの値を tperrno に設定します。

リターン値	リターン値（数値）	意味
TPEINVAL	4	引数 svcname が NULL か NULL 文字（""）が設定されています。
TPENOENT	6	引数 svcname は，現在このサーバで広告されていません。
TPEOS	7	オペレーティングシステムにエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。
TPEPROTO	9	tpunadvertise()が間違った状況で呼ばれました。
TPESYSTEM	12	コミュニケーションリソースマネジャシステムでエラーが起こりました。厳密なエラーの性質は，product-specific な方法で定義されます。

関連項目

tpadvertise()

『OpenTP1 で使う場合の注意事項』

1. 『一つのノードで負荷分散している場合（マルチサーバ），tpunadvertise()をどれか一つのプロセスが呼び出すと，負荷分散しているプロセスすべてでサービスを受け付けられなくなります。その後，tpadvertise()で再びサービスを広告すれば，サービス要求を受け付けられるようになります。』
2. 『OSI TP 通信をする XATMI のエラーは，従来の TCP/IP とエラー動作が異なる場合があります。』

TX インタフェースのアプリケーションプログラミングインタフェース（tx_～）

TX インタフェースの API（関数）の文法について説明します。この節の記述は，X/Open 発行の「X/Open CAE Specification Distributed TP : The TX (Transaction Demarcation) Specification」の文法部である「Chapter 5　C Reference Manual Pages」の記述を，日本語訳したものです。

なお，OpenTP1 の UAP で TX_関数を使うときに注意する項目として追加した文章は，『　』で示します。

TX インタフェースの関数を次に示します。

- tx_begin－トランザクションの開始
- tx_close－リソースマネジャ集合のクローズ
- tx_commit－トランザクションのコミット
- tx_info－現在のトランザクションに関する情報の返却
- tx_open－リソースマネジャ集合のオープン
- tx_rollback－トランザクションのロールバック
- tx_set_commit_return－commit_return 特性の設定
- tx_set_transaction_control－transaction_control 特性の設定
- tx_set_transaction_timeout－transaction_timeout 特性の設定

TX インタフェースの関数（tx_～）は，TP1/Server Base と TP1/LiNK のどちらの UAP でも使えます。

tx_begin

名称

トランザクションの開始

形式

ANSI C，C++ の形式

```
#include <tx.h>
int  tx_begin(void)
```

K&R 版 の形式

```
#include <tx.h>
int  tx_begin()
```

機能

関数 tx_begin()は，呼び出し元スレッドをトランザクションモードにします。呼び出しスレッドは，まずリンクされたリソースマネジャがオープンされていることを（tx_open()で）保証しなければなりません。そのあとで，トランザクションの開始が可能になります。

関数 tx_begin()の呼び出し元が，すでにトランザクションモードに入っているか，または tx_open()をまだ呼び出していない場合，関数 tx_begin()はエラーリターンします（TX_PROTOCOL_ERROR が返ります）。

tx_begin()でトランザクションモードに入ったスレッドは，tx_commit()，tx_rollback()を発行して現在のトランザクションを必ず完了させてください。

トランザクションの開始では，呼び出し元が明示的に tx_begin()の呼び出しを必要としない，トランザクション連鎖にかかわる場合があります。

トランザクション連鎖の詳細は，tx_commit()，tx_rollback()を参照してください。

『tx_begin()は MHP から呼び出せません。』

『次に示す関数で設定した値は，tx_begin()の処理に影響を与えます。』

- tx_set_transaction_timeout()

リターン値

成功した場合，tx_begin()は TX_OK を返します。これは，負ではないリターン値です。『0 が返ります。』

エラー

次のような場合，tx_begin()はエラーリターンして，次のうちどれか一つの値を返します。これは，負のリターン値です。

リターン値	リターン値（数値）	意味
TX_OUTSIDE	-1	呼び出し元のスレッドが現在，グローバルトランザクションの外部で，一つ以上のリソースマネジャに関連する作業をしています。この場合トランザクションマネジャはグローバルトランザクションを開始できません。このような作業は，グローバルトランザクションが開始される前に完了していなければなりません。 呼び出し元の状態が，ローカルトランザクション中にあることに変わりはありません。
TX_PROTOCOL_ERROR	-5	関数は，適切でないコンテクストで呼ばれました（例えば，呼び出し元がすでにトランザクションモードである場合）。 呼び出し元の状態がトランザクションモードになっているかどうかは変わりません。
TX_ERROR	-6	トランザクションマネジャ，または一つ以上のリソースマネジャが，トランザクションを開始しようとした際に一時的なエラーが起こりました。 このエラーが返った場合は，呼び出し元の状態はトランザクションモードではありません。
TX_FAIL	-7	トランザクションマネジャ，または一つ以上のリソースマネジャで，回復できないエラーが起こりました。トランザクションマネジャ，一つ以上のリソースマネジャのどちらか，またはその両方が，すでにアプリケーションに代わって作業できないことがエラーの原因です。 このエラーが返った場合は，呼び出し元はトランザクションモードではありません。

参照

tx_commit()，tx_open()，tx_rollback()，tx_set_transaction_timeout()

『指定例』

```
『if (tx_info(NULL)== 0 && tx_begin() < 0)
    fputs(”cannot begin transaction¥n”,stderr);』
```

注意事項

グローバルトランザクションに参加するには，XA に準拠したリソースマネジャが正常にオープンされていることが前提です。詳細は tx_open()を参照してください。

『OpenTP1 で使う場合の注意事項』

1. 『SPP からトランザクション処理を開始する場合は，tx_begin()を必ず呼び出します。SPP の場合は，呼び出し元で tx_begin()を呼び出していれば，トランザクション処理になります。』
2. 『tx_begin()でトランザクションを生成するプロセスは，このマニュアルの記述に従って正しく結合された UAP の実行形式ファイルを起動したものでなければなりません。』
3. 『tx_begin()と dc_trn_～の関数は併用できません。』

tx_close

名称

リソースマネジャ集合のクローズ

形式

ANSI C，C++ の形式

```
#include <tx.h>
int  tx_close(void)
```

K&R 版 C の形式

```
#include <tx.h>
int  tx_close()
```

機能

関数 tx_close()は，互換性がある方法で，リソースマネジャの集合をクローズします。tx_close()でリソースをクローズすると，トランザクションマネジャがリソースマネジャ固有の情報をトランザクションマネジャ独自の方法で読み，この情報を，呼び出し元にリンクされているリソースマネジャに渡します。

関数 tx_close()は，呼び出し元にリンクされているすべてのリソースマネジャをクローズします。tx_close()は，リソースマネジャ固有のクローズ呼び出しの代わりに発行され，アプリケーションプログラムからの互換性を阻害する呼び出しを不要にします。

リソースマネジャは，終了セマンティクスがそれぞれ異なっています。そのため，特定のリソースマネジャをクローズするのに必要な固有な情報は，それぞれのリソースマネジャから公開されてなければなりません。

アプリケーションスレッドで，もはやグローバルトランザクションに入るつもりがないときには，関数 tx_close()を呼び出します。

呼び出し元がトランザクションモードにいる場合は，関数 tx_close()は失敗します（TX_PROTOCOL_ERROR を返します）。すなわち，幾つかのリソースマネジャが現在のトランザクションに参加していない場合も，リソースマネジャはクローズされません。呼び出し元にリンクされているすべてのリソースマネジャがクローズできれば，tx_close()は正常にリターンします（TX_OK が返ります）。

リターン値

成功した場合，tx_close()は TX_OK を返します。これは，負ではないリターン値です。『0 が返ります。』『関数の呼び出し元に結合していたリソースマネジャの集合はクローズされます。』

エラー

次のような場合，tx_close()はエラーリターンして，次のうちどれか一つの値を返します。これは，負のリターン値です。

リターン値	リターン値（数値）	意味
TX_PROTOCOL_ERROR	-5	関数は，適切でないコンテクストで呼ばれました（例えば，呼び出し元がトランザクションモードにある場合）。リソースマネジャは一つもクローズされません。
TX_ERROR	-6	トランザクションマネジャ，または一つ以上のリソースマネジャで一時的なエラーが起こりました。クローズできるリソースマネジャはすべてクローズされます。
TX_FAIL	-7	トランザクションマネジャ，または一つ以上のリソースマネジャで，回復できないエラーが起こりました。トランザクションマネジャ，または一つ以上のリソースマネジャのどちらか，またはその両方が，すでにアプリケーションに代わって作業できないことがエラーの原因です。

参照

tx_open()

『指定例』

```
『if (tx_info(NULL) == 0 && tx_close() < 0)
    fputs(”cannot close resource manager¥n”,stderr);』
```

『OpenTP1 で使う場合の注意事項』

1. 『tx_close()でクローズできるのは，X/Open の XA インタフェースに準拠しているリソースマネジャだけです。』

tx_commit

名称

トランザクションのコミット

形式

ANSI C，C++ の形式

```
#include <tx.h>
int  tx_commit(void)
```

K&R 版 C の形式

```
#include <tx.h>
int  tx_commit()
```

機能

関数 tx_commit()は，呼び出し元のスレッドのトランザクションの作業をコミットします。

transaction_control 特性が，TX_UNCHAINED に設定されていた場合に，tx_commit()がリターンすると，呼び出し元スレッドはトランザクションモードにはいません。しかし，transaction_control 特性が TX_CHAINED に設定されていた場合は，tx_commit()がリターンすると，呼び出し元スレッドは新しいトランザクションのためにトランザクションモードにとどまります。transaction_control 特性については，tx_set_transaction_control()を参照してください（リターン値を参照）。

『次に示す関数で設定した値は，tx_commit()の処理に影響を与えます。』

- tx_set_commit_return()
- tx_set_transaction_control()
- tx_set_transaction_timeout()

リターン値

成功した場合，tx_commit()は TX_OK を返します。これは，負ではないリターン値です。『0 が返ります。』『transaction_control 特性が TX_CHAINED に設定されていた場合，新しいグローバルトランザクションが開始します。』

エラー

次のような場合，tx_commit()はエラーリターンして，次のうちどれか一つの値を返します。これは，負のリターン値です。

リターン値	リターン値（数値）	意味
TX_ROLLBACK	-2	トランザクションはコミットできないで，ロールバックしました。 transaction_control 特性が TX_CHAINED に設定されている場合は，新しいトランザクションが開始されます。
TX_MIXED	-3	トランザクションは部分的にコミット，部分的にロールバックしました。transaction_control 特性が TX_CHAINED が設定されている場合は，新しいトランザクションが開始されます。
TX_HAZARD	-4	障害が原因で，トランザクションは部分的にコミット，部分的にロールバックされた可能性があります。 transaction_control 特性が TX_CHAINED に設定されている場合は，新しいトランザクションが開始されます。 トランザクションサービス定義で，trn_extend_function オペランドに 00000001 を指定し，1 相コミット時にリソースマネジャからのリターン値が XAER_NOTA の場合も，TX_HAZARD を返します。
TX_PROTOCOL_ERROR	-5	関数は，適切でないコンテクストで呼ばれました（例えば，呼び出し元がトランザクションモードにいない場合）。呼び出し元のトランザクションに対する状態は変わりません。
TX_FAIL	-7	トランザクションマネジャ，または一つ以上のリソースマネジャで，回復できないエラーが起こりました。トランザクションマネジャ，または一つ以上のリソースマネジャのどちらか，またはその両方が，すでにアプリケーションに代わって作業できないことがエラーの原因です。呼び出し元のトランザクションに対する状態は不明です。
TX_NO_BEGIN	-100	トランザクションはコミット終了しましたが，新しいトランザクションは開始できないで，呼び出し元はすでにトランザクションモードにいません。 このリターン値は，トランザクション特性が TX_CHAINED のときだけ返ります。
TX_ROLLBACK_NO_BEGIN	-102	トランザクションはコミットできないで，ロールバックしました。新しいトランザクションは開始できないで，呼び出し元はすでにトランザクションモードではありません。 このリターン値は，トランザクション特性が TX_CHAINED のときだけ返ります。
TX_MIXED_NO_BEGIN	-103	トランザクションは部分的にコミット，部分的にロールバックしました。新しいトランザクションは開始できないで，呼び出し元はすでにトランザクションモードにはいません。 このリターン値は，トランザクション特性が TX_CHAINED のときだけ返ります。
TX_HAZARD_NO_BEGIN	-104	障害が原因で，トランザクションは部分的にコミット，部分的にロールバックしました。新しいトランザクションは開始でき

リターン値	リターン値（数値）	意味
TX_HAZARD_NO_BEGIN	-104	ないで，呼び出し元はすでにトランザクションモードにはいません。 このリターン値は，トランザクション特性が TX_CHAINED のときだけ返ります。 トランザクションサービス定義で，trn_extend_function オペランドに 00000001 を指定し，1 相コミット時にリソースマネジャからのリターン値が XAER_NOTA の場合も，TX_HAZARD_NO_BEGIN を返します。

参照

tx_begin()，tx_set_commit_return()，tx_set_transaction_control()，tx_set_transaction_timeout()

『指定例』

```
『if (tx_info(NULL) == 1 && tx_commit() < 0)
   fputs("cannot commit transaction¥n",stderr);』
```

『OpenTP1 で使う場合の注意事項』

1. 『tx_commit()は，グローバルトランザクションを開始した UAP（tx_begin()を呼び出した UAP）のプロセスからだけ呼び出せます。』
2. 『tx_commit()を呼び出すプロセスは，このマニュアルの記述に従って正しく結合された UAP の実行形式ファイルを起動したものでなければなりません。』
3. 『tx_commit()と dc_trn_～の関数は併用できません。』

tx_info

名称

現在のトランザクションに関する情報の返却

形式

ANSI C，C++ の形式

```
#include <tx.h>
int  tx_info(TXINFO *info)
```

K&R 版 C の形式

```
#include <tx.h>
int  tx_info(info)
TXINFO   *info;
```

機能

関数 tx_info()は，グローバルトランザクションの情報を，info で指す構造体に返します。さらに，tx_info()は呼び出し元が現在トランザクションモードにあるかどうかを示す値を返します。

『引数』

『●info』

info に null 以外を設定した場合は，tx_info()は info で示す TXINFO 構造体へ，グローバルトランザクションの情報を設定します。TXINFO 構造体は次のような要素を含んでいます。

```
XID                   xid;
COMMIT_RETURN         when_return;
TRANSACTION_CONTROL   transaction_control;
TRANSACTION_TIMEOUT   transaction_timeout;
TRANSACTION_STATE     transaction_state;
```

トランザクションモードで tx_info()を呼び出した場合は，xid には現在のトランザクションブランチの識別子と，transaction_state の値が設定されます。

トランザクションモードでない呼び出し元から tx_info()を呼び出した場合には，xid に null XID が設定されます（詳細は<tx.h>を参照）。さらに，呼び出し元がトランザクションモードであるかないかにかかわらず，when_return，transaction_control，transaction_timeout は，現在設定されている when_return，transaction_control と秒単位のトランザクションタイムアウト値と同じ値になります。

返されたトランザクションタイムアウト値は，次のトランザクションモードが開始されたときに設定されて反映されます。しかし，呼び出し元の現在のトランザクションでは，タイムアウト値に反映されていな

い場合もあります。その理由は，現在のトランザクションの開始後 tx_set_timeout()が呼び出された場合，timeout 値を変更しているかもしれないからです。

info に null を設定した場合は，TX_INFO 構造体は返されません。

リターン値

関数の呼び出し元がトランザクションモードの場合は，1 が返ります。関数の呼び出し元がトランザクションモードでない場合は，0 が返ります。

エラー

次のような場合，tx_info()はエラーリターンして，次のうちどれか一つの値を返します。これは，負のリターン値です。

リターン値	リターン値（数値）	意味
TX_PROTOCOL_ERROR	-5	関数は，適切でないコンテクストで呼ばれました（例えば，呼び出し元がまだ tx_open()を呼び出していない）。
TX_FAIL	-7	トランザクションマネジャで，回復できないエラーが発生しました。トランザクションマネジャが，すでにアプリケーションに代わって作業できないことがエラーの原因であることを示します。

アプリケーションの使い方

同じグローバルトランザクション内で，複数回の tx_info()を呼び出すと，tx_info()への呼び出しでは，gtrid（グローバルトランザクション識別子）に同じ XID が供給されることを保証されています。しかし，bqual（ブランチ限定子）が同じかどうかは一概にはいえません。

参照

tx_open()，tx_set_commit_return()，tx_set_transaction_control()，tx_set_transaction_timeout()

『指定例』

```
『if (tx_info(NULL) != 1)
   fputs(”not transaction mode¥n”,stderr);』
```

tx_open

名称

リソースマネジャ集合のオープン

形式

ANSI C，C++ の形式

```
#include <tx.h>
int  tx_open(void)
```

K&R 版 C の形式

```
#include <tx.h>
int  tx_open()
```

機能

関数 tx_open()は，互換性がある方法で，リソースマネジャの集合をオープンします。tx_open()でリソースをオープンすると，トランザクションマネジャがリソースマネジャ固有の情報をトランザクションマネジャ独自の方法で読み，この情報を，呼び出し元にリンクされているリソースマネジャに渡します。

関数 tx_open()は，呼び出し元にリンクされているすべてのリソースマネジャをオープンしようとします。tx_open()は，リソースマネジャ固有のオープン呼び出しの代わりに使われて，アプリケーションプログラムからの互換性を阻害する呼び出しを不要にします。

リソースマネジャは，初期化セマンティクスがそれぞれ異なっています。そのため，特有のリソースマネジャをオープンするのに必要な情報は，それぞれのリソースマネジャから公開されてなければなりません。

tx_open()が TX_ERROR を返したら，リソースマネジャは一つもオープンされていません。tx_open()が TX_OK を返したら，一つ以上のリソースマネジャがオープンされています。アプリケーションからアクセスしたときにリソースマネジャがオープンしていないと，リソースマネジャ固有のエラーを返します。あるスレッドがグローバルトランザクションに参加する前には，tx_open()が成功してリターンしていなければなりません。

関数 tx_open()が成功したあとでも，繰り返し tx_open()を（tx_close()を呼び出す前なら）呼び出せます。繰り返し tx_open()を呼び出しても正常にリターンしますが，トランザクションマネジャでは，リソースマネジャの再オープン処理は一切しません。

リターン値

成功した場合，tx_open()は TX_OK を返します。これは，負ではないリターン値です。『0 が返ります。』『関数の呼び出し元に結合している一つ以上のリソースマネジャの集合は，オープンされます。』

エラー

次のような場合，tx_open()はエラーリターンして，次のうちどれか一つの値を返します。これは，負のリターン値です。

リターン値	リターン値（数値）	意味
TX_ERROR	-6	トランザクションマネジャ，または一つ以上のリソースマネジャで一時的なエラーが起こりました。リソースマネジャは一つもオープンできませんでした。
TX_FAIL	-7	トランザクションマネジャ，または一つ以上のリソースマネジャで，回復できないエラーが起こりました。トランザクションマネジャ，一つ以上のリソースマネジャのどちらか，またはその両方が，アプリケーションに代わって作業できないことがエラーの原因です。 または，実行環境がジャーナルファイルレスモードのため，トランザクションマネジャでエラーが起こりました。

参照

tx_close()

『指定例』

```
『if ( tx_open() < 0)
   fputs(”cannot open resource manager¥n”,stderr);』
```

『OpenTP1 で使う場合の注意事項』

1. 『tx_open()でオープンできるのは，X/Open の XA インタフェースに準拠しているリソースマネジャだけです。』
2. 『tx_open()と dc_trn_～の関数は併用できません。』

tx_rollback

名称

トランザクションのロールバック

形式

ANSI C，C++ の形式

```
#include <tx.h>
int  tx_rollback(void)
```

K&R 版 C の形式

```
#include <tx.h>
int  tx_rollback()
```

機能

関数 tx_rollback()は，呼び出しスレッドのトランザクションの作業をロールバックします。

transaction_control 特性が，TX_UNCHAINED に設定されていた場合で，tx_rollback()がリターンすると，呼び出し元スレッドはトランザクションモードにはいません。しかし，transaction_control 特性が TX_CHAINED に設定されていた場合で，tx_rollback()がリターンすると，呼び出し元スレッドは新しいトランザクション下のトランザクションモードにとどまります。transaction_control 特性については，tx_set_transaction_control()を参照してください。

『次に示す関数で設定した値は，tx_rollback()の処理に影響を与えます。』

- tx_set_commit_control()
- tx_set_transaction_timeout()

『tx_rollback()は，MHP では使えません。』

リターン値

成功した場合，tx_rollback()は TX_OK を返します。これは，負ではないリターン値です『0 が返ります』『transaction_control 特性が TX_CHAINED に設定されていた場合は，新しいトランザクションが開始されます』『tx_rollback()を呼び出した SPP がルートトランザクションブランチでない場合は，実際のロールバック処理はしないで，そのトランザクションブランチを rollback_only 状態にするだけです。ルートトランザクションブランチの同期点処理でロールバックの指示があるまで，トランザクションモードのままとどまります。』

エラー

次のような場合，tx_rollback()はエラーリターンして，次のうちどれか一つの値を返します。これは，負のリターン値です。

リターン値	リターン値（数値）	意味
TX_MIXED	-3	トランザクションは部分的にコミット，部分的にロールバックしました。transaction_control 特性が TX_CHAINED と設定されていれば，新しいトランザクションが開始されます。
TX_HAZARD	-4	障害が原因で，トランザクションは部分的にコミット，部分的にロールバックした可能性があります。transaction_control 特性が TX_CHAINED に設定されていれば，新しいトランザクションが開始されます。
TX_PROTOCOL_ERROR	-5	関数は，適切でないコンテクストで呼ばれました（例えば，呼び出し元がトランザクションモードでない）。
TX_FAIL	-7	トランザクションマネジャ，または一つ以上のリソースマネジャで，回復できないエラーが起こりました。トランザクションマネジャ，または一つ以上のリソースマネジャのどちらか，またはその両方が，すでにアプリケーションに代わって作業できないことがエラーの原因です。呼び出し元のトランザクションに対する状態は不明です。
TX_COMMITTED	-9	トランザクションは，ヒューリスティックにコミットしました。 transaction_control 特性が TX_CHAINED に設定されていれば，新しいトランザクションが開始されます。
TX_NO_BEGIN	-100	トランザクションはロールバックしましたが，新しいトランザクションを開始できないで，呼び出し元はすでにトランザクションモードにいません。 このリターン値は，transaction_control 特性が TX_CHAINED のときだけ返ります。
TX_MIXED_NO_BEGIN	-103	トランザクションは部分的にコミット，部分的にロールバックしました。新しいトランザクションは開始できないで，呼び出し元は，すでにトランザクションモードにはいません。 このリターン値は，transaction_control 特性が TX_CHAINED のときだけ返ります。
TX_HAZARD_NO_BEGIN	-104	障害が原因で，トランザクションは部分的にコミット，部分的にロールバックした可能性があります。新しいトランザクションは開始できないで，呼び出し元はすでにトランザクションモードではありません。 このリターン値は，transaction_control 特性が TX_CHAINED のときだけ返ります。
TX_COMMITTED_NO_BEGIN	-109	トランザクションは，ヒューリスティックにコミットしました。新しいトランザクションは開始できないで，呼び出し元はすでにトランザクションモードにはいません。このリターン値

リターン値	リターン値（数値）	意味
TX_COMMITTED_NO_BEGIN	-109	は，transaction_control 特性が TX_CHAINED のときだけ返ります。

参照

tx_begin()，tx_set_transaction_control()，tx_set_transaction_timeout()

『指定例』

```
『if (tx_info(NULL) == 1 && tx_rollback() < 0)
    fputs(”cannot rollback transaction¥n”,stderr);』
```

『OpenTP1 で使う場合の注意事項』

1. 『transaction_control 特性が TX_CHAINED に設定されていた場合，tx_rollback()を使えるのはルートトランザクションブランチ（tx_begin()を呼び出した UAP）だけです。』
2. 『transaction_control 特性が TX_UNCHAINED に設定されていた場合，tx_rollback()はルートトランザクションブランチでなくても呼び出せますが，呼び出したトランザクションブランチによって処理が異なります。tx_rollback()の呼び出し元がルートブランチの場合，非ルートブランチに対して，RPC の関数を介してロールバック要求をしますが，非ルートブランチで tx_rollback()を呼び出した場合，呼び出し元の非ルートブランチは rollback_only を記録するだけで，ルートブランチに対して RPC の関数を介してロールバック要求をしません。呼び出し元の非ルートブランチは，ルートブランチの指示を待ったあと，ロールバック処理をします。』
3. 『tx_rollback()と dc_trn_～の関数は併用できません。』

tx_set_commit_return

名称

commit_return 特性の設定

形式

ANSI C，C++ の形式

```
#include <tx.h>
int  tx_set_commit_return(COMMIT_RETURN  when_return)
```

K&R 版 C の形式

```
#include <tx.h>
int  tx_set_commit_return(when_return)
COMMIT_RETURN   when_return;
```

機能

関数 tx_set_commit_return()は，commit_return 特性を when_return に指定した値に設定します。commit_return 特性は，tx_commit()が呼び出し元に制御を戻す方法に対して影響します。

tx_set_commit_return()は，呼び出し元がトランザクションモードかどうかに関係なく呼び出せます。when_return に設定した commit_return 特性は，あとから呼び出される tx_set_commit_return()で変更されるまで引き継がれて，効力を発揮します。

commit_return 特性の初期設定値は，製品の仕様に依存します。『OpenTP1 の場合は，TX_COMMIT_COMPLETED です。』

『引数』

『●when_return』

次の二つのどちらかの値が，when_return の妥当な設定です。

```
{TX_COMMIT_DECISION_LOGGED|TX_COMMIT_COMPLETED}
```

- TX_COMMIT_DECISION_LOGGED『この引数は，OpenTP1 の該当バージョンでは使えません。when_return に TX_COMMIT_DECISION_LOGGED を設定した場合は，リターン値 TX_NOT_SUPPORTED でエラーリターンします。』
  このフラグは，コミット決定を 2 相コミットプロトコルの 1 相目でジャーナルに書いたが，2 相目がまだ完了していないときに，tx_commit()がリターンすることを示します。このフラグを設定すれば，tx_commit()の呼び出し元に早くリターンできますが，トランザクションの参加者がヒューリスティック決定をしてしまう危険性があります。この場合，tx_commit()はすでにリターンしているので『トランザクションマネジャは，それ独自の方法でリソースマネジャがヒューリスティック決定をしたことを

示すことに注意』，ヒューリスティック決定が起こると，呼び出し元にリターンコードで示すことはできません。

通常の条件では，1 相目にコミットを約束した参加者は，2 相目の処理の間にコミットをします。一定の特殊な条件下では，長期間のネットワーク，サイト障害などによって，2 相目の完了ができないで，ヒューリスティック決定が起こるようなこともあります。トランザクションマネジャは，このフラグは選択しないことにした場合，この値が設定されていないことを示すため，TX_NOT_SUPPORTED を返します。

- TX_COMMIT_COMPLETED

  このフラグは，2 相コミットプロトコルが完全に終了したあとで tx_commit()がリターンすることを示します。このフラグを設定すると，コミットの 2 相目でヒューリスティックな決着が起こっても，tx_commit()の呼び出し元にリターンコードで知らせることができます。トランザクションマネジャは，この機能を使えないようにできます。そして TX_NOT_SUPPORTED を返す理由にこのフラグを使えない意味を含めることができます。

リターン値

『リターン値が 0 で』成功した場合，tx_set_commit_return()は TX_OK を返します。これは，負ではないリターン値です。『この場合，commit_return 特性の設定は，when_return に設定した値に変更されます。』

『リターン値が正の値で』成功した場合，tx_set_commit_return()は TX_NOT_SUPPORTED を返します。これは，負ではないリターン値です。『when_return に設定した値は，システムのトランザクションマネジャでは使えません。』

この場合の commit_return 特性は，変更されません。トランザクションマネジャは TX_COMMIT_COMPLETED または TX_COMMIT_DECISION_LOGGED のどちらかは，when_return の値として使えるようにしておく必要があります。『OpenTP1 の場合は，TX_COMMIT_RETURN です。』

エラー

次のような場合，tx_set_commit_return()はエラーリターンして，次のうちどれか一つの値を返します。これは，負のリターン値です。負のリターン値が返った場合は，tx_set_commit_return()は commit_return 特性を変更しません。

リターン値	リターン値（数値）	意味
TX_PROTOCOL_ERROR	-5	関数は，適切でないコンテクストで呼ばれました（例えば，呼び出し元が tx_open()を呼び出していない）。
TX_FAIL	-7	トランザクションマネジャで，回復できないエラーが起こりました。トランザクションマネジャが，すでにアプリケーションに代わって作業できないことが，エラーの原因です。

リターン値	リターン値（数値）	意味
TX_EINVAL	-8	when_return に設定した値が，TX_COMMIT_DECISION_LOGGED でも，TX_COMMIT_COMPLETED でもありません。

参照

tx_commit()，tx_open()，tx_info()

『指定例』

```
『if (tx_set_commit_return(TX_COMMIT_COMPLETED) == 0 &&
                                        tx_commit() < 0 )
   fputs(”cannot commit transaction¥n”,stderr);』
```

『OpenTP1 で使う場合の注意事項』

1.『tx_set_commit_return()と dc_trn_～の関数は併用できません。』

tx_set_transaction_control

名称

transaction_control 特性の設定

形式

ANSI C，C++ の形式

```
#include <tx.h>
int  tx_set_transaction_control(TRANSACTION_CONTROL  control)
```

K&R 版 C の形式

```
#include <tx.h>
int  tx_set_transaction_control(control)
TRANSACTION_CONTROL   control;
```

機能

関数 tx_set_transaction_control()は，transaction_control 特性を，control に指定した値に設定します。transaction_control 特性は，tx_commit()，tx_rollback()が呼び出し元にリターンする前に，新しいトランザクションが開始するかどうかを決定します。

関数 tx_set_transaction_control()は，呼び出し元がトランザクションモードかどうかに関係なく呼び出せます。control に設定した transaction_control 特性は，あとから呼び出される tx_set_transaction_control()で変更されるまで引き継がれて，効力を発揮します。

transaction_control 特性の初期設定は，TX_UNCHAINED です。

『引数』

『●control』

次の二つのどちらかの値が，control の妥当な設定です。

```
{TX_UNCHAINED|TX_CHAINED}
```

- TX_UNCHAINED

  このフラグは，tx_commit()と tx_rollback()が呼び出し元にリターンする前に，新たなトランザクションを開始しないことを示します。呼び出し元は，新しいトランザクションを開始する際には，tx_begin()を呼び出さなければなりません。

- TX_CHAINED

  このフラグは，tx_commit()と tx_rollback()が呼び出し元にリターンする前に，新しいトランザクションを開始することを示します。

リターン値

成功した場合，tx_set_transaction_control()は TX_OK を返します。これは，負ではないリターン値です。『0 が返ります。』『transaction_control 特性の設定は，control の値に変更されました。』

エラー

次のような場合，tx_set_transaction_control()はエラーリターンして，次のうちどれか一つの値を返します。これは，負のリターン値です。負のリターン値が返った場合は，tx_set_transaction_control()は transaction_control 特性を変更しません。

リターン値	リターン値（数値）	意味
TX_PROTOCOL_ERROR	-5	関数は，適切でないコンテクストで呼ばれました（例えば，呼び出し元がまだ tx_open()を呼び出していない）。
TX_FAIL	-7	トランザクションマネジャで，回復できないエラーが起こりました。トランザクションマネジャが，すでにアプリケーションに代わって作業できないことがエラーの原因です。
TX_EINVAL	-8	control に設定した値は，TX_UNCHAINED でも，TX_CHAINED でもありません。

参照

tx_begin()，tx_commit()，tx_open()，tx_rollback()，tx_info()

『指定例』

```
『if (tx_set_transaction_return(TX_UNCHAINED) == 0 &&
                                        tx_commit() < 0 )
   fputs(”cannot commit transaction¥n”,stderr);』
```

『OpenTP1 で使う場合の注意事項』

1.『tx_set_transaction_control()と dc_trn_～の関数は併用できません。』

tx_set_transaction_timeout

名称

transaction_timeout 特性の設定

形式

ANSI C，C++ の形式

```
#include <tx.h>
int  tx_set_transaction_timeout(TRANSACTION_TIMEOUT  timeout)
```

K&R 版 C の形式

```
#include <tx.h>
int  tx_set_transaction_timeout(timeout)
TRANSACTION_TIMEOUT   timeout;
```

機能

関数 tx_set_transaction_timeout()は，timeout に指定した値に transaction_timeout 特性を設定します。transaction_timeout 特性の値には，トランザクションを完了させなければならない時間，つまりトランザクションをタイムアウトする時間を設定します。ここで示す時間とは，UAP が tx_begin()を呼び出してから tx_commit()を呼び出すまで，または tx_begin()を呼び出してから tx_rollback()を呼び出すまでの時間を示します。

tx_set_transaction_timeout()は，呼び出し元がトランザクションモードにいるかどうかに関係なく呼び出せます。tx_set_transaction_timeout()がトランザクションモードから呼び出されても，新しいタイムアウトの値は，次のトランザクションまで効果を発揮しません。

transaction_timeout 特性の初期設定は，0（タイムアウトなし）です。

『システム定義の trn_expiration_time に値を指定している場合は，その値が初期設定になります。』

『引数』

「●timeout

トランザクションがタイムアウトとなる時間を，秒数で設定します。秒数には，そのシステムで定義された long 型の最大値までの値が設定できます。0 を設定した場合は，タイムアウトとなりません。』

リターン値

成功した場合，tx_set_transaction_timeout()は TX_OK を返します。これは，負ではないリターン値です。『0 が返ります。』『transaction_timeout 特性の設定は，timeout の値に変更されました。』

エラー

次のような場合，tx_set_transaction_timeout()はエラーリターンして，次のうちどれか一つの値を返します。これは，負のリターン値です。負のリターン値が返った場合は，tx_set_transaction_timeout()はtransaction_timeout 特性を変更しません。

リターン値	リターン値（数値）	意味
TX_PROTOCOL_ERROR	-5	関数は，適切でないコンテクストで呼ばれました（例えば，呼び出し元が tx_open()をまだ呼び出していない）。
TX_FAIL	-7	トランザクションマネジャで，回復できないエラーが起こりました。トランザクションマネジャが，すでにアプリケーションに代わって作業できないことがエラーの原因です。
TX_EINVAL	-8	timeout に設定した値が間違っています。

参照

tx_begin()，tx_commit()，tx_open()，tx_rollback()，tx_info()

『指定例』

```
『if (tx_set_transaction_timeout(TRANSACTION_TIMEOUT)
                                        == 0 && tx_commit() < 0 )
   fputs(”cannot commit transaction¥n”,stderr);』
```

『OpenTP1 で使う場合の注意事項』

1.『tx_set_transaction_timeout()と dc_trn_～の関数は併用できません。』

5 OpenTP1 のライブラリ関数の文法（アソシエーションの状態の通知）

通信プロトコルに OSI TP を使ったクライアント/サーバ形態の通信では，通信イベント処理用 SPP が必要です。この章では，通信イベント処理用 SPP で使うライブラリ関数と，受信する通信イベントの形式について説明します。

アソシエーションの操作（dc_xat_～）

通信イベント処理用 SPP で使う，アソシエーションを操作する関数について説明します。アソシエーションを操作する関数を次に示します。

- dc_xat_connect － アソシエーションの確立

アソシエーションの操作の関数（dc_xat_～）は，TP1/Server Base の場合にだけ使えます。TP1/LiNK では，アソシエーションを操作する関数は使えません。

アソシエーションを操作する関数を呼び出せるのは，**通信イベント処理用 SPP だけ**です。ほかの OpenTP1 の UAP（SUP，SPP，MHP）では，アソシエーションを操作する関数は使えません。

通信イベント処理用 SPP のユーザサービス定義の server_type オペランドには，必ず"betran"を指定してください。

dc_xat_connect

名称

アソシエーションの確立

形式

ANSI C，C++の形式

```
#include <dcxat.h>
int  dc_xat_connect(char *svcname,char *aso_name,DCLONG flags)
```

K&R 版 C の形式

```
#include <dcxat.h>
int  dc_xat_connect(svcname,aso_name,flags)
char       *svcname;
char       *aso_name;
DCLONG     flags;
```

機能

svcname に設定した XATMI 通信サービスに，aso_name に設定したアソシエーションの確立を要求します。

dc_xat_connect 関数は，相手システムにアソシエーションの確立要求を送信してからリターンします。アソシエーション確立の通知は，dc_xat_connect 関数では受け取れません。

dc_xat_connect 関数は，TP1/NET/OSI-TP-Extended を使った OSI TP 通信をするときにだけ使えます。

dc_xat_connect 関数は，トランザクションの処理の範囲からでも範囲外からでも呼び出せます。

UAP で値を設定する引数

●svcname

アソシエーションの確立を要求する先の XATMI 通信サービス名を設定します。XATMI 通信サービス名として，XATMI 通信サービス定義ファイル名（_xatc）を設定します。

●aso_name

確立するアソシエーション名を設定します。アソシエーション名として，

TP1/NET/OSI-TP-Extended の定義のプロトコル固有定義 nettalccn オペランドの-c オプションに指定したコネクション名を設定します。

●flags

DCNOFLAGS を設定します。

リターン値

リターン値	リターン値（数値）	意味
DC_OK	0	正常に終了しました。
DCXATER_INVAL	-4570	引数に設定した値が間違っています。
DCXATER_MEMORY	-4571	メモリが不足しました。
DCXATER_PROTO	-4572	dc_rpc_open 関数を呼び出していません。
DCXATER_NOT_FOUND	-4575	XATMI 通信サービスのアドレス情報を取得できません。
DCXATER_TERMINATING	-4576	XATMI 通信サービスは終了処理中です。
DCXATER_COMM_SEND	-4577	XATMI 通信サービスへの送信時に，サービス要求が失敗しました。
DCXATER_COMM_RECV	-4578	XATMI 通信サービスからの受信時に，サービス要求が失敗しました。XATMI 通信サービスがコネクション確立要求を実行している可能性があります。
DCXATER_ASO_NAME	-4580	設定したアソシエーション名を定義していません。
DCXATER_ASO_CONNECT_ALREADY	-4581	アソシエーションはすでに確立しています。
DCXATER_ASO_CONNECTING	-4582	アソシエーションは確立処理中です。
DCXATER_ASO_DISCONNECTING	-4583	アソシエーションは解放処理中です。
DCXATER_ASO_INITIATE	-4584	アソシエーションは着呼モードのため，確立できません。

受信する通信イベントの形式

アソシエーションの状態を示す通信イベントの形式について説明します。通信イベントを受信するときは，**通信イベント処理用 SPP のサービスグループ名とサービス名を XATMI 通信サービス定義**に指定しておきます。このとき，どのオペランドにサービスグループ名とサービス名を指定するかで，受け取れる通信イベントが異なります。

xat_aso_con_event_svcname オペランド

　アソシエーションの確立通知の通信イベント

xat_aso_discon_event_svcname オペランド

　アソシエーションの正常解放の通信イベント

xat_aso_failure_event_svcname オペランド

　アソシエーションの異常解放の通信イベント

複数のオペランドに同じサービスグループ名とサービス名を指定すると，一つの通信イベント処理用 SPP が複数の通信イベントを受信できるようになります。

通信イベントは，**構造体**で通知されます。通信イベントの構造体は，ヘッダファイル**<dcxat.h>**で定義してあります。通信イベント処理用 SPP では，#include で<dcxat.h>をインクルードしてください。

```
struct dc_xat_event_type{
     int     event_code;       … 通信イベント識別コード
     char    aso_name[9];      … アソシエーション名
     char    reserve1[3];      … 予備領域1
     int     aso_initiative;   … アソシエーション発呼，着呼種別
     DCULONG reason_code;      … 理由コード
     char    xatc_svcname[9];  … XATMI通信サービス名
     char    reserve2[63];     … 予備領域2
};
```

引数の内容

●event_code

通信イベントを識別するコードが設定されます。( ) 内は該当するコードの数値表示（10 進数）を示します。

DCXAT_ASO_CONNECT（00000001）

　アソシエーションの確立

DCXAT_ASO_DISCONNECT（00000002）

　アソシエーションの正常解放

DCXAT_ASO_FAILURE（00000003）

　アソシエーションの異常解放

●aso_name

通信イベントで状態が通知されるアソシエーションの，アソシエーション名が設定されます。

●reserve1

予備の領域です。

●aso_initiative

確立したアソシエーションで，自システムが発呼となるか着呼となるかを示す値が設定されます。()内は該当するコードの数値表示（10 進数）を示します。

DCXAT_ASO_INIT（00000001）
　自システムが発呼側

DCXAT_ASO_RESP（00000002）
　自システムが着呼側

●reason_code

アソシエーションが解放されたときの，理由コードが設定されます。( ) 内は該当するコードの数値表示（10 進数）を示します。

アソシエーションの正常解放の場合は次の値のどれかが設定されます。

DCXAT_RSN_COMMAND（00000001）
　コマンド実行によるアソシエーションの解放

DCXAT_RSN_XATMI（00000005）
　XATMI によるアソシエーションの解放

DCXAT_RSN_REMOTE（00000007）
　相手システムからのアソシエーションの正常解放

DCXAT_RSN_TP_NORMAL（00000008）
　TP 層によるアソシエーションの正常解放

アソシエーションの異常解放の場合は次の値のどれかが設定されます。

DCXAT_RSN_COMMAND（00000001）
　コマンド実行によるアソシエーションの強制解放

DCXAT_RSN_LOWER（00000003）
　下位層の障害（回線障害，通信管理の障害など）

DCXAT_RSN_OSITP（00000004）
　TP1/NET/OSI-TP-Extended の障害（プロトコルエラーなど）

DCXAT_RSN_XATMI（00000005）
　XATMI 通信サービスによるアソシエーションの強制解放

DCXAT_RSN_FAILURE（00000006）

アソシエーション確立の失敗

DCXAT_RSN_REMOTE（00000007）

相手システムからのアソシエーションの強制解放

●xatc_svcname

XATMI 通信サービス名が設定されます。

●reserve2

予備の領域です。

6 X/Open に準拠したアプリケーション間通信（TxRPC）

この章では，X/Open で規定するアプリケーション間通信（TxRPC）で使うインタフェース定義ファイル（IDL ファイル）と IDL コンパイラ（txidl コマンド）の文法について説明します。

6.1　TxRPC で通信するときの準備手順

TxRPC で通信するための準備作業について説明します。

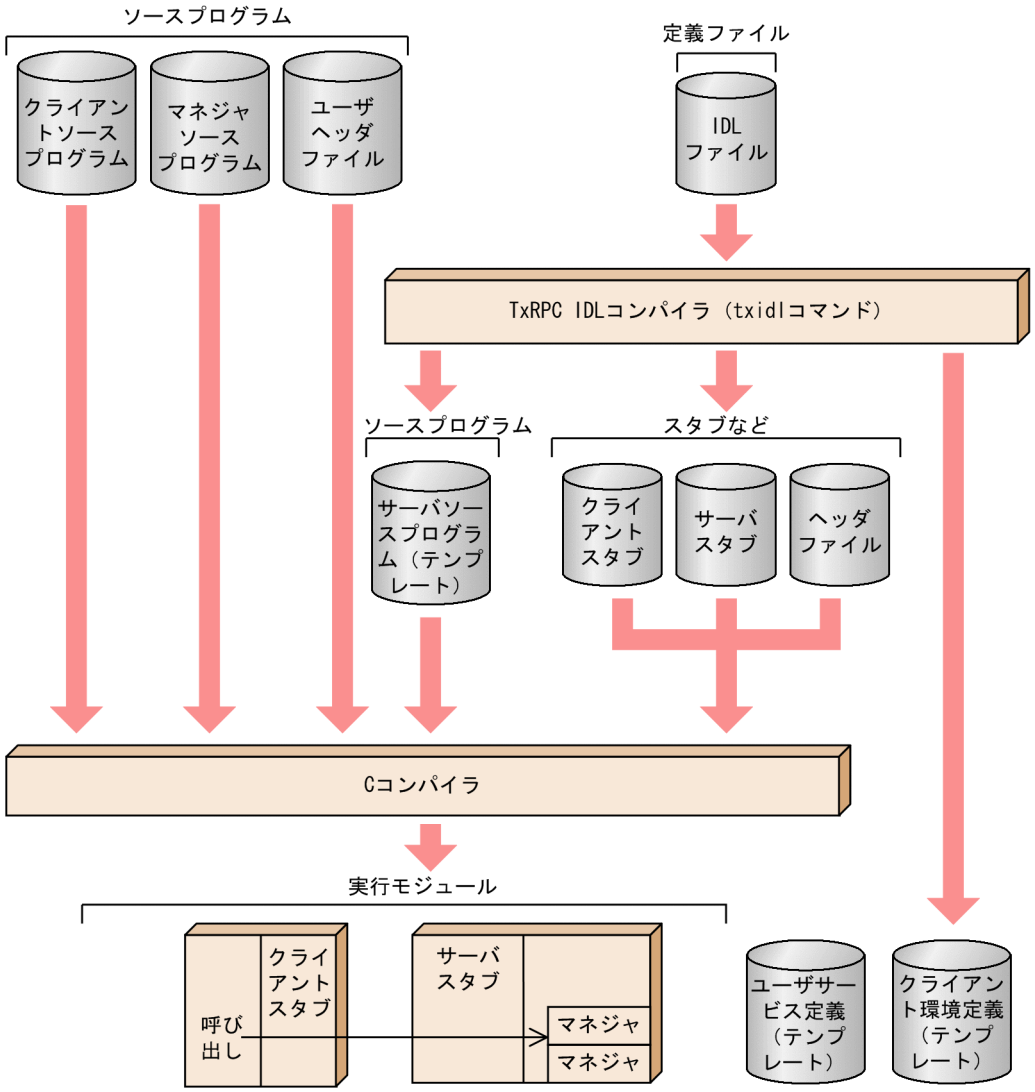
6.1.1　IDL-only TxRPC を使う場合の手順

IDL-only TxRPC の通信をする UAP を作成するときの手順は，次のとおりです。

1. **インタフェース定義言語ファイル（IDL ファイル）**を作成します。
2. IDL ファイルを **IDL コンパイラ（txidl コマンド）**でコンパイルします。
3. txidl コマンドで生成されたサーバ UAP のテンプレートを基に，**プログラムをコーディング**します。一緒にクライアント UAP もコーディングします。
4. txidl コマンドで生成されたスタブとコーディングしたプログラムを，**C コンパイラ**でコンパイル，リンケージします。

IDL-only TxRPC で通信する UAP を作成する手順を次の図に示します。

図 6-1　IDL-only TxRPC で通信する UAP を作成する手順



(1)　ユーザが作成するファイル

ユーザは，次に示すファイルを作成します。

- **クライアントプログラム**

  クライアント UAP のプログラムです。

- **マネジャプログラム**

  OpenTP1 のサービス関数に該当するオペレーション関数を含むプログラムです。要求されたサービスを処理します。

- **ユーザヘッダファイル**

  クライアントプログラム，マネジャプログラムで使うヘッダファイルを作成します。

- **インタフェース定義言語ファイル（IDL ファイル）**

  通信インタフェースを定義するファイルです。

(2) IDL コンパイラが生成するファイル

IDL コンパイラ（txidl コマンド）が生成するファイルを次に示します。

- **クライアントスタブ**
  クライアントプログラムにリンケージするスタブです。
- **サーバスタブ**
  サーバプログラムにリンケージするスタブです。
- **ヘッダファイル**
  TxRPC で使う宣言を定義したファイルです。
- **サーバプログラムのテンプレート**
  ユーザの業務をするサーバプログラムのテンプレートです。
- **ユーザサービス定義のテンプレート**
  ユーザが作成したプログラムについて設定したユーザサービス定義のテンプレートです。
- **クライアント環境定義のテンプレート**
  ユーザが作成したプログラムについて設定した，TP1/Client のクライアント環境定義のテンプレートです。txidl コマンドにゲートウェイプログラムを作成するときのオプションを指定したときに作成されます。

これらのうち，サーバプログラムのテンプレート，ユーザサービス定義のテンプレートおよびクライアント環境定義のテンプレートは，ユーザが修正して使います。テンプレートについては，「7.5　TxRPC の例題（IDL コンパイラが生成するテンプレート）」を参照してください。

6.2　アプリケーションプログラムを作るときの注意

TxRPC で通信する UAP をコーディングするときの留意事項について説明します。

6.2.1　TxRPC の通信で使うプログラムの名称の付け方の注意

オペレーション関数（サービス関数）には，先頭が英字で始まる英数字で，任意の名称を付けることができます。ただし，次の名称は使わないでください。

- "dc"で始まる名称
- "CBLDC"で始まる名称
- "tx"，または"TX"で始まる名称
- "tp"，または"TP"で始まる名称

そのほかの名称（外部変数名，定数名）については，OpenTP1 のライブラリを使った UAP と同じです。名称の制限については，「1.1.2　コーディング規約」を参照してください。

また，コーディングするプログラムや，ヘッダファイルの識別子にも，上記の名称は使えません。

6.2.2　TxRPC の通信以外のプログラムで使えない名称

TxRPC ではインタフェース名を，OpenTP1 内部の処理で使っています。ほかのプログラムの処理で，この名称をサービスグループ名として使わないでください。

（例）

インタフェース名"timope"の場合は，"timope"は使えません。

6.2.3　TxRPC の制限事項

TxRPC の通信では，次に示す制限事項があります。

1. IDL-only TxRPC では，dc_rpc_open 関数，dc_adm_complete 関数などを UAP で呼び出す必要があります。
2. コンテキストハンドルは使えません。
3. IDL ファイルでは，#ifdef でマクロ変数の宣言はできません。
4. C コンパイラの仕様によっては，生成したスタブでコンパイルエラーになる場合があります。

5. txidl コマンドは，ファイルの内容が ANSI 仕様に準拠しているかどうかをチェックしません。そのため，ANSI 仕様でだけ有効な記述がある IDL ファイルをコンパイルした場合，生成されるスタブは，ANSI 仕様に準拠した C コンパイラだけしか使えません。
6. UAP とスタブは，同じ C コンパイラでコンパイルしてください。

6.3　インタフェース定義言語ファイル（IDL ファイル）の作成

IDL ファイルの作り方について説明します。

6.3.1　文法規則

IDL ファイルは，次に示す規則に従って作成してください。

(1)　ファイル名

ファイル名には，".idl"というサフィックスを付けます。クライアント UAP とサーバ UAP に同じ IDL ファイルを組み込んでおく必要があります。

(2)　字句要素

(a)　識別子

識別子には，次に示す文字を使ってください。

- アルファベット（大文字と小文字）
- 0 から 9 までの数字
- アンダスコア（ _ ）

さらに，最初の文字はアルファベットを使ってください。また，特に記述がないかぎり，31 文字までの長さで指定してください。

(b)　使えない用語（キーワード）

IDL ファイルを記述するときは，幾つかの識別子をキーワードとして予約しています。このキーワードは変更できません。

(c)　句読点記号

次に示す図形文字を使えます。

" ' ( ) * , . / : ; | = [＼] { }

(d)　空白

次の文字を空白と見なします。

- 空白
- 改行
- 水平タブ

- 行の先頭の改ページ
- コメント行
- これらの空白構造体のうち一つまたは複数が連続したもの

次に示す部分には，空白が必要です。

- 句読点記号が前にないキーワード，識別子，数字の前には空白が必要です。
- キーワード，識別子，数字の後ろに句読点記号がない場合は，後ろに空白が必要です。
- ほかの条件がないかぎり，句読点の前，または後ろ，または前後の両方に空白が必要です。

空白をダブルクォーテーションマーク（"），またはシングルクォーテーションマーク（’）で囲んだ場合は，空白を文字として扱います。それ以外は，空白はほかの字句要素を区切る役割だけとして，無視されます。

(e) コメント

コメントは"/*"が始まりを，"*/" が終わりを示します。コメントはネストできません。

(3) 文法の形式の表記

このマニュアルでは，IDL ファイルを作成する文法説明の表記を，次のように使い分けています。

abc：この字体は，文法説明の表記どおりに記述することを示します。

abc：この字体は，特定の値を代入して記述することを示します。
　代入する文字列は，文法の説明を参照してください。

このマニュアルでは，IDL ファイルを作成する文法説明に，次に示す特殊記号を使っています。

[ ]：この字体の角括弧は，構文に必要な部分を示します。
　項目を記述する場合，[ ] も記述してください。

〔 〕：この字体のきっ甲は，項目を省略できることを示します。
　項目を記述する場合，〔 〕 は記述しないでください。

6.3.2 インタフェース定義言語の形式

IDL ファイルに記述する，インタフェース定義言語（IDL）の形式について説明します。インタフェース定義は，次のものから構成されます。

- **インタフェース定義ヘッダ**
  インタフェース全体にかかわる仕様を定義します。
- **インタフェース定義本体**

個々の型，オペレーションの仕様などを定義します。インタフェース定義本体は，次に示す四つの宣言があります。

インポート宣言

定数宣言

型宣言

オペレーション宣言

オペレーション宣言の中では，**パラメタ宣言**もあります。

インタフェース定義ヘッダとインタフェース定義本体で，指定に矛盾があった場合は，インタフェース定義本体の宣言が有効となります。

インタフェース定義言語の構成を次に示します。〔 〕で囲まれた部分は省略できます。

インタフェース定義ヘッダ

```
〔[interface_attribute,…]〕 interface  interface_name
```

インタフェース定義本体

```
{
 インポート宣言
 定数宣言
 型宣言
 オペレーション宣言
 パラメタ宣言
}
```

6.3.3　インタフェース定義言語の文法

インタフェース定義言語の文法について説明します。説明形式を次に示します。

形式

OpenTP1 の IDL-only TxRPC のインタフェース定義ヘッダ，およびインタフェース定義本体の各宣言の形式を示します。

意味

OpenTP1 の IDL-only TxRPC のインタフェース定義ヘッダ，およびインタフェース定義本体の各宣言の意味を示します。

指定する項目

形式に示す項目に指定する属性，データ型，および宣言子を示します。属性については「6.6　属性」を，データ型については「6.7　データ型」を，型宣言子については「6.8　型宣言子」を参照してください。

説明

宣言についての説明を示します。

OpenTP1 の IDL-only TxRPC の制限事項

OpenTP1 の IDL-only TxRPC と，X/Open が規定する IDL-only TxRPC との仕様の違いについて示します。

6.4　インタフェース定義ヘッダの文法

インタフェース定義ヘッダの定義形式について説明します。

インタフェース定義ヘッダ

形式

```
〔[interface_attribute,…]〕 interface　interface_name
```

意味

インタフェース名とその属性を定義します。

指定する項目

●interface_attribute

インタフェースの属性を定義します。指定できる属性の値を次に示します。

- version
  インタフェースのバージョンを指定します。
- pointer_default
  省略時仮定値のポインタセマンティクスを指定します。
- transaction_mandatory
  必ずトランザクションを拡張することを指定します。
- transaction_optional
  トランザクションの処理であればトランザクションを拡張することを指定します。

transaction_mandatory と transaction_optional は，同時に指定できません。どちらか片方だけを指定してください。

OpenTP1 の IDL-only TxRPC の制限事項

- 一つのサーバに対しては，一つのインタフェースしか定義できません。
- uuid 属性を指定する必要はありません。uuid 属性を指定してもエラーにはなりません。ただし，uuid 属性の形式に従っていない場合は，エラーになります。
- local 属性は使えません。local 属性を使うとエラーになります。
- endpoint 属性は使えません。endpoint 属性を使うとエラーになります。
- transaction_mandatory と transaction_optional は，通信するプロセスが両方とも ndce プロセスの場合だけ有効です。

6.5　インタフェース定義本体の文法

インタフェース定義本体には，次に示すうちの一つ，または複数の宣言をします。

- インポート宣言
- 定数宣言
- 型宣言
- オペレーション宣言（オペレーション宣言には，パラメタ宣言があります）

それぞれの宣言の終わりには，セミコロンを付けます。インタフェース定義本体は｛｝で囲みます。

インポート宣言は，ほかの宣言よりも前で宣言します。そのほかの宣言は型や定数が定義されていれば，順序に制限はありません。

インポート宣言

形式

```
import file,…;
```

意味

使う型と定数を宣言しているインタフェース定義ファイルをインポート（取り込み）します。

指定する項目

●file

ファイル名を指定します。インポートするIDLファイルのファイル名をダブルクォーテーションマーク（"）で囲んで指定します。

txidlコンパイラの-Iオプションで親ディレクトリを参照して，インポートするファイル名を決めることもできます。

説明

1. オペレーション宣言はインポートされません。
2. インタフェースは何回インポートしても結果は同じになります。
3. インポートするファイルは，あらかじめtxidlコマンドでコンパイルしておく必要があります（ヘッダファイルだけ生成しておくだけでかまいません）。

（例）

```
import  "garlic.idl", "oil.idl";
```

OpenTP1 の IDL－only TxRPC の制限事項

- インポートできるファイルは，100 個までです。

定数宣言

形式

```
const  integer_type_spec  identifier=integer|value;
const  boolean  identifier=TRUE|FALSE|value;
const  char  identifier=character|value;
const  char*  identifier=string|value;
const  void*  identifier=NULL|value;
```

意味

定数を宣言します。

指定する項目

次に示す整数定数のデータ型を宣言できます。

- integer_type_spec：整数定数（除く hyper）
- boolean：ブール定数
- char：文字定数
- char＊：文字型定数
- void＊：ヌル定数

●identifier

定数の名前を指定します。

●integer，character，string，value

定数に割り当てる値を指定します。value は事前に定義されている値であれば何でもかまいません。

説明

1. hyper は指定できません。
2. 定数宣言はスタブ内で#define で定義されているため，この定数を UAP で使うと展開されます。

（例）

```
const short TEN = 10;
const boolean FAUX = FALSE;
const char CHAR = 'A';
const char* DSCH = "abcde";
```

OpenTP1 の IDL－only TxRPC の制限事項

- 数式を整数定数として指定できません。
- オーバフローはチェックされません。適切でない大きさの値を定義した場合の動作は保証しません。

型宣言

形式

```
typedef 〔[type_attribute,…]〕 type_specifier type_declarator,…;
```

意味

インタフェースで使う型を定義します。

指定する項目

●type_attribute

宣言する型の属性を指定します。指定できる属性は，次のとおりです。

- string：文字列を示します。
- ptr：完全ポインタを示します。
- ref：参照ポインタを示します。

●type_specifier

データ型を指定します。基本型と構成型（構造体だけ），およびこれらを事前に定義した型を指定できます。

●type_declarator

定義する型の宣言子を指定します。指定できる内容を次に示します。

- 単純宣言子
- 固定長一次元配列
- ポインタ

説明

1. string 属性を指定できるのは，char と byte の配列だけです。
2. ptr 属性と ref 属性は，基本型と構造体型のポインタに対してだけ指定できます。

OpenTP1 の IDL－only TxRPC の制限事項

- 構成型のうち，union と enum は使えません。
- 宣言子として，関数のポインタ，配列のポインタは指定できません。
- 整合配列，可変長配列は使えません。
- 多次元配列は使えません。
- 次に示す型属性は使えません。
  transmit_as，handle，context_handle，vl_struct，vl_array，vl_string，vl_enum
- 指定できるポインタは，一つだけです。
- 構造体のメンバには，ポインタは指定できません。
- 構造体のメンバには，構造体は指定できません。
- string 属性を指定してもエラーにはなりませんが，無視されます。

オペレーション宣言

形式

```
〔[operation_attribute,…]〕 type_specifier
          operation_identifier(parameter_declaration,…);
〔[operation_attribute,…]〕 type_specifier
          operation_identifier( 〔 void 〕 );
```

意味

実際の処理をする関数を定義します。

指定する項目

●operation_attribute

オペレーション属性を指定します。指定できる属性を次に示します。

- transaction_mandatory
  必ずトランザクション拡張します。
- transaction_optional
  トランザクションの処理であれば，トランザクション拡張します。

●type_specifier

データ型を指定します。オペレーションが返すデータ型があれば，そのデータ型を指定します。この型はスカラ型か，前に定義されている型を指定します。結果を返さない場合は，void を必ず指定します。指定できる型は整数型です。

●operation_identifier

オペレーション名を指定します。文字の長さは最大 30 文字指定できます。

●parameter_declaration

パラメタ宣言を指定します。オペレーションのパラメタを宣言します。

説明

1. transaction_mandatory 属性と transaction_optional 属性を同時に指定できません。
2. オペレーションのリターン値をポインタにする場合は，必ず完全ポインタにしてください。

OpenTP1 の IDL－only TxRPC の制限事項

- context_handle 属性は使えません。
- ptr 属性は使えません。
- string 属性は使えません。
- transaction_mandatory 属性と transaction_optional 属性は，通信するプロセスが両方とも ndce プロセスの場合だけ有効です。
- 該当バージョンでは，type_specifier には error_status_t だけしか使えません。システムまたはスタブでエラーが起こった場合は，そのエラーコードが返されます。正常終了した場合にだけ，オペレーション関数のリターン値が返されます。また，error_status_t にポインタや配列は指定できません。

パラメタ宣言

形式

```
[parameter_attribute,…] type_specifier  parameter_declarator;
```

意味

オペレーション宣言の中で，オペレーションのパラメタを定義します。

指定する項目

●parameter_attribute

パラメタ属性を指定します。指定できる属性を次に示します。

- in：入力パラメタを指定します。
- out：出力パラメタを指定します。
- ptr：完全ポインタを指定します。
- ref：参照ポインタを指定します。
- string：文字列を指定します。

●type_specifier

パラメタのデータ型を指定します。指定できる型を次に示します。

- 基本型，構造体

●parameter_declarator

パラメタ宣言子を指定します。指定できる値を次に示します。

- 単純宣言子
- ポインタ
- 固定長一次元配列

説明

1. in または out のどちらかは，必ず指定してください。
2. out 属性のパラメタは，配列または明示的に宣言されたポインタでなければなりません。明示的に宣言されたポインタとは，"*" で宣言されたポインタのことです。

OpenTP1 の IDL－only TxRPC の制限事項

- 構成型のうち，union と enum は使えません。
- 宣言子として，関数のポインタ，配列のポインタは指定できません。
- 整合配列，可変長配列は使えません。
- 多次元配列は使えません。
- 次に示す型属性は使えません。
  配列属性，context_handle，vl_struct，vl_array，vl_string，vl_enum
- string 属性を指定してもエラーにはなりませんが，無視されます。

6.6 属性

IDL ファイルの宣言に使う，各種の属性について説明します。OpenTP1 の TxRPC で使える属性を次に示します。

- version 属性
- pointer_default 属性
- transaction_mandatory 属性
- transaction_optional 属性
- in 属性
- out 属性
- ポインタ属性

OpenTP1 の IDL-only TxRPC の制限事項

- 次に示す属性は，IDL-only TxRPC では無視されます。
  uuid
- 次に示す属性は，IDL-only TxRPC ではエラーとなります。
  endpoint，local，context_handle，transmit_as，vl_array，vl_enum，vl_string，vl_struct，配列属性
- 次に示す属性は，通信するプロセスが両方とも ndce プロセスの場合だけ有効です。
  transaction_mandatory，transaction_optional

説明形式を次に示します。

形式

属性の形式を示します。

属性の意味

属性の意味を示します。

指定する項目

属性として指定する項目を示します。

説明

属性の説明を示します。

指定例

属性の指定例を示します。

version 属性

形式

```
version(major 〔,minor〕)
```

属性の意味

リモートインタフェースの特定のバージョンを指定します。

指定する項目

●major

0 以上 65535 以下の整数で指定します。

●minor

0 以上 65535 以下の整数で指定します。

説明

- バージョン番号は，主バージョン番号と副バージョン番号を指定した一組の整数，または主バージョン番号だけを指定した整数一つのどちらかで指定します。主バージョン番号と副バージョン番号を指定する場合，二つの整数は空白を入れないでピリオドで区切ります。副バージョンを指定しない場合は，0 が仮定されます。
- version 属性の指定を省略すると，省略時仮定値として 0.0 が設定されます。
- 次の条件を満たした場合に，クライアントとサーバが通信できます。
  1. クライアントが呼び出すインタフェースと，サーバが広告するインタフェースが同じ主バージョンである。
  2. クライアントが呼び出すインタフェースの副バージョン番号が，サーバが広告するインタフェースの副バージョンの番号以下にする。

指定例

```
version(1.1)
version(3)
```

pointer_default 属性

形式

```
pointer_default(pointer_attribute)
```

属性の意味

IDL で使える二つのポインタセマンティクスのうち，どちらを省略時仮定値とするかを指定します。

指定する項目

●pointer_attribute

次に示すポインタ属性のどちらかを指定します。

ref：参照ポインタ

ptr：完全ポインタ

説明

- 省略時仮定値のポインタセマンティクスは，次に示す場合に使われます。
  1. 構造体のメンバの宣言に使うポインタ
  2. 複数のポインタ演算子が宣言するオペレーションパラメタのトップレベル以外に使うポインタ
- オペレーションのリターン値であるポインタは，常に完全ポインタです。そのため，pointer_default 属性は使われません。
- pointer_default 属性の指定よりも，ポインタ属性が優先されます。
- インタフェース定義で pointer_default 属性を指定しないで，省略時仮定値のポインタセマンティクスを必要とする宣言をすると，コンパイラでエラーになります。

transaction_mandatory 属性

形式

```
transaction_mandatory
```

属性の意味

グローバルトランザクションの一部で実行することを指定します。

説明

- この属性を指定したインタフェースまたはオペレーションは，必ずグローバルトランザクション内から呼び出してください。トランザクションの外から呼び出した場合，エラーとなってサービスは実行されません。
- transaction_optional 属性と同時に指定できません。

transaction_optional 属性

形式

```
transaction_optional
```

属性の意味

グローバルトランザクションの一部として実行されるかどうかが，呼び出した環境がトランザクション内かどうかで決めることを指定します。

説明

- この属性を設定したインタフェースまたはオペレーションは，グローバルトランザクション内から呼び出された場合は，そのトランザクションの一部として実行されます。トランザクションの処理の外から呼び出された場合は，トランザクションでない RPC として実行されます。
- transaction_mandatory 属性と同時に指定できません。

in 属性

形式

```
in
```

属性の意味

パラメタは入力であることを指定します。

説明

- パラメタには，in 属性，または out 属性のどちらかを必ず指定してください。

out 属性

形式

```
out
```

属性の意味

パラメタは出力であることを指定します。

説明

- パラメタには，in 属性，または out 属性のどちらかを必ず指定してください。

ポインタ属性

形式

```
ref
ptr
```

属性の意味

ポインタクラスを指定します。ポインタクラスには参照ポインタ（rtr）と完全ポインタ（ptr）があります。

説明

- ポインタ属性は，パラメタ，構造体メンバ，および型定義に使います。txidl コマンドがポインタの使用方法から適当なポインタクラスを判断する場合もありますが，ほとんどの場合，次に示す方式のうち一つを使ってポインタクラスを指定する必要があります。
  1. ポインタ宣言に ref 属性か ptr 属性を使う。
  2. IDL インタフェースのヘッダ部に pointer_default 属性を使う。省略時仮定値のポインタクラスは，pointer_default 属性で決定。
- ポインタ属性は，宣言中のトップレベルポインタにだけ有効です。複数のポインタが一つの宣言の中で宣言されているときは，確立された pointer_default がトップレベルポインタ以外のすべてのポインタに有効となります。
- ref 属性と ptr 属性は同時に指定できません。

参照ポインタの説明

参照ポインタは，単純な形式のポインタです。このクラスのポインタで一般的な使用方法は，参照によって整数を渡すなどの受け渡しです。

参照ポインタは，完全ポインタよりも高性能ですが，次に示す制限があります。

1. 参照ポインタには NULL 値がないため，リンケージを終了できません。
2. 参照ポインタを使ってリンケージ済みのリストは作成できません。

参照ポインタには，次に示す特性があります。

- 参照ポインタは，常に有効なストレージをポイントします。NULL 値は持てません。参照ポインタに NULL 値を入れた場合は，動作は保証しません。

- 参照ポインタの値は，関数の呼び出し中には変更されません。呼び出しから戻ったときも，常に呼び出しの開始時と同じ領域をポイントします。
- 別名化は使えません。同じオペレーションのパラメタが使う，ほかのポインタが指した領域はポイントできません。

完全ポインタの説明

完全ポインタは，複雑な形式のポインタです。完全ポインタは，ポインタに関連するすべての機能を使えます。例えば，完全ポインタを使うとリンケージ済みのリストやツリー，キュー，任意のグラフなどの複雑なデータ構造体を作れます。

完全ポインタには，次に示す特性があります。

- 完全ポインタの値は，関数の呼び出し中でも変更できます。
- IDL-only TxRPC では，別名化は使えません。
- 同じオペレーションのパラメタが使っているほかの完全ポインタを指すストレージ領域もポイントできます。ただし，この場合は，ポインタは構造体の開始をポイントする必要があります。例えば，インタフェース定義コードが次のコードを組み込んでいる場合など，基礎構造体や重複したストレージ領域へのポインタは使えません。

6.7　データ型

IDL ファイルの宣言に使う，データ型について説明します。OpenTP1 で使える TxRPC のデータ型を次に示します。

- 整数型（基本データ型）
- 浮動小数点型（基本データ型）
- 文字型（基本データ型）
- ブール型（基本データ型）
- バイト型（基本データ型）
- void 型（基本データ型）
- エラー状態型（基本データ型）
- 多国語に関する型（基本データ型）
- 構造体（構成データ型）

OpenTP1 の IDL-only TxRPC の制限事項

- string 属性を指定してもエラーにはなりませんが，無視されます。
- 構造体のメンバにポインタは指定できません。
- 構造体のメンバに構造体は指定できません。
- 可変長配列，整合配列は使えません。
- 多次元配列は使えません。
- union，enum は使えません。
- ハンドル型は使えません。

説明形式を次に示します。

形式

データ型の形式を示します。

データ型の説明

データ型について説明します。

整数型（基本データ型）

形式

```
int_size 〔 int 〕
unsigned int_size 〔 int 〕
int_size unsigned 〔 int 〕
```

データ型の説明

int_size は，次に示す値となります。

- hyper（64 ビット）
- long（32 ビット）
- short（16 ビット）
- small（8 ビット）

キーワード int は，オプションで意味を持ちません。キーワード unsigned は符号なし整数型を表しますが，サイズキーワードの前でもあとでもかまいません。

浮動小数点型（基本データ型）

形式

```
float
double
```

データ型の説明

2 とおりの長さの浮動小数点データ型を使えます。float は 32 ビット，double は 64 ビットです。

文字型（基本データ型）

形式

```
〔 unsigned 〕 char
```

データ型の説明

キーワード unsigned はオプションで，意味を持ちません。符号付き文字型は使えません。符号付き 8 ビット整数を表記するためには，small データ型を使ってください。

ブール型（基本データ型）

形式

```
boolean
```

データ型の説明

8 ビットで表します。ゼロは FALSE，ゼロ以外は TRUE とします。

バイト型（基本データ型）

形式

```
byte
```

データ型の説明

- 8 ビットで表します。byte データのデータ表記形式は，RPC を使ってデータを伝送しても変更されないように保証されています。
- 整数型，文字型，および浮動小数点型，またはこれらを組み合わせた構成型は，すべて異なるデータ表記形式を使うホスト間で伝送すると，形式が変換されることがあります。データの形式が変換されないようにしたいときは，型を byte 型の配列として伝送します。
- フォーマット変換処理をしないため，ほかのデータ型よりも性能は良くなります。

void 型（基本データ型）

形式

```
void
```

データ型の説明

void 型の使い方を次に示します。

- 値を返さないオペレーションの型を指定します。または，パラメタのないオペレーションのパラメタを示します。

エラー状態型（基本データ型）

形式

```
error_status_t
```

データ型の説明

RPC の通信状態の情報を保持するために，事前に定義されたデータ型です。

多国語に関する型（基本データ型）

形式

```
ISO_LATIN_1
ISO_MULTI_LINGUAL
ISO_UCS
```

データ型の説明

システムファイルに文字や文字列の表記に関して，現在の国際規格，および今後の新しい国際規格を使えるようにするために事前に定義してある型です。

- char 型データは，RPC メカニズムによって伝送すると，ASCII-EBCDIC 変換されることがあります。事前に定義された多国語に関する型は，基本型である byte から構成されているため，データ表記形式は変換されません。それぞれの型は，次に示すように事前に定義されています。

  ```
  typedef  byte  ISO_LATIN_1
  typedef  struct  {
    byte  row,column;
  }  ISO_MULTI_LINGUAL
  typedef  struct  {
    byte  group,plane,row,column;
  }  ISO_UCS
  ```

- IDL-only　TxRPC では，char 型データ型が ASCII-EBCDIC 変換されることはありません。そのため，この型の定義は特に意味を持ちません。

構造体（構成データ型）

形式（その 1）

```
struct[tag]
{
```

```
 〔[struct_member_attribute,…]〕  type_specifier declarator,…;
}
```

形式（その 2）

```
struct  tag
```

データ型の説明

（その 1）の形式で，指定子に tag を指定した場合は，tag 以降の一連のメンバ宣言の略式表記になります。この tag は，これ以降（その 2）の形式の指定子として使えます。

●struct_member_attribute

該当バージョンでは，指定できる属性はありません。

6.8　型宣言子

IDL ファイルの宣言に使う，型宣言子について説明します。OpenTP1 の TxRPC で使える型宣言子を次に示します。

- 配列
- 文字列
- ポインタ

OpenTP1 の IDL-only TxRPC の制限事項

ポインタで使えるアスタリスクは，一つまでです。

説明形式を次に示します。

形式

データ型の形式を示します。

型宣言子の説明

型宣言子について説明します。

配列

形式

IDL 配列は，次のような構文の array_declarator 構造体を通して宣言されます。

```
array_identifier array_bounds_declarator…
```

型宣言子の説明

次に示す配列型が使えます。

- 固定

  IDL で配列のサイズが定義されていて，配列内のすべてのデータが関数の呼び出し中に転送されます。

●array_bounds_declarator

配列の各次元について指定します。一次元配列に対する array_bounds_declarator は，次に示すうちのどれか一つの形式にします。

[lower..upper]：下方限界は lower に，上方限界は upper に指定します。

[size]：下方限界は 0 に，上方限界は size-1 に指定します。

IDL では，lower に対する正規の値は 0 だけです。

- 配列限界には，必ず整数を指定してください。配列属性は構造体メンバや整数項のパラメタだけを参照できます。

文字列

形式

```
char
byte
```

型宣言子の説明

IDL では，文字列を string 属性が割り当てられた一次元配列としています。配列の要素型は，必ず次に示す値にしてください。

- byte 型であるメンバ
- byte 型になるように事前に定義された型であるメンバをすべて持つ構造体
- char，byte のどちらか一つになる事前定義型

ポインタ

形式

IDL ポインタの宣言には次の構文を使います。

```
* 〔* ...〕 pointer_identifier
```

型宣言子の説明

アスタリスクはポインタ演算子で，アスタリスクが複数あると複数レベルの間接参照があることを示します。

6.9 属性定義言語

IDL-only TxRPC では，属性定義言語は使えません。

6.10　IDL コンパイラ（txidl コマンド）

IDL コンパイラ（txidl コマンド）の文法について説明します。説明形式を次に示します。

形式

IDL コンパイラの指定形式を示します。

機能

IDL コンパイラの機能を示します。

argument に指定する引数

形式に示す引数 dargument に指定する引数を示します。

説明

IDL コンパイラの説明を示します。

メッセージ

IDL コンパイラが出力するメッセージを示します。

関連するファイル

IDL コンパイラに関連するファイルを示します。

注意事項

IDL コンパイラの注意事項を示します。

txidl（IDL コンパイラ）

形式

```
txidl　filename　〔argument〕…
```

機能

TxRPC 用インタフェース定義言語コンパイラを起動します。

argument に指定する引数

●-cptype process_type

クライアントのプロセスタイプを指定します。process_type には，次に示すどれかを指定します。

- ndce
  このプロセスは TP1/Server Base のライブラリを使います。
- nbet
  このプロセスは DCE のライブラリだけを使います。

何も指定しなかった場合は，ndce として扱います。間違ったプロセスタイプを指定してコンパイルしたプログラムは動作しません（例えば，nbet を指定してコンパイルしたスタブに TP1/Server Base ライブラリを組み込んでも動作しません）。

●-sptype process_type

サーバのプロセスタイプを指定します。process_type は，-cptype と同じです。

何も指定しなかった場合は，ndce として扱います。間違ったプロセスタイプを指定してコンパイルしたプログラムは動作しません（例えば，nbet を指定してコンパイルしたスタブに TP1/Server Base ライブラリを組み込んでも動作しません）。

●-client file_type

どのクライアントファイルを生成するかを決めます。この引数を指定しない場合，または file_type を指定しない場合は，コンパイラはすべてのクライアントファイルを生成します。file_type には，次に示す値を指定します。

- none
  ファイルを生成しません。
- stub
  スタブファイルだけを生成します。
- all
  スタブおよびクライアント生成ファイルを生成します。

●-server file_type

どのサーバファイルを生成するかを決定します。この引数を指定しない場合，または file_type を指定しない場合は，コンパイラはすべてのサーバファイルを生成します。file_type は，-client と同じです。

●-cstub filename

クライアントスタブのパス名を指定します。

ファイル名を指定するときは，ファイル拡張子は指定しないでください。C 言語のソースファイルには".c" を txidl コンパイラが付けます。-cstub オプションを使わない場合は，_cstub.c を txidl コンパイラが付けます。

クライアントのプロセスタイプが gateway，サーバのプロセスタイプが dce の場合，2 種類のスタブファイルが生成されます。この場合，OpenTP1 のスタブファイル名は，filename の先頭に大文字の"B"を付けた名称になります。

●-sstub filename

サーバスタブのパス名を指定します。ファイル名を指定するときは，ファイル拡張子は指定できません。C 言語のソースファイルには".c"を txidl コンパイラが付けます。-sstub オプションを使わないと，_sstub.c を txidl コンパイラが付けます。

●-header header_file

生成されるヘッダファイルのパス名を指定します。

ファイル名を指定するときは，ファイル拡張子は指定しないでください。省略時仮定値として，IDL ファイルのベース名に".h"を txidl コンパイラが付けます。

●-cconf conffile

クライアントプログラムのユーザサービス定義ファイル，または環境設定ファイルのパス名を指定します。-cconf オプションを使わないと，IDL ファイルのベース名の先頭に"C"を付けた名称のファイルが生成されます。このオプションは，プロセスタイプの組み合わせが IDL-only TxRPC のときだけ有効です。それ以外で指定した場合は，エラーにはならないで，無視されます。

●-sconf conffile

サーバプログラムのユーザサービス定義ファイルのパス名を指定します。-sconf オプションを使わないと，IDL ファイルのベース名の先頭に"S"を付けた名称のファイルが生成されます。このオプションは，プロセスタイプの組み合わせが IDL-only TxRPC のときだけ有効です。それ以外で指定した場合は，エラーにはならないで，無視されます。

●-out directory

指定したディレクトリに出力ファイルを生成します。省略時仮定値では，コンパイラはカレントディレクトリに出力ファイルを生成します。

ほかのオプションでパス名が指定されている場合は，指定した順番に関係なく，そちらが優先されます。

●-Idirectory

インポートするインタフェース定義ファイルを含むディレクトリ名を指定します。コマンド行に追加の-Idirectory 引数を指定して，複数のディレクトリを指定できます。コンパイラは，この引数に指定した順番にディレクトリを検索します。

一つのファイルが複数のディレクトリ内にある場合，コンパイラは最初に現れるファイルをインポートします。

この引数を省略した場合，次に示す順番でディレクトリを検索します。

1. カレントディレクトリを検索
2. 指定されたすべてのディレクトリを検索
3. システム IDL ディレクトリ（$DCDIR/include）を検索

●-no_def_idir

コンパイラがカレントディレクトリだけでインポートファイルを検索します。

-Idirectory と一緒にこのオプションを指定すると，コンパイラはユーザが指定したディレクトリだけを検索して，カレントディレクトリやシステムディレクトリを検索しません。

●-noconf

OpenTP1 のユーザサービス定義や環境設定ファイルのテンプレートを生成しません。プロセスタイプの組み合わせが IDL-only TxRPC のときだけ有効です。

●-noserver

サーバプログラムのテンプレートを生成しません。プロセスタイプの組み合わせが IDL-only TxRPC のときだけ有効です。

●-syntax_only

IDL ファイルの構文だけをチェックして，ファイルは出力しないことを指定します。

説明

- txidl コマンドは，IDL で書かれたインタフェース定義を解析して，ヘッダファイル，サーバスタブファイル，クライアントスタブファイル，補助ファイル，OpenTP1 用定義ファイルのテンプレートなどを生成します。
- IDL コンパイラは，関連する ACF を各ディレクトリで検索します。例えば，source.idl という名称のファイルをコンパイルすると，コンパイラは自動的に source.acf という名称のファイルを検索します。また，インポートされた IDL ファイル（および関連する ACF）も検索します。コンパイラは，次の順番でこれらのファイルを検索します。

  1. カレントディレクトリ

     -no_def_idir および-Idirectory 引数を一緒に指定しないかぎり，コンパイラは常にこのディレクトリを検索します。

  2. インポートされたディレクトリ

     コンパイラは-Idirectory 引数に指定する各ディレクトリを検索します。

  3. システム IDL ディレクトリ

     コンパイラは，システム IDL ディレクトリ内にある dctrpb.idl を自動的にインポートします。-no_def_idir 引数を指定しないかぎり，コンパイラは常にこのディレクトリを検索します。

  4. ソースファイル名に指定されたディレクトリ

     ソース IDL パス名に明示的にディレクトリを指定すると，対応する ACF がそのディレクトリで検索されます。

- IDL-only TxRPC では，自動的に txidl コマンドが OpenTP1 用定義ファイルを生成します。txidl コマンドに指定したオプションによって，生成しないようにもできます。
- オペレーション名を変更した場合は，OpenTP1 定義ファイルも生成し直す必要があります。

メッセージ

txidl コンパイラが出力するメッセージには，次の 3 種類があります。次に示す該当するマニュアルを参照してください。

1. txidl コンパイラ自身が出力するメッセージ

マニュアル「OpenTP1 メッセージ」を参照してください。

2. txidl コンパイラが起動する DCE idl が出力するメッセージ
   DCE の該当するマニュアルを参照してください。
3. DCE idl が起動する cpp または cc が出力するメッセージ
   各コマンドに該当するマニュアルを参照してください。

関連するファイル

IDL-only TxRPC に関連するファイルを次に示します。

$DCDIR/bin/txidl：IDL コンパイラ

$DCDIR/include/dctrpb.idl：システム用 IDL ファイル

$DCDIR/include/dctrp.h：ヘッダファイル

注意事項

- IDL コンパイラは，ANS　C コードを生成します。C コンパイラによるスタブのコンパイル中に警告メッセージは返りませんが，完全に ANSI C 仕様でない C コンパイラは，次のメッセージを通知する場合があります。

  warning：& before array or function： ignored

  warning：enumeration type clash，operator ＝
- オプションとパラメタの間は空白を入れてください。

  （例）-out ×××（-out×××とは指定できません）
- 次に示すファイル名は，IDL コンパイラで予約しています。この中のどれかを使って IDL ファイルに名前を付けた場合は，動作は保証しません。

  iovector.idl，lbase.idl，nbase.idl，ncastat.idl，rpc.idl，rpcbase.idl，rpcpvt.idl，rpcsts.idl，rpctypes.idl，twr.idl，uuid.idl，dctrpb.idl
- このバージョンでは RPC TxRPC をサポートしていません。したがって，txidl コマンドの-cptype オプション，および-sptype オプションのプロセスタイプに nbet を指定しても，生成されたスタブファイルは使用できません。

6.11　TxRPC のエラーコード

OpenTP1 の TxRPC 用のシステムサービスから返されるエラーについて説明します。

TxRPC のエラーコード一覧を次の表に示します。IDL-only TxRPC の場合は，次の表に示す dc_rpc_call 関数の同等のリターン値を参考にして，エラー時の処理を作成してください。

表 6-1　TxRPC のエラーコード一覧

エラーコード	意味
txrpc_s_not_in_transaction	transaction_mandatory を指定したオペレーションが，グローバルトランザクションの外から呼び出されました。
txrpc_s_no_tx_open_done	OpenTP1 の TxRPC 用システムサービスでマネジャを呼び出したときに，tx_open 関数を呼び出していない状態でオペレーションが実行されました。
DCTRPER_PROTO	プロトコルエラーが起こりました。
rpc_s_comm_failure	通信関連のエラーが起こりました。次に示す dc_rpc_call 関数のリターン値に相当します。 • DCRPCER_SYSERR • DCRPCER_SYSERR_RB • DCRPCER_SYSERR_AT_SERVER • DCRPCER_SYSERR_AT_SERVER_RB • DCRPCER_SERVICE_TERMINATING • DCRPCER_SERVICE_NOT_UP • DCRPCER_SERVICE_CLOSED • DCRPCER_OLTF_NOT_UP • DCRPCER_OLTF_INITIALIZING
rpc_s_no_memory	メモリが不足しました。次に示す dc_rpc_call 関数のリターン値に相当します。 • DCRPCER_NO_BUFS
rpc_s_fault_remote_no_memory	サーバ側でメモリが不足しました。次に示す dc_rpc_call 関数のリターン値に相当します。 • DCRPCER_NO_BUFS_RB • DCRPCER_NO_BUFS_AT_SERVER
rpc_s_call_timeout	タイムアウトが起こりました。次に示す dc_rpc_call 関数のリターン値に相当します。 • DCRPCER_TIMED_OUT
rpc_s_in_args_too_big	引数に指定した値が大き過ぎます。次に示す dc_rpc_call 関数のリターン値に相当します。 • DCRPCER_MESSAGE_TOO_BIG
rpc_s_entry_not_found	サービスエントリが見つかりません。次に示す dc_rpc_call 関数のリターン値に相当します。 • DCRPCER_NO_SUCH_SERVICE_GROUP • DCRPCER_NO_SUCH_SERVICE

エラーコード	意味
rpc_s_mgmt_op_disallowed	サーバがソケット受信型サーバで，サービス要求を受信できません。または，サーバ側が OpenTP1 のセキュリティ機能で保護されていて，クライアントにアクセス権がありません。次に示す dc_rpc_call 関数のリターン値に相当します。 • DCRPCER_SERVER_BUSY
rpc_s_binding_has_no_auth	サーバで OpenTP1 のセキュリティ機能を使っています。セキュリティ機能でアクセスエラーが起こりました。次に示す dc_rpc_call 関数のリターン値に相当します。 • DCRPCER_SECCHK
rpc_s_fault_unspec	OpenTP1 のシステムで，次に示す dc_rpc_call 関数のリターン値に相当するエラーが起こりました。 • DCRPCER_TESTMODE • DCRPCER_INVALID_REPLY • DCRPCER_REPLY_TOO_BIG • DCRPCER_REPLY_TOO_BIG_RB または，マーシャリング/アンマーシャリングに失敗しました。または，通信データが破壊されました。
rpc_s_unknown_stub_rtl_if_vers	OpenTP1 のライブラリでバージョンが一致していません。
rpc_s_unknown_if	インタフェース定義でバージョンが一致していません。

7 コーディング例

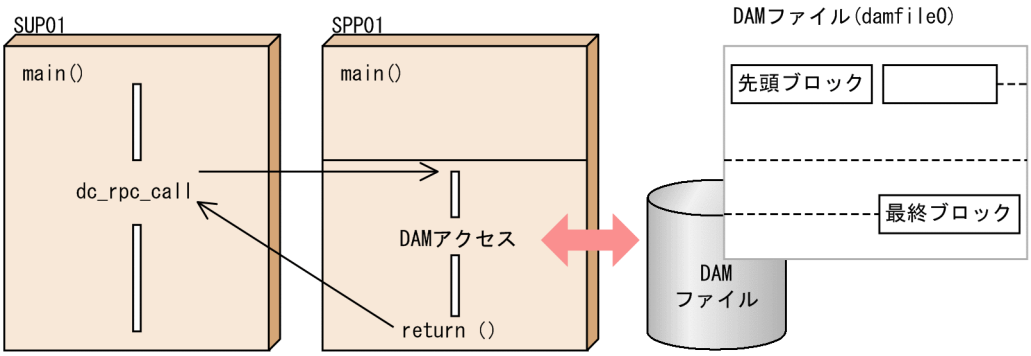
この章では，アプリケーションプログラム（UAP）のコーディング例について説明します。

この章で示すアプリケーションプログラムのコーディング例（7.1～7.4）は，K&R 版の C 言語の形式で示します。

7.1 クライアント/サーバ形態の UAP のコーディング例（SUP，SPP　DAM アクセス）

クライアント/サーバ形態の UAP 構成例を次の図に示します。

図 7-1　クライアント/サーバ形態の UAP 構成例（DAM アクセス）



（説明）

DAM ファイル damfile0 には，先頭ブロックに管理部があり，2 番目以降のブロックをデータレコードとしています。サービスの処理は，先頭ブロックを入力して更新したあと（dc_dam_read 関数，dc_dam_rewrite 関数），2 番目以降のブロックは dc_dam_write 関数で，直接，更新します。

ここでは，図に示した構成例のコーディング例を示します。

7.1.1 SUP の例

SUP のコーディング例を次に示します。

```
 10  /*
 20   * SUP01
 30   */
 40  #include <stdio.h>
 50  #include <string.h>
 60  #include <dcrpc.h>
 70  #include <dctrn.h>
 80
 90  main()
100  {
110  /*
120   *変数の定義
130   */
140    static char   in_buf[1024];
150    static DCLONG in_buf_len;
160    static char   out_buf[1024];
170    static DCLONG out_buf_len;
180    int rc;
190  /*
200   *RPC-OPEN(UAPの開始)
```

```
210   */
220    rc = dc_rpc_open(DCNOFLAGS);
230    /*OpenTP1の各種関数を使う準備をする(各機能の初期化)*/
240    if(rc != DC_OK) {
250      printf("SUP01:dc_rpc_openに失敗しました。CODE = %d ¥n",rc);
260      goto PROG_END;
270    }
280  /*
290   *ADM-COMPLETE(ユーザサーバの開始処理完了の報告)
300   */
310    rc = dc_adm_complete(DCNOFLAGS);
320    if(rc != DC_OK){
330      printf("SUP01:dc_adm_completeに失敗しました。CODE = %d ¥n",rc);
340      goto PROG_END;
350    }
360  /*
370   *TRN_BEGIN(トランザクションの開始)
380   */
390    rc = dc_trn_begin();
400    if(rc != DC_OK) {
410      printf("SUP01:dc_trn_beginに失敗しました。CODE = %d ¥n",rc);
420      goto TRAN_END;
430    }
440    /*
450     *RPC-CALL(遠隔サービスの要求)
460     */
470    strcpy(in_buf,"SUP01:DATA OpenTP1!!");
480    in_buf_len = strlen(in_buf) + 1;
490    out_buf_len = 1024;
500    rc = dc_rpc_call("spp01grp","svr01",in_buf,&in_buf_len,
510    out_buf,&out_buf_len,DCNOFLAGS);
520    if(rc != DC_OK) {
530      printf("SUP01:サービス要求に失敗しました。CODE = %d ¥n",rc);
540      goto TRAN_END;
550    }
560    printf("SUP01:SERVICE FUNCTION RETURN = %s¥n",out_buf);
570  /*
580   *TRN-UNCHAINED-COMMIT(非連鎖モードのコミット)
590   */
600    TRAN_END:
610    rc = dc_trn_unchained_commit();
620    if(rc != DC_OK) {
630      printf("SUP01:dc_trn_unchained_commitに失敗しました。CODE = %d ¥n",rc);
640    }
650  /*
660   *RPC-CLOSE(UAPの終了)
670   */
680    PROG_END:
690    dc_rpc_close(DCNOFLAGS);
700    printf("SUP01:処理を終了しました。¥n");
710    exit(0);
720  }
```

7.1.2 SPP の例（メイン関数）

SPP のメイン関数のコーディング例を次に示します。

```
 10  /*
 20   *SPP01 メイン関数
 30   */
 40  #include <stdio.h>
 50  #include <dcrpc.h>
 60  #include <dcdam.h>
 70  #define DAMFILE "damfile0"
 80
 90  int damfd;   /* damfile file-id */
100
110  main()
120  {
130  /*
140   *リターン値を格納する領域の定義
150   */
160    int rc;
170  /*
180   *RPC-OPEN(UAPの開始)
190   */
200    rc = dc_rpc_open(DCNOFLAGS);
210    if(rc != DC_OK) {
220      printf("SPP01:dc_rpc_openに失敗しました。CODE = %d ¥n",rc);
230      goto PROG_END;
240    }
250  /*
260   *DAM-OPEN(論理ファイルのオープン)
270   */
280    rc = dc_dam_open(DAMFILE,DCDAM_BLOCK_EXCLUSIVE);
290    if(rc < DC_OK) {
300      printf("SVR01:dc_dam_openに失敗しました。CODE = %d ¥n",rc);
310      goto DAM_END;
320    }
330    damfd = rc;
340  /*
350   *RPC-MAINLOOP(SPPのサービス開始)
360   */
370    printf("SPP01:mainloopに入ります。¥n");
380    rc = dc_rpc_mainloop(DCNOFLAGS);
390    if(rc != DC_OK) {
400      printf("SPP01:dc_rpc_mainloop ¥
410          に失敗しました。CODE = %d ¥n",rc);
420    }
430  /*
440   *DAM-CLOSE(論理ファイルのクローズ)
450   */
460    DAM_END:
470    rc = dc_dam_close(damfd,DCNOFLAGS);
480    if(rc != DC_OK) {
490      printf("SVR01:dc_dam_closeに失敗しました。CODE = %d¥n",rc);
500    }
510  /*
520   *RPC-CLOSE(UAPの終了)
530   */
```

```
540    PROG_END:
550    dc_rpc_close(DCNOFLAGS);
560    printf("SPP01:SPPのサービス処理を終了します。¥n");
570    exit(0);
580  }
```

7.1.3　SPP の例（サービス関数）

SPP のサービス関数のコーディング例を次に示します。

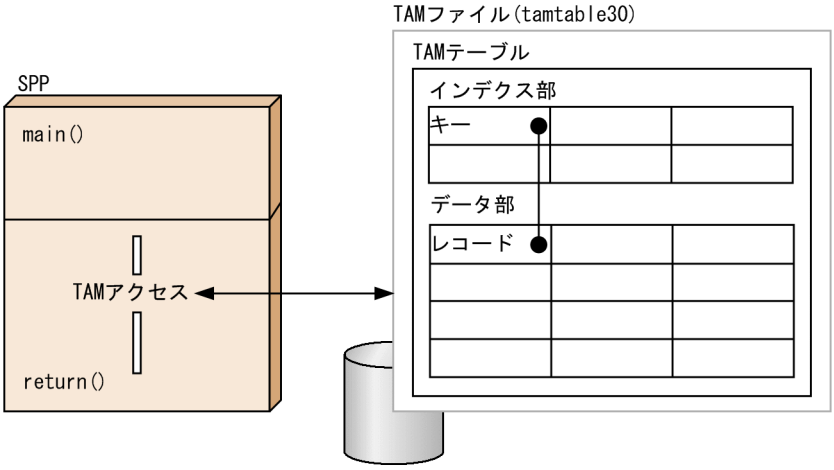
```
 10  /*
 20   *SVR01 サービス関数
 30   */
 40  #include <stdio.h>
 50  #include <string.h>
 60  #include <dcrpc.h>
 70  #include <dcdam.h>
 80  #define DAMFILE "damfile0"
 90  #define DAM_BLK_SIZE 504
100  #define REWRITE_LEN 19
110  extern int damfd;
120
130  void svr01(in_data,in_leng,out_data,out_leng)
140    char   *in_data;
150    DCLONG *in_leng;
160    char   *out_data;
170    DCLONG *out_leng;
180  {
190  /*
200   *変数の定義
210   */
220    static struct DC_DAMKEY keyptr;
230    static char *damc_buf;
240    static char dam_cntl_buf[DAM_BLK_SIZE];
250    static char write_buf[DAM_BLK_SIZE];
260    struct dam_cntl_p {
270      int w_point;
280      char rewrite_data[REWRITE_LEN];
290    } *dam_cntl_p;
300    int rc;
310    int write_size;
320    int rewrite_size;
330    int damc_buf_size;
340
350    keyptr.fstblkno = 0;
360    keyptr.endblkno = 0;
370    damc_buf_size = DAM_BLK_SIZE;
380    printf("SVR01:処理の開始 ¥n");
390  /*
400   *DAM_READ(論理ファイルからブロックの入力)
410   */
420    rc = dc_dam_read(damfd,&keyptr,1,dam_cntl_buf,
430    damc_buf_size,DCDAM_MODIFY);
440    if(rc != DC_OK) {
```

```
450       printf("SVR01:dc_dam_readに失敗しました。CODE = %d ¥n",rc);
460       strcpy(out_data,"SVR01:DAM READ FAILED");
470       *out_leng = strlen(out_data);
480       goto PROG_END;
490     }
500   /*
510    *DAM_WRITE(論理ファイルへブロックの出力)
520    *DAM_REWRITE(論理ファイルのブロックの更新)
530    */
540     DAM_WRITE:
550     dam_cntl_p = (struct dam_cntl_p *)dam_cntl_buf;
560     write_size = DAM_BLK_SIZE;
570     memcpy(write_buf,in_data,*in_leng);
580     dam_cntl_p->w_point = dam_cntl_p->w_point + 1;
590     keyptr.fstblkno = dam_cntl_p->w_point;
600     keyptr.endblkno = 0;
610     rc = dc_dam_write(damfd,&keyptr,1,write_buf,
620     write_size,DCNOFLAGS);
630     if(rc != DC_OK) {
640       if(rc == DCDAMER_BNOER) {
650         dam_cntl_p->w_point = 0;
660         goto DAM_WRITE;
670       }
680       printf("SVR01:dc_dam_writeに失敗しました。CODE = %d ¥n",rc);
690       strcpy(out_data,"SVR01;DAM WRITE FAILED");
700       *out_leng = strlen(out_data);
710       goto PROG_END;
720     }
730     keyptr.fstblkno = 0;
740     keyptr.endblkno = 0;
750     damc_buf_size = DAM_BLK_SIZE;
760     sprintf(dam_cntl_p->rewrite_data,"REWRITE COMPLETE¥n");
770     rc = dc_dam_rewrite(damfd,&keyptr,1,dam_cntl_buf,
780     damc_buf_size,DCDAM_UPDATE);
790     if(rc != DC_OK) {
800       printf("SVR01:dc_dam_rewriteに失敗しました。CODE = %d¥n",rc);
810       strcpy(out_data,"SVR01:DAM REWRITE FAILED");
820       *out_leng = strlen(out_data);
830     }
840     strcpy(out_data,"SVR01:PROCESS COMPLETE");
850     *out_leng = strlen(out_data);
860     PROG_END:
870     printf("SVR01:処理を終了します。¥n");
880     return;
890   }
```

7.2　クライアント/サーバ形態の UAP のコーディング例（SPP　TAM アクセス）

クライアント/サーバ形態の UAP 構成例を次の図に示します。ここでは，SPP のコーディング例だけを掲載します。この SPP にサービスを要求する SUP は，「7.1　クライアント/サーバ形態の UAP のコーディング例（SUP，SPP　DAM アクセス）」と同じ SUP からサービス要求されるものとします。

図 7-2　クライアント/サーバ形態の UAP 構成例（TAM アクセス）



ここでは，図に示した構成例のコーディング例を示します。

7.2.1　SPP の例（メイン関数）

SPP のメイン関数のコーディング例を次に示します。

```
 10  /*
 20   * spp01 メイン関数
 30   */
 40  #include <stdio.h>
 50  #include <dcrpc.h>
 60  #include <dctam.h>
 70  #define  TAMTABLE    "tamtable30"
 80
 90  long  tamfd ;      /* tamfile file-id  */
100
110  main()
120  {
130
140  /*
150   * リターンコード格納変数の定義
160   */
170    int    rcd ;
180  /*
190   * RPC-OPEN(UAPの開始)
200   */
210    rcd = dc_rpc_open(DCNOFLAGS) ;
```

```
220    if(rcd != DC_OK) {
230      printf("SPP01:dc_rpc_openに失敗しました。code = %d ¥n", rcd) ;
240      goto PROG_END ;
250    }
260  /*
270   * TAM-OPEN(TAMテーブルのオープン)
280   */
290    rcd = dc_tam_open(TAMTABLE, DCTAM_REC_EXCLUSIVE) ;
300    if(rcd <= 0) {
310      printf("SVR01:dc_tam_openに失敗しました。code = %d ¥n", rcd) ;
320      goto TAM_END ;
330    }
340    tamfd = (long)rcd ;
350  /*
360   * RPC-MAINLOOP(SPPのサービス開始)
370   */
380    rcd = dc_rpc_mainloop(DCNOFLAGS) ;
390    if(rcd != DC_OK) {
400      printf("SPP01:dc_rpc_mainloopに失敗しました。code = %d ¥n", rcd) ;
410    }
420  /*
430   * TAM-CLOSE(TAMテーブルのクローズ)
440   */
450    rcd = dc_tam_close(tamfd, DCNOFLAGS) ;
460    if(rcd != DC_OK) {
470      printf("SVR01:dc_tam_closeに失敗しました。code = %d ¥n", rcd) ;
480    }
490  TAM_END :
500  /*
510   * RPC-CLOSE(UAPの終了)
520   */
530    dc_rpc_close(DCNOFLAGS) ;
540  PROG_END :
550    printf("SPP01:SPPのサービス処理を終了します。¥n") ;
560    exit(0) ;
570  }
```

7.2.2 SPP の例（サービス関数）

SPP のサービス関数のコーディング例を次に示します。

```
 10  /*
 20   * srv01 サービス関数
 30   */
 40  #include <stdio.h>
 50  #include <string.h>
 60  #include <dctam.h>
 70  #define  TAM_REC_SIZE  128
 80
 90  extern  long tamfd ;   /* tamfile file-id */
100
110  void svr01(in_data, in_leng, out_data, out_leng)
120    char *in_data ;
130    long *in_leng ;
```

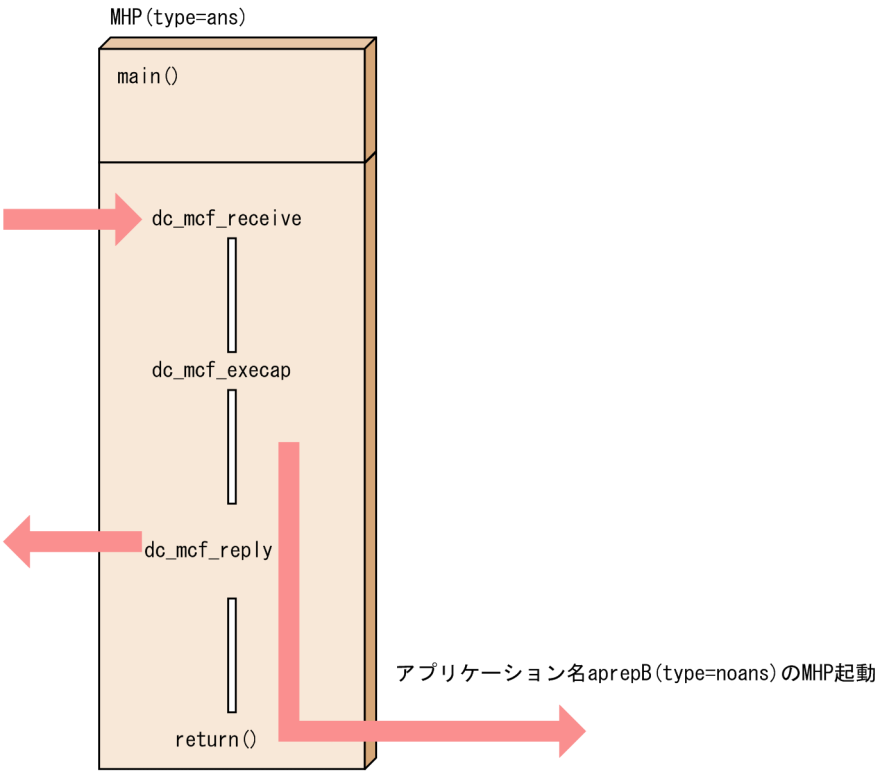
```
140    char *out_data ;
150    long *out_leng ;
160  {
170
180  /*
190   * 変数の定義
200   */
210    static struct DC_TAMKEY keyptr ;
220    static char   *tamc_buf ;
230    static char   tam_cntl_buf[TAM_REC_SIZE] ;
240    static char   write_buf[TAM_REC_SIZE] ;
250    struct tam_cntl_p {
260                        char keyname[10] ;
270                        char filler[118] ;
280                       } *tam_cntl_p ;
290    int    rcd ;
300    int    write_size ;
310    int    tamc_buf_size ;
320    static char keypar[4][10] = {
330                                  { 0x00, 0x00, 0x00, 0x00, 0x00,
340                                    0x00, 0x00, 0x00, 0x00, 0x01} ,
350                                  { 0x00, 0x00, 0x00, 0x00, 0x00,
360                                    0x00, 0x00, 0x00, 0x00, 0x02} ,
370                                  { 0x00, 0x00, 0x00, 0x00, 0x00,
380                                    0x00, 0x00, 0x00, 0x00, 0x03} ,
390                                  { 0x00, 0x00, 0x00, 0x00, 0x00,
400                                    0x00, 0x00, 0x00, 0x00, 0x04} ,
410                                } ;
420    printf("SVR01:処理の開始 ¥n") ;
430  /*
440   * TAM_READ(TAMテーブルから第1レコードを入力)
450   */
460    keyptr.keyname = keypar[0] ;
470    tamc_buf_size = TAM_REC_SIZE ;
480    rcd = dc_tam_read(tamfd, &keyptr, 1, tam_cntl_buf,
490                      tamc_buf_size, DCTAM_EQLSRC | DCTAM_MODIFY) ;
500    if(rcd != DC_OK) {
510      printf("SVR01:dc_tam_readに失敗しました。code = %d ¥n", rcd) ;
520      strcpy(out_data, "SVR01:TAM READ FAILED") ;
530      *out_leng = strlen(out_data) ;
540      goto PROG_END ;
550    }
560  /*
570   * TAM_REWRITE(TAMテーブルの第1レコードを検索前提の更新)
580   */
590    tam_cntl_p = (struct tam_cntl_p *)tam_cntl_buf ;
600    memcpy(tam_cntl_p->filler, in_data, *in_leng) ;
610    rcd = dc_tam_rewrite(tamfd, &keyptr, 1, tam_cntl_buf,
620                                tamc_buf_size, DCNOFLAGS) ;
630    if(rcd != DC_OK) {
640      printf("SVR01:dc_tam_rewriteに失敗しました。code = %d ¥n", rcd) ;
650      strcpy(out_data, "SVR01:TAM REWRITE FAILED") ;
660      *out_leng = strlen(out_data) ;
670      goto PROG_END ;
680    }
690  /*
700   * TAM_WRITE(TAMテーブルの第2レコードを更新)
710   */
```

```
 720    keyptr.keyname = keypar[1] ;
 730    tam_cntl_p = (struct tam_cntl_p *)write_buf ;
 740    memcpy(tam_cntl_p->keyname, keypar[1], 10) ;
 750    memcpy(tam_cntl_p->filler, in_data, *in_leng) ;
 760    write_size = TAM_REC_SIZE ;
 770    rcd = dc_tam_write(tamfd, &keyptr, 1, tam_cntl_p,
 780                              write_size, DCTAM_WRITE) ;
 790    if(rcd != DC_OK) {
 800      printf("SVR01:dc_tam_writeに失敗しました。code = %d ¥n", rcd) ;
 810      strcpy(out_data, "SVR01:TAM WRITE FAILED") ;
 820      *out_leng = strlen(out_data) ;
 830      goto PROG_END ;
 840    }
 850  /*
 860   * TAM_READ(TAMテーブルから第3レコードを入力)
 870   */
 880    keyptr.keyname = keypar[2] ;
 890    tamc_buf_size = TAM_REC_SIZE ;
 900    rcd = dc_tam_read(tamfd, &keyptr, 1, tam_cntl_buf,
 910                      tamc_buf_size, DCTAM_EQLSRC | DCTAM_MODIFY) ;
 920    if(rcd != DC_OK) {
 930      printf("SVR01:dc_tam_readに失敗しました。code = %d ¥n", rcd) ;
 940      strcpy(out_data, "SVR01:TAM READ FAILED") ;
 950      *out_leng = strlen(out_data) ;
 960      goto PROG_END ;
 970    }
 980  /*
 990   * TAM_READ_CANCEL(TAMテーブルの第3レコードの検索を取り消す)
1000   */
1010    rcd = dc_tam_read_cancel(tamfd, &keyptr, 1, DCNOFLAGS) ;
1020    if(rcd != DC_OK) {
1030      printf("SVR01:dc_tam_read_cancelに失敗しました。code = %d ¥n",
1040                                                          rcd) ;
1050      strcpy(out_data, "SVR01:TAM READ CANCEL FAILED") ;
1060      *out_leng = strlen(out_data) ;
1070      goto PROG_END ;
1080    }
1090  /*
1100   * TAM_delete(TAMテーブルの第4レコードを削除)
1110   */
1120    keyptr.keyname = keypar[3] ;
1130    rcd = dc_tam_delete(tamfd, &keyptr, 1,
1140                               NULL, 0, DCTAM_NOOUTREC) ;
1150    if(rcd != DC_OK) {
1160      printf("SVR01:dc_tam_deleteに失敗しました。code = %d ¥n", rcd) ;
1170      strcpy(out_data, "SVR01:TAM DELETE FAILED") ;
1180      *out_leng = strlen(out_data) ;
1190      goto PROG_END ;
1200    }
1210    strcpy(out_data, "SVR01:PROCESS COMPLETE") ;
1220    *out_leng = strlen(out_data) ;
1230  PROG_END :
1240    printf("SVR01:処理を終了します。¥n") ;
1250    return ;
1260  }
```

7.3　メッセージ送受信形態の UAP のコーディング例（MHP）

メッセージ送受信形態の UAP の例を次の図に示します。

図 7-3　メッセージ送受信形態の UAP の例（MHP）



ここでは，図に示した構成例のコーディング例を示します。

7.3.1　MHP の例（メイン関数）

MHP のメイン関数のコーディング例を次に示します。

```
 10  /*
 20   * MHP メイン関数
 30   */
 40  #include <stdio.h>
 45  #include <stdlib.h>
 50  #include <dcrpc.h>
 60  #include <dcmcf.h>
 70
 80  main()
 90  {
100     int rtn_cod ;
110
120     printf("******  RPC OPEN   *****¥n") ;
130  /*
140   * RPC-OPEN (UAPの開始)
150   */
```

```
160     rtn_cod = dc_rpc_open(DCNOFLAGS) ;
170     if(rtn_cod != DC_OK) {
180       printf("dc_rpc_open 失敗 !! CODE = %d ¥n", rtn_cod) ;
190       goto PROG_END ;
200     }
210
220     printf("******  MCF OPEN   *****¥n") ;
230   /*
240    * MCF-OPEN (MCF環境のオープン)
250    */
260     rtn_cod = dc_mcf_open(DCNOFLAGS) ;
270     if(rtn_cod != DC_OK) {
280       printf("dc_mcf_open 失敗 !! CODE = %d ¥n", rtn_cod) ;
290       goto PROG_END ;
300     }
310
320     printf("******  MCF MAINLOOP   *****¥n") ;
330   /*
340    * MCF-MAINLOOP (MHPサービスの開始)
350    */
360     rtn_cod = dc_mcf_mainloop(DCNOFLAGS) ;
370     if(rtn_cod != DC_OK) {
380       printf("dc_mcf_mainloop 失敗 !! CODE = %d ¥n", rtn_cod) ;
390     }
400
410     printf("******  MCF CLOSE   *****¥n") ;
420   /*
430    * MCF-CLOSE (MCF環境のクローズ)
440    */
450     dc_mcf_close(DCNOFLAGS) ;
460
470   PROG_END :
480     printf("******  RPC CLOSE   *****¥n") ;
490   /*
500    * RPC-CLOSE (UAPの終了)
510    */
520     dc_rpc_close(DCNOFLAGS) ;
530     exit(0) ;
540   }
```

7.3.2 MHP の例（サービス関数）

MHP のサービス関数のコーディング例を次に示します。

```
 10  /*
 20   * MHP サービス関数
 30   */
 40  #include <stdio.h>
 50  #include <sys/types.h>
 60  #include <dcmcf.h>
 70  #include <dcrpc.h>
 80
 90  void svrA()
100  {
```

```
110       DCLONG action ;
120       DCLONG commform ;
130       DCLONG opcd ;
140       DCLONG active ;
150       char recvdata[1024] ;
160       DCLONG rdataleng ;
170       DCLONG time ;
180       DCLONG inbufleng ;
190       int  rtn_cod ;
200       DCLONG cdataleng ;
210       char termnam[10] ;
220       static char execdata[32] = "        SVRA EXECAP DATA" ;
230       static char senddata[32] = "        SVRA REPLY DATA1" ;
240       static char resv01[9] = "¥0" ;
250       static char resv02[9] = "¥0" ;
260       static char resv03[9] = "¥0" ;
270       static char apnam[9] = "aprepB" ;
280
290       printf("*****   UAP START   *****¥n") ;
300
310       printf("*****   MCF RECEIVE   *****¥n") ;
320   /*
330    * MCF-RECEIVE (メッセージの受信)
340    */
350       action = DCMCFFRST ;
360       commform = DCNOFLAGS ;
370       inbufleng = sizeof(recvdata) ;
380       rtn_cod = dc_mcf_receive(action, commform, termnam, resv01, recvdata,
390                  &rdataleng, inbufleng, &time) ;
400       if(rtn_cod != DCMCFRTN_00000) {
410   /*
420    * MCF-ROLLBACK (エラー処理)
430    */
440       printf("dc_mcf_receive 失敗 !! CODE = %d ¥n", rtn_cod) ;
450       rtn_cod = dc_mcf_rollback(DCMCFNRTN) ;
460       }
470
480       printf("*****   MCF EXECAP   *****¥n") ;
490   /*
500    * MCF-EXECAP (アプリケーションプログラム起動)
510    */
520       action = DCMCFEMI | DCMCFJUST ;
530       commform = DCNOFLAGS ;
540       active = 0 ;
550       cdataleng = 16 ;
560       rtn_cod = dc_mcf_execap(action, commform, resv01, active,
570                  apnam, execdata, cdataleng) ;
580       if(rtn_cod != DCMCFRTN_00000) {
590   /*
600    * MCF-ROLLBACK (エラー処理)
610    */
620       printf("dc_mcf_execap 失敗 !! CODE = %d ¥n", rtn_cod) ;
630       rtn_cod = dc_mcf_rollback(DCMCFNRTN) ;
640       }
650
660       printf("*****   MCF REPLY   *****¥n") ;
670   /*
680    * MCF-REPLY (応答メッセージの送信)
```

```
690   */
700       action = DCMCFEMI ;
710       commform = DCNOFLAGS ;
720       opcd = DCNOFLAGS ;
730       cdataleng = 16 ;
740       rtn_cod = dc_mcf_reply(action, commform, resv01, resv02,
750                 senddata, cdataleng, resv03, opcd) ;
760       if(rtn_cod != DCMCFRTN_00000) {
770   /*
780    * MCF-ROLLBACK (エラー処理)
790    */
800       printf("dc_mcf_reply 失敗 !! CODE = %d ¥n", rtn_cod) ;
810       rtn_cod = dc_mcf_rollback(DCMCFNRTN) ;
820       }
830   }
```

7.4.1　XATMI インタフェースの例

(1)　リクエスト/レスポンス型サービスの通信の例

(a)　処理の概要

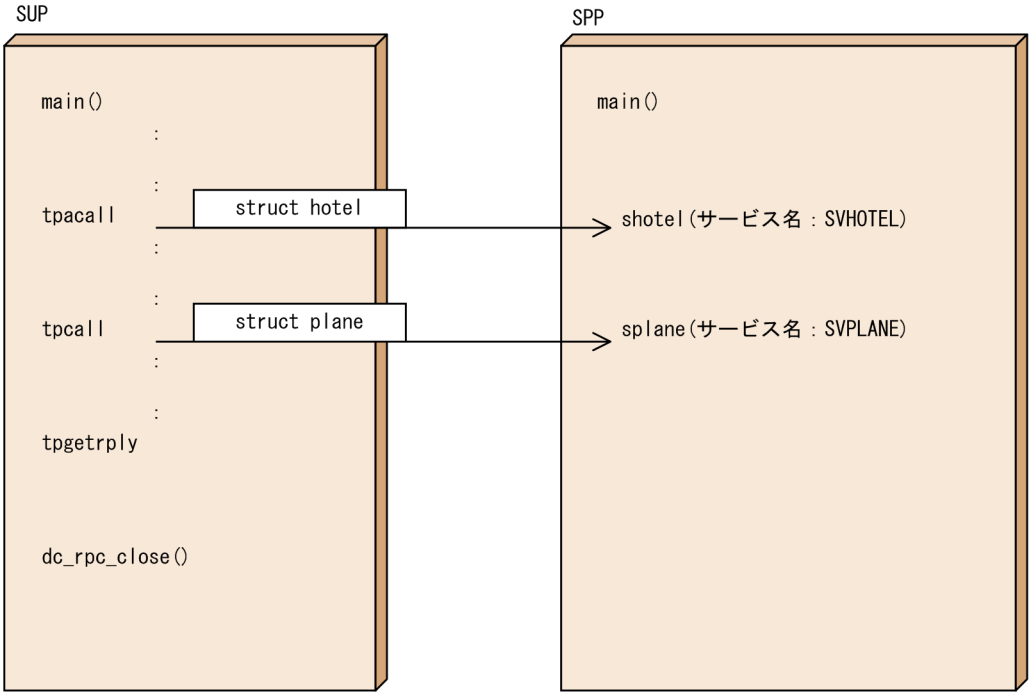
ここで示す例題の処理概要を，次に説明します。

［説明］宿泊施設の空き状況を調べるサービスと，飛行機の空き状況を調べるサービスを，SUP から呼びます。前者は非同期に，後者は同期的に応答を受信します。

(b)　UAP の構成

例題の UAP の構成を次の図に示します。

図 7-4　同期的に応答を受信するリクエスト/レスポンス型サービスの通信形態



(c)　通信に使う型付きバッファ

通信に使う型付きバッファの構造体を次に示します。

```
struct hotel {
    DCLONG date;
    char place[128];
    char hname[128];
```

```
struct plane {
    DCLONG date;
    char dest[128];
    DCLONG departure;
```

```
    DCLONG status;                    DCLONG status;
}                                 }
```

(d) SUP の例

- XATMI インタフェース定義の例

  リクエスト/レスポンス型サービスの例題で示す SUP の XATMI インタフェース定義を次に示します。

```
 10  /* SUPのXATMIインタフェース定義の例(rrsup.defファイル)*/
 20  called_servers = { ”rrspp.def” };
```

- SUP のコーディング例

  リクエスト/レスポンス型サービスの例題で示す SUP のコーディング例を次に示します。

```
 10  /* SUPの例(rrsup.cファイル)*/
 20  #include <stdio.h>
 30  #include <stdlib.h>
 40  #include <string.h>
 50  #include <dcrpc.h>
 60  #include <xatmi.h>
 70  #include <dcadm.h>
 80  /*
 90   * XATMIスタブヘッダファイル
100   */
110  #include ”rrsup_stbx.h”
120  main()
130  {
140  /*
150   * 変数の定義
160   */
170    struct hotel *hptr;
180    struct plane *pptr;
190    struct  errmsg *werrmsg ;
200    long  hlen, plen ;
210    int  cd ;
220    int rc;
230  /*
240   * RPC-OPEN(UAPの開始)
250   */
260       rc = dc_rpc_open(DCNOFLAGS);
270       if(rc != DC_OK){
280           printf(”dc_rpc_openに失敗しました。¥
290                   ERROR CODE = %d ¥n”, rc);
300           goto PROG_END;
310       }
320  /*
330   * ADM-COMPLETE(ユーザサーバの開始処理完了の報告)
340   */
350       rc = dc_adm_complete(DCNOFLAGS);
360       if(rc != DC_OK){
370           printf(”dc_adm_completeに失敗しました。¥
380                   ERROR CODE = %d ¥n”,  rc);
390           goto PROG_END;
400       }
410  /*
420   * TPALLOC(型付きバッファの確保)
```

```
 */
  /* 宿泊施設空き状況検索サービス用 */
    hptr = (struct hotel *)tpalloc("X_COMMON", "hotel", 0);
    if(hptr == NULL){
        printf("tpallocに失敗しました。¥
                ERROR CODE = %d ¥n", tperrno);
        goto PROG_END;
    }
  /* 飛行機空き状況検索サービス用 */
    pptr = (struct plane *)tpalloc("X_COMMON", "plane", 0);
    if(pptr == NULL){
        printf("tpallocに失敗しました。¥
                ERROR CODE = %d ¥n", tperrno);
        goto PROG_END;
    }
/*
 * データの設定
 */
  hptr->date = 940415 ;
  strcpy(hptr->place, "SAPPORO") ;
  strcpy(hptr->hname, "PRINCE") ;
  hptr->status = 0 ;
  pptr->date = 940415 ;
  strcpy(pptr->dest, "CHITOSE") ;
  pptr->departure = 1540 ;
  pptr->status = 0 ;
/*
 * TPACALL(サービスリクエストを送信する)
 */
    cd = tpacall("SVHOTEL", (char *) hptr, 0, 0);
    if(cd == -1){
       printf("宿泊施設空状況検索サービスの呼び出しに失敗しました。¥
               ERROR CODE = %d ¥n", tperrno);
       goto PROG_END;
    }
    printf("宿泊施設空状況検索サービスの呼び出しに成功しました。¥n");
/*
 * TPCALL(サービスリクエストを送信して,応答を待つ)
 */
    rc = tpcall("SVPLANE", (char *) pptr, 0, (char **) &pptr, &plen, 0);
    if(rc != 0){
        if(tperrno == TPESVCFAIL){
           werrmsg = (struct errmsg *) pptr ;
           printf("%s ERROR CODE = %d USER CODE = %d¥n",
                   werrmsg->errmessage, tperrno, tpurcode);
           goto PROG_END ;
        }else{
           printf("飛行機空き状況検索サービスの呼び出しに失敗しました。 ¥
                   ERROR CODE = %d", tperrno);
           goto PROG_END;
        }
    }
    printf("飛行機空き状況検索サービスの呼び出しの応答受信に成功しました。¥n");
    if(pptr->status == 1){
        printf("飛行機空き状況 : 満席です。 ¥n");
    } else {
        printf("飛行機空き状況 : 空席があります。 ¥n");
    }
```

```
1010  /*
1020   * TPGETRPLY(応答を受信する)
1030   */
1040      rc = tpgetrply(&cd, (char **) &hptr, &hlen, 0);
1050      if(rc != 0){
1060          if(tperrno == TPESVCFAIL){
1070             werrmsg = (struct errmsg *) hptr ;
1080             printf("%s ERROR CODE = %d USER CODE = %d¥n",
1090                     werrmsg->errmessage, tperrno, tpurcode);
1100             goto PROG_END ;
1110          }else{
1120             printf("宿泊施設空状況検索サービスに失敗しました。 ¥
1130                     ERROR CODE = %d", tperrno);
1140             goto PROG_END;
1150          }
1160      }
1170      printf("宿泊施設空状況検索サービスの応答受信に成功しました。¥n");
1180      if(hptr->status == 1){
1190          printf("宿泊施設空き状況 : 満室です。 ¥n");
1200      } else {
1210          printf("宿泊施設空き状況 : 空室があります。 ¥n");
1220      }
1230  /*
1240   * 型付きバッファの解放
1250   */
1260      tpfree((char *) hptr);
1270      tpfree((char *) pptr);
1280  /*
1290   * RPC-CLOSE(UAPの終了)
1300   */
1310      PROG_END:
1320      dc_rpc_close(DCNOFLAGS);
1330      printf("またのご利用をお待ちしています。¥n");
1340      exit(0);
1350  }
```

- ユーザサービス定義の例

  リクエスト/レスポンス型サービスの例題で示す SUP のユーザサービス定義例を次に示します。

```
  10  #ユーザサービス定義の例(rrsupファイル)
  20  set  module              = "rrsup"
  30  set  receive_from        = none
  40  set  trn_expiration_time = 180
  50  set  trn_expiration_time_suspend = Y
  60  set  xat_osi_usr = Y
```

(e) SPP の例

- XATMI インタフェース定義の例

  リクエスト/レスポンス型サービスの例題で示す SPP の XATMI インタフェース定義を次に示します。

```
  10  /* XATMIインタフェース定義の例(rrspp.defファイル)*/
  20  X_COMMON hotel {
  30      long   date;
  40      char   place[128];
  50      char   hname[128];
```

```
 60      long   status;
 70  };
 80  X_COMMON plane {
 90      long   date;
100      char   dest[128];
110      long   departure;
120      long   status;
130  };
140  X_COMMON errmsg {
150      char   errmessage[128];
160  };
170  service shotel(X_COMMON hotel) ;
180  service splane(X_COMMON plane) ;
```

- SPP のコーディング例（メイン関数）

  リクエスト/レスポンス型サービスの例題で示す SPP のコーディング例（メイン関数）を次に示します。

```
 10  /* SPPのメイン関数例(rrspp.cファイル)*/
 20  #include <stdio.h>
 30  #include <stdlib.h>
 40  #include <dcrpc.h>
 50  #include <xatmi.h>
 60  #include <dcadm.h>
 70  /*
 80   * XATMIスタブヘッダファイル
 90   */
100  #include "rrspp_stbx.h"
110  main()
120  {
130  /*
140   * 変数の定義
150   */
160     int rc;
170  /*
180   * RPC-OPEN(UAPの開始)
190   */
200      rc = dc_rpc_open(DCNOFLAGS);
210      if(rc != DC_OK){
220          printf("dc_rpc_openに失敗しました。¥
230                  ERROR CODE = %d ¥n", rc);
240          goto PROG_END;
250      }
260  /*
270   * RPC-MAINLOOP(SPPのサービス開始)
280   */
290      rc = dc_rpc_mainloop(DCNOFLAGS);
300      if(rc != DC_OK){
310          printf("dc_rpc_mainloopに失敗しました。¥
320                  ERROR CODE = %d ¥n", rc);
330      }
340  /*
350   * RPC-CLOSE(UAPの終了)
360   */
370  PROG_END:
380      dc_rpc_close(DCNOFLAGS);
390      exit(0);
400  }
```

- SPP のコーディング例（サービス関数）

  リクエスト/レスポンス型サービスの例題で示す SPP のコーディング例（サービス関数）を次に示します。

```
 10  /* SPPのサービス関数例(rrsvc.cファイル)*/
 20  #include <stdio.h>
 30  #include <dcrpc.h>
 40  #include <xatmi.h>
 50  #include <dcadm.h>
 60  /*
 70   * XATMIスタブヘッダファイル
 80   */
 90  #include "rrspp_stbx.h"
100  void shotel(svcinfo)
110  TPSVCINFO *svcinfo;
120  {
130  /*
140   * 変数の定義
150   */
160      struct hotel *hptr;
170
180      hptr = (struct hotel *) svcinfo->data;
190      /* このサービスは空き状況を検索して，満室ならば status = 1 ,
200       * 空室があれば status = 0, エラーが起こったときは，メッセージを返します。
210       * ここでは，検索した結果，満室だったとします。*/
220      hptr->status = 1 ;
230      tpreturn(TPSUCCESS, 0, (char *)hptr, 0, 0);
240      return ; /* OpenTP1の場合はtpreturn後にreturnの発行が必要です。 */
250  }
260  void splane(svcinfo)
270  TPSVCINFO *svcinfo;
280  {
290      struct plane *pptr;
300      pptr = (struct plane *) svcinfo->data;
310      /* このサービスは空き状況を検索して，満席ならば status = 1,
320       * 空席があれば status = 0 , エラーが起こったときは，メッセージを返します。
330       * ここでは，検索した結果，満席だったとします。*/
340      pptr->status = 1 ;
350      tpreturn(TPSUCCESS, 0, (char *)pptr, 0, 0);
360      return ;
370  }
```

- ユーザサービス定義の例

  リクエスト/レスポンス型サービスの例題で示す SPP のユーザサービス定義例を次に示します。

```
 10  #ユーザサービス定義の例(rrsppファイル)
 20  set    service_group      = "rrspp_svg"
 30  set    module             = "rrspp"
 40  set    service            = "SVHOTEL=shotel", "SVPLANE=splane"
 50  set    trn_expiration_time = 180
 60  set    trn_expiration_time_suspend = Y
 70  set    server_type = "xatmi"
 80  set    xat_osi_usr = Y
```

(2)　会話型サービスの通信の例

(a)　処理の概要

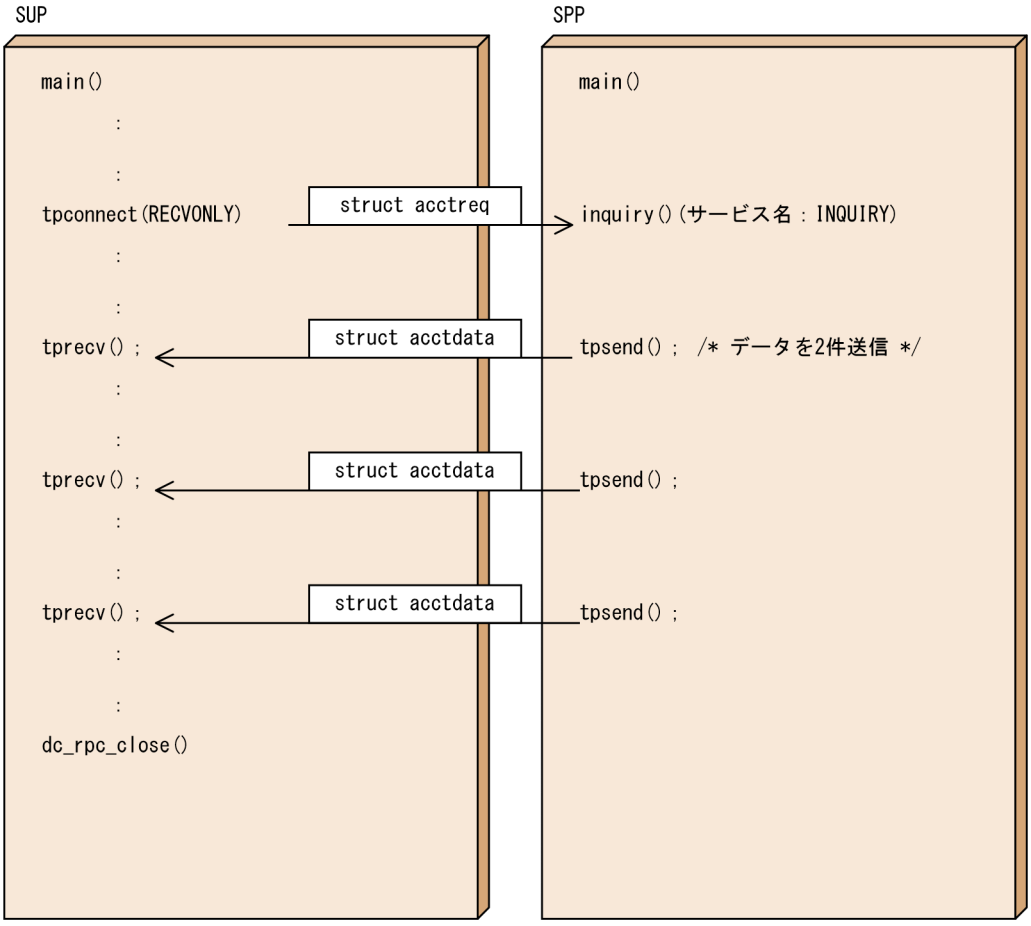
ここで示す例題の処理概要を，次に説明します。

［説明］acctreq 構造体の構造を持つ型付きバッファでサービス関数を起動します。acctreq のメンバは口座番号の上限と下限を示します。サービス関数では，この範囲にある口座データを acctdata 構造体の構造を持つ型付きバッファに設定して，会話のオリジネータに送信します。

(b)　UAP の構成

例題の UAP の構成を次の図に示します。

図 7-5　会話型サービスの通信形態



(c)　通信に使う型付きバッファ

通信に使う型付きバッファの構造体を次に示します。

- サービス関数起動時のデータ

```
struct acctreq{
  DCLONG upper_no;
```

```
  DCLONG lower_no;
}
```

- 会話型サービスの通信をするときのデータ

```
struct acctdata{
  DCLONG acct_no;
  char name[128];
  short amount;
  char dc_dummy0[2];
}
```

(d) SUP の例

- XATMI インタフェース定義の例

  会話型サービスの例題で示す SUP の XATMI インタフェース定義を次に示します。

```
  10  /* SUPのXATMIインタフェース定義の例(convsup.defファイル)*/
  20  called_servers = { ”convspp.def” };
```

- SUP のコーディング例

  会話型サービスの例題で示す SUP のコーディング例を次に示します。

```
  10  /* SUPのコーディング例(convsup.cファイル)*/
  20  #include <stdio.h>
  30  #include <stdlib.h>
  40  #include <dcrpc.h>
  50  #include <xatmi.h>
  60  #include <tx.h>
  70  #include <dcadm.h>
  80  /*
  90   * XATMIスタブヘッダファイル
 100   */
 110  #include ”convsup_stbx.h”
 120  main()
 130  {
 140  /*
 150   * 変数の定義
 160   */
 170      struct   acctreq  *rptr;
 180      struct   acctdata *dptr;
 190      long     wlen;
 200      int      cd;
 210      int      rc;
 220      long     revent;
 230      long     size = 0 ;
 240  /*
 250   * RPC-OPEN(UAPの開始)
 260   */
 270      rc = dc_rpc_open(DCNOFLAGS);
 280      if(rc != DC_OK){
 290          printf(”dc_rpc_openに失敗しました。ERROR CODE = %d ¥n”, rc);
 300          goto PROG_END;
 310      }
 320  /*
 330   * ADM-COMPLETE(ユーザサーバの開始処理完了の報告)
```

```
340   */
350      rc = dc_adm_complete(DCNOFLAGS);
360      if(rc != DC_OK){
370          printf("dc_adm_completeに失敗しました。ERROR CODE = %d ¥n", rc);
380          goto PROG_END;
390      }
400  /*
410   * TPALLOC(型付きバッファの確保)
420   */
430    /*サーチする口座番号の上限，下限の設定用*/
440      rptr = (struct acctreq *)tpalloc(X_COMMON, "acctreq", 0);
450
460      if(rptr == NULL){
470          printf("tpallocに失敗しました。ERROR CODE = %d ¥n", tperrno);
480          goto PROG_END;
490      }
500    /*サーチ結果の口座データ用*/
510      dptr = (struct acctdata *)tpalloc(X_COMMON, "acctdata", 0) ;
520      if(dptr == NULL){
530          printf("tpallocに失敗しました。ERROR CODE = %d ¥n", tperrno);
540          goto PROG_END;
550      }
560  /*
570   * データの設定
580   * サーチする範囲を指定
590   */
600      rptr->lower_no = 10000000L;
610      rptr->upper_no = 20000000L;
620    /* トランザクション開始  */
630      tx_begin() ;
640  /*
650   * TPCONNECT(会話サービスの呼び出し)
660   * INQUIRYを呼ぶ。
670   */
680      cd = tpconnect("INQUIRY", (char *) rptr, 0, TPRECVONLY);
690      if(cd == -1){
700          printf("tpconnectに失敗しました。ERROR CODE = %d ¥n", tperrno);
710          goto PROG_END;
720      }
730  /*
740   * エラーが起こるまで(イベントの発生も含む),
750   * TPRECV(メッセージを受信)
760   */
770      while(rc != -1){
780          rc = tprecv(cd, (char **) &dptr, &wlen, 0, &revent);
790          /*
800           *エラーが起こっていなければ,
810           *受信した口座情報を出力する。
820           */
830          if(rc != -1) {
840              printf("サービスから口座情報を受信しました。¥n");
850              printf("口座番号 = %d ¥n", dptr->acct_no);
860              printf("名前 = %s ¥n", dptr->name);
870              printf("預金 = %d ¥n", dptr->amount);
880          }
890      }
900  /*
910   * サービスの結果の出力
```

```
 920   */
 930      if(tperrno == TPEEVENT){
 940          if(revent == TPEV_SVCSUCC){
 950          /* サービスは成功した。*/
 960              printf("サービスは成功しました。¥n");
 970              /* トランザクションのコミット */
 980              tx_commit() ;
 990          }else{
1000              printf("何らかのイベントが発生しています。revent = %d¥n",
1010              revent);
1020              /* トランザクションのロールバック*/
1030              tx_rollback() ;
1040          }
1050    }
1060  /*
1070   * 型付きバッファの解放
1080   */
1090      tpfree((char *) rptr);
1100      tpfree((char *) dptr);
1110  /*
1120   * RPC-CLOSE(UAPの終了)
1130   */
1140      PROG_END:
1150      dc_rpc_close(DCNOFLAGS);
1160      exit(0);
1170  }
```

- ユーザサービス定義の例

  会話型サービスの例題で示す SUP のユーザサービス定義例を次に示します。

```
  10  #ユーザサービス定義の例(convsupファイル)
  20  set module              = "convsup"         #実行形式ファイル名
  30  set watch_time          = 180               #最大応答待ち時間
  40  set receive_from        = none              #受信方法
  50  set trn_expiration_time = 180
  60                           #トランザクションブランチ限界経過時間
  70  set trn_expiration_time_suspend = Y         #必ず Y を指定
```

(e) SPP の例

- XATMI インタフェース定義の例

  会話型サービスの例題で示す SPP の XATMI インタフェース定義を次に示します。

```
  10  /* SPPのXATMIインタフェース定義の例(convspp.defファイル)*/
  20  X_COMMON acctreq {
  30      long   upper_no;
  40      long   lower_no;
  50  };
  60  X_COMMON acctdata {
  70      long   acct_no;
  80      char   name[128];
  90      short  amount;
 100  };
 110  service inquiry(X_COMMON acctreq) ;
```

- SPP のコーディング例（メイン関数）

会話型サービスの例題で示す SPP のコーディング例（メイン関数）を次に示します。

```
 10  /* SPPのメイン関数例(convspp.cファイル)*/
 20  #include <stdio.h>
 30  #include <stdlib.h>
 40  #include <dcrpc.h>
 50  #include <xatmi.h>
 60  #include <dcadm.h>
 70  /*
 80   * XATMIスタブヘッダファイル
 90   */
100  #include "convspp_stbx.h"
110  main()
120  {
130  /*
140   * 変数の定義
150   */
160      int rc;
170  /*
180   * RPC-OPEN(UAPの開始)
190   */
200      rc = dc_rpc_open(DCNOFLAGS);
210      if(rc != DC_OK){
220          printf("dc_rpc_openに失敗しました。ERROR CODE = %d ¥n", rc);
230          goto PROG_END;
240      }
250
260  /*
270   * RPC-MAINLOOP(SPPのサービス開始)
280   */
290      rc = dc_rpc_mainloop(DCNOFLAGS);
300      if(rc != DC_OK){
310          printf("dc_rpc_mainloopに失敗しました。ERROR CODE = %d ¥n",rc);
320      }
330  /*
340   * RPC-CLOSE(UAPの終了)
350   */
360      PROG_END:
370      dc_rpc_close(DCNOFLAGS);
380      exit(0);
390  }
```

- SPP のコーディング例（サービス関数）

  会話型サービスの例題で示す SPP のコーディング例（サービス関数）を次に示します。

```
 10  /* SPPのサービス関数例(convsvc.cファイル)*/
 20  #include <stdio.h>
 30  #include <stdlib.h>
 40  #include <string.h>
 50  #include <dcrpc.h>
 60  #include <xatmi.h>
 70  #include <dcadm.h>
 80  /*
 90   * XATMIスタブヘッダファイル
100   */
110  #include "convspp_stbx.h"
120  /*
```

```
 * DEPOSITSVC  サービス関数
 * tpconnect()によって口座番号の上限，下限を受取り
 * その範囲にある口座の情報を送信する。
 */
void inquiry(svcinfo)
TPSVCINFO *svcinfo;
{
/*
 * 変数の定義
 */
    struct   acctreq  *rptr;
    struct   acctdata *dptr;
    char     type[9];
    char     subtype[17];
    long     revent, rval;
    int      size;
/*
 * サービス要求が受け付けられた
 */
    rptr = (struct acctreq *) svcinfo->data;
/*
 * オリジネータに返すデータのtypedバッファの確保
 */
    dptr = (struct acctdata *)tpalloc("X_COMMON", "acctdata", 0);
    if(dptr == NULL){
        printf("tpallocでエラーが起こりました。tperrno = %d ¥n",
               tperrno);
        abort();
    }
/*
 * ユーザ処理
 * データファイルを検索して口座番号が範囲内の口座情報を返す。
 * ここでは2件見つかったこととしてデータを送信する。
 */

    dptr->acct_no = 10000001L;
    strcpy(dptr->name, "Hitachi Hanako");
    dptr->amount = 20000;
/*
 * TPSEND(メッセージの送信)
 */
    tpsend(svcinfo->cd, (char *) dptr, 0, 0, &revent);
    if(tperrno != -1){
        rval = TPSUCCESS;
    }else{
        rval = TPFAIL;
        goto SVC_END;
    }
    dptr->acct_no = 10000002L;
    dptr->amount = 10000;
    strcpy(dptr->name, "Hitachi Tarou");
/*
 * TPSEND(メッセージの送信)
 */
    tpsend(svcinfo->cd, (char *) dptr, 0, 0, &revent);
    if(tperrno != -1){
        rval = TPSUCCESS;
    }else{
```

```
710           rval = TPFAIL;
720           goto SVC_END;
730       }
740   SVC_END:
750       tpreturn(rval, 0, NULL, 0, 0);
760       return; /* OpenTP1の場合 tpreturn後にreturnが必要です。*/
770   }
```

- ユーザサービス定義の例

  会話型サービスの例題で示す SPP のユーザサービス定義例を次に示します。

```
 10  # ユーザサービス定義の例(convsppファイル)
 20    set  service_group     = ”convspp_svg”      #サービスグループ名
 30    set  module            = ”convspp”          #実行形式ファイル名
 40    set  service  = ”INQUIRY=inquiry”
 50                              #サービス名=エントリポイント名
 60    set  watch_time        = 180                #最大応答待ち時間
 70    set  trn_expiration_time = 240
 80                            #トランザクションブランチ限界経過時間
 90    set  trn_expiration_time_suspend = Y        #必ず Y を指定
100    set  server_type = ”xatmi”                  #サーバタイプ
110    set  receive_from = ”socket”                #受信方法
```

7.4.2 TX インタフェースの例

X/Open の TX インタフェースを使用した，SUP のコーディング例を次に示します。この SUP は，7.1 で示す SUP の処理を，TX インタフェースでトランザクション制御したものです。処理の構成図と，サービス要求先の SPP については，「7.1 クライアント/サーバ形態の UAP のコーディング例（SUP，SPP DAM アクセス)」を参照してください

```
 10
 20  /*
 30   * SUP01
 40   */
 50  #include <stdio.h>
 60  #include <string.h>
 70  #include <dcrpc.h>
 80  #include <tx.h>
 90
100  main()
110  {
120  /*
130   * 変数の定義
140   */
150      static char in_buf [1024];
160      static long in_buf_len;
170      static char out_buf [1024];
180      static long out_buf_len;
190      int rc;
200    TRANSACTION_TIMEOUT trn_timeout = 180; /* 監視時間 180 秒*/
210    TXINFO info;
220  /*
230   * RPC-OPEN(UAPの開始)
```

```
240   */
250      rc = dc_rpc_open(DCNOFLAGS);
260      if(rc != DC_OK){
270          printf("SUP01:dc_rpc_openに失敗しました。CODE = %d ¥n",rc);
280          goto PROG_END;
290      }
300   /*
310    * TX-OPEN(リソースマネジャのオープン)
320    */
330      rc = tx_open();
340      if(rc != TX_OK){
350          printf("SUP01:tx_openに失敗しました。CODE = %d ¥n",rc);
360          goto PROG_END;
370      }
380   /*
390    * TX-SET-TRANSACTION-TIMEOUT(トランザクション監視時間の設定)
400    */
410      rc = tx_set_transaction_timeout(trn_timeout);
420      if(rc != TX_OK){
430          printf("SUP01:tx_set_transaction_timeoutに失敗しました。CODE = %d ¥n",rc);
440          goto PROG_END;
450      }
460   /*
470    * ADM-COMPLETE(ユーザサーバの開始処理完了の報告)
480    */
490      rc = dc_adm_complete(DCNOFLAGS);
500      if(rc != DC_OK){
510          printf("SUP01:dc_adm_completeに失敗しました。CODE = %d ¥n",rc);
520          goto PROG_END;
530      }
540   /*
550    * TX-BEGIN(トランザクションの開始)
560    */
570      rc = tx_begin();
580      if(rc != TX_OK){
590          printf("SUP01:tx_beginに失敗しました。CODE = %d ¥n",rc);
600          goto TRAN_END;
610      }
620
630   /*
640    * TX-INFO(トランザクション情報の取得)
650    */
660      rc = tx_info(&info);
670      if(rc <= 0){
680          printf("SUP01:現在トランザクションモードではありません。CODE = %d ¥n",rc);
690          goto PROG_END;
700      }else if (rc == 1){
710      printf("SUP01:return=%d,control=%d,timeout=%d,state=%d¥n",
720        info.when_return,info.transaction_control,
730        info.transaction_timeout, info.transaction_state);
740    }
750   /*
760    * RPC-CALL(遠隔サービスの要求)
770    */
780      strcpy(in_buf,"SUP01:DATA OpenTP1!!");
790      in_buf_len = strlen(in_buf) + 1;
800      out_buf_len = 1024;
810      rc = dc_rpc_call("svr01","svr01",in_buf,&in_buf_len,
```

```
 820      out_buf,&out_buf_len,DCNOFLAGS);
 830      if(rc != DC_OK){
 840          printf("SUP01:サービスの要求が失敗しました。 CODE = %d ¥n",rc);
 850          goto TRAN_END;
 860      }
 870      printf("SUP01:SERVICE FUNCTION RETURN = %s¥n",out_buf);
 880  /*
 890   * TX-SET-TRANSACTION-CONTROL(非連鎖モード設定)
 900   */
 910      TRAN_END:
 920      rc = tx_set_transaction_control(TX_UNCHAINED);
 930      if(rc != TX_OK){
 940          printf("SUP01:tx_set_transaction_controlに失敗しました。CODE = %d ¥n",rc);
 950      }
 960  /*
 970   * TX-COMMIT(非連鎖モードのコミット)
 980   */
 990      rc = tx_commit();
1000      if(rc != TX_OK){
1010          printf("SUP01:tx_commitに失敗しました。CODE = %d ¥n",rc);
1020      }
1030  /*
1040   * TX-CLOSE(リソースマネジャのクローズ)
1050   */
1060      PROG_END:
1070      rc = tx_close();
1080      if(rc != TX_OK){
1090          printf("SUP01:tx_closeに失敗しました。CODE = %d ¥n",rc);
1100          goto PROG_END;
1110      }
1120  /*
1130   * RPC-CLOSE(UAPの終了)
1140   */
1150      dc_rpc_close(DCNOFLAGS);
1160      printf("SUP01:処理を終了しました。¥n");
1170      exit(0);
1180  }
```

7.5　TxRPC の例題（IDL コンパイラが生成するテンプレート）

IDL コンパイラが出力するテンプレートについて説明します。このテンプレートを業務に応じて修正してください。

7.5.1　作成手順の概要

作成手順の概要について説明します。

(1)　スタブの作成と UAP のコーディング

スタブを作成して，UAP をコーディングするまでの手順を説明します。

(a)　IDL-only TxRPC の場合

1. 次に示すファイルを作成します。

   (1)IDL ファイル

   (2)クライアントプログラム

   (3)マネジャプログラム

2. txidl コンパイラで，手順 1 で作成した IDL ファイルをコンパイルします。この結果，次に示すファイルが生成されます。( ) は省略時仮定値の名称です（xxxx は，IDL ファイル名を示します）。

   (4)サーバプログラムのテンプレート（名称は serv.c で固定されています）

   サーバプログラムのテンプレートは，そのままでも使えます。内容を変更するときは，必要に応じて名称を変更してから，追加する処理をコーディングしてください。

   (5)ユーザサービス定義のテンプレート

   ユーザサービス定義のテンプレートは，修正しないと使えません。マニュアル「OpenTP1 システム定義」のユーザサービス定義の説明を参照して必要事項を定義してください。

   (6)環境定義ファイルのテンプレート（-cptype wdce のオプションを指定したとき）

   環境定義ファイルのテンプレートは，修正しないと使えません。マニュアル「OpenTP1 クライアント使用の手引 TP1/Client/W，TP1/Client/P 編」のクライアント環境定義についての説明を参照して，必要事項を定義してください。

   (7)クライアントスタブ（xxxx_cstub.c）

   (8)サーバスタブ（xxxx_sstub.c）

   (9)ヘッダファイル（xxxx.h）

(2)　UAP のコンパイルとリンケージ

プログラムを C コンパイラでコンパイルします。指定したプロセスタイプに応じてリンケージするライブラリが異なります。リンケージするライブラリを次に示します。

-lbetran：TP1/Server Base のライブラリ

-lclt：TP1/Client のライブラリ

指定するプロセスタイプと，クライアントプログラムとサーバプログラムに必要なライブラリを次に示します。

- -cptype ndce と-sptype ndce
  クライアント側：-lbetran と-ltactk
  サーバ側：-lbetran と-ltactk
- -cptype wdce と-sptype ndce
  クライアント側：-ltp1dce と-lclt，および DCE 関連ライブラリ
  サーバ側：-lbetran と-ltactk
- -cptype ndce と-sptype wdce
  クライアント側：-lbetran と-ltactk
  サーバ側：-ltp1dce と-lbetran，および DCE 関連ライブラリ
- -cptype wdce と-sptype nbet
  クライアント側：DCE 関連ライブラリ
  サーバ側：DCE 関連ライブラリ
- -cptype nbet と-sptype wdce
  クライアント側：DCE 関連ライブラリ
  サーバ側：DCE 関連ライブラリ
- -cptype nbet と-sptype nbet
  クライアント側：DCE 関連ライブラリ
  サーバ側：DCE 関連ライブラリ

7.5.2 各ファイルの例題

ファイルの例を次に示します。ここでは，次に示す例を示します。

- IDL ファイル
- クライアントプログラム
- マネジャプログラム
- ACF ファイル
- サーバプログラムのテンプレート
- ユーザサービス定義のテンプレート
- 環境定義のテンプレート

(1) IDL ファイルの例

IDL ファイルの例を次に示します。

```
 10  /*
 20   *  (1)IDLファイルの例 sample.idl
 30   */
 40  [
 50  uuid(f990a82a-10e5-11ce-9b02-0000870000ff),
 60  version(1.0),
 70  transaction_mandatory
 80  ]
 90  interface sample_ope
100  {
110    const  long  NAME_LENGTH = 20;  /* size of name field in record */
120    const  long  AGE_LEN = 3;       /* size of age field in record */
130    const  long  MAXRECORD = 10;    /* max number of record in database */
140
150    /*struct info : record format of customer information database */
160    typedef struct  info{
170      char  name[NAME_LENGTH];  /* name (20 byte) */
180      char  sex;                /* sex (1 byte) */
190      char  age[AGE_LEN];       /* age (3 byte) */
200      long  sale;               /* sale (4 byte) */
210    }info_t;
220
230     error_status_t  getinfo
240    (
250      [in] unsigned   char  name[NAME_LENGTH],  /* input parameter */
260      [out] info_t    *ptr                      /* output parameter */
270    );
280  }
290  /* EOF */
```

(2) クライアントプログラムの例

クライアントプログラムの例を次に示します。

```
 10  /*
 20   *    (2)クライアントプログラムの例
 30   *  注意事項:ndceタイプの場合は dc_rpc_open(),dc_adm_complete(),
 40   *           dc_rpc_close()が必要です。
 50   *           wdceタイプの場合は dc_clt_cltin(),dc_rpc_open(),
 60   *           dc_rpc_close(),dc_clt_cltin()が必要です。インクルードする
 70   *           ヘッダファイルは,TP1/Clientのライブラリを使ってください。
 80   *  clt.c
 90   *  Functions = main()
100   */
110
120  #include <stdio.h>
130  #include <dcrpc.h>
140  #include <dctrp.h>
150  #include <dcadm.h>
160  #include <tx.h>
170  #include "sample.h"
```

```
180
190   /*
200    *  Program Specification
210    *  construct customer information database. allow actions noted the following.
220    *    * reference processing
230    *      refer the information using "name" as a key.
240    *
250    *  customer information database
260    *  *-----------------------------------------*
270    *  | name         | sex    | age    | sale      |
280    *  |-----------------------------------------|
290    *  | Suzuki       | Male   | 30     | 1,000,000 |
300    *  | Okada        | Female | 23     | 1,500,000 |
310    *  | Yoshida      | Female | 26     |   800,000 |
320    *  | Saitoh       | Male   | 24     | 1,000,000 |
330    *  | Itoh         | Male   | 35     | 1,800,000 |
340    *  | Nishikawa    | Male   | 20     |   300,000 |
350    *  | Katoh        | Female | 28     | 1,000,000 |
360    *  | Satoh        | Female | 27     | 2,100,000 |
370    *  | Hasegawa     | Male   | 25     |   600,000 |
380    *  | Tumura       | Male   | 24     | 1,100,000 |
390    *  *-----------------------------------------*
400    *
410    *  This program requires service.
420    *    <refer>  refer Tumura's information.
430    *
440    */
450   /*
460    *  name = main()
470    *  func = Client program for sample_ope interface
480    *       (1)service requirement(reference)
490    *       (2)output result of service requirement
500    *  arg = nothing
510    *  return = void
520    */
530
540   int  main()
550   {
560     static unsigned char name[] = "Tumura";  /* input parameter */
570     info_t out_data;                         /* output parameter */
580     error_status_t status;                   /* return code for server */
590     int     rc;                              /* return code */
600
610   /*
620    *  Start UAP
630    */
640     rc = dc_rpc_open(DCNOFLAGS);
650     /* error processing */
660     if(rc != DC_OK){
670       fprintf(stderr,"client:dc_rpc_open failed. rc = %d¥n",rc);
680       goto END;
690     }
700
710     /*
720     **  Post the completion of user process start processing
730     */
740     rc = dc_adm_complete(DCNOFLAGS);
750     /* error processing */
```

```
 760     if(rc != DC_OK){
 770       fprintf(stderr,"client:dc_adm_complete failed. rc = %d¥n",rc);
 780       goto END;
 790     }
 800
 810   /*
 820    *  Begin  transaction
 830    */
 840
 850     rc = tx_begin();
 860     /* error processing */
 870     if(rc != DC_OK){
 880       fprintf(stderr,"client:tx_begin failed. rc = %d¥n",rc);
 890       goto END;
 900     }
 910
 920   /*
 930    *  getinfo:
 940    *  get information for input parameter
 950    */
 960     status = getinfo(name,&out_data);
 970     if(status != 0){
 980       fprintf(stderr,"client:getinfo failed.rc = %d¥n",status);
 990     }else{
1000       fprintf(stdout,"NAME: %s SEX: %c AGE: %s SALE:%ld¥n",
1010           out_data.name,
1020           out_data.sex,
1030           out_data.age,
1040           out_data.sale);
1050     }
1060   /*
1070    *  commit  transaction
1080    */
1090
1100     rc = tx_commit();
1110     /* error processing */
1120     if(rc != DC_OK){
1130       fprintf(stderr,"client:tx_commit failed. rc = %d¥n",rc);
1140       goto END;
1150     }
1160   /*
1170    *  Termination processing
1180    */
1190   END:
1200     dc_rpc_close(DCNOFLAGS);
1210     return(0);
1220   }
```

(3)　マネジャプログラムの例

マネジャプログラムの例を次に示します。

```
  10   /*
  20    *
  30    *    (3)マネジャプログラムの例
  40    *  sv.c
```

```
 50    *  Data Table = customers
 60    *  Functions = main()
 70    *        getinfo()
 80    */
 90
100   #include <stdio.h>
110   #include <string.h>
120   #include "sample.h"
130
140   /*
150    *  name   = customers
160    *  func   = customer information database
170    *  field  = name(20byte)
180    *      sex(1byte)
190    *      age(3byte)
200    *      sale(4byte)
210    *  record  = 10 record(1 record = 28 byte)
220    */
230   static info_t customers[MAXRECORD] =
240           { {"Suzuki",  'M',"30",1000000},
250             {"Okada",    'F',"23",1500000},
260             {"Yoshida",  'F',"26", 800000},
270             {"Saitoh",  'M',"24",1000000},
280             {"Itoh",    'M',"35",1800000},
290             {"Nishikawa",  'M',"20", 300000},
300             {"Katoh",    'F',"28",1000000},
310             {"Satoh",    'F',"27",2100000},
320             {"Hasegawa",  'M',"25", 600000},
330             {"Tumura",  'M',"24",1100000}
340       };
350
360   /*
370    *  name = getinfo()
380    *  func = Manager routine for sample_ope interface
390    *    (1)search suitable record.
400    *    (2)set found record to output parameter.
410    *  arg =   name   :i: name
420    *      out_data:o: information for input parameter
430    *  return = result
440    *      0 : success getinfo
450    */
460
470   error_status_t  getinfo(name,out_data)
480   unsigned char  *name;
490   info_t  *out_data;
500   {
510     int i;          /* counter of for loop */
520     info_t *ptr;    /* pointer for search record */
530
540   /* point 1st record of database(customers) */
550     ptr = customers;
560
570   /* search until find record with same name or end of database */
580     for (i = 0; i < MAXRECORD; i++, ptr++) {
590       /* compare name */
600       if(strcmp(name,ptr->name) == 0) {
610         memcpy(out_data,ptr,sizeof(info_t));
620         return (0);
```

```
630         }
640     }
650     return(1);
660   }
```

(4) ACF ファイルの例

ACF ファイルの例を次に示します。

```
 10   /*
 20    *
 30    *    (4) ACFファイルの例 RPC TxRPCでだけ使えます。
 40    *  sample.acf
 50    */
 60
 70   [auto_handle] interface sample_ope
 80   {
 90     [comm_status, fault_status] getinfo();
100   }
```

(5) サーバプログラムのテンプレート例

txidl コマンドの引数に指定する値によって，サーバプログラムのテンプレート例は異なります。-sptype ndce オプションを指定した場合の例を次に示します。

```
 10   /*
 20    *
 30    *    (5)サーバプログラムのテンプレート(名称 : serv.c)
 40    *    <-sptype ndceの場合>
 50    */
 60
 70   #include <dctrp.h>
 80
 90   main()
100   {
110     idl_long_int rc;
120     rc = dc_rpc_open(DCNOFLAGS);
130     if(rc != DC_OK) {
140       printf("server : dc_rpc_open failed. rc=%d¥n", rc);
150       goto end_of_program;
160     }
170     rc=dc_rpc_mainloop(DCNOFLAGS);
180     if(rc != DC_OK) {
190       printf("server : dc_rpc_mainloop failed. rc=%d¥n", rc);
200     }
210   end_of_program:
220     dc_rpc_close(DCNOFLAGS);
230     exit(0);
240   }
```

(6) ユーザサービス定義のテンプレート例

txidl コマンドの引数に指定する値によって，ユーザサービス定義のテンプレート例は異なります。それぞれのオプションを指定した場合の例を次に示します。

- -cptype ndce オプションを指定した場合

```
 10  /*
 20   *    (6)ユーザサービス定義 テンプレートの例
 30   *   <-cptype ndce の場合>
 40   */
 50
 60  #Don't change the 2 definitions below.
 70
 80  set atomic_update = Y
 90
100  set trn_expiration_time_suspend = Y
110
120  # If this program is SUP, set none. If other, set queue or socket.
130
140  set receive_from = none
150
160  #Set your modulename.
170
180  set module = "modulename"
190
200  #Set non-zero value.
210
220  set trn_expiration_time = 180
230
240  #Add any definition you need.
```

- -sptype ndce オプションを指定した場合

```
 10  /*
 20   *    (6)ユーザサービス定義 テンプレートの例
 30   *       <-sptype ndce の場合>
 40   */
 50
 60  #Don't change the  4 definitions below.
 70
 80
 90  set atomic_update = Y
100
110  set trn_expiration_time_suspend = Y
120
130  set service_group = "sample_ope"
140
150  set service = "_getinfo=_getinfo"
160
170  #Set your modulename.
180
190  set module = "modulename"
200
210  #Set non-zero value.
220
230  set trn_expiration_time = 180
```

```
240
250   #Add any definition you need.
```

- -sptype wdce オプションを指定した場合

```
 10   /*
 20    *    (6)ユーザサービス定義 テンプレートの例
 30    *       <-sptype wdce の場合>
 40    */
 50
 60   #Don't change the  4 definitions below.
 70
 80   set atomic_update = N
 90
100   set receive_from = queue
110
120   set service_group = "sample_ope"
130
140   set service = "_getinfo=_getinfo"
150
160   #Set your modulename.
170
180   set module = "modulename"
190
200   #Add any definition you need.
```

(7)　環境定義のテンプレートの例

環境定義のテンプレートの例を次に示します。

```
 10   /*
 20    *    (7)環境定義 テンプレートの例
 30    *       <-cptype wdce の場合>
 40    */
 50
 60   #Set the 2 definition below
 70
 80   #DCNAMPORT =
 90
100   #DCHOST =
110
120   #Add any definition you need.
```

8 アプリケーション起動関連のリファレンス

この章では，TP1/Message Control を使った場合の，アプリケーションプログラムを起動する機能に関連するユーザオウンコーディングと MCF イベントのリファレンス情報について説明します。

タイマ起動引き継ぎ決定 UOC の関数形式

タイマ起動引き継ぎ決定 UOC は，次に示す形式で呼び出します。

形式

ANSI C，C++ の形式

```
#include <dcmpsv.h>
DCLONG  uoc_func(dcmpsv_uoc_rtime  *parm)
```

K&R 版 C の形式

```
#include <dcmpsv.h>
DCLONG  uoc_func(parm)

dcmpsv_uoc_rtime    *parm ;
```

説明

タイマ起動の dc_mcf_execap 関数を呼び出したあとで，障害が起こって OpenTP1 を再開始（リラン）した場合に，タイマ起動の環境を変更する UOC です。この UOC を作成しておくと，次に示すことができます。

- タイマ起動を引き継ぐか，または取り消す。
- 引き継いだタイマ起動を即時起動とする。
- 起動するアプリケーション名を変更する。

タイマ起動引き継ぎ決定 UOC を MCF に組み込むときは，アプリケーション起動サービス用の MCF メイン関数に，UOC 関数のアドレスを設定します。アプリケーション起動サービス用の MCF メイン関数は，通信プロトコルに依存しません。

アプリケーション起動サービス用の MCF メイン関数の作成方法については，マニュアル「OpenTP1 運用と操作」を参照してください。

uoc_func（タイマ起動引き継ぎ決定 UOC）を呼び出すときには，parm には次に示すパラメタが MCF から設定されます。

パラメタの内容

dcmpsv_uoc_rtime の内容

```
typedef  struct {
     char  le_name[9];   … 入力元論理端末名称
     char  reserve1[7];  … 予備
     char  ap_name[9];   … アプリケーション名
     char  reserve2[7];  … 予備
     DCLONG  exec_time;    … タイマ起動時刻
```

```
    char  ap_type;       … アプリケーションの型
                             'a':ans型  'n':noans型
    char  time_type;     … タイマ起動の種別
                             'i':経過時間指定のタイマ起動
                             't':時刻指定のタイマ起動
    char  reserve3[26]; … 予備
} dcmpsv_uoc_rtime;
```

MCF から UOC へ値が渡される引数

●le_name

入力元の論理端末名称が設定されます。dc_mcf_execap 関数を SPP から呼び出している場合は，'*' が設定されます。

●ap_name

UAP がタイマ起動の dc_mcf_execap 関数に設定した，アプリケーション名が設定されます。

●exec_time

UAP がタイマ起動の dc_mcf_execap 関数に設定した，MHP を起動する時間が，1970 年 1 月 1 日 0 時 0 分 0 秒からの通算秒で設定されます。

●ap_type

タイマ起動の dc_mcf_execap 関数を呼び出した UAP の，アプリケーションの型が設定されます。

'a'：ans 型

'n'：noans 型

●time_type

UAP がタイマ起動の dc_mcf_execap 関数に設定した，タイマ起動の種別が設定されます。

'i'：経過時間指定のタイマ起動

't'：時刻指定のタイマ起動

UOC で値を設定する引数

●ap_name

タイマ起動で開始させるアプリケーション名を変更するときに，変更後のアプリケーション名を設定します。ここに指定した名称は，リターン値に DCMPSV_UOC_TIME_JUST を設定したときに有効となります。

リターン値

uoc_func()は，次に示す値でリターンしてください。

リターン値	意味
DCMPSV_UOC_TIME_CONTINUE	タイマ起動を継続します。
DCMPSV_UOC_TIME_JUST	即時起動として起動します。
DCMPSV_UOC_TIME_DEQ	タイマ起動を取り消します。

uoc_func()からリターンした値による MCF の処理を次に示します。

- DCMPSV_UOC_TIME_CONTINUE

  UOC からこの値をリターンすると，MCF は時刻を 1970 年 0 時 0 分 0 秒からの通算秒で取得して，dc_mcf_execap 関数に設定された時刻と比べます。

  比べた結果，関数に設定した時刻を過ぎていた場合，該当する MHP を即時に起動します。設定した時刻を過ぎていない場合は，タイマ起動となります。

- DCMPSV_UOC_TIME_JUST

  UOC からこの値をリターンすると，MCF は該当する MHP を即時に起動します。この値をリターンする場合は，即時起動する MHP のアプリケーション名を，UOC で変更できます。ただし，MCF イベント処理用 MHP を起動するような変更はできません。変更したアプリケーション名が定義されていない名称の場合は，ERREVT4 が通知されます。

  UOC で即時起動する UAP のアプリケーション名を変更した場合に，変更前と変更後で起動する MHP のアプリケーションの型が異なるときは，タイマ起動するセグメントは出力キューから削除されて，警告メッセージ（KFCA10711-W）が出力されます。

- DCMPSV_UOC_TIME_DEQ

  UOC からこの値をリターンすると，MCF はタイマ起動を取り消します。タイマ起動するセグメントは出力キューから削除されて，情報メッセージ（KFCA10700-I）が出力されます。

上記以外の値を UOC からリターンすると，タイマ起動するセグメントは出力キューから削除されて，警告メッセージ（KFCA10710-W）が出力されます。

UOC 作成上の注意事項

- UOC で使用できる関数

  UOC を作成する場合，UOC では次に示す関数だけが使用できます。ほかの関数を使用した場合，正常に動作しないことがあるため注意してください。

  - メモリ操作をする関数

    データ領域管理（例：malloc，free）

    共有メモリ管理（例：shmctl，shmget，shmop）

    メモリ操作（例：memcpy）

    文字列操作（例：strcpy）

  - 時間取得関数

- UOC の異常処理

UOC で異常を検知した場合，MCF の所定のリターンコードを使用して MCF に異常の発生を通知してください。UOC でプロセス終了となるシグナル，または abort()を発行すると，MCF が異常終了します。

- UOC の実行タイミング

  MCF が起動する UOC の実行タイミングは，OpenTP1 システム，および UAP の開始，終了シーケンスと同期しない場合があります。UAP より先に UOC が実行されたり，UAP がすべて終了してから UOC が呼び出されたりしてもよいように作ってください。

- UOC のローカル変数サイズ

  UOC で使用するローカル変数のサイズの合計は，各 UOC で 1 キロバイト以内になるよう設計してください。また，UOC の中で関数の再帰呼出しはしないでください。

タイマ起動メッセージ廃棄通知イベント（ERREVT4）の構造体形式

タイマ起動メッセージ廃棄通知イベント（ERREVT4）の先頭セグメントとして渡される，構造体の形式を次に示します。

この構造体はヘッダファイル<dcmcf.h>で定義されています。MCF イベント情報を処理する MHP では，#include で<dcmcf.h>をインクルードしてください。

ERREVT4 以外の MCF イベント情報の形式は，マニュアル「OpenTP1 プロトコル」の該当するプロトコル編を参照してください。

MCF イベント情報の共通ヘッダ

```
struct  dc_mcf_evtheader {
     char  mcfevt_name[9] ;       …MCFイベントコード
     char  le_name[16] ;          …入力元論理端末名称
     char  cn_name[9] ;           …コネクション名
     unsigned  char  format_kind ;…MCFで使う領域
     char  reserve01 ;            …予備
     DCLONG  time ;                 …メッセージ入力時刻
} ;
```

ERREVT4 の形式

```
struct  dc_mcf_evt4_type {
     struct  dc_mcf_evtheader evtheader ; …MCFイベント共通ヘッダ
     char  reserve01[12] ;                …予備
     char  reserve02[10] ;                …予備
     char  reserve03[2] ;                 …予備
     char  ap_name[10] ;                  …アプリケーション名
                                            （タイマ起動メッセージに
                                              対応する
                                              アプリケーション名）
     short reason_code ;                  … 理由コード
} ;
```

引数の内容

●le_name

メッセージを入力した論理端末名称が設定されます。次に示す場合には'*'が設定されます。

- SPP から dc_mcf_execap 関数を呼び出して起動した MHP で，障害が発生した場合
- 上記の障害が発生したあとに，MCF イベントとして起動した MHP からさらに dc_mcf_execap 関数を呼び出して起動した MHP で，障害が発生した場合

●cn_name

コネクション名が設定されます。次に示す場合には，'*'が設定されます。

- SPP から dc_mcf_execap 関数を呼び出して起動した MHP で，障害が発生した場合
- 上記の障害が発生したあとに，MCF イベントとして起動した MHP からさらに dc_mcf_execap 関数を呼び出して起動した MHP で，障害が発生した場合

●time

メッセージを入力した時刻が，1970 年 1 月 1 日 0 時 0 分 0 秒からの通算の秒数で設定されます。

●ap_name

異常が起こったタイマ起動の dc_mcf_execap 関数に設定したアプリケーション名が設定されます。

●reason_code

ERREVT4 が通知された理由を示すコードが設定されます。理由コードの内容を次に示します。

C 言語の理由コード（16 進数）	ERREVT4 の通知理由
DCMCF_SCD_ERR（0020）	RPC 障害，サーバ未起動などが原因で，MHP の起動に失敗しました。
DCMCF_QUE_BUF_ERR（0030）	メモリ不足のため，入力キューへの書き込みに失敗しました。
DCMCF_QUE_FIL_OVER（0031）	キューファイルが満杯のため，入力キューへの書き込みに失敗しました。
DCMCF_QUE_LIMIT_OVER（0032）	入力メッセージ最大格納数の定義指定値を超えたため，入力キューに書き込みませんでした。
DCMCF_QUE_IO_ERR（0033）	入力キューへの書き込み時に障害が起こりました。
DCMCF_AP_CLOSE（0040）	MHP のアプリケーションが閉塞中です。
DCMCF_AP_SECURE（0041）	MHP のアプリケーションがセキュア状態です。
DCMCF_SERV_CLOSE（0042）	MHP のサービスまたはサービスグループが閉塞中です。
DCMCF_SERV_SECURE（0043）	MHP のサービスグループがセキュア状態です。

付録

付録 A　OpenTP1 のリモートプロシジャコールと XATMI インタフェースの関数を併用する場合

OpenTP1 のプロセス間通信（OpenTP1 のリモートプロシジャコールと XATMI インタフェースの関数）を併用する方法について説明します。

XATMI インタフェースと併用できるのは，OpenTP1 独自のインタフェースだけです。TxRPC と XATMI インタフェースは併用できません。

付録 A.1　併用する形態

併用する形態には，次に示す 2 とおりがあります。

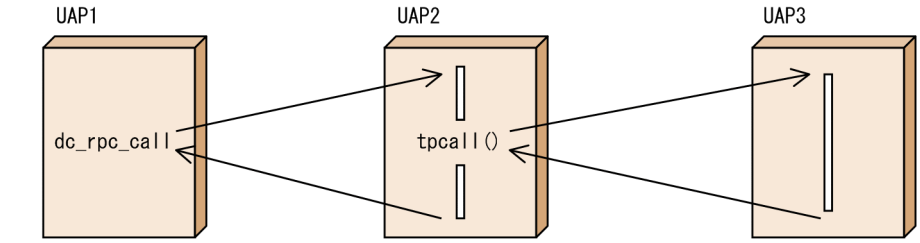
1. OpenTP1 の RPC のサーバであり，XATMI インタフェース通信のクライアントである場合
2. XATMI インタフェース通信のサーバであり，OpenTP1 の RPC のクライアントである場合

このうち，1.の形態では，スタブを作成するときに RPC インタフェース定義と XATMI インタフェース定義を一つのファイルに指定して，stbmake コマンドまたは tpstbmk コマンドを実行します。

通信を併用する形態と必要なインタフェース定義を次の図に示します。

図 A-1　通信を併用する形態と必要なインタフェース定義

1. OpenTP1のRPCのサーバであり，XATMIインタフェース通信のクライアントである場合



● UAP1に必要な
インタフェース定義

なし

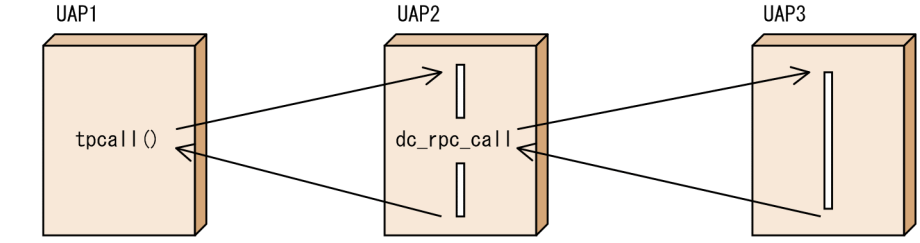
● UAP2に必要な
インタフェース定義

・RPCインタフェース定義
・XATMIインタフェース定義
（クライアント用）

● UAP3に必要な
インタフェース定義

・XATMIインタフェース定義
（サーバUAP用）

2. XATMIインタフェース通信のサーバであり，OpenTP1のRPCのクライアントである場合



● UAP1に必要な
インタフェース定義

・XATMIインタフェース定義
（クライアントUAP用）

● UAP2に必要な
インタフェース定義

・XATMIインタフェース定義
（サーバUAP用）

● UAP3に必要な
インタフェース定義

・RPCインタフェース定義

付録 A.2　併用するアプリケーションプログラムのスタブの作成手順

dc_rpc_call 関数で呼ばれて，XATMI インタフェースの関数（tpcall()など）を呼び出す UAP のスタブを作成する方法について説明します。

1. インタフェース定義ファイルを作成します

作成するファイルには，RPC インタフェース定義と XATMI インタフェース定義のクライアント用の定義を指定します。ファイル名は".def"で終わらせてください。

2. stbmake コマンドまたは tpstbmk コマンドを実行します

スタブを生成するコマンド（stbmake コマンドまたは tpstbmk コマンド）に必要な引数を指定して実行します。コマンドを実行すると，次のファイルが作成されます。xxxxx はインタフェース定義ファイルのファイル名から".def"を除いた文字列を示します。

- OpenTP1 の RPC 用スタブソースファイル
  （デフォルトのファイル名：xxxxx_sstb.c）
- XATMI スタブソースファイル

（デフォルトのファイル名：xxxxx_stbx.c）

- XATMI スタブヘッダファイル
  （デフォルトのファイル名：xxxxx_stbx.h）

RPC インタフェース定義と XATMI インタフェース定義が混在している場合には，XATMI スタブソースファイルと XATMI スタブヘッダファイルが作成されます。

3. スタブソースファイルをコンパイルして，UAP にリンケージさせます

2.で作成したソースファイルを C コンパイラでコンパイルして，UAP にリンケージさせます。

付録 A.3　呼び出せる XATMI インタフェースの関数

dc_rpc_call 関数で呼び出された SPP が使える XATMI インタフェースの関数を次の表に示します。これらの関数を呼び出す SPP にはスタブを結合していることが前提となります。詳細については，「付録 A.2 併用するアプリケーションプログラムのスタブの作成手順」を参照してください。

表 A-1　dc_rpc_call 関数で呼び出された SPP が使える XATMI インタフェースの関数

XATMI インタフェースの関数	使用可否
tpacall	○
tpadvertise	−
tpalloc	○
tpcall	○
tpcancel	○
tpconnect	○
tpdiscon	○
tpgetrply	○
tpfree	○
tprecv	○
tprealloc	○
tpreturn	○
tpsend	○
tpservice	−
tptypes	○
tpunadvertise	−

（凡例）
○：関数を使えます。

－：関数を使えません。

注

tpservice は，サービス関数の実体を示します。

付録 B　インタフェースの変更一覧（バージョン 6 以前から移行する場合）

TP1/Message Control のバージョン 6 以前から移行する場合に，32 ビットアーキテクチャ以外では，C 言語のソースファイルを見直す必要があります。ここでは，バージョン 6 以前のインタフェースの変更一覧を示します。

ここで説明するインタフェースを次に示します。

表 B-1　インタフェースの変更一覧

変更されたインタフェース		バージョン 7 のマニュアルの該当個所
メッセージ送受信インタフェース	dc_mcf_ap_info	2.　dc_mcf_ap_info
	dc_mcf_ap_info_uoc	2.　dc_mcf_ap_info_uoc
	dc_mcf_close	2.　dc_mcf_close
	dc_mcf_commit	2.　dc_mcf_commit
	dc_mcf_contend	2.　dc_mcf_contend
	dc_mcf_execap	2.　dc_mcf_execap
	dc_mcf_mainloop	2.　dc_mcf_mainloop
	dc_mcf_open	2.　dc_mcf_open
	dc_mcf_receive	2.　dc_mcf_receive
	dc_mcf_rollback	2.　dc_mcf_rollback
	dc_mcf_tempget	2.　dc_mcf_tempget
	dc_mcf_tempput	2.　dc_mcf_tempput
	dc_mcf_timer_cancel	2.　dc_mcf_timer_cancel
	dc_mcf_timer_set	2.　dc_mcf_timer_set
ユーザオウンコーディング	タイマ起動引き継ぎ決定 UOC	8.　タイマ起動引き継ぎ決定 UOC の関数形式
MCF イベントインタフェース		8.　タイマ起動メッセージ廃棄通知イベント（ERREVT4）の構造体形式
MHP サービス関数のコーディング例		7.3.2　MHP の例（サービス関数）

以降，バージョン 6 以前のインタフェースと，バージョン 7 以前のインタフェースの変更一覧を示します。変更個所には，下線を付与しています。

付録 B.1　メッセージ送受信インタフェース

ここでは，メッセージ送受信インタフェースの変更一覧を示します。

(1) dc_mcf_ap_info －アプリケーション情報通知

(a) ANSI C，C++の形式

バージョン 6 以前

＜ 32 ビットアーキテクチャの場合＞

```
int  dc_mcf_ap_info(long flags,
                    char *mcfid,
                    char *apname,
                    struct DC_MCFAPINFO *apinfo,
                    char *resv01,
                    long resv02)
```

＜ 64 ビットアーキテクチャの場合＞

```
int  dc_mcf_ap_info(int flags,
                    char *mcfid,
                    char *apname,
                    struct DC_MCFAPINFO *apinfo,
                    char *resv01,
                    int resv02)
```

バージョン 7

```
int  dc_mcf_ap_info(DCLONG flags,
                    char *mcfid,
                    char *apname,
                    struct DC_MCFAPINFO *apinfo,
                    char *resv01,
                    DCLONG resv02)
```

(b) K&R 版 C の形式

バージョン 6 以前

＜ 32 ビットアーキテクチャの場合＞

```
int  dc_mcf_ap_info(flags,
                    mcfid,
                    apname,
                    apinfo,
                    resv01,
                    resv02)
long flags;
char *mcfid;
char *apname;
struct DC_MCFAPINFO  *apinfo;
char *resv01;
long resv02;
```

＜ 64 ビットアーキテクチャの場合＞

```
int  dc_mcf_ap_info(flags,
                    mcfid,
                    apname,
                    apinfo,
                    resv01,
                    resv02)
int flags;
char *mcfid;
char *apname;
struct DC_MCFAPINFO  *apinfo;
char *resv01;
int resv02;
```

バージョン 7

```
int  dc_mcf_ap_info(flags,
                    mcfid,
                    apname,
                    apinfo,
                    resv01,
                    resv02)
DCLONG flags;
char *mcfid;
char *apname;
struct DC_MCFAPINFO  *apinfo;
char *resv01;
DCLONG resv02;
```

(c) OpenTP1 から値が返される引数

●apinfo

バージョン 6 以前	バージョン 7

＜32 ビットアーキテクチャの場合＞
`struct DC_MCFAPINFO {`
`char mcf_apinfo[4];`
`long mcf_resv00;`
`char mcf_ap_name[9];`
`char mcf_ap_mcfid[3];`
`char mcf_resv01[4];`
`long mcf_ap_stat;`
`long mcf_ap_type;`
`char mcf_sg_name[32];`
`long mcf_sg_stat;`
`long mcf_sg_hold;`
`char mcf_sv_name[32];`
`long mcf_sv_stat;`
`long mcf_ap_ntmetim;`
`long mcf_ap_tempsize;`
`long mcf_ap_msgcnt;`
`long mcf_ap_trnmode;`
`long mcf_ap_quekind;`
`char mcf_resv02[72];`
`}`
	`struct DC_MCFAPINFO {`
	`char mcf_apinfo[4];`
	`DCLONG mcf_resv00;`
	`char mcf_ap_name[9];`
	`char mcf_ap_mcfid[3];`
	`char mcf_resv01[4];`
	`DCLONG mcf_ap_stat;`
	`DCLONG mcf_ap_type;`
	`char mcf_sg_name[32];`
	`DCLONG mcf_sg_stat;`
	`DCLONG mcf_sg_hold;`
	`char mcf_sv_name[32];`
	`DCLONG mcf_sv_stat;`
	`DCLONG mcf_ap_ntmetim;`
	`DCLONG mcf_ap_tempsize;`
	`DCLONG mcf_ap_msgcnt;`
	`DCLONG mcf_ap_trnmode;`
	`DCLONG mcf_ap_quekind;`
	`char mcf_resv02[72];`
	`}`

＜64 ビットアーキテクチャの場合＞
`struct DC_MCFAPINFO {`
`char mcf_apinfo[4];`
`int mcf_resv00;`
`char mcf_ap_name[9];`
`char mcf_ap_mcfid[3];`
`char mcf_resv01[4];`
`int mcf_ap_stat;`
`int mcf_ap_type;`
`char mcf_sg_name[32];`
`int mcf_sg_stat;`
`int mcf_sg_hold;`
`char mcf_sv_name[32];`
`int mcf_sv_stat;`
`int mcf_ap_ntmetim;`
`int mcf_ap_tempsize;`
`int mcf_ap_msgcnt;`
`int mcf_ap_trnmode;`
`int mcf_ap_quekind;`
`char mcf_resv02[72];`
`}`	

(2) dc_mcf_ap_info_uoc − UOC へのアプリケーション情報通知

(a) ANSI C，C++の形式

バージョン 6 以前	バージョン 7

＜32 ビットアーキテクチャの場合＞
`int dc_mcf_ap_info_uoc(long flags,`
`char *apname,`
	`int dc_mcf_ap_info_uoc(DCLONG flags,`
	`char *apname,`
	`struct DC_MCFAPINFO_UOC *ap`
	`info)`

バージョン 6 以前	バージョン 7
struct DC_MCFAPINFO_UOC *apinf o)	int dc_mcf_ap_info_uoc(DCLONG flags, char *apname, struct DC_MCFAPINFO_UOC *ap info)
＜ 64 ビットアーキテクチャの場合＞ int dc_mcf_ap_info_uoc(int flags, char *apname, struct DC_MCFAPINFO_UOC *apinf o)	

(b) K&R 版 C の形式

バージョン 6 以前	バージョン 7
＜ 32 ビットアーキテクチャの場合＞ int dc_mcf_ap_info_uoc(flags,apname, apinfo) long flags; char *apname; struct DC_MCFAPINFO_UOC *apinfo;	int dc_mcf_ap_info_uoc(flags,apname, apinfo) DCLONG flags; char *apname; struct DC_MCFAPINFO_UOC *apinfo;
＜ 64 ビットアーキテクチャの場合＞ int dc_mcf_ap_info_uoc(flags,apname, apinfo) int flags; char *apname; struct DC_MCFAPINFO_UOC *apinfo;	

(c) OpenTP1 から値が返される引数

●apinfo

バージョン 6 以前	バージョン 7

＜ 32 ビットアーキテクチャの場合＞
struct DC_MCFAPINFO_UOC {
char mcf_apinfo[4];
long mcf_resv00;
char mcf_ap_name[9];
char mcf_ap_mcfid[3];
char mcf_resv01[4];
long mcf_ap_stat;
long mcf_ap_type;
long mcf_ap_msgcnt;
char mcf_sg_name[32];
long mcf_sg_stat;
long mcf_sg_hold;
long mcf_sg_msgcnt;
char mcf_sv_name[32];
long mcf_sv_stat;
long mcf_ap_ntmetim;
long mcf_ap_tempsize;
long mcf_ap_max_msgcnt;
long mcf_ap_trnmode;
long mcf_ap_quekind;
char mcf_resv02[64];
};
	struct DC_MCFAPINFO_UOC {
	char mcf_apinfo[4];
	DCLONG mcf_resv00;
	char mcf_ap_name[9];
	char mcf_ap_mcfid[3];
	char mcf_resv01[4];
	DCLONG mcf_ap_stat;
	DCLONG mcf_ap_type;
	DCLONG mcf_ap_msgcnt;
	char mcf_sg_name[32];
	DCLONG mcf_sg_stat;
	DCLONG mcf_sg_hold;
	DCLONG mcf_sg_msgcnt;
	char mcf_sv_name[32];
	DCLONG mcf_sv_stat;
	DCLONG mcf_ap_ntmetim;
	DCLONG mcf_ap_tempsize;
	DCLONG mcf_ap_max_msgcnt;
	DCLONG mcf_ap_trnmode;
	DCLONG mcf_ap_quekind;
	char mcf_resv02[64];
	};

バージョン 6 以前	バージョン 7

＜64 ビットアーキテクチャの場合＞
struct DC_MCFAPINFO_UOC {
char mcf_apinfo[4];
int mcf_resv00;
char mcf_ap_name[9];
char mcf_ap_mcfid[3];
char mcf_resv01[4];
int mcf_ap_stat;
int mcf_ap_type;
int mcf_ap_msgcnt;
char mcf_sg_name[32];
int mcf_sg_stat;
int mcf_sg_hold;
int mcf_sg_msgcnt;
char mcf_sv_name[32];
int mcf_sv_stat;
int mcf_ap_ntmetim;
int mcf_ap_tempsize;
int mcf_ap_max_msgcnt;
int mcf_ap_trnmode;
int mcf_ap_quekind;
char mcf_resv02[64];
};
	struct DC_MCFAPINFO_UOC {
	char mcf_apinfo[4];
	DCLONG mcf_resv00;
	char mcf_ap_name[9];
	char mcf_ap_mcfid[3];
	char mcf_resv01[4];
	DCLONG mcf_ap_stat;
	DCLONG mcf_ap_type;
	DCLONG mcf_ap_msgcnt;
	char mcf_sg_name[32];
	DCLONG mcf_sg_stat;
	DCLONG mcf_sg_hold;
	DCLONG mcf_sg_msgcnt;
	char mcf_sv_name[32];
	DCLONG mcf_sv_stat;
	DCLONG mcf_ap_ntmetim;
	DCLONG mcf_ap_tempsize;
	DCLONG mcf_ap_max_msgcnt;
	DCLONG mcf_ap_trnmode;
	DCLONG mcf_ap_quekind;
	char mcf_resv02[64];
	};

(3) dc_mcf_close − MCF 環境のクローズ

(a) ANSI C，C++の形式

バージョン 6 以前	バージョン 7
＜32 ビットアーキテクチャの場合＞ void dc_mcf_close(long flags)	void dc_mcf_close(DCLONG flags)
＜64 ビットアーキテクチャの場合＞ void dc_mcf_close(int flags)	

(b) K&R 版 C の形式

バージョン 6 以前	バージョン 7
＜32 ビットアーキテクチャの場合＞ void dc_mcf_close(flags) long flags;	void dc_mcf_close(flags) DCLONG flags;
＜64 ビットアーキテクチャの場合＞ void dc_mcf_close(flags) int flags;	

(4) dc_mcf_commit − MHP のコミット

(a) ANSI C，C++の形式

バージョン 6 以前	バージョン 7
＜32 ビットアーキテクチャの場合＞ `int dc_mcf_commit(long action)`	`int dc_mcf_commit(DCLONG action)`
＜64 ビットアーキテクチャの場合＞ `int dc_mcf_commit(int action)`	

(b) K&R 版 C の形式

バージョン 6 以前	バージョン 7

＜32 ビットアーキテクチャの場合＞
`int dc_mcf_commit(action)`
`long action;`
	`int dc_mcf_commit(action)`
	`DCLONG action;`

＜64 ビットアーキテクチャの場合＞
`int dc_mcf_commit(action)`
`int action;`	

(5) dc_mcf_contend − 継続問い合わせ応答の終了

(a) ANSI C，C++の形式

バージョン 6 以前	バージョン 7

＜32 ビットアーキテクチャの場合＞
`int dc_mcf_contend(long action,`
`char *resv01)`
	`int dc_mcf_contend(DCLONG action,`
	`char *resv01)`

＜64 ビットアーキテクチャの場合＞
`int dc_mcf_contend(int action,`
`char *resv01)`	

(b) K&R 版 C の形式

バージョン 6 以前	バージョン 7

＜32 ビットアーキテクチャの場合＞
`int dc_mcf_contend(action,resv01)`
`long action;`
`char *resv01;`
	`int dc_mcf_contend(action,resv01)`
	`DCLONG action;`
	`char *resv01;`

＜64 ビットアーキテクチャの場合＞	

バージョン 6 以前	バージョン 7

`int  dc_mcf_contend(action,resv01)`
`int action;`
`char *resv01;`
	`int  dc_mcf_contend(action,resv01)`
	`DCLONG action;`
	`char *resv01;`

(6)　dc_mcf_execap － アプリケーションプログラムの起動

(a)　ANSI C，C++の形式

バージョン 6 以前	バージョン 7

＜ 32 ビットアーキテクチャの場合＞
`int  dc_mcf_execap(long action,`
`long commform,`
`char *resv01,`
`long active,`
`char *apnam,`
`char *comdata,`
`long cdataleng)`
＜ 64 ビットアーキテクチャの場合＞
`int  dc_mcf_execap(int action,`
`int commform,`
`char *resv01,`
`int active,`
`char *apnam,`
`char *comdata,`
`int cdataleng)`
	`int  dc_mcf_execap(DCLONG action,`
	`DCLONG commform,`
	`char *resv01,`
	`DCLONG active,`
	`char *apnam,`
	`char *comdata,`
	`DCLONG cdataleng)`

(b)　K&R 版 C の形式

バージョン 6 以前	バージョン 7

＜ 32 ビットアーキテクチャの場合＞
`int  dc_mcf_execap(action,commform,`
`resv01,active,apnam,`
`comdata,cdataleng)`
`long action;`
`long commform;`
`char *resv01;`
`long active;`
`char *apnam;`
`char *comdata;`
`long cdataleng;`
＜ 64 ビットアーキテクチャの場合＞
`int  dc_mcf_execap(action,commform,`
`resv01,active,apnam,`
`comdata,cdataleng)`
`int action;`
`int commform;`
`char *resv01;`
`int active;`
`char *apnam;`
`char *comdata;`
`int cdataleng;`
	`int  dc_mcf_execap(action,commform,`
	`resv01,active,apnam,`
	`comdata,cdataleng)`
	`DCLONG action;`
	`DCLONG commform;`
	`char  *resv01;`
	`DCLONG active;`
	`char *apnam;`
	`char *comdata;`
	`DCLONG cdataleng;`

(7) dc_mcf_mainloop − MHP のサービス開始

(a) ANSI C，C++の形式

バージョン 6 以前	バージョン 7
＜ 32 ビットアーキテクチャの場合＞ `int dc_mcf_mainloop(long flags)`	`int dc_mcf_mainloop(DCLONG flags)`
＜ 64 ビットアーキテクチャの場合＞ `int dc_mcf_mainloop(int flags)`	

(b) K&R 版 C の形式

バージョン 6 以前	バージョン 7

＜ 32 ビットアーキテクチャの場合＞
`int dc_mcf_mainloop(flags)`
`long flags;`
	`int dc_mcf_mainloop(flags)`
	`DCLONG flags;`

＜ 64 ビットアーキテクチャの場合＞
`int dc_mcf_mainloop(flags)`
`int flags;`	

(8) dc_mcf_open − MCF 環境のオープン

(a) ANSI C，C++の形式

バージョン 6 以前	バージョン 7
＜ 32 ビットアーキテクチャの場合＞ `int dc_mcf_open(long flags)`	`int dc_mcf_open(DCLONG flags)`
＜ 64 ビットアーキテクチャの場合＞ `int dc_mcf_open(int flags)`	

(b) K&R 版 C の形式

バージョン 6 以前	バージョン 7

＜ 32 ビットアーキテクチャの場合＞
`int dc_mcf_open(flags)`
`long flags;`
	`int dc_mcf_open(flags)`
	`DCLONG flags;`

＜ 64 ビットアーキテクチャの場合＞
`int dc_mcf_open(flags)`
`int flags;`	

(9) dc_mcf_receive － メッセージの受信

(a) ANSI C，C++の形式

バージョン 6 以前	バージョン 7

＜32 ビットアーキテクチャの場合＞
`int dc_mcf_receive(long action,`
`long commform,`
`char *termnam,`
`char *resv01,`
`char *recvdata,`
`long *rdataleng,`
`long inbufleng,`
`long *time)`
	`int dc_mcf_receive(DCLONG action,`
	`DCLONG commform,`
	`char *termnam,`
	`char *resv01,`
	`char *recvdata,`
	`DCLONG *rdataleng,`
	`DCLONG inbufleng,`
	`DCLONG *time)`

＜64 ビットアーキテクチャの場合＞
`int dc_mcf_receive(int action,`
`int commform,`
`char *termnam,`
`char *resv01,`
`char *recvdata,`
`int *rdataleng,`
`int inbufleng,`
`int *time)`	

(b) K&R 版 C の形式

バージョン 6 以前	バージョン 7

＜32 ビットアーキテクチャの場合＞
`int dc_mcf_receive(action,commform,`
`termnam,resv01,`
`recvdata,rdataleng,`
`inbufleng,time)`
`long action;`
`long commform;`
`char *termnam;`
`char *resv01;`
`char *recvdata;`
`long *rdataleng;`
`long inbufleng;`
`long *time;`
	`int dc_mcf_receive(action,commform,`
	`termnam,resv01,`
	`recvdata,rdataleng,`
	`inbufleng,time)`
	`DCLONG action;`
	`DCLONG commform;`
	`char *termnam;`
	`char *resv01;`
	`char *recvdata;`
	`DCLONG *rdataleng;`
	`DCLONG inbufleng;`
	`DCLONG *time;`

＜64 ビットアーキテクチャの場合＞
`int dc_mcf_receive(action,commform,`
`termnam,resv01,`
`recvdata,rdataleng,`
`inbufleng,time)`
`int action;`
`int commform;`
`char *termnam;`
`char *resv01;`
`char *recvdata;`
`int *rdataleng;`
`int inbufleng;`
`int *time;`	

(10) dc_mcf_rollback − MHP のロールバック

(a) ANSI C，C++の形式

バージョン 6 以前	バージョン 7
＜ 32 ビットアーキテクチャの場合＞ `int dc_mcf_rollback(long action)`	`int dc_mcf_rollback(DCLONG action)`
＜ 64 ビットアーキテクチャの場合＞ `int dc_mcf_rollback(int action)`	

(b) K&R 版 C の形式

バージョン 6 以前	バージョン 7

＜ 32 ビットアーキテクチャの場合＞
`int dc_mcf_rollback(action)`
`long action;`
	`int dc_mcf_rollback(action)`
	`DCLONG action;`

＜ 64 ビットアーキテクチャの場合＞
`int dc_mcf_rollback(action)`
`int action;`	

(11) dc_mcf_tempget − 一時記憶データの受け取り

(a) ANSI C，C++の形式

バージョン 6 以前	バージョン 7

＜ 32 ビットアーキテクチャの場合＞
`int dc_mcf_tempget(long action,`
`char *getdata,`
`long gtempleng,`
`long *gdataleng,`
`char *resv01)`
	`int dc_mcf_tempget(DCLONG action,`
	`char *getdata,`
	`DCLONG gtempleng,`
	`DCLONG *gdataleng,`
	`char *resv01)`

＜ 64 ビットアーキテクチャの場合＞
`int dc_mcf_tempget(int action,`
`char *getdata,`
`int gtempleng,`
`int *gdataleng,`
`char *resv01)`	

(b) K&R 版 C の形式

バージョン 6 以前	バージョン 7
＜ 32 ビットアーキテクチャの場合＞	`int dc_mcf_tempget(action,getdata,` `gtempleng,gdataleng,`

バージョン 6 以前	バージョン 7
int dc_mcf_tempget(action,getdata, gtempleng,gdataleng, resv01) long action; char *getdata; long gtempleng; long *gdataleng; char *resv01; ＜ 64 ビットアーキテクチャの場合＞ int dc_mcf_tempget(action,getdata, gtempleng,gdataleng, resv01) int action; char *getdata; int gtempleng; int *gdataleng; char *resv01;	resv01) DCLONG action; char *getdata; DCLONG gtempleng; DCLONG *gdataleng; char *resv01;

(12) dc_mcf_tempput － 一時記憶データの更新

(a) ANSI C，C++の形式

バージョン 6 以前	バージョン 7
＜ 32 ビットアーキテクチャの場合＞ int dc_mcf_tempput(long action, char *putdata, long pdataleng, char *resv01) ＜ 64 ビットアーキテクチャの場合＞ int dc_mcf_tempput(int action, char *putdata, int pdataleng, char *resv01)	int dc_mcf_tempput(DCLONG action, char *putdata, DCLONG pdataleng, char *resv01)

(b) K&R 版 C の形式

バージョン 6 以前	バージョン 7
＜ 32 ビットアーキテクチャの場合＞ int dc_mcf_tempput(action,putdata, pdataleng,resv01) long action; char *putdata; long pdataleng; char *resv01; ＜ 64 ビットアーキテクチャの場合＞ int dc_mcf_tempput(action,putdata, pdataleng,resv01) int action; char *putdata;	int dc_mcf_tempput(action,putdata, pdataleng,resv01) DCLONG action; char *putdata; DCLONG pdataleng; char *resv01;

バージョン 6 以前	バージョン 7
int pdataleng; char *resv01;	int dc_mcf_tempput(action,putdata, pdataleng,resv01) DCLONG action; char *putdata; DCLONG pdataleng; char *resv01;

(13) dc_mcf_timer_cancel － ユーザタイマ監視の取り消し

(a) ANSI C，C++の形式

バージョン 6 以前	バージョン 7
＜32 ビットアーキテクチャの場合＞ int dc_mcf_timer_cancel(long flags, long timer_id, char *lename) ＜64 ビットアーキテクチャの場合＞ int dc_mcf_timer_cancel(int flags, int timer_id, char *lename)	int dc_mcf_timer_cancel(DCLONG flags, DCLONG timer_id, char *lename)

(b) K&R 版 C の形式

バージョン 6 以前	バージョン 7
＜32 ビットアーキテクチャの場合＞ int dc_mcf_timer_cancel(flags, timer_id, lename) long flags; long timer_id; char *lename; ＜64 ビットアーキテクチャの場合＞ int dc_mcf_timer_cancel(flags, timer_id, lename) int flags; int timer_id; char *lename;	int dc_mcf_timer_cancel(flags, timer_id, lename) DCLONG flags; DCLONG timer_id; char *lename;

(14) dc_mcf_timer_set － ユーザタイマ監視の設定

(a) ANSI C，C++の形式

バージョン 6 以前	バージョン 7
＜32 ビットアーキテクチャの場合＞	int dc_mcf_timer_set(DCLONG flags, DCLONG timer,

バージョン 6 以前	バージョン 7
int dc_mcf_timer_set(long flags, long timer, long timer_id, char *lename, char *apname, char *data, long data_leng, long resv01, long resv02)	DCLONG timer_id, char *lename, char *apname, char *data, DCLONG data_leng, DCLONG resv01, DCLONG resv02)
＜ 64 ビットアーキテクチャの場合＞ int dc_mcf_timer_set(int flags, int timer, int timer_id, char *lename, char *apname, char *data, int data_leng, int resv01, int resv02)	

(b)　K&R 版 C の形式

バージョン 6 以前	バージョン 7
＜ 32 ビットアーキテクチャの場合＞ int dc_mcf_timer_set(flags,timer, timer_id,lename, apname,data,data_leng, resv01,resv02) long flags; long timer; long timer_id; char *lename; char *apname; char *data; long data_leng; long resv01; long resv02;	int dc_mcf_timer_set(flags,timer, timer_id,lename,apname, data,data_leng,resv01, resv02) DCLONG flags; DCLONG timer; DCLONG timer_id; char *lename; char *apname; char *data; DCLONG data_leng; DCLONG resv01; DCLONG resv02;
＜ 64 ビットアーキテクチャの場合＞ int dc_mcf_timer_set(flags,timer, timer_id,lename, apname,data,data_leng, resv01,resv02) int flags; int timer; int timer_id; char *lename; char *apname; char *data; int data_leng; int resv01; int resv02;	

付録 B.2　ユーザオウンコーディング

ここでは，ユーザオウンコーディングの変更一覧を示します。

(1)　タイマ起動引き継ぎ決定 UOC

(a)　形式

ANSI C，C++の形式

バージョン 6 以前	バージョン 7
＜ 32 ビットアーキテクチャの場合＞ `long  uoc_func(dcmpsv_uoc_rtime  *parm)`	`DCLONG uoc_func(dcmpsv_uoc_rtime  *parm)`
＜ 64 ビットアーキテクチャの場合＞ `int  uoc_func(dcmpsv_uoc_rtime  *parm)`	

K&R 版 C の形式

バージョン 6 以前	バージョン 7

＜ 32 ビットアーキテクチャの場合＞
`long  uoc_func(parm)`
`dcmpsv_uoc_rtime *parm ;`
	`DCLONG uoc_func(parm)`
	`dcmpsv_uoc_rtime *parm ;`

＜ 64 ビットアーキテクチャの場合＞
`int  uoc_func(parm)`
`dcmpsv_uoc_rtime *parm ;`	

(b)　パラメタの内容

dcmpsv_uoc_rtime の内容

バージョン 6 以前	バージョン 7

＜ 32 ビットアーキテクチャの場合＞
`typedef  struct {char le_name[9];`
`char reserve1[7];`
`char ap_name[9];`
`char reserve2[7];`
`long exec_time;`
`char ap_type;`
`char time_type;`
`char reserve3[26];`
`} dcmpsv_uoc_rtime;`
	`typedef  struct {char le_name[9];`
	`char reserve1[7];`
	`char ap_name[9];`
	`char reserve2[7];`
	`DCLONG exec_time;`
	`char ap_type;`
	`char time_type;`
	`char reserve3[26];`
	`} dcmpsv_uoc_rtime;`

＜ 64 ビットアーキテクチャの場合＞
`typedef  struct {char le_name[9];`
`char reserve1[7];`
`char ap_name[9];`
`char reserve2[7];`	

バージョン 6 以前	バージョン 7
int exec_time; char ap_type; char time_type; char reserve3[26]; } dcmpsv_uoc_rtime;	typedef struct {char le_name[9]; char reserve1[7]; char ap_name[9]; char reserve2[7]; DCLONG exec_time; char ap_type; char time_type; char reserve3[26]; } dcmpsv_uoc_rtime;

付録 B.3 MCF イベントインタフェース

ここでは，MCF イベントインタフェースの変更一覧を示します。

(1) タイマ起動メッセージ廃棄通知イベント（ERREVT4）の構造体形式

(a) MCF イベント情報の共通ヘッダの形式

バージョン 6 以前	バージョン 7

＜ 32 ビットアーキテクチャの場合＞
struct dc_mcf_evtheader {
char mcfevt_name[9] ;
char le_name[16] ;
char cn_name[9] ;
unsigned char format_kind;
char reserve01;
long time ;
};
	struct dc_mcf_evtheader {
	char mcfevt_name[9] ;
	char le_name[16] ;
	char cn_name[9] ;
	unsigned char format_kind;
	char reserve01;
	DCLONG time ;
	};

＜ 64 ビットアーキテクチャの場合＞
struct dc_mcf_evtheader {
char mcfevt_name[9] ;
char le_name[16] ;
char cn_name[9] ;
unsigned char format_kind;
char reserve01;
int time ;
};	

付録 B.4 MHP サービス関数のコーディング例

ユーザアプリケーションプログラムの作成例の変更一覧を示します。下線部分が変更個所です。

```
10  /*
20   * MHP  サービス関数
30   */
40  #include <stdio.h>
50  #include <sys/types.h>
```

```
 60  #include <dcmcf.h>
 70  #include <dcrpc.h>
 80
 90  void svrA()
100  {
110     DCLONG action ;
120     DCLONG commform ;
130     DCLONG opcd ;
140     DCLONG active ;
150     char recvdata [1024] ;
160     DCLONG rdataleng ;
170     DCLONG time ;
180     DCLONG inbufleng ;
190     int  rtn_cod ;
200     DCLONG cdataleng ;
210     char termnam [10]  ;
220     static char execdata [32]  = "        SVRA EXECAP DATA" ;
230     static char senddata [32]  = "        SVRA REPLY DATA1" ;
240     static char resv01 [9]  = "¥0" ;
250     static char resv02 [9]  = "¥0" ;
260     static char resv03 [9]  = "¥0" ;
270     static char apnam [9]  = "aprepB" ;
280
290     printf("*****   UAP START   *****¥n") ;
300
310     printf("*****   MCF RECEIVE   *****¥n") ;
320  /*
330   * MCF- RECEIVE (メッセージの受信)
340   */
350       action = DCMCFFRST ;
360       commform = DCNOFLAGS ;
370       inbufleng = sizeof(recvdata) ;
380       rtn_cod = dc_mcf_receive(action, commform, termnam, resv01,
390                 recvdata, &rdataleng, inbufleng, &time) ;
400       if(rtn_cod != DCMCFRTN_00000) {
410  /*
420   * MCF- ROLLBACK (エラー処理)
430   */
440       printf("dc_mcf_receive 失敗 !! CODE = %d ¥n", rtn_cod) ;
450       rtn_cod = dc_mcf_rollback(DCMCFNRTN) ;
460       }
470
480       printf("*****   MCF EXECAP    *****¥n") ;
490  /*
500   * MCF-EXECAP (アプリケーションプログラム起動)
510   */
520       action = DCMCFEMI | DCMCFJUST ;
530       commform = DCNOFLAGS ;
540       active = 0 ;
550       cdataleng = 16 ;
560       rtn_cod = dc_mcf_execap(action, commform, resv01, active,
570                 apnam, comdata, cdataleng) ;
580       if(rtn_cod != DCMCFRTN_00000) {
590  /*
600   * MCF- ROLLBACK (エラー処理)
610   */
620       printf("dc_mcf_execap 失敗 !! CODE = %d ¥n", rtn_cod) ;
630       rtn_cod = dc_mcf_rollback(DCMCFNRTN) ;
```

```
640       }
650
660       printf("*****   MCF REPLY   *****¥n") ;
670   /*
680    * MCF-REPLY（応答メッセージの送信）
690    */
700       action = DCMCFEMI ;
710       commform = DCNOFLAGS ;
720       opcd = DCNOFLAGS ;
730       cdataleng = 16 ;
740       rtn_cod = dc_mcf_reply(action, commform, resv01, resv02,
750                 senddata, cdataleng, resv03, opcd) ;
760       if(rtn_cod != DCMCFRTN_00000) {
770   /*
780    * MCF- ROLLBACK（エラー処理）
790    */
800       printf("dc_mcf_reply 失敗 !! CODE = %d ¥n", rtn_cod) ;
810       rtn_cod = dc_mcf_rollback(DCMCFNRTN) ;
820       }
830   }
```

索引

E

T

U

V

W

X

あ

い

う

え

し

す

せ

そ

た

つ

て

も

ゆ

ら

り

れ

ろ