

# Tmax リファレンスガイド

Tmax v6.0



Copyright © 2016 TmaxSoft Co., Ltd. All Rights Reserved.

## Copyright Notice

Copyright © 2016 TmaxSoft Co., Ltd. All Rights Reserved.

45, Jeongjail-ro, Bundang-gu, Seongnam-si, Gyeonggi-do, 13613, South Korea

## Restricted Rights Legend

All TmaxSoft Software (Tmax®) and documents are protected by copyright laws and international convention. TmaxSoft software and documents are made available under the terms of the TmaxSoft License Agreement and this document may only be distributed or copied in accordance with the terms of this agreement. No part of this document may be transmitted, copied, deployed, or reproduced in any form or by any means, electronic, mechanical, or optical, without the prior written consent of TmaxSoft Co., Ltd.

Nothing in this software document and agreement constitutes a transfer of intellectual property rights regardless of whether or not such rights are registered) or any rights to TmaxSoft trademarks, logos, or any other brand features. This document is for information purposes only. The company assumes no direct or indirect responsibilities for the contents of this document, and does not guarantee that the information contained in this document satisfies certain legal or commercial conditions. The information contained in this document is subject to change without prior notice due to product upgrades or updates. The company assumes no liability for any errors in this document.

このソフトウェア(Tmax®)マニュアルの内容とプログラムは、日本国の著作権法および国際条約によって保護されています。マニュアルの内容とプログラムは、TmaxSoft Co., Ltd.との使用許諾契約書の下でのみ使用することができ、マニュアルは使用許諾契約で許可されている範囲を除いては、配布または複製することができません。TmaxSoftの書面による事前の承諾を得ることなく、このマニュアルの全部または一部を電子的または機械的な方法を問わず、転送、複製、配布したり、または二次的著作物を作成する等の行為を一切禁じます。

このソフトウェアのマニュアルとプログラムの使用許諾契約は、いかなる場合においても、マニュアル及びプログラムと関連する知的財産権(登録の有無を問わず)を譲渡するものと解釈されず、TmaxSoftのブランド、ロゴ、商標等の使用権限を与えるものではありません。マニュアルは、情報を提供する目的でのみ提供しており、これに伴う契約上の直接的ないしは間接的な責任を負わず、マニュアルの内容は法律上もしくは商業的な特定の条件が満たされることを保証しません。マニュアルの内容は、製品のアップグレード及び修正により、その内容が予告なく変更されることがあり、内容上の誤りがないことを保証しません。

## Trademarks

Tmax®, Tmax WebtoB® and JEUS® are registered trademark of TmaxSoft Co., Ltd. Other products, titles or services may be registered trademarks of their respective companies.

Tmax®, Tmax WebtoB®, JEUS® は、TmaxSoft Co., Ltd.の登録商標です。その他、記載されている会社名、製品名などは、各社の商標または登録商標です。

## Open Source Software Notice

Some modules or files of this product are subject to the terms of the following licenses. : openssl-0.9.7.m, zlib-1.1.4, expat-2.0.0, net-snmp, DCE1.0, pthread, google-diff-match-patch, libevent, getopt

---

Detailed Information related to the license can be found in the following directory :  
\${INSTALL\_PATH}/license/oss\_licenses

この製品の一部ファイルまたはモジュールは、openssl-0.9.7.m、zlib-1.1.4、expat-2.0.0、net-snmp、DCE1.0、pthread、google-diff-match-patch、libevent、getoptライセンスを遵守します。

詳細情報については、製品ディレクトリーの\${INSTALL\_PATH}/license/oss\_licensesに記載されている事項を参照してください。

## 文書情報

文書名: Tmax リファレンスガイド

発行日: 2016年8月5日

ソフトウェアバージョン: Tmax v6.0

ガイドバージョン: v2.1.1

---



# 目次

このガイドについて .....	xv
<b>第1章 概要 .....</b>	<b>1</b>
1.1. コマンド .....	1
1.2. 関数 .....	2
1.2.1. サーバー/クライアント使用関数 .....	2
1.2.2. サーバー関数 .....	7
1.2.3. クライアント関数 .....	9
1.2.4. TCP/IPゲートウェイ関数 .....	10
1.2.5. TDL関数 .....	11
1.2.6. Windows関連関数 .....	12
1.2.7. その他の関数 .....	13
<b>第2章 コマンド .....</b>	<b>15</b>
2.1. cfl .....	15
2.2. fdlc .....	18
2.3. gst .....	21
2.4. hkcfll .....	23
2.5. idlc .....	23
2.6. mkcli .....	25
2.7. mkacl .....	26
2.8. mkgrp .....	27
2.9. mkpw .....	28
2.10. mksvr .....	30
2.11. racd .....	33
2.12. racdr .....	36
2.13. rcakill .....	39
2.14. rcastat .....	40
2.15. sdlc .....	41
2.16. svcrrpt .....	44
2.17. tdlclean .....	45
2.18. tdlinit .....	47
2.19. tdlrm .....	48
2.20. tdlrm .....	49
2.21. tdlseqno .....	49
2.22. tdlshm .....	50
2.23. tdlsync .....	52
2.24. tdltrace .....	53
2.25. tdlupdate .....	54
2.26. tencrypt .....	55
2.27. tadmin .....	57
2.28. tmapm .....	60

2.29.	tmaxlibver .....	61
2.30.	tmaxtrace .....	62
2.31.	tmboot .....	65
2.32.	tmd .....	71
2.33.	tmdown .....	73
2.34.	tmmbfgen .....	77
2.35.	tmsnmpd .....	78
2.36.	tperr .....	81
2.37.	twagent .....	83
2.38.	uncfl .....	84
2.39.	untmmbfgen .....	85
2.40.	xwsdlgen .....	85
<b>第3章 関数 .....</b>		<b>87</b>
3.1.	サーバー/クライアント関数 .....	87
3.1.1.	gettperrno .....	87
3.1.2.	tgsterror .....	88
3.1.3.	tmax_chk_conn .....	88
3.1.4.	tmax_gq_count .....	91
3.1.5.	tmax_gq_get .....	92
3.1.6.	tmax_gq_getkeylist .....	92
3.1.7.	tmax_gq_keygen .....	93
3.1.8.	tmax_gq_purge .....	94
3.1.9.	tmax_gq_put .....	95
3.1.10.	tmax_grid_create .....	96
3.1.11.	tmax_grid_create2 .....	98
3.1.12.	tmax_grid_destroy .....	100
3.1.13.	tmax_grid_set .....	101
3.1.14.	tmax_grid_get .....	103
3.1.15.	tmax_grid_is_exist .....	104
3.1.16.	tmax_grid_count .....	105
3.1.17.	tmax_grid_lock .....	106
3.1.18.	tmax_grid_unlock .....	107
3.1.19.	tmax_grid_set_watcher .....	108
3.1.20.	tmax_grid_wait_watcher .....	111
3.1.21.	tmax_grid_enqueue .....	112
3.1.22.	tmax_grid_dequeue .....	114
3.1.23.	tmax_grid_get_children .....	115
3.1.24.	tmax_grid_get_child_with_index .....	116
3.1.25.	tmax_grid_keylist_free .....	117
3.1.26.	tmax_get_sessionid .....	118
3.1.27.	tmax_keylist_count .....	118
3.1.28.	tmax_keylist_free .....	119

3.1.29.	tmax_keylist_getakey .....	120
3.1.30.	tmax_sq_count .....	121
3.1.31.	tmax_sq_get .....	121
3.1.32.	tmax_sq_getkeylist .....	122
3.1.33.	tmax_sq_keygen .....	123
3.1.34.	tmax_sq_purge .....	124
3.1.35.	tmax_sq_put .....	125
3.1.36.	tmaxlastsvc .....	126
3.1.37.	tmaxreadenv .....	127
3.1.38.	tp_sleep .....	128
3.1.39.	tp_usleep .....	129
3.1.40.	tpabort .....	131
3.1.41.	tpacall .....	132
3.1.42.	tpacallsvg .....	136
3.1.43.	tpalivechk .....	138
3.1.44.	tpalloc .....	140
3.1.45.	tpbegin .....	142
3.1.46.	tpbroadcast .....	144
3.1.47.	tpcall .....	146
3.1.48.	tpcallsvg .....	151
3.1.49.	tpcancel .....	153
3.1.50.	tpcommit .....	155
3.1.51.	tpconnect .....	156
3.1.52.	tpdeq .....	159
3.1.53.	tpdeq_ctl .....	162
3.1.54.	tpdiscon .....	167
3.1.55.	tpenq .....	169
3.1.56.	tpenq_ctl .....	172
3.1.57.	tperrordetail .....	176
3.1.58.	tpextsvinfo .....	177
3.1.59.	tpextsvcname .....	179
3.1.60.	tpfree .....	181
3.1.61.	tpget_timeout .....	182
3.1.62.	tpgetactivesvr .....	183
3.1.63.	tpgetcliaddr .....	186
3.1.64.	tpgetcliaddr_ipv6 .....	187
3.1.65.	tpgetctxt .....	189
3.1.66.	tpgetenv .....	192
3.1.67.	tpgetlev .....	193
3.1.68.	tpgetpeername .....	194
3.1.69.	tpgetrcahseqno .....	196
3.1.70.	tpgetrcainfo .....	197
3.1.71.	tpgetrply .....	197

3.1.72.	tpgetsockname .....	201
3.1.73.	tpgetsprlist .....	203
3.1.74.	tpgetsvglist .....	205
3.1.75.	tpgprio .....	206
3.1.76.	tpmcall .....	208
3.1.77.	tpmcallx .....	210
3.1.78.	tpnotify .....	213
3.1.79.	tppost .....	215
3.1.80.	tpputenv .....	217
3.1.81.	tpqstat .....	218
3.1.82.	tpqsvcstat .....	220
3.1.83.	tprealloc .....	221
3.1.84.	tprecv .....	223
3.1.85.	tpreissue .....	227
3.1.86.	tpremoteconnect .....	228
3.1.87.	tpscmt .....	229
3.1.88.	tpsend .....	230
3.1.89.	tpsetsvctimeout .....	234
3.1.90.	tpset_timeout .....	235
3.1.91.	tpsetfd .....	237
3.1.92.	tpsleeper .....	240
3.1.93.	tpspracall .....	242
3.1.94.	tpsprio .....	244
3.1.95.	tpsetctxt .....	246
3.1.96.	tpstrerror .....	249
3.1.97.	tpsubqname .....	250
3.1.98.	tpsubscribe .....	251
3.1.99.	tptypes .....	253
3.1.100.	tpunsubscribe .....	254
3.1.101.	tuxgetenv .....	255
3.1.102.	tuxputenv .....	256
3.1.103.	tuxreadenv .....	257
3.1.104.	tx_begin .....	259
3.1.105.	tx_commit .....	261
3.1.106.	tx_info .....	263
3.1.107.	tx_rollback .....	265
3.1.108.	tx_set_commit_return .....	268
3.1.109.	tx_set_transaction_control .....	270
3.1.110.	tx_set_transaction_timeout .....	273
3.1.111.	ulogsync .....	275
3.1.112.	userlog .....	276
3.1.113.	UserLog .....	278
3.2.	サーバー関数 .....	279



3.2.1.	_tmax_check_license .....	279
3.2.2.	_tmax_event_handler .....	280
3.2.3.	_tmax_main .....	282
3.2.4.	tmadmin .....	284
3.2.5.	tmax_get_db_passwd .....	290
3.2.6.	tmax_get_db_tnsname .....	292
3.2.7.	tmax_get_db_username .....	294
3.2.8.	tmax_get_svccnt .....	295
3.2.9.	tmax_get_svclist .....	296
3.2.10.	tmax_is_restarted .....	298
3.2.11.	tmax_is_xa .....	298
3.2.12.	tmax_my_svrinfo .....	299
3.2.13.	tmgetsmgid .....	301
3.2.14.	tmget_smtrclog .....	307
3.2.15.	tmget_smtrclog_count .....	310
3.2.16.	tpadvertise .....	311
3.2.17.	tpcancelctx .....	313
3.2.18.	tpchkclid .....	314
3.2.19.	tpclrfd .....	315
3.2.20.	tpclrfd_w .....	317
3.2.21.	tpforward .....	319
3.2.22.	tpgetclid .....	322
3.2.23.	tpgetctx .....	323
3.2.24.	tpgetdbsessionid .....	324
3.2.25.	tpgetmaxsvr .....	325
3.2.26.	tpgetmaxuser .....	326
3.2.27.	tpgetminsvr .....	327
3.2.28.	tpgetmynode .....	328
3.2.29.	tpgetmysvgno .....	329
3.2.30.	tpgetmysvrid .....	331
3.2.31.	tpgetnodename .....	332
3.2.32.	tpgetnodeno .....	333
3.2.33.	tpgetorgclh .....	334
3.2.34.	tpgetorgnode .....	335
3.2.35.	tpgetpeer_ipaddr .....	336
3.2.36.	tpgetsvcname .....	338
3.2.37.	tpgetsvrseqno .....	340
3.2.38.	tpissetfd .....	341
3.2.39.	tpissetfd_w .....	344
3.2.40.	tpprechk .....	344
3.2.41.	tpregcb .....	345
3.2.42.	tprelay .....	347
3.2.43.	tpresumetx .....	349

3.2.44.	tpreturn .....	350
3.2.45.	tpsavectx .....	354
3.2.46.	tpschedule .....	356
3.2.47.	tpsendtocli .....	358
3.2.48.	tpsetdbsessionid .....	360
3.2.49.	tpsuspendtx .....	361
3.2.50.	tpsvctimeout .....	363
3.2.51.	tpsvrdone .....	364
3.2.52.	tpsvrdown .....	365
3.2.53.	tpsvrinit .....	366
3.2.54.	tpsvrthrdone .....	368
3.2.55.	tpsvrthrinit .....	369
3.2.56.	tptsleep .....	371
3.2.57.	tpunadvertise .....	372
3.2.58.	tpunregcb .....	374
3.2.59.	tpuschedule .....	375
3.2.60.	tx_close .....	377
3.2.61.	tx_open .....	378
3.2.62.	TPADMNTOI .....	380
3.2.63.	tpregancb .....	381
3.2.64.	tpunregancb .....	383
3.3.	クライアント関数 .....	384
3.3.1.	gettpurcode .....	384
3.3.2.	tpchkunsol .....	385
3.3.3.	tpend .....	387
3.3.4.	tpgethostaddr .....	389
3.3.5.	tpgetunsol .....	390
3.3.6.	tpinit .....	393
3.3.7.	tpreset .....	394
3.3.8.	tpsetunsol .....	395
3.3.9.	tpsetunsol_flag .....	397
3.3.10.	tpstart .....	399
3.3.11.	tpterm .....	403
3.3.12.	tpbackup .....	404
3.3.13.	tpgetclid .....	405
3.4.	TCP/IPゲートウェイ関数 .....	406
3.4.1.	init_remote_info .....	406
3.4.2.	remote_connected .....	406
3.4.3.	remote_connected_ipv6 .....	407
3.4.4.	remote_closed .....	408
3.4.5.	allow_connection .....	409
3.4.6.	allow_connection_ipv6 .....	410
3.4.7.	get_msg_length .....	412

3.4.8.	get_msg_info .....	412
3.4.9.	get_channel_num .....	413
3.4.10.	put_msg_info .....	414
3.4.11.	put_msg_complete .....	414
3.4.12.	get_service_name .....	415
3.4.13.	prepare_shutdown .....	415
3.4.14.	set_service_timeout .....	415
3.4.15.	chk_end_msg .....	416
3.4.16.	inmsg_recovery .....	416
3.4.17.	outmsg_recovery .....	417
3.4.18.	get_extmsg_info .....	418
3.4.19.	put_extmsg_info .....	419
3.4.20.	set_ping_msg .....	419
3.4.21.	chk_pong_msg .....	420
3.4.22.	set_extping_msg .....	421
3.4.23.	chk_extpong_msg .....	423
3.4.24.	reset_ping_msg .....	424
3.4.25.	reset_extping_msg .....	424
3.4.26.	set_error_msg .....	426
3.4.27.	get_msg_security .....	427
3.4.28.	put_msg_security .....	428
3.5.	TDL関数 .....	429
3.5.1.	tdlcall .....	429
3.5.2.	tdlcall2 .....	430
3.5.3.	tdlcall2s .....	432
3.5.4.	tdlcall2v .....	433
3.5.5.	tdlcallva .....	434
3.5.6.	tdlcallva2 .....	437
3.5.7.	tdlcreate .....	438
3.5.8.	tdldestroy .....	440
3.5.9.	tdlend .....	442
3.5.10.	tdlerror .....	442
3.5.11.	tdlresume .....	443
3.5.12.	tdlstart .....	444
3.5.13.	tdlsuspend .....	444
3.5.14.	tdlgetseqno .....	445
3.5.15.	tdlclose .....	445
3.5.16.	tdlload .....	446
3.5.17.	tdlload2 .....	446
3.5.18.	tdlinit .....	447
3.5.19.	tdldone .....	447
3.5.20.	tdlfind .....	448
3.5.21.	tdlfind2 .....	449

3.5.22.	tdlstat .....	449
3.5.23.	tdlstat2 .....	450
3.6.	Windows関連関数 .....	451
3.6.1.	WinTmaxAcall .....	451
3.6.2.	WinTmaxAcall2 .....	455
3.6.3.	WinTmaxEnd .....	459
3.6.4.	WinTmaxSend .....	460
3.6.5.	WinTmaxSetContext .....	464
3.6.6.	WinTmaxStart .....	466
3.7.	セキュリティ関数 .....	467
3.7.1.	tmax_init_crypt .....	467
3.7.2.	tmax_fini_crypt .....	468
3.7.3.	tmax_accept_crypt .....	469
3.7.4.	tmax_wrap .....	469
3.7.5.	tmax_unwrap .....	470
3.7.6.	tmax_init_auth .....	471
3.7.7.	tmax_fini_auth .....	472
3.7.8.	tmax_chk_authen .....	472
3.7.9.	tmax_req_auth .....	473
3.7.10.	tmax_chk_author .....	473
3.7.11.	tmax_crypt_plugin_init .....	474
3.7.12.	tmax_crypt_plugin_fini .....	475
3.7.13.	tmax_auth_plugin_init .....	476
3.7.14.	tmax_tencrypt .....	476
3.8.	その他の関数 .....	478
3.8.1.	tlog_close .....	478
3.8.2.	tlog_find .....	479
3.8.3.	tlog_nodeno .....	482
3.8.4.	tlog_open .....	483
3.8.5.	Usiginit .....	485
3.8.6.	Usignal .....	486
3.8.7.	Uunixerr .....	488
3.8.8.	Uunix_err .....	488
3.8.9.	Ustrerror .....	489
3.8.10.	tmaxoserrno .....	489
<b>付録 A.</b>	<b>エラー別対応方法 .....</b>	<b>491</b>
<b>付録 B.</b>	<b>ヘッダー・ファイルの例 .....</b>	<b>497</b>
B.1.	<atmi.h> .....	497
B.2.	<tmaxapi.h> .....	503
<b>索引</b>	<b>.....</b>	<b>513</b>

## 図目次

[図 2.1]	マルチノード監視環境 .....	15
[図 2.2]	Tmax SNMPのOID .....	80
[図 3.1]	tpforward .....	320



# このガイドについて

## 対象読者

本書は、Tmax<sup>®</sup>(以下Tmax)を利用してプログラムを開発するユーザーを対象としており、Tmaxアプリケーションの開発に使用されるコマンドの概念、使用方法およびその例について記述しています。なお、クライアントとサーバーの接続、通信に使用される関数の使用方法とその例についても記述しています。

## 前提知識

本書は、Tmaxシステムについての全般的な理解と、Tmaxシステムが提供する各種機能および特性を習得するための基本ガイドです。

本書を理解するためには、以下の事項について熟知する必要があります。

- ミドルウェア(Middleware)およびUNIXシステムについて
- Tmaxの基本概念について
- Java、Cプログラミングについて

## 制限事項

本書で記述しているコマンドと関数は、4GLに関連した部分を含んでいません。本書に含まれていない関数については、該当モジュールのガイドを参照してください。実務での具体的な使用方法、管理および運用については、各製品のガイドを参照してください。

---

### 参考

Tmaxシステム開発についての基本的な内容は、『Tmax 運用ガイド』もしくは『Tmax アプリケーション開発ガイド』を参照してください。

---

## 本書の構成

本書は、計3つの章と付録で構成されています。

各章の主な内容は以下のとおりです。

- 第1章: 概要

本ガイドで扱うコマンドと関数の一覧を紹介します。

- 第2章: コマンド

Tmaxで使用するコマンドについて記述します。

- 第3章: 関数

Tmaxで使用する関数について記述します。

- 付録A. エラー別対応方法

Tmaxで発生するエラーの説明と対応方法について記述します。

- 付録B. ヘッダー・ファイルの例

Tmaxベースのプログラミングのために熟知する必要がある各モジュールのヘッダー・ファイルについて記述します。



## 表記上の規則

表記	意味
<AaBbCc123>	プログラム・ソースコードのファイル名、ディレクトリー
<Ctrl>+C	CtrlキーとCキーを同時に押す
[Button]	GUIのボタン、メニュー名
太字	強調
「」、『』（鍵カッコ）	関連文書、あるいはガイド内の他の章および節の表示
「入力項目」	画面UI上の入力項目
<ハイパーリンク>	メール・アカウント、Webサイト
>	メニューの実行順
+----	下位ディレクトリー/ファイル有り
----	下位ディレクトリー/ファイル無し
<div>参考</div>	参照/注意事項
[図 1.1]	図の名称
[表 1.1]	表の名称
AaBbCc123	コマンド、コマンド実行結果の画面出力、サンプル・コード
[ ]	オプション・パラメータ値
	選択パラメータ値

## システム要件

	要求事項
プラットフォーム	IBM AIX 5.x / 6.1 / 7.1
	HP-UX 11.xx
	SunOS 5.7~5.9 / SunOS 5.10 / SunOS 5.11
ハードウェア	1GB以上のハードディスク空き容量
	512MB以上のメモリー空き容量
データベース	Oracle 9~12
	Tibero 4~5
	DB2
	Informix

## 関連文書

ガイド	説明
Tmax メッセージリファレンスガイド	Tmax製品の使用時に発生する可能性のあるメッセージ(エラー・メッセージを含む)と、その対応方法について説明しています
Tmax FDLリファレンスガイド	TmaxFDL関数の定義とサンプル・プログラムを利用して、FDLが提供する機能を活用する方法について説明しています
Tmax 運用ガイド	Tmaxを利用するための環境設定ファイルとシステム運用方法について説明しています
Tmax プログラミングガイド(ダイナミックライブラリ)	TmaxのTDL(Tmax Dynamic Library)を使用してプログラムを開発するユーザー向けに、TDLを使用するための環境設定と提供されるAPIの使用方法について説明しています
Tmax アプリケーション開発ガイド	Tmaxアプリケーション・プログラムの開発で使用するAPIの概念と使用方法および例について説明しています
Tmax ゲートウェイガイド(Webサービス)	Tmaxサービスを変更せずにWebサービスで使用するために提供されるWebサービス・ゲートウェイについて説明しています
Tmax プログラミングガイド(SQ)	TmaxのSQ(Session Queue)の概念と使用方法について説明しています

# お問合せ先

## Korea

TmaxSoft Co., Ltd.  
45, Jeongjail-ro, Bundang-gu,  
Seongnam-si, Gyeonggi-do, 13613  
South Korea  
Tel: +82-31-8018-1000  
Fax: +82-31-8018-1115  
Email: [info@tmax.co.kr](mailto:info@tmax.co.kr)  
Web (Korean): <http://www.tmaxsoft.com>  
TechNet: <http://technet.tmaxsoft.com>

## USA

TmaxSoft Inc.  
101 North Wacker Drive, Suite 2014,  
Chicago, IL 60606  
U.S.A  
Tel: +1-312-525-8330  
Email: [info@tmaxsoft.com](mailto:info@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/us\\_en/home](http://www.tmaxsoft.com/us_en/home)

## Japan

TmaxSoft Japan Co., Ltd.  
5F Sanko Bldg, 3-12-16 Mita, Minato-Ku, Tokyo, 108-0073  
Japan  
Tel: +81-3-5765-2550  
Fax: +81-3-5765-2567  
Email: [info@tmaxsoft.co.jp](mailto:info@tmaxsoft.co.jp)  
Web (Japanese): <http://www.tmaxsoft.co.jp>

## China

Beijing TmaxSoft System Software Co., Ltd.  
Room103, No.2 Huizhong Building, Seven Street Shangdi,  
Haidian District, Beijing, 100085  
P.R.China  
Tel: +86-10-6298-8827  
Email: [info@tmaxsoft.com.cn](mailto:info@tmaxsoft.com.cn)  
Web (Chinese): [http://www.tmaxsoft.com/cn\\_en/home\\_cn\\_en](http://www.tmaxsoft.com/cn_en/home_cn_en)

## Brazil

Tmax Brasil Sistemas e Serviços Ltda.  
Av. Copacabana, 177, sala 32~35 Empresarial 18 do Fortel  
Alphaville Barueri, Sao Paulo, 06472-001  
Brazil  
Tel: +55-11-4191-3100  
Fax: +55(11) 4191-3705 (extension#112)  
Email: [info.bra@tmaxsoft.com](mailto:info.bra@tmaxsoft.com)  
Web (Portuguese): [http://www.tmaxsoft.com/br\\_en/home\\_br\\_en](http://www.tmaxsoft.com/br_en/home_br_en)

## Russia

Tmax Rus L.L.C.  
Leninsky prospekt, 113/1 (Park Place Moscow),  
Office 318e, Moscow, 117198  
Russia  
Tel: +7(495)970-01-35  
Email: [info.rus@tmaxsoft.com](mailto:info.rus@tmaxsoft.com)  
Web (Russian): [http://www.tmaxsoft.com/ru\\_ru/home\\_ru\\_ru](http://www.tmaxsoft.com/ru_ru/home_ru_ru)

## Singapore

Tmax Singapore Pte. Ltd.  
430 Lorong 6, Toa Payoh #10-02,  
OrangeTee Building, 319402  
Singapore  
Tel: +65-6259-7223  
Fax: +65-6258-7112  
Email: [info.sg@tmaxsoft.com](mailto:info.sg@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/sg\\_en/home\\_sg\\_en](http://www.tmaxsoft.com/sg_en/home_sg_en)

## United Kingdom

TmaxSoft UK Ltd.  
215 Knyvett House, Watermans Business Park,  
The Causeway, Staines TW18 3BAB  
United Kingdom  
Tel: +44-1784-895005  
Email: [info.uk@tmaxsoft.com](mailto:info.uk@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/gb\\_en/home\\_gb\\_en](http://www.tmaxsoft.com/gb_en/home_gb_en)

## Canada

TmaxSoft Canada, Inc.  
2425 Matheson Blvd East, 8th floor,  
Unit 824 Mississauga, ON, L4W 5K4  
Canada  
Tel: +1-905-361-2888  
Email: [info.canada@tmaxsoft.com](mailto:info.canada@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/ca\\_en/home\\_ca\\_en](http://www.tmaxsoft.com/ca_en/home_ca_en)

## Australia

TmaxSoft Proprietary Limited  
L32, 101 Miller Street, North Sydney 2060  
Australia  
Tel: +91-9845-330-704  
Email: [info.aus@tmaxsoft.com](mailto:info.aus@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/au\\_en/home\\_au\\_en](http://www.tmaxsoft.com/au_en/home_au_en)

## India

TmaxSoft Technologies Private Limited  
Sobha Alexander Plaza, 3rd Floor,  
16/2 Commissariat Road, Bangalore-560025  
India  
Tel: +91-9845-330-704  
Email: [info.india@tmaxsoft.com](mailto:info.india@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/in\\_en/home\\_in\\_en](http://www.tmaxsoft.com/in_en/home_in_en)

## Turkey

TmaxSoft Co., Ltd. Turkey Liaison Office  
Windowist Tower. Eski Buyukdere Cad. No:26,  
Maslak 34467 Istanbul  
Turkey  
Tel: +90-544-553-6045  
Email: [cslee@tmaxsoft.com](mailto:cslee@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/tr\\_en/home\\_tr\\_en](http://www.tmaxsoft.com/tr_en/home_tr_en)





# 第1章 概要

Tmaxシステムを使用するために提供するコマンドと関数について説明します。

## 1.1. コマンド

以下は、Tmaxで提供するコマンドの一覧です。

コマンド	説明
<a href="#">cfl</a>	テキスト形式のTmax環境ファイルをコンパイルし、tmconfig(バイナリTmax環境ファイル)を作成します
<a href="#">fdlc</a>	フィールド・キー表をコンパイルします
<a href="#">gst</a>	バイナリTmax環境ファイルを参照し、サービス表を作成します
<a href="#">hkcfli</a>	テキスト形式のホストリンク環境設定ファイルをコンパイルしてバイナリのホストリンク環境設定ファイル(hlinkcfg)を作成します
<a href="#">idlc</a>	RPCモジュールの定義ファイルを実際にプログラミング可能なソースに変換します
<a href="#">mkcli</a>	Tmaxクライアント・モジュールを作成します
<a href="#">mkacl</a>	ACL(Access Control List)を作成します
<a href="#">mkgrp</a>	ユーザー・グループを作成します
<a href="#">mkpw</a>	暗号を管理します
<a href="#">mksvr</a>	Tmaxサーバー・モジュールを作成します
<a href="#">racd</a>	マルチノードで分散された環境で中央集中管理します
<a href="#">racdr</a>	マルチノード、マルチドメインに分散された環境でファイルのコピー、起動を実行します
<a href="#">rcakill</a>	RCAの終了時、RCAが使用するリソースを削除するために使用します
<a href="#">rcastat</a>	RCAの設定内容を確認したり、RCAに接続したクライアント数などをモニタリングする際に使用します
<a href="#">sdlc</a>	構造体を定義したファイルをコンパイルします
<a href="#">svcrpt</a>	Tmaxシステムの運用時、サービス実行に関連したログ記録を分析して出力します
<a href="#">tdlclean</a>	runディレクトリーの旧バージョンのライブラリー・ファイルや不要なファイルを整理します
<a href="#">tdlinit</a>	TDL共有メモリーおよび動的モジュールの初期化を実行します
<a href="#">tdlnm</a>	指定したライブラリーに対して自動エクスポートされる関数の一覧を照会します

コマンド	説明
<a href="#">tdlrm</a>	TDLを今後使用しない場合、共有メモリーを完全に削除します
<a href="#">tdlseqno</a>	TDLのシーケンス番号を照会します
<a href="#">tdlshm</a>	TDL共有メモリー情報を照会します。あるいは、統計モニタリングの有効化の可否およびモジュールの有効化の可否を設定します
<a href="#">tdlsync</a>	TDL共有メモリーとバックアップ・ファイルの同期化を実行します
<a href="#">tdltrace</a>	TDLの環境情報と統計情報を照会します
<a href="#">tdlupdate</a>	指定した動的モジュールをアップデートします
<a href="#">tencrypt</a>	環境設定のOPENINFOセクションを暗号化します
<a href="#">tmadmin</a>	Tmaxシステム管理をします
<a href="#">tmapm</a>	シグアラーが使用できない場合や、シグアラーの使用が容易でない場合に、サービス・タイムアウトを設定して使用することができる別途のサーバーです
<a href="#">tmaxlibver</a>	Tmaxライブラリーのバージョン情報を照会します
<a href="#">tmboot</a>	Tmaxシステムの全体または一部分を実行します
<a href="#">tmd</a>	サーバー・プログラムをテストするために、クライアントをシミュレーションします
<a href="#">tmdown</a>	Tmaxシステム全体、あるいは一部分を終了させます
<a href="#">tmmbfgen</a>	サービス情報ファイル(text)をサービス情報バイナリ・ファイルで作成します
<a href="#">tmaxtrace</a>	アプリケーションに対してランタイム・トラッキングします
<a href="#">tmsnmpd</a>	標準SNMPプロトコルに基づいて、Tmaxの構成および性能情報を照会します
<a href="#">tperr</a>	Tmaxのエラー番号とエラータイプを利用し、エラーに関する詳細情報を照会します
<a href="#">twagent</a>	TmaxのWeb Agentと接続するデーモン・プロセスを起動します
<a href="#">uncfl</a>	テキスト形式のTmax環境ファイルをコンパイルして作成されたtmconfigを、テキスト形式の環境ファイルにします
<a href="#">untmmbfgen</a>	サービス情報バイナリ・ファイルをサービス情報ファイル(text)にします
<a href="#">xwsdlgen</a>	Webサービスのスペックのうち、明細書の役割をするWSDL文書を作成します

## 1.2. 関数

以下は、各用途別に使用される関数についての説明です。

### 1.2.1. サーバー/クライアント使用関数

関数	説明
<a href="#">gettperrno</a>	Tmaxシステムの呼び出し時に設定されたエラー・コード(errno)を返します

関数	説明
<a href="#">gq2sterror</a>	GQ2 APIを使用時に発生するgqerrnoに該当する番号のメッセージを出力します
<a href="#">tmax_chk_conn</a>	クライアントの接続状態をチェックします。tpstartの実行可否のチェック、ソケット状態のチェック、メッセージ転送によって接続状態をチェックする役割をします
<a href="#">tmax_get_sessionid</a>	現行セッションのIDを返します
<a href="#">tmax_gq_count</a>	GQに保存されたデータ数を返します
<a href="#">tmax_gq_get</a>	GQからデータを取得します。キーを指定した場合、該当キーのデータを取得します
<a href="#">tmax_gq_getkeylist</a>	GQのキー・リストを取得します
<a href="#">tmax_gq_keygen</a>	システム・キーを作成および取得します
<a href="#">tmax_gq_purge</a>	GQのデータを削除します
<a href="#">tmax_gq_put</a>	データをGQに保存します。キーとデータ値を渡します
<a href="#">tmax_grid_create</a>	キーを作成します
<a href="#">tmax_grid_create2</a>	キーを作成し、キーに値を設定します
<a href="#">tmax_grid_destroy</a>	キーを削除します
<a href="#">tmax_grid_set</a>	キーに値を設定します
<a href="#">tmax_grid_get</a>	値を取得し、当該キーの値を削除します
<a href="#">tmax_grid_is_exist</a>	キーが存在するかチェックします
<a href="#">tmax_grid_count</a>	キーの全体数を参照します
<a href="#">tmax_grid_lock</a>	キーの名前でロックを実行します
<a href="#">tmax_grid_unlock</a>	キーの名前でロックを解除します
<a href="#">tmax_grid_set_watcher</a>	当該キーのイベントが発生時に呼び出す関数を登録します
<a href="#">tmax_grid_wait_watcher</a>	イベントが発生するまでタイムアウトに指定した時間の間待機します
<a href="#">tmax_grid_enqueue</a>	キーの名前でデータを入力します
<a href="#">tmax_grid_dequeue</a>	tmax_grid_enqueue()により入力した値のうち最初に入力した値を参照します
<a href="#">tmax_grid_get_children</a>	キーの名前で子キーの名前のリストを参照します
<a href="#">tmax_grid_get_child_with_index</a>	子キーの情報を含むgrid_KEYLIST_Tハンドラーでnth番目のキー情報をgrid_KEYINFO_T構造体に保存します
<a href="#">tmax_grid_keylist_free</a>	子キーの情報を含むgrid_KEYLIST_Tハンドラーのリソースを解除します
<a href="#">tmax_keylist_count</a>	keylistハンドルからキー・リストの数を返します
<a href="#">tmax_keylist_free</a>	keylistハンドルのメモリーやその他のリソースを解除します
<a href="#">tmax_keylist_getakey</a>	keylistハンドルからn番目のキー情報を取得します
<a href="#">tmax_sq_count</a>	現行SQに保存されているデータ数を返します
<a href="#">tmax_sq_get</a>	データをセッション・キューに保存します

関数	説明
<a href="#">tmax_sq_getkeylist</a>	現行セッションのSQのキー・リストを取得します
<a href="#">tmax_sq_keygen</a>	システム・キーを作成および取得します
<a href="#">tmax_sq_purge</a>	SQのデータを削除します
<a href="#">tmax_sq_put</a>	サーバーのデータをSQに保存します
<a href="#">tmaxlastsvc</a>	最後に実行されたサービス名を照会します。エラーが発生したサービス名または最後にルーチンを実行したサービス名を返します
<a href="#">tmaxreadenv</a>	接続するシステムの情報を読み込み、環境変数に新規値を設定します
<a href="#">tp_sleep</a>	データの到着を秒単位で待機します
<a href="#">tp_usleep</a>	データの到着を1000000(百万)分の1秒単位で待機します
<a href="#">tpabort</a>	グローバル・トランザクションをロールバックします
<a href="#">tpacall</a>	非同期サービスの要求を送受信します
<a href="#">tpacallsvg</a>	COUSINIにまとめられたマルチサーバー・グループ環境で、特定サーバー・グループに属するサービスに非同期型の通信でサービス要求を送信します
<a href="#">tpalivechk</a>	クライアントの接続状態をチェックする関数であり、ソケットの状態をチェックする役割をします
<a href="#">tpalloc</a>	型付きバッファ(typed buffer)を割り当てます
<a href="#">tpbegin</a>	トランザクション時間の設定およびグローバル・トランザクションを開始します
<a href="#">tpbroadcast</a>	Tmaxシステムに登録されているクライアントに要求していないメッセージを送信します
<a href="#">tpcall</a>	同期型のサービス要求を送信します
<a href="#">tpcallsvg</a>	特定サーバー・グループに属しているサービスをサーバーとクライアントで呼び出します
<a href="#">tpcancel</a>	応答をキャンセルします
<a href="#">tpcommit</a>	グローバル・トランザクションをコミットします
<a href="#">tpconnect</a>	プログラムが対話型サービスのsvcと通信を接続します
<a href="#">tpdeq</a>	RQからデータをロードします
<a href="#">tpdeq_ctl</a>	トランザクションをサポートし、RQからデータをロードします
<a href="#">tpdiscon</a>	対話型通信の接続を終了します
<a href="#">tpenq</a>	RQにデータを保存します
<a href="#">tpenq_ctl</a>	トランザクションをサポートし、RQデータを保存します
<a href="#">tperrordetail</a>	サーバーとクライアントでTmaxシステムの呼び出し時に発生したエラーの詳細情報を取得する際に使用します
<a href="#">tpextsvcinfo</a>	tpdeq()を使用してRQからデータを読み込んだ場合、該当データについての詳細情報を提供します

関数	説明
<a href="#">tpextsvcname</a>	RQに保存されているデータのうち、サービス名を出力します
<a href="#">tpfree</a>	型付きバッファー(typed buffer)に割り当てられたメモリーを解除します
<a href="#">tpget_timeout</a>	ブロック・タイムアウト時間を返します
<a href="#">tpgetactivesvr</a>	現在アクティブなサーバーの一覧を照会します
<a href="#">tpgetcliaddr</a>	Tmaxシステムに接続しているクライアントのうち、clidに該当するクライアントのIPとポート番号を取得します
<a href="#">tpgetcliaddr_ipv6</a>	Tmaxシステムに接続されているクライアントのうち、clidに該当するクライアントのIPとポート番号を取得します。IPv6環境で使⽤します
<a href="#">tpgetctxt</a>	現行コンテキストを返します
<a href="#">tpgetenv</a>	nameという名前で登録されている環境変数の値を返します
<a href="#">tpgetlev</a>	トランザクション・モードの可否を確認します
<a href="#">tpgetpeername</a>	接続されている相手方のソケット・アドレスを取得します
<a href="#">tpgetrcahseqno</a>	tpgetsvrseqno APIと同様にRCAHプロセス番号を返します
<a href="#">tpgetrcainfo</a>	RCAHのスレッド情報を通知します
<a href="#">tpgetrply</a>	非同期的に要求したサービスに対する応答を受信します
<a href="#">tpgetsockname</a>	Tmaxシステム内部で使⽤されるソケット・アドレスを取得します
<a href="#">tpgetsprlist</a>	サーバー・プロセス単位で呼び出しを行うために、該当サービスが属しているサーバー・プロセスの索引を取得します
<a href="#">tpgetsvglist</a>	該当サービスが属しているサーバー・グループと、該当サービス・グループのCOUSINIに設定されているサーバー・グループについての情報を提供します
<a href="#">tpgprio</a>	要求を受けたサービスの優先順位を出力します
<a href="#">tpmcall</a>	COUSINIにまとめられたすべてのサーバー・グループ・サーバーのサービス呼び出します
<a href="#">tpmcallx</a>	既存のtpmcall()の拡張機能提供を目的とする関数です。既存とは異なり、COUSINサーバー・グループのサービスから応答をすべて受けるまで待機します
<a href="#">tpnotify</a>	サーバーで指定されているクライアントに非要求メッセージを送信します
<a href="#">tppost</a>	特定イベントを発生させ、メッセージを送信します
<a href="#">tpputenv</a>	環境変数値を再設定します
<a href="#">tpqstat</a>	RQに保存されているデータの統計を要求します
<a href="#">tpqsvcstat</a>	RQに保存されているデータのうち、指定したサービスの統計を要求します。現行RQに保存されているデータの統計を求めます
<a href="#">tprealloc</a>	RQに保存されているデータ件数を照会します
<a href="#">tprecv</a>	対話型通信を行う場合、メッセージを受信します

関数	説明
<a href="#">tpreissue</a>	該当RQのフェイル・キューにたまっている要求データを、再度リクエスト・キューに入れます
<a href="#">tpremoteconnect</a>	リモート・ホストとTCPで接続する関数です
<a href="#">tpscmt</a>	コミット方法を再設定します
<a href="#">tpsend</a>	対話型通信でメッセージを送信します
<a href="#">tpset_timeout</a>	ブロック・タイムアウト時間を設定します
<a href="#">tpsetctxt</a>	現行コンテキストを設定します
<a href="#">tpsetfd</a>	ソケットfdをUCSプロセスのスケジューラーに登録します
<a href="#">tpsetsvctimeout</a>	サーバーのサービス・タイムアウト時間を設定します
<a href="#">tpsleap</a>	データが到着するまで待機します
<a href="#">tpspracall</a>	tpgetsprlist()を使用して取得したサーバー・プロセスの索引のうち、特定プロセスにサービスを呼び出します
<a href="#">tpsprio</a>	サービス要求の優先順位を設定します
<a href="#">tpsterror</a>	エラー番号に該当するメッセージを出力します
<a href="#">tpsubqname</a>	サブ・キュー番号に該当するキューの名前を返します
<a href="#">tpsubscribe</a>	特定イベントのメッセージに対する要求を登録します
<a href="#">tptypes</a>	バッファのタイプおよびサブタイプについての情報を照会します
<a href="#">tpunsubscribe</a>	特定イベントのメッセージに対する登録を解除します
<a href="#">tuxgetenv</a>	環境変数値を返します
<a href="#">tuxputenv</a>	環境変数を適用します
<a href="#">tuxreadenv</a>	ファイルの環境変数を読み込みます
<a href="#">tx_begin</a>	グローバル・トランザクションを開始します
<a href="#">tx_commit</a>	グローバル・トランザクションをコミットします
<a href="#">tx_info</a>	グローバル・トランザクション情報を返します
<a href="#">tx_rollback</a>	グローバル・トランザクションをロールバックします
<a href="#">tx_set_commit_return</a>	commit_returnの特性を設定します
<a href="#">tx_set_transaction_control</a>	transaction_controlの特性をcontrol値に設定します
<a href="#">tx_set_transaction_timeout</a>	transaction_timeoutの特性をtimeout値に設定します
<a href="#">ulogsync</a>	メモリー・バッファ上のulogをファイルに保存します
<a href="#">userlog</a>	メモリー・バッファにulogを保存します
<a href="#">UserLog</a>	ulogを即時ファイルに保存します

## 1.2.2. サーバー関数

関数	説明
<a href="#">_tmax_check_license</a>	AnyLinkおよびOpenFrameで該当ライセンスが発行されたかどうかをチェックします
<a href="#">_tmax_event_handler</a>	SVRTYPEがEVT_SVRの場合、SLOGの発生時に呼び出されるコールバック関数です
<a href="#">_tmax_main</a>	ユーザー・プログラムにmain()が含まれている場合に使用します
<a href="#">tmadmin</a>	Tmaxシステム管理ツールのtmadminで照会できる統計情報を出力します
<a href="#">tmax_get_db_passwd</a>	現在Tmaxが接続しているデータベースのusernameのパスワードを照会します
<a href="#">tmax_get_db_tnsname</a>	現在Tmaxが接続しているデータベースのtnsnameを照会します
<a href="#">tmax_get_db_username</a>	現在Tmaxが接続しているデータベースのusernameを照会します
<a href="#">tmax_get_svccnt</a>	自身が属しているサーバーのサービス数を返します
<a href="#">tmax_get_svclist</a>	自身が属しているサーバーのサービス一覧を取得します
<a href="#">tmax_is_restarted</a>	Tmax APサーバー・ルーチン内で自身が属しているサーバー・プロセスが異常終了後に再起動したかどうかを判断できるようにします
<a href="#">tmax_is_xa</a>	現在自身が属しているサーバーがXAであるのか、あるいはNON-XAであるのかをチェックします
<a href="#">tmax_my_svrinfo</a>	サーバー・プロセスのシステム設定情報を取得します
<a href="#">tmgetsmgid</a>	SysMaster trace機能をサポートします。現在の自身のGIDを取得します
<a href="#">tmget_smtrclog</a>	ロギング・データを構造体バッファに保存します
<a href="#">tmget_smtrclog_count</a>	データ数を返します
<a href="#">tpadvertise</a>	サーバー・プロセスが提供するサービスをサーバーに登録します
<a href="#">tpcancelctx</a>	サーバー・ライブラリー内のCTX_T構造体の内容を削除します
<a href="#">tpchkclid</a>	UCSプロセス内でデータの送信前に、該当クライアントが正常に接続され、非要求データを受信できるかどうかを判別します
<a href="#">tpclrfd</a>	UCS方式プロセス内部のfdsetのソケットFDをオフするのに使用されます
<a href="#">tpclrfd_w</a>	UCS方式プロセス内部のwritable fdsetのソケットFDをオフするのに使用されます
<a href="#">tpforward</a>	サービス要求を他のサービス・ルーチンに渡します
<a href="#">tpgetclid</a>	Tmaxシステムに接続されているクライアントの番号を表示します
<a href="#">tpgetctx</a>	サーバー・ライブラリー内のCTX_T構造体の値をユーザー変数に保存します
<a href="#">tpgetdbsessionid</a>	RMセッション情報を取得します
<a href="#">tpgetmaxsvr</a>	サーバー・プロセスの最大実行数を出力します



関数	説明
<a href="#">tpgetmaxuser</a>	サーバー・プロセスが属しているノードの最大同時接続者数を出力します
<a href="#">tpgetminsvr</a>	サーバー・プロセスの最小実行数を出力します
<a href="#">tpgetmynode</a>	サーバーで特定のノード名とノード番号を取得します
<a href="#">tpgetmysvgno</a>	現在自身が属しているサーバー・グループの番号を出力します
<a href="#">tpgetmysvrid</a>	サーバー・プロセスIDを出力します
<a href="#">tpgetnodename</a>	サーバーで指定したノード名を取得します
<a href="#">tpgetnodeno</a>	サーバーでノード名をもつノードの番号を取得します
<a href="#">tpgetorgclh</a>	該当クライアントが現在接続されているCLH番号を返します
<a href="#">tpgetorgnode</a>	該当クライアントが接続されているノード番号(node number)を返します
<a href="#">tpgetpeer_ipaddr</a>	接続されている相手方のIPアドレスを出力します
<a href="#">tpgetsvcname</a>	サービス索引からサービス名を取得します
<a href="#">tpgetsvrseqno</a>	同じサーバー・プロセス間のサーバー・プロセスのシリアル番号を返します
<a href="#">tpissetfd</a>	サーバーにてUCSプロセスでソケットFDIにデータが到着したかどうかをチェックします
<a href="#">tpissetfd_w</a>	UCS方式サーバー・プロセスのFDSETをチェックし、パラメータ値として与えられたソケットFDIに送信するデータがあるかどうかを確認します
<a href="#">tpprechk</a>	RMの状態をチェックするためのユーザー・コールバック関数です。tpprechk()はTmaxシステムの接続前に呼び出されます
<a href="#">tpregcb</a>	サーバーでUCSの非同期型の要求に対する応答を受けるルーチンを設定します
<a href="#">tprelay</a>	UCS形式のサーバー・プロセスでサービスを要求したクライアントの情報を保存し、他のサービスを要求します
<a href="#">tpresumetx</a>	現在中止しているグローバル・トランザクションを再開します。中止しているグローバル・トランザクションを、 <a href="#">tpresumetx()</a> 、 <a href="#">tpsuspendxt()</a> を使用して再開できます
<a href="#">tpreturn</a>	サーバーのサービスを終了します
<a href="#">tpsavectx</a>	UCSプロセスで使用され、クライアントの情報を内部的に管理します
<a href="#">tpschedule</a>	UCSサーバー・プロセスでデータの到着を待機します
<a href="#">tpsendtocli</a>	指定されたクライアントに非要求メッセージを送信します
<a href="#">tpsetdbsessionid</a>	UCSプロセスでデータの到着を待機します
<a href="#">tpsuspendtx</a>	既存のグローバル・トランザクションを中止します
<a href="#">tpsvctimeout</a>	UCSサーバー・プロセスをダウンします
<a href="#">tpsvrdone</a>	Tmaxサーバー・プロセス終了ルーチンを呼び出します
<a href="#">tpsvrdown</a>	サーバー・プロセスをダウンします



関数	説明
<a href="#">tpsvrinit</a>	Tmaxサーバー・プロセスを初期化します
<a href="#">tpsvrthrdone</a>	マルチスレッドおよびマルチコンテキスト・サーバーで、サーバー・プロセスを終了する際、tpsvrdone関数を実行する前にサービス・スレッドを終了させる関数です
<a href="#">tpsvrthrinit</a>	マルチスレッドおよびマルチコンテキスト・サーバーでスレッドを初期化します
<a href="#">tptsleep</a>	TMMからサーバー・プロセス終了のイベントを待機します
<a href="#">tpunadvertise</a>	サーバー・プロセスが提供するサービスをサーバーで登録解除(unadvertise)します
<a href="#">tpunregcb</a>	サーバーにてUCSで非同期型の要求に対する応答を受けるルーチンを再設定します
<a href="#">tpuschedule</a>	UCSサーバー・プロセスでデータの到着を待機します
<a href="#">tx_close</a>	リソース・マネージャーとの接続を終了します
<a href="#">tx_open</a>	関連するリソース・マネージャーと接続します

### 1.2.3. クライアント関数

関数	説明
<a href="#">gettpurcode</a>	urcodeサービスに設定されているurcodeをクライアントに返します
<a href="#">tpchkunsol</a>	関数の呼び出し時にサーバーから非要求メッセージがある場合、該当メッセージを処理するための関数を呼び出します
<a href="#">tpend</a>	クライアントでTmaxシステムとの接続を解除します
<a href="#">tpgethostaddr</a>	TmaxクライアントとTmaxシステムの接続可否を確認します。あるいは、接続したTmaxシステムのIPとポート番号についての情報を取得します
<a href="#">tpgetunsol</a>	クライアントの要求なく、一方的に渡されたメッセージを処理します
<a href="#">tpinit</a>	Tmaxシステムに接続します
<a href="#">tpreset</a>	現在接続されているクライアントの接続を解除します
<a href="#">tpsetunsol</a>	非要求受信メッセージを処理するルーチンを設定します
<a href="#">tpsetunsol_flag</a>	非要求メッセージの受信フラグを変更します
<a href="#">tpstart</a>	Tmaxシステムに接続します
<a href="#">tpterm</a>	Tmaxシステムとの接続を解除します
<a href="#">tptobackup</a>	Tmaxバックアップ・システムに接続します

## 1.2.4. TCP/IPゲートウェイ関数

関数	説明
<a href="#">init_remote_info</a>	リモート・ノードと接続を確立する前に呼び出される関数です。接続情報などを保存するために共有メモリーを作成するロジックを実装することができます
<a href="#">remote_connected</a>	リモート・ノードと接続を確立したときに呼び出される関数です。ユーザーがリモート・ノードとの接続を確立した時に行う作業を実装できます
<a href="#">remote_connected_ipv6</a>	IPv6プロトコル環境でリモート・ノードと接続を確立したときに呼び出される関数です。ユーザーがリモート・ノードとの接続を確立した時に行う作業を実装できます
<a href="#">remote_closed</a>	リモート・ノードとの接続を終了したときに呼び出される関数です。ユーザーがリモート・ノードとの接続の終了時に行う作業を実装できます
<a href="#">allow_connection</a>	リモート・ノードと新しい接続を確立する前に呼び出されます。当該接続要求を許可するかどうかを戻り値で指定します
<a href="#">allow_connection_ipv6</a>	IPv6プロトコル環境でリモート・ノードと新しい接続を確立する前に呼び出されます。当該接続要求を許可するかどうかを戻り値で指定します
<a href="#">get_msg_length</a>	リモート・ノードから要求や応答が到着する場合に呼び出す関数であり、返された値の分実データを再び読み込みます
<a href="#">get_msg_info</a>	リモート・ノードから要求や応答が到着してデータを読み込んだ後、TCPGWライブラリーとcustom.cとのインターフェースの役割をするinfoを参照または加工します
<a href="#">get_channel_num</a>	Tmaxサービスやクライアントから要求したデータをリモート・ノードに送信時に、ユーザーがチャンネルを選択できるようにする関数です
<a href="#">put_msg_info</a>	リモート・ノードにメッセージを送信したい場合に呼び出します
<a href="#">put_msg_complete</a>	リモート・ノードにメッセージを送信した後呼び出される関数です。データが完全にリモート・ノードに送信されたことを通知します
<a href="#">get_service_name</a>	tpreply()またはtpacall()を実行するサービスの名前をエラーコードによって設定する関数です
<a href="#">prepare_shutdown</a>	TCPGWが終了する直前に呼び出される関数です。init_remote_info()関数で作成した共有メモリーを解除します
<a href="#">set_service_timeout</a>	サービス・タイムアウトが発生したときに呼び出される関数です。ユーザーがサービス・タイムアウトが発生したときに行う作業を実装できます
<a href="#">chk_end_msg</a>	リモート・ノードからデータを受信時にデータの末尾を示す特定の文字またはビット・ストリームが存在する場合、ユーザーが呼び出す関数です
<a href="#">inmsg_recovery</a>	リモート・ノードからの要求を処理したときに、エラーが発生する場合に呼び出す関数です
<a href="#">outmsg_recovery</a>	リモート・ノードに要求を送信した時にエラーが発生した場合に呼び出される関数です

関数	説明
<a href="#">get_extmsg_info</a>	get_msg_infoと一部の機能を除いては基本的に同じ機能を持ちます
<a href="#">put_extmsg_info</a>	put_msg_infoと一部の機能を除いては基本的に同じ機能を持ちます
<a href="#">set_ping_msg</a>	チャンネル障害の監視のために送信するメッセージの設定および周期、タイムアウトなどを設定するユーザー関数です
<a href="#">chk_pong_msg</a>	リモート・サーバーから受信したメッセージがチャンネル障害検知のための応答メッセージであるかどうかを確認する関数です
<a href="#">set_extping_msg</a>	サーバーのチャンネル障害を検知した場合に送信するメッセージを設定します
<a href="#">chk_extpong_msg</a>	チャンネル障害を検知した場合にサーバーから受信したメッセージを確認します
<a href="#">reset_ping_msg</a>	TCP/IPゲートウェイでPING(チャンネル障害の検知)メッセージの送信の可否と送信メッセージの再設定のために周期的に呼び出される関数です
<a href="#">reset_extping_msg</a>	reset_ping_msgと一部の機能を除いては基本的に同じ機能を持ちます
<a href="#">set_error_msg</a>	リモート・ノードとのトランザクションの途中にエラーが発生した場合に自動で呼び出される関数です
<a href="#">get_msg_security</a>	get_msg_infoが呼び出された後にデータを加工します
<a href="#">put_msg_security</a>	put_msg_infoが呼び出される前にデータを加工します

## 1.2.5. TDL関数

名前	説明
<a href="#">tdlcall</a>	最新バージョンの動的モジュール関数を呼び出します
<a href="#">tdlcall2</a>	最新バージョンの動的モジュール関数を呼び出します
<a href="#">tdlcall2s</a>	最新バージョンの動的モジュール関数を呼び出します
<a href="#">tdlcall2v</a>	最新バージョンの動的モジュール関数を呼び出します
<a href="#">tdlcallva</a>	最新バージョンの動的モジュール関数を呼び出します
<a href="#">tdlcallva2</a>	最新バージョンの動的モジュール関数を呼び出します
<a href="#">tdlcreate</a>	最新バージョンの動的モジュールで、クラス・ファクトリーを使用してクラス・インスタンスを作成する関数です。TDL環境ファイル(tdl.cfg)にVERSION=4と設定されている場合に使用できます
<a href="#">tdldestroy</a>	最新バージョンの動的モジュールで、クラス・ファクトリーを使用してクラス・インスタンスを破棄する関数です。TDL環境ファイル(tdl.cfg)にVERSION=4と設定されている場合に使用できます
<a href="#">tdlend</a>	明示的なバージョン整合性(Explicit Version Consistency)の維持を終了します

名前	説明
<a href="#">tdlerror</a>	tdlcall()に対するエラーが発生時、文字列形式を返します
<a href="#">tdlresume</a>	一時的に中止していたバージョン整合性の維持を再開します
<a href="#">tdlstart</a>	明示的なバージョン整合性(Explicit Version Consistency)の維持を開始します
<a href="#">tdlsuspend</a>	一時的にバージョン整合性の維持を中止します
<a href="#">tdlgetseqno</a>	グローバル・シーケンス番号を取得します
<a href="#">tdlclose</a>	当該モジュールの参照カウントを0に初期化するか、モジュールを直接メモリーから解除します
<a href="#">tdlload</a>	tdlcall()を行う前に、ハッシュ表の検索およびライブラリーのロードを実行し、ローカル・キャッシュに当該モジュールの情報を保存します
<a href="#">tdlload2</a>	tdlcall()を行う前に、ハッシュ表の検索およびライブラリーのロードを実行し、ローカル・キャッシュに当該モジュールの情報を保存します
<a href="#">tdlinit</a>	共有メモリーを初期設定します
<a href="#">tdldone</a>	共有メモリーを初期化します
<a href="#">tdlfind</a>	モジュールの索引を探します
<a href="#">tdlfind2</a>	モジュールの索引を探します
<a href="#">tdlstat</a>	TDLの統計情報を出力します
<a href="#">tdlstat2</a>	TDLの統計情報を出力します

## 1.2.6. Windows関連関数

名前	説明
<a href="#">WinTmaxAcall</a>	マルチスレッド環境での非同期サービスの送信要求です
<a href="#">WinTmaxAcall2</a>	マルチスレッド環境での非同期サービスの送信要求です
<a href="#">WinTmaxEnd</a>	Tmaxシステムと接続を終了します
<a href="#">WinTmaxSend</a>	データを送信します
<a href="#">WinTmaxSetContext</a>	ウィンドウ・ハンドルとメッセージ型を設定します
<a href="#">WinTmaxStart</a>	マルチウィンドウ環境でTmaxシステムとの接続に使用されます

## 1.2.7. その他の関数

名前	説明
<a href="#">tlog_close</a>	トランザクション・ログを分析するための関数の1つで、該当ログ・ファイルを閉じます
<a href="#">tlog_find</a>	該当トランザクション・ログ・ファイルでentryに指定された情報と一致するentryを検索します
<a href="#">tlog_nodeno</a>	XIDを使用してトランザクションが開始したノード番号を検索します
<a href="#">tlog_open</a>	トランザクション・ログを分析する関数であり、該当ログ・ファイルを開きます
<a href="#">Usiginit</a>	Tmaxシグナル・ハンドラーを初期化します
<a href="#">Usignal</a>	ユーザー・シグナル・ハンドリングに必要なマクロ設定に使用されます。Windowsシステム環境では使用されません
<a href="#">Uunixerr</a>	システム呼出しの途中にエラーが発生した場合、統合されたエラー番号が設定される変数です
<a href="#">Uunix_err</a>	ATMI API呼出しに失敗し、tperrnoがTPEOSで設定された場合、システム・エラーの種類をstderrに出力します
<a href="#">Ustrerror</a>	システムのエラー・コード(errno)に対する統合エラー・メッセージを返します



## 第2章 コマンド

本章では、Tmaxで使用可能なコマンドについて説明します。

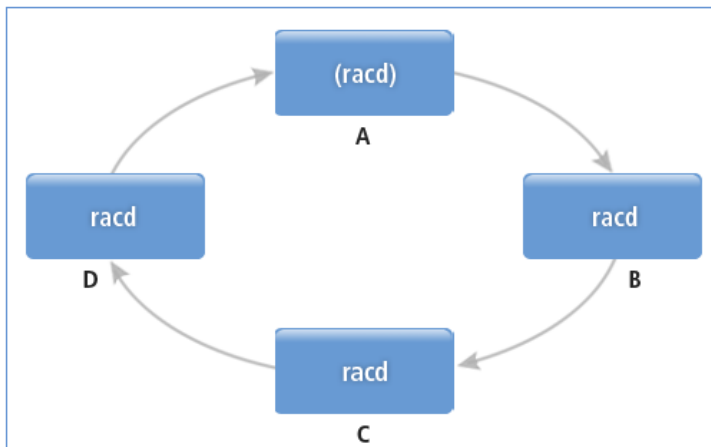
### 2.1. cfl

テキスト形式のTmax環境ファイルをコンパイルし、**tmconfig** (バイナリTmax環境ファイル)を作成するコマンドです。プログラムを動作させるためには、プログラムに合う環境ファイルの定義が必要です。環境ファイルの定義後、環境ファイルが正しく作成されたかどうかを検証します。cflは、テキスト形式で作成されたTmax環境ファイルをバイナリ・ファイルにコンパイルするコマンドです。

コンパイル中にエラーが検出された場合、バイナリ環境ファイルを作成せずにコンパイルを中断します。一方、エラーが検出されなかった場合は、バイナリ・ファイルに変換されたTmax環境ファイルが作成されます。コンパイルが完了したバイナリTmax環境ファイルは、**gst**、**tmboot**、**tmdown**コマンドなどで使用します。マルチノード環境で特定ノードの環境ファイルをコンパイルしたい場合、[ -n *node\_name* ]オプションを使用します。

以下は、マルチノード監視環境について説明しています。

[図 2.1] マルチノード監視環境



#### • 使用方法

```
$ cfl [-i テキストTmax環境ファイル名] [-o バイナリTmax環境ファイル名]  
      [-h] [-V] [-n node_name] [-A] [-v num] [-I] [-r] [-x]
```

項目	説明
[ -i テキストTmax環境ファイル名 ]	コンパイル対象となる元の設定ファイルであるテキスト形式のTmax環境ファイル名を設定します。ディレクトリーはユーザーが指定できます。指定されていない場合、環境設定ディレクトリーは基本的に <code>\${TMAXDIR}/config</code> です。元の設定ファイルが見つからなかった場合、警告メッセージを出力します
[ -o バイナリTmax環境ファイル名 ]	コンパイル結果であるバイナリTmax環境ファイル名を設定します。パスと一緒に指定でき、パスが指定されていない場合、 <code>\${TMAXDIR}/config</code> にバイナリTmax環境ファイルが生成されます。オプションが省略された場合、ファイル名は <code>tmconfig</code> で生成されます
[ -h ]	コマンドのヘルプ・オプションです
[ -V ]	実行ファイルのバージョンを確認できます
[ -n node_name ]	マルチノード環境で特定ノードの環境ファイルをコンパイルしたい場合に使用します。マルチノード環境で他のノードを管理するには <code>racd</code> を設定する必要があります
[ -A ]	<p>サービス・アクセス制御(第3段階セキュリティ)を使用する場合にのみ有効なオプションです。</p> <p>1つのドメイン内にあるACLサービスに対するアクセス権限は、すべてのノードで同様に適用される必要があります。そのため、現行ノードの<code>\${TMAXDIR}/config/group、acl、user</code>ファイルを、同じドメインに属している他のノードの<code>\${TMAXDIR}/config</code>に配布するためのオプションです。オプションを使用して環境ファイルをコンパイルする場合、<code>mkgrp、mkpw、mkacl</code>コマンドを使用して、あらかじめ<code>group、acl、user</code>ファイルが生成されている必要があります。</p> <p>生成されたファイルは、1つのドメイン内にあるすべてのノードが共通して使用するため、ファイルを作成する場合はドメインに属しているノードのすべてのユーザーを考慮する必要があります。group_name、group_id、user_name、user_idは、1つのドメインで一意である必要があります。passwdファイルはコピーできないため、ノード別に作成するか、別途コピーする必要があります</p>
[ -v num ]	<p>numでは、0と1を指定できます(デフォルト値: 1)</p> <ul style="list-style-type: none"> <li>0: マルチノードを構成しているノードがすべて同じタイプのマシンで、管理者がノード別に環境ファイルを管理する場合には、0を設定します。コンパイルされたバイナリ・ファイルは、それぞれのマシンにコピーして管理します</li> <li>1: <code>racd</code>を使用して環境ファイルが自動的にマシンからコンパイルされます(デフォルト値)</li> </ul>
[ -I ]	<code>cf</code> を実行時、環境ファイルのDOMAINセクションのSHMKEY項目に設定した値を現在使用している場合は、該当値のUIDを比較し、異なる場合は、エラー・メッセージが発生します。該当オプションは、共有メモリー値の現在の使用可否およびUID同一可否をチェックしない場合に使用します



項目	説明
[-r]	<p>cflの実行段階で、ulimit -nで照会時に出力される現行システムの、使用可能な最大FD数をあらかじめチェックし、ユーザーに通知します。CLH1個あたり開けられる最大FD数をあらかじめ計算してチェックします。</p> <p>Tmaxシステムで使用するFD値が、システムで使用可能なFDより大きく設定されている場合、次のようなエラーが発生します</p> <pre>(E) CFL9990 Current Tmax configuration contains more servers or nodes than current system can support[CFL5056]</pre>
[-a Tmax環境ファイル名]	<p>動的に追加するTmax環境ファイル名を指定します。</p> <p>サーバーに動的に追加する場合、cflを利用してバイナリ環境ファイルを作成時には、[-a]オプションを必ず使用する必要があります。このオプションを使用せずに動的に追加した場合には、(E) TMAX00157エラーが発生します。詳細については、『Tmax 運用ガイド』の「5.5.6. cfgadd(ca)」を参照してください</p>
[-Z]	<p>現在cflでMAXSACALL/MAXCACALLが1024以下に制限されていますが、この制限を解除するオプションです</p> <p>例)</p> <pre>- cfl -Z -i sample.m</pre>
[-x]	<p>SERVERセクションのTARGET項目、CLOPT項目の -xオプションを使用する場合、実際に存在するサーバーまたはサービスの名前であることをチェックします。</p> <p>– SERVERセクションのTARGET設定に環境ファイルで宣言されていないサーバー名が設定された場合、以下のようなエラー・メッセージを出力します</p> <pre>line %d(サーバーが宣言された行): invalid target server name %s(TARGET項目に設定したサーバー名)</pre> <p>– SERVICEセクションのCLOPT設定に環境ファイルで宣言されていないサービス名で -xオプションが設定された場合、以下のようなエラー・メッセージを出力します</p> <pre>line %d(サーバーが宣言された行): invalid service name %s(-xで設定したサービス名)</pre> <p>行番号はTARGET項目やCLOPT項目が設定された行ではなく、サーバーが宣言された行の番号が出力されます</p> <p>例)</p>

項目	説明
	- cfl -x -i sample.m

- 適用環境

Tmaxシステムがインストールされている運用システム環境で使用できます。

- 例

- 以下は、/user1/tmax/tempディレクトリーにある<basic>Tmax環境ファイル(テキスト形式)をコンパイルし、現行ディレクトリーにデフォルト値で<tmconfig>というバイナリTmax環境ファイルを作成する例です。

```
$ cfl -i /user1/tmax/temp/basic
```

- 以下は、ディレクトリーの<ex\_config>というテキスト形式のTmax環境ファイルをコンパイルし、/user1/tmax/binディレクトリーに<tmconfig>という名前でバイナリTmax環境ファイルを作成する例です。

```
$ cfl -i ex_config -o /user1/tmax/bin/tmconfig
```

---

## 参考

gst、tmboot、tmdownコマンドについての詳細は、それぞれ[「2.3. gst」](#)、[「2.31. tmboot」](#)、[「2.33. tmdown」](#)を参照してください。

---

## 2.2. fdlc

フィールド・キー表をコンパイルするコマンドです。フィールド・キー方式は、サーバーとクライアント間のデータ通信で構造体を転送する方法のように全体構造体の項目を渡さず、必要な項目のみ渡します。項目を渡すには、それぞれの項目を区別できる一意キーが必要です。fdlcは、テキスト形式で定義されているフィールド・キー表をコンパイルし、フィールド・キーを作成するコマンドです。

- 使用方法

```
$ fdlc {-a|c|d|u} [-f] [-h ヘッダー・ファイル名] {-i フィールド・キー表のファイル名}
        [-jc|ji] [-o 結果名] [-p パッケージ名] [-x] [-r 出力ファイル名] [-V]
```

項目	説明
{-a}	生成されたバイナリ形式のファイルに、テキストで作成されたフィールド・キー表をコンパイルし、必要な部分を追加します。

項目	説明
	[ <b>-f</b> ]オプションを使用して対象ファイルを指定します。デフォルト・ファイルは <b>&lt;tmx.fdl&gt;</b> です。重複するフィールドの場合、新しい値で代替されます
{ <b>-c</b> }	テキストで作成されたフィールド・キー表をコンパイルし、バイナリ形式のファイルを作成します。すでにバイナリ・ファイルが存在する場合、新規の内容で代替されます(デフォルト値)
{ <b>-d</b> }	生成されたバイナリ形式のファイルに、テキストで作成されたフィールド・キー表をコンパイルし、必要な部分を削除します。[ <b>-f</b> ]オプションを使用して対象ファイルを指定します。デフォルト・ファイルは <b>&lt;tmx.fdl&gt;</b> です
{ <b>-u</b> }	生成されたバイナリ形式のファイルに、テキストで作成されたフィールド・キー表をコンパイルし、必要な部分を修正または追加します。[ <b>-f</b> ]オプションを使用して対象ファイルを指定します。デフォルト・ファイルは <b>&lt;tmx.fdl&gt;</b> です
[ <b>-f</b> ]	テキストで作成されたフィールド・キー表をコンパイルし、必要な部分を追加、修正、および削除する場合、対象ファイルを指定します
[ <b>-h</b> ヘッダー・ファイル名]	ヘッダー・ファイル名を別名に変更する場合に使用します。[ <b>-h</b> ]オプション値をコンパイルした場合、FDLヘッダー・ファイル名は <b>&lt;フィールド・キー表名_fdl.h&gt;</b> です
{ <b>-i</b> フィールド・キー表のファイル名 }	クライアントプログラムとサーバープログラムで使用されるフィールド・キー表を定義したファイルを指定します。必須オプションで、パスと一緒に指定できます
[ <b>-jiljc</b> ]	WebTで使用するオプションで、生成されるフィールド定義クラスの形式を指定します – <b>ji</b> : インターフェース形式のフィールド定義クラスファイルを作成します – <b>jc</b> : クラス形式のフィールド・キー定義のJavaファイルを作成します
[ <b>-o</b> 結果名 ]	コンパイル結果を別名に変更する場合に使用します。[ <b>-o</b> ]オプションなしでコンパイルした場合、fdlファイル名は <b>&lt;tmx.fdl&gt;</b> です
[ <b>-p</b> パッケージ名 ]	WebTで使用するオプションで、生成されたフィールド定義クラスのパッケージ名を与えられた値に設定します
[ <b>-x</b> ]	必ず[ <b>-a</b> ]オプションと一緒に使用します。該当オプションが使用された場合、重複するフィールドについては元の値を維持します
[ <b>-r</b> 出力ファイル名]	fdlファイルをテキストの出力ファイル名ファイルに戻すオプションです。  「 <b>-f</b> fdl名」オプションと一緒に使用される必要があります。  「 <b>fdlc -f</b> fdl名 <b>-r</b> 出力ファイル名」コマンドでfdlファイルをテキスト・ファイルにデコードすることができます
[ <b>-V</b> ]	実行ファイルのバージョンを確認できます

- 適用環境

UNIX運用システムとMS-DOS運用システムでサポートされます。

- 例

– 以下は、[-jclji]オプションを設定して作成したJavaファイルの例です。

「**number**」値に使用できる値の範囲は0 ~ 2<sup>24</sup>-1(0~16777215)です。

<demo.f>

```
#demo.f
#name    number  type    flags    comments
INPUT    101     string  -        -
OUTPUT    102     string  -        -

$> fdlc -c -i demo.f -ji -jc webtdemo
```

– 以下は、生成された<demo\_fdl.java>の内容です。

```
package webtdemo;
public interface demo_fdl {
    public int INPUT = (469762149);    /* number: 101 type: string */
    public int OUTPUT = (469762150); /* number: 102 type: string */
}
```

– 以下は、現在ディレクトリにある「demo.f」フィールド・キー表のファイルをfdlcコマンドでコンパイルする例です。コンパイル後、<tmax.fdl>と<demo\_fdl.h>ファイルが生成されます。<demo\_fdl.h>は、フィールド・キーが定義されているヘッダー・ファイルです。

```
$ fdlc -c -i demo.f
```

## フィールド・キー

サーバー/クライアントデータ通信でフィールド・キー・バッファを使用してデータを送信する場合、FDL方式を使用します。FDL(Field Definition Language)は、フィールド・キー・バッファに2つのペアで構成されている識別子とその識別子に対応するデータ値を保存し、互いに異なるプロセス間にデータを交換する方式です。フィールド・キー・バッファでそれぞれの識別子は、データ型と重複していない識別番号の組み合わせで作成されます。この識別子が**フィールド・キー**です。

アプリケーションではフィールド・キーの代わりに、プログラムに対する判読性を高めるためのフィールド名を使用します。それぞれのフィールド名は、実行中にフィールド・キー・バイナリ・ファイルを参照し、フィールド・キーに転換されてバッファに保存されます。

FDL方式は、該当フィールドを使用するプログラムの変更がなくてもフィールドの型(type)と長さを変更することができ、各フィールドの長さを可変的に使用できます。データ型は、一般C言語で提供されているChar、Short、Integer、Long、Float、Double、String、Carrayなどの型をサポートし、フィールド名は16文字まで使用可能です。

fdlcコンパイラは、テキスト形式で定義されたフィールド・キー定義ファイルをコンパイルし、ユーザープログラムで参照されているフィールド・キー・バイナリ・ファイルを作成します。使用されるフィールド・キー定義ファイルはテキストファイルで、表ファイル名は<f>形式のファイルです。デフォルト結果は、<tmax.fdl>と<フィールド・キー定義ファイル名\_fdl.h>です。サーバープログラムは、生成されたヘッダー・ファイルを参照してコンパイルされ、フィールドのデータの保存や読み込みができます。クライアントは、fdlcコマンドで生成されたファイルをFDLFILEという環境変数に必ず登録します。

フィールド・キー・バッファの長所は、データの独立性です。構造体バッファの場合、使用されていないフィールドが存在するとしても、構造体全体を転送するしかありません。しかし、フィールド・キー・バッファの場合は、必要なフィールドのみを選択して転送できます。また、開発の生産性向上のため、多様なフィールド・キー操作関数を提供しています。

注意事項：fdl number値が16777216以上の場合は許容しません。

---

#### 参考

フィールド・キー操作関数の詳細内容は『Tmax FDL リファレンスガイド』を参照してください。

---

## 2.3. gst

バイナリTmax環境ファイルを参照し、サービス表の作成に使用されるコマンドです。gstコマンドは、cflコマンドによって作成されたバイナリTmax環境ファイルのSERVERセクションとSERVICEセクションを参照し、各サーバー別にサービス表を作成します。この表はサーバー・プロセスで提供するサービス一覧で、サーバー・プログラムと共に作成し、プログラムがコンパイルされるときに一緒にコンパイルされます。また、サーバー・プロセスが動作する際、サービス位置の検索に使用されます。

gstコマンドの結果ファイルは、指定されたTMAXDIRディレクトリー下位のsvctディレクトリーに<サーバー名\_svctab.c>で生成されます。TMAXDIRは、source configファイルのNODEセクションを参照します。サーバー名はSERVERセクションに登録されている名前です。各ファイルの内容は、SERVICEセクションに登録されている該当サーバーが提供するサービス名です。

Tmax環境ファイルに登録されているサービスは、該当サーバーのサービス表にも必ず登録します。Tmax環境ファイルのSERVERセクションやSERVICEセクションのサーバー名、サービス名、サービスのSVRNAMEなどが変更された場合、gstコマンドを使用してサービス表とサーバー・プログラムをコンパイルする必要があります。

#### ● 使用方法

```
$ gst [-f バイナリTmaxファイル名] -v server_name | -h | -n node_name | -V]
```

項目	説明
[ -f バイナリTmax環境ファイル名 ]	参照するバイナリTmax環境ファイル名を設定します。  cflコマンドの結果で、tmbootコマンドとtmdownコマンドでも参照されるファイルです。パスと一緒に指定できます。省力した場合は、デフォルト値として <b>\${TMAXDIR}/config</b> に指定されたディレクトリー下位のconfigディレクトリーで <b>tmconfig</b> を参照します
[ -v server_name ]	サーバー名に該当するサービス表を設定します
[ -h ]	コマンドのヘルプ・オプションです
[ -n node_name ]	マルチノード環境で他ノードのサーバー・プロセスのサービス表を現行ノードの <b>\${TMAXDIR}/svct</b> に作成するようにします
[ -V ]	実行ファイルのバージョンを確認できます

- 適用環境

Tmaxシステムがインストールされた運用システム環境で使用できます。

- 例

以下は、/user1/park/tmax/binディレクトリーの<exconfig>環境ファイルを参照し、サーバー別にサービス表を作成する例です。

```
$ gst -f /user1/park/tmax/bin/exconfig
```

たとえば、環境ファイルを以下のページのように登録した場合、/user1/park/tmax/svctディレクトリーに<svr1\_svctab.c>と<svr2\_svctab.c>というサービス表が生成されます。

```
...
*SERVER
svr1      SVGNAME = svg1
svr2      SVGNAME = svg1
*SERVICE
svc1      SVRNAME = svr1
svc2      SVRNAME = svr2
```

---

## 参考

cfl、tmboot、tmdownコマンドについての詳細は、それぞれ[「2.1. cfl」](#)、[「2.31. tmboot」](#)、[「2.33. tmdown」](#)を参照してください。

---

## 2.4. hkcfl

**hkcfl**は、テキスト形式のホストリンク環境設定ファイルをコンパイルして、バイナリ・ホストリンク環境設定ファイル(hlinkcfg)を作成します。コンパイル中にエラーが発生すると、コンパイルを中断します。コンパイルが正常に完了すると、バイナリ環境設定ファイルが生成されます。

- 使用方法

```
hkcfl [-h] [-p] [-o バイナリ・ホストリンク環境設定ファイル名] -i テキスト・ホストリンク環境設定ファイル名
      -c コメント処理文字
```

項目	説明
[-h]	コマンドのヘルプ・オプションです
[-p]	ホストリンク環境設定ファイルのコンパイル実行内容を画面に出力します
[バイナリ・ホストリンク環境設定ファイル名]	コンパイル結果生成されるバイナリ・ホストリンク環境設定ファイルの名前を指定します。パスと一緒に指定することができます。パスを指定しない場合、\${TMAXDIR}/configに結果ファイルが生成されます(デフォルト値: 'hlinkcfg')
-i テキスト・ホストリンク環境設定ファイル名	テキスト形式のホストリンク環境設定ファイル名を指定します。必須オプションであり、ディレクトリーを指定しない場合、\${TMAXDIR}/configに設定されます。パスと一緒に指定することができます。ソース・ファイルが見つからない場合、警告メッセージが出力されます
-c コメント処理文字	テキスト形式の環境設定ファイルのコメント文字を変更します。デフォルトでは「#」文字がコメント文字として使用されます

## 2.5. idlc

Enteraの変換時に、定義ファイルを実際にプログラミングが可能なソース(stub)に変換するユーティリティーです。

以下は、idlcで使用可能なオプションとその説明です。

- 使用方法

```
idlc -inf インターフェース名
      -sql SQL インターフェース名
      -dbtype ora | tbr | syb
      -clang c | delphi | vbasic | pbuilder | java | delphi6
      -xa
      -header ファイル名
      -sstub ファイル名
      -cstub ファイル名
```

項目	説明
-inf インターフェース名	DAサーバーを使用する場合のインターフェース名を指定します
-sql SQL インターフェース名	DAサーバーとTPサーバーを使用する場合のインターフェース・ファイル名を指定します
-dbtype	DAサーバーを使用する場合のDBの種類を選択します。RPCがサポートするOSとDBの種類は『Tmax プログラミングガイド』の「1.3. 制約事項」を参照してください。  <div> <div>– ora : Oracle</div> <div>– syb : Sybase</div> <div>– tbr : Tiberio</div> </div>
-clang	作成するクライアントのソースStubファイルの言語を選択します  <div> <div>– c</div> <div>– delphi</div> <div>– vbasic</div> <div>– pbuilder</div> <div>– java</div> <div>– delphi6</div> </div>
-xa	XAサーバーで作成する場合に指定します
-headerファイル名	作成するヘッダー・ファイル名を指定します
-sstubファイル名	作成するサーバーStubファイル名を指定します
-cstubファイル名	作成するクライアントStubファイル名を指定します

#### ● 例

以下は、サーバーの種類別の使用例です。

#### – DAサーバー

Oracleを使用し、dstest.sqlを業務に使用し、クライアントはC言語を使用している場合、以下のようにオプションを設定します。

```
idlc -inf dstest
      -sql dstest.sql
      -dbtype ora
      -clang c
```



## – TSサーバー

- 「.def」ファイルを使用する場合

「.def」ファイルを使用する場合、以下のようにオプションを設定します。

```
idlc -xa
      -clang c ttest.def
```

- DAサーバーを使用する場合

TPサーバーで呼び出すDAサーバーを使用する場合、以下のようにオプションを設定します。

```
idlc -inf db1
      -sql db1.sql
      -dbtype ora
      -clang c
      -xa
      -tps
```

## – ファンクション・サーバー

ファンクション・サーバーは、TPサーバーと同様にオプションを設定して実行します。

```
idlc -xa
      -clang c fstest.def
```

## 2.6. mkcli

Tmaxクライアント・モジュールを作成するコマンドです。パラメータは最大1024文字まで使用できます。このコマンドを実行すれば、ccコンパイラによってクライアント・モジュールが生成されます。

- 使用方法

```
$ mkcli {-o outfile} {-f firstfiles} [-v] [-l lastfiles] [(-32)|-64] [-V]
      [-u UserCompileOption]
```

項目	説明
{-o outfile}	クライアント・モジュール名をoutfileに指定します
{-f firstfiles}	ユーザー定義ファイルを指定すると、Tmaxが提供するクライアント・ライブラリーより先にリンクされ、コンパイルされます。一般的に、ユーザー定義ファイルとしてクライアント・プログラムを指定します
[-v]	verboseモードでコンパイルされ、コンパイルするプロセスがすべてコンソール・ウィンドウに出力されます

項目	説明
[ -l <i>lastfiles</i> ]	ユーザー定義ファイルを指定すると、Tmaxが提供するクライアント・ライブラリーより後にリンクされ、コンパイルされます
[ (-32)   -64 ]	作成するクライアント・モジュールが32Bitと64Bitのどちらであるかを指定します – [-32]: Tmaxライブラリー・パスが <code>\${TMAXDIR}/lib</code> として設定されます(デフォルト値) – [-64]: Tmaxライブラリー・パスが <code>\${TMAXDIR}/lib64</code> として設定されます
[ -V ]	実行ファイルのバージョンを確認できます
[ -u <i>UserCompileOption</i> ]	mkcliコマンド内部で設定したプラットフォームに適合した基本コンパイル・オプションを除去し、ユーザーが指定したコンパイル・オプションで代替します。ライブラリーおよびインクルード(include)関連オプションは適用されないのに注意が必要です。  2つ以上のオプションを指定する必要がある場合、以下のように設定します  <pre>-u opt1 opt2</pre>

- 適用環境

Tmaxがインストールされている運用システム環境で使用できます。

## 2.7. mkacl

ACL(access control list)を作成するコマンドです。サービス・アクセス権限制御(第3段階セキュリティ)はユーザー・グループ別に行われます。サービスは、アクセス権限が許容されているグループに該当するユーザーのみアクセスが可能です。したがって、該当機能を使用するには、**group**ファイルを作成する必要があり、該当グループに属しているユーザー・ファイルが必要です。また、サービス別にアクセス可能なユーザー・グループを指定する**acl**ファイルが必要です。

aclファイルを作成するには、mkacclコマンドを使用します。

- 使用方法

```
$ mkaccl [-a] [-d] [-G group_name] [-t type] [-s service_name] [-h] [-V]
```

項目	説明
[ -a ]	既存のACLファイルに新しいACLサービスを追加するためのオプションです。ACLファイルが存在する場合、ファイルの一番最後に新規サービスが追加され、存在しない場合、新規ACLファイルの作成後に追加されます
[ -d ]	生成されたACLサービスを削除するためのオプションで、[-G]、[-s]、[-t]オプションと一緒に指定する必要があります。ACLファイルから[-s]オプションで指定したACL

項目	説明
	サービスを削除します。[-G]オプションは、該当サービスと一致する必要があります
[-G group_name]	[-s]で指定したサービスに対して、アクセスが許容されているグループ名を指定します。必ず <code>\$(TMAXDIR)/config/group</code> に指定されている名前を使用します
[-t type]	ACLのタイプを指定します。現在はSERVICEのみサポートします
[-s service_name]	ACLを適用するサービス名を指定します。[-G]オプションで指定されている1つのサーバー・グループに属しているユーザーのみアクセス可能です
[-h]	コマンドのヘルプ・オプションです
[-V]	コマンドをサポートするTmaxバージョン情報を表示します

- 結果

`$(TMAXDIR)/config/acl`ファイルが生成されます。その形式と内容は以下のとおりです。

```
service_name:type:group_id
```

```
TOUPPER1:SERVICE:1
TOUPPER5:SERVICE:5
TOUPPER7:SERVICE:7
TOUPPER9:SERVICE:9
```

---

## 参考

マルチノードで使用する場合、[「2.1. cfl」](#)の[-A]オプションの説明を参照してください。

---

## 2.8. mkgrp

ユーザー・グループを作成するコマンドです。サービス・アクセス権限制御(第3段階セキュリティ)はユーザー・グループ別に行われます。1つのサービスは、サービスに対してアクセス権限が許容されているグループに該当するユーザーのみアクセスが可能です。したがって、該当機能を使用するには、**group**ファイルを作成する必要があり、該当グループに属しているユーザー・ファイルが必要です。また、サービス別にアクセス可能なユーザー・グループを指定する**acl**ファイルが必要です。groupファイルを作成するには、mkgrpコマンドを使用します。

- 使用方法

```
$ mkgrp {-a} {-G group_name} {-g group_id} [-d] [-h] [-V]
```

項目	説明
{ -a }	既存のgroupファイルに新しいユーザー・グループを追加します。既存のgroupファイルが存在する場合、ファイルの一番最後に新規グループが追加され、存在しない場合、新規groupファイルの作成後に追加されます
{ -G group_name }	作成するグループの名前を指定します。他のグループと重複しないようにしてください
{ -g group_id }	作成するグループのIDを指定します。他のグループと重複しないようにしてください
[ -d ]	生成されている既存グループを削除します。[-g]、[-G]オプションと一緒に指定する必要があります、[-g]オプションで指定したIDのユーザー・グループを削除します
[ -h ]	コマンドのヘルプ・オプションです
[ -V ]	コマンドをサポートするTmaxバージョン情報を表示します

## ● 結果

**`${TMAXDIR}/config/group`**ファイルが生成されます。その形式と内容は以下のとおりです。

```
Group_name:x:Group_id
```

```
grp1:x:1
grp2:x:2
grp3:x:3
grp4:x:4
grp5:x:5
grp6:x:6
grp7:x:7
grp8:x:8
grp9:x:9
grp10:x:10
```

## 2.9. mkpw

パスワードを管理するコマンドです。Tmaxユーザーが選択できるセキュリティ確認メカニズムは4種類あります。

1. セキュリティー不使用
2. ドメイン・セキュリティの確認
3. ユーザー・セキュリティの確認
4. サービス・アクセスの制御

セキュリティー確認メカニズムを使用するには、ユーザー名とパスワードを\$(TMAXDIR)/configディレクトリーのpasswdファイルに登録します。mkpwコマンドは、passwdファイルの管理に使われます。

## ● 使用方法

```
$ mkpw [-f file_name] [-a | -d | -i | -n | -p | -h | -V]
        [-G group_name] [-u uid]
```

項目	説明
[ -f filename ]	passwdファイル名を指定します。デフォルトは\$(TMAXDIR)/config/passwdです
[ -a ]	ユーザーを追加します
[ -d ]	ユーザーを削除します
[ -i ]	ユーザーの情報を変更します
[ -n ]	passwdファイルを新規作成します
[ -p ]	ユーザーのパスワードを変更します。パラメータが指定されていない場合、デフォルトとして\$ mkpw -f \$(TMAXDIR)/config/passwd -pを実行します
[ -h ]	使用方法を表示します
[ -V ]	実行ファイルのバージョンを確認できます
[ -G group_name ]	該当ユーザーが属するグループ名を指定します。サービス・アクセス・セキュリティー機能を使用する場合、 <b>groupファイル</b> を作成する必要があります、該当グループに属するユーザーが存在する必要があります。したがって、 <b>userファイル</b> を作成する必要があります。userファイルは、mkpwコマンドを使用してpasswdファイルと一緒に生成されます。  passwdファイルを作成する場合、[-G]オプションを使用し、以前のバージョンとは異なる方式で作成します。ACLで指定されたサービスを呼び出すには、1つのグループに属する必要があるためです。[-G]オプションを使用してグループ名を指定する場合、groupファイルに指定されたグループの中から1つを選択して指定する必要があります、[-u]オプションと一緒に指定します
[ -u uid ]	該当ユーザーの一意のIDを指定します。[-G]オプションと一緒に指定し、第3段階セキュリティーのサービス・アクセス制御を使用する場合にのみ指定します

## ● 結果

以下は、[-G], [-u]オプションを使用した場合のpasswdファイルとuserファイルの形式および例です。

### – passwdファイル

形式は以下のとおりです。

```
username:passwd:user_id:group_id:Description:x:x
```

#### <passwdファイルの例>

```
starbj1:UnQGcdDkNqXNc:1:1:starbj1:x:x
starbj2:mPLY7VZtNvRXs:2:2:starbj2:x:x
starbj3:aiu6Mt36rqwe6:3:3:starbj3:x:x
starbj4:vVdS9naV02jA.:4:4:starbj4:x:x
starbj5:568kCzyzYXriQ:5:5:starbj5:x:x
starbj6:ouKrHf/89QMW6:6:6:starbj6:x:x
starbj7:Mx8PaESrqWR4I:7:7:starbj7:x:x
starbj8:LL59popHJp59U:8:8:starbj8:x:x
starbj9:RG/S5BetAPeFs:9:9:starbj9:x:x
starbj10:Ebbzv1EcX0abE:10:10:starbj10:x:x
```

#### – userファイル

形式は以下のとおりです。

```
userファイル:username:user_id:group_id
```

#### <userファイルの例>

```
starbj1:1:1
starbj2:2:2
starbj3:3:3
starbj4:4:4
starbj5:5:5
starbj6:6:6
starbj7:7:7
starbj8:8:8
starbj9:9:9
starbj10:10:10
```

- 適用環境

Tmaxがインストールされている運用システム環境で使用できます。

## 2.10. mksvr

Tmaxサーバー・モジュールを作成するコマンドです。mksvrコマンドを使用してサーバー・モジュールを作成した場合、Tmaxの環境ファイルにサービスを登録しなくても動的にサービスを登録することができます。パラメータは最大1024文字まで使用できます。該当コマンドを実行すれば、ccコンパイラによってサーバー・モジュールが生成されます。オプションは以下のとおりです。

- 使用方法

```
$ mksvr {-s { @filename | service[,service...] [:func]}} [-o outfile]
        [-f firstfiles] [-v] [-r rmname ] [-S sdlfilename] [-l lastfiles]
        [-t servertime] [(-32)|-64] [-V] [-a autotran] [-T svctime]
        [-u UserCompileOption] [-d]
```

項目	説明
[-s { @filename   service[,service...] [:func] }]	<p>サービス名をserviceに指定します。</p> <p>filenameのファイルにサービス名を1行に1つずつ記録し、該当ファイルの名前を「@」の後ろに追加して、サーバー・モジュールを作成できます。サービスのfuncを指定した場合、サービスに対する要求が来た際、該当処理をfuncで行うことができます。サービス名をセミコロン(;)の後ろに続けて書き込む場合、セミコロンの後ろにスペースを入れてはいけません。オプションは複数回指定できます。</p> <p>例)</p> <pre>-s TOUPPER -s TOLOWER</pre> <p>AUTOTRANとSVCTIMEに関する内容を一緒に設定することもできます。その場合、常に二重引用符(" ")を使用する必要があります。「-s TOUPPER -a 1 -T 10」に設定したものと同一と見なされます。Function Listファイル内でも、上記と同一に設定できます</p> <p>例)</p> <pre>-s "TOUPPER 1 10"</pre>
[-o outfile]	サーバー・モジュール名をoutfileに指定します
[-f firstfiles]	ユーザー定義ファイルを指定すると、Tmaxが提供するクライアント・ライブラリーより先にリンクされ、コンパイルされます。一般的に、ユーザー定義ファイルとしてクライアント・プログラムを指定します
[-v]	verboseモードでコンパイルされ、コンパイルするプロセスがすべてコンソール・ウィンドウに出力されます
[-r rmname]	<p>作成するサーバー・モジュールと接続されるリソース・マネージャーを指定します。</p> <p>rmnameは、<b>\$(TMAXDIR)/config/RM</b>ファイルに以下の形式で指定します。</p> <pre>rmname:stub:XA libraries</pre> <p>stubには、Tmaxが提供するDBMS stub(liboras.so、libsybs.so、libinfs.so、libdb2s.so)が設定されます。</p> <p>mksvrコマンドを使用する場合、[-r rmname]を指定すると、StubとXA librariesがccコンパイル時にリンクされ、コンパイルされます。</p>

項目	説明
	[-r]オプションが指定されていない場合、デフォルトとしてlibnodb.soがリンクされます
[-S sdlfilename]	SDLバッファを使用するサーバー・モジュールの場合、sdlファイル(.s)のobject fileをsdlfilenameに設定します。[-S]オプションが指定されていない場合、デフォルトとして\${TMAXDIR}/lib/sdl.oがリンクされます
[-l lastfiles]	ユーザー定義ファイルを指定すると、Tmaxが提供するサーバー・ライブラリーより後にリンクされ、コンパイルされます
[-t servertype]	<p>Tmax環境ファイルのSERVERセクションに「SVRTYPE」と指定されている値をmksvrコマンドで設定します</p> <ul style="list-style-type: none"> <li>– [-t]オプションが指定されていない場合、サーバー型をSTDと認識し、ccコンパイル時に&lt;libsvr.so&gt;がリンクされます</li> <li>– [-t]オプションが指定されている場合、設定されたTmaxライブラリーがccコンパイル時にリンクされます</li> </ul>
[( -32 )   -64 ]	<p>作成するサーバー・モジュールが32Bitと64Bitのどちらであるかを指定します</p> <ul style="list-style-type: none"> <li>– [-32]: Tmaxライブラリー・パスが\${TMAXDIR}/libに設定されます(デフォルト値)</li> <li>– [-64]: Tmaxライブラリー・パスが\${TMAXDIR}/lib64に設定されます</li> </ul>
[-V]	実行ファイルのバージョンを確認できます
[-a autotran]	<p>動的に登録したサービスのAUTOTRAN有無を設定します</p> <ul style="list-style-type: none"> <li>– 0 : NO</li> <li>– 1: YES</li> </ul>
[-T svctime]	<p>動的に登録したサービスのSVCTIMEを設定します</p> <ul style="list-style-type: none"> <li>– 0 : 無限大でSVCTIMEを設定します</li> <li>– 0より大きい数: SVCTIMEを該当値の分設定します</li> </ul>
[-u UserCompileOption]	<p>mksvrコマンド内部で設定したプラットフォームに適合した基本コンパイル・オプションを除去し、ユーザーが指定したコンパイル・オプションで代替します。ライブラリーおよびインクルード(include)関連オプションは適用されないので注意が必要です。</p> <p>2つ以上のオプションを指定する必要がある場合、以下のように設定します</p> <pre>-u opt1 opt2</pre>



項目	説明
[-d]	mksvrコマンドでビルド時に、既存のsvctab.cを参照せずにサービスをビルドできるようにします。既存の環境設定の影響を受けずに、サービスを動的に登録する場合に使用します

#### ● 適用環境

Tmaxがインストールされている運用システム環境で使用できます。Tmax 3.8.16バージョンからWindows NTとWindows 2000環境で使用可能です。

#### ● 注意事項

mksvrコマンドは、COUSINあるいはBACKUPサーバー・グループ形式のサーバー・モジュールにサービスを動的に登録した場合、使用に制約があります。COUSINあるいはBACKUPサーバー・グループの場合、複数ノードに登録されているサービスの情報を管理する必要があるため、mksvrコマンドで特定サービスを動的に登録した場合、該当サービスを解除できません。

一般サーバー・グループでは、該当サーバーをtmddownコマンド実行後にmksvrコマンドで再登録すると、サービスの移動と削除が可能です。しかし、COUSIN/BACKUPサーバー・グループでは登録されているサービスの解除が許容されていないため、サービスの移動と削除が不可能です。

mksvrコマンドを使用したサービスの動的登録を行う場合、次の事項を考慮して使用してください。

- 該当サーバーのサーバー・グループ・タイプ
- 該当サービスの登録可否

## 2.11. racd

マルチノードで分散された環境で各ノードを中央管理するためのコマンドです。racdコマンドは、複数ノードを1つのドメインでTmaxシステムを構築した場合、1つのノードでTmaxシステムを集中管理するために、それぞれのノードであらかじめ起動されるデーモン・プロセスです。Tmaxシステムを管理するノードでは、racdコマンドを実行しなくても構いません。

racdコマンドは、ドメイン内の1ノードでtadminコマンドを使用して全体ノードを管理したり、またはcflコマンドで環境ファイルをドメイン内のすべてのノードに同一内容で適用可能にします。

IPバージョンがIPv6またはInfiniBandのSDP(Socket Direct Protocol)である場合、環境ファイルに下記を指定する必要があります。

```
SYSTEM_PROTOCOL=" IPV6 "
SYSTEM_PROTOCOL=" SDP "
```

racdを-kオプションと一緒に使用する場合は、環境変数に下記を指定する必要があります。

```
TMAX_RAC_IPV6="IPV6"
TMAX_RAC_IPV6="SDP"
```

以下は、racdコマンドの使用方法についての説明です。

- 使用方法

```
$ racd [-d] [-f バイナリTmax環境ファイル名] [-h] [-k]
      [-i filename] [-l Label] [-P umask] [-V]
```

項目	説明
[-d]	デバッグ・モードでracdコマンドを実行する場合に設定します
[-f バイナリTmax環境ファイル名]	参照するバイナリTmax環境ファイル名を設定します。cflコマンドの結果で、 <b>tmboot</b> コマンドと <b>tmdown</b> コマンドでも参照されるファイルです。パスと一緒に指定でき、省略した場合、デフォルトとしてTMAXDIRと指定されたディレクトリ下位のconfigディレクトリで <b>tmconfig</b> を参照します
[-h]	コマンドのヘルプ・オプションです
[-k]	バイナリTmax環境ファイルの参照可否を設定します。  オプションを指定した場合、バイナリTmax環境ファイルを参照しません。一般的に、racdコマンドはこのオプションを使用して実行します(passive listen mode)。  IPバージョンがIPv6またはInfiniBandのSDP(Socket Direct Protocol)である場合、このオプションを使用すると、環境ファイルを参照しないため、環境変数に「TMAX_RAC_IPV6=IPv6」または「TMAX_RAC_IPV6=SDP」を指定する必要があります
[-i filename]	1つの物理的なマシンで複数の論理的ノードを定義する場合に使用します。  論理ノードのNODETYPEがSHM_RACDの場合、各論理ノードあたり1つずつのracdを実行する必要があるため、RACPORTはすべて異なる必要があります。racdコマンドを実行前に、環境変数のTMAX_RAC_PORT変数を設定しなければなりません。しかし、論理ノードの数が多い場合はすべて変えることができないため、TMAXHOME、TMAXDIR、TMAX_RAC_PORTが定義されたファイル名を指定できます。詳細内容は例を参照してください
[-l Label]	Labelは、ファイル内に登録された環境情報の記述子です。2個以上のシステム情報を1つのファイルに登録する場合、それぞれのシステムを区別できる値です
[-P umask]	racdコマンドを使用して起動するプロセスの場合、umaskをユーザーが設定して、必要な権限のファイルを実行できるようにします
[-V]	実行ファイルのバージョンを確認できます

- 適用環境

Tmaxがインストールされている運用システム環境で使用できます。

- 例

– 以下は、racdコマンドに[-P]オプションを指定し、umaskを設定する例です。

<tmax.racd>

```
[tmaxs1]
TMAXHOME = /user2/starbj81/tmax32
TMAXDIR = /user2/starbj81/tmax32
TMAX_RAC_PORT = 3333
[NODE1]
TMAXHOME = /user2/starbj81/tmax32
TMAXDIR = /user2/starbj81/proj1
TMAX_RAC_PORT = 4335
[NODE2]
TMAXHOME = /user2/starbj81/tmax32
TMAXDIR = /user2/starbj81/proj2
TMAX_RAC_PORT = 4337
```

1. 環境ファイルを作成します。

```
*DOMAIN
tmax1      SHMKEY = @SHMEMKY@, MINCLH = 1, MAXCLH = 3,
           TPORTNO = @TPORTNO@, BLOCKTIME = 30, RACPORT = 3255

*NODE
@HOSTNAME@ TMAXDIR = "@TMAXDIR@",
           APPDIR = "@TMAXDIR@/appbin",
           PATHDIR = "@TMAXDIR@/path",

@RMTNAME@  TMAXDIR = "@RMTDIR@",
           APPDIR = "@RMTDIR@/appbin",
           PATHDIR = "@RMTDIR@/path",

*SVRGROUP
svg1      NODENAME = "@HOSTNAME@", COUSIN = "svg2"
svg2      NODENAME = "@RMTNAME@"

*SERVER
svr2      SVGNAME = svg1, CLOPT = "-o $(SVR).out -e $(SVR).err"

*SERVICE
TOUPPER   SVRNAME = svr2
```

2. リモートノードのracdを起動します。

```
$ export TMAX_RAC_PORT = 3255
$ racd -k -P 055
```

3. HOSTノードで全体Tmaxを起動します。(tmboot)

4. RMTノードのULOGDIRにsvr2.outのファイル権限を確認します。

– 以下は、NODE1のRACDを実行させる例です。

```
$ racd -k -i tmax.racd -l NODE1
```

– 以下は、環境ファイルを参照せずに、他のコマンド(tmboot)で使用した情報のみを利用する例です。

```
$ racd -k
```

---

#### 参考

1. tmboot、tmdownコマンドについての詳細は、「[2.31. tmboot](#)」、「[2.33. tmdown](#)」を参照してください。
  2. 論理ノード設定についての詳細は、『*Tmax 運用ガイド*』の「3.2 NODEセクション」のHOSTNAME項目の説明を参照してください。
- 

## 2.12. racdr

マルチノード、マルチドメインの分散環境で、各ノードにコマンドの実行、ファイルの送信を実行するためのコマンドです。管理を行うノードにはracdデーモンが起動されている必要があります。環境設定ファイルを一定の形式に合わせてあらかじめ作成します。

以下は、racdコマンドの使用方法についての説明です。

#### ● 使用方法

```
$ racdr [-f racdr 環境ファイル名] [-n ノード名]
        [[[-W | -w] -e] -c コマンド] | [-s ファイル名] [-d 保存先]
        [-r]] [-h] [-V]
```

項目	説明
[-f racdr 環境ファイル名]	他のドメインのノード情報を保存した環境ファイルを指定します。  指定しない場合、現在のディレクトリーのracdr.cfg、または\${TMAXDIR}/config/racdr.cfgファイルを使用します

項目	説明
[ -n ノード名 ]	環境ファイルで定義したノード名を指定します。すべての要求はそのノードのracdに送信され実行されます
[ -W シェル・プログラム ]	<p>-e、-cオプションでコマンドを実行時に、指定したシェル・プログラムにより実行されるようにします。</p> <p>ksh、tsh、bashなど、使用したいシェル・プログラムの名前を引数に指定します</p>
[ -w ]	<p>-e、-cオプションでコマンドを実行時に、「sh」シェル・プログラムにより実行されるようにします。他のシェル・プログラムを使用するには、このオプションの代わりに -Wオプションを使用します。</p> <p>コマンドのパラメータにワイルドカード文字を使用する場合は、ワイルドカード文字をそのままracdに送信するために、必ずコマンドまたはコマンドラインのパラメータを引用符(' ')で囲む必要があります</p>
[ -e ]	-cオプションで許容しないコマンドを実行できるようにします。-cオプションの前に指定する必要があります。当該ノードでシェル・プログラムによりコマンドが実行される必要がある場合は、-wオプションと一緒に使用します
[ -c コマンド ]	<p>racdでコマンドを実行します。</p> <p>-cオプション以降のパラメータはすべてコマンドラインのパラメータとしてみなします。そのため、racdrの他のオプションを指定するときは、必ず -cオプションの前に指定する必要があります。コマンドは、tmboot、tmdown、cfl、gstのみ許容されます。その他のコマンドやシェル・スクリプトを実行するには、-eオプションを指定します。</p> <p>注意事項としては、racdでtmboot、tmdownコマンドを実行する場合、コマンドライン・パラメータの末尾に内部で使用するオプションを追加することです。もし引数の指定が必須のオプションを最後のパラメータとして使用していて引数を指定しなかった場合、予想外の動作が発生する可能性があるので注意が必要です。</p> <p>たとえば、「racdr -c tmboot -d」コマンドを実行すると、元々はtmbootのパラメータの誤った使用(-dオプションは必ず引数を入力する必要があります)によってエラーが出力されるはずですが、racdによって実行された場合は、-dの後に追加されるオプションによって異常な値を処理し、コマンドが実行されることがあります。したがって、コマンドを入力するときは、パラメータの入力に繊細な注意を払う必要があります</p>
[ -s ファイル名 ]	ファイルを該当ノードに送信します。-dオプションと一緒に使用する必要があります。

項目	説明
	<p>保存される場所にすでにファイルが存在する場合、そのファイルの名前を変更し、送信されたファイルを既存のファイル名で保存します。既存のファイル名は後ろに現在の「_年月日時分秒」が追加され、同じ名前のファイルがさらに存在する場合は、その後ろに「_番号」を追加します。</p> <p>ファイルの送信中に失敗すると、送信中のファイルを削除し、既存のファイルを復元します。既存のファイルを復元したくない場合は、-rオプションと一緒に指定します</p>
[-d 保存先]	<p>ファイルを保存するパスを環境ファイルに設定します。パスは、最大3つまで指定できます。</p> <p>dest_noは、設定したディレクトリーの番号を指定します。最初のパス(dest1)は1、2番目のパス(dest2)は2、3番目のパス(dest3)は3を指定します</p>
[-r]	ファイルが保存されるノードに同じ名前のファイルがすでに存在しても既存のファイルのバックアップを作成しません
[-h]	Usage画面を出力します
[-V]	実行ファイルのバージョンを確認できます

- 適用環境

racdrとracdr環境ファイルがあれば実行できます。

- 注意事項

- コマンドを実行するとき、コマンド引数のパラメータに必ず -nオプションを設定して、指定したノードでのみ実行します。

- 入力を要求するコマンドは処理できません。入力を要求ないようにコマンド引数を設定する必要があります。

```
例) $ tmdown
      Do you really want to down whole Tmax? (y : n):
--> $ tmdown -n node1
```

- 実行結果画面は、コマンドがすべて実行され終了されるときにクライアントに送信されます。

- 実行結果画面の中で、STDOUTとSTDERRの出力は実際の画面と異なる順序で出力されることがあります。

- racdrの実行結果においてtmbootのSTDERRの内容は出力されません。

- コマンドの実行中にracdは他の要求は処理できません。

- 例

- 以下は、3つのノードを管理するracdr環境設定ファイルの例でs。

<racdr.cfg>

#nodename	ip-address	racport	dest1(appbin)	dest2(applib)	dest3(config)
tmnode1	192.168.1.1	5000	/tmax/appbin	/tmax/applib	/tmax/etc
tmnode2	192.168.1.2	5000	/tmax/appbin	/tmax/applib	/tmax/config
othnode	192.168.10.2	9999	/tmax/appbin	/tmax/applib	/tmax/config

- /app/svr2ファイルをtmnode1ノードの /tmax/appbinディレクトリーに送信します。

```
$ racdr -f config.txt -n tmnode1 -s /app/svr2 -d 1
```

- tmnode1ノードにcflコマンドを実行します。

```
$ racdr -f config.txt -n tmnode1 -c "cfl -i tmaxconfig.m"
```

- tmnode1ノードにtmbootコマンドを実行します。

```
$ racdr -f config.txt -n tmnode1 -c "tmboot -n node1"
```

- tmnode1ノードで特定のサーバーを終了します。

```
$ racdr -f config.txt -n tmnode1 -c "tmdown -n node1 -S svr2"
```

## 2.13. rcakill

RCAを終了したい場合、RCAが使用しているリソースを削除するために使用します。

- 使用方法

```
rcakill [-h] [-p pid] [-n rcah_no] [-k shmkey]
```

項目	説明
[-h]	コマンドのヘルプ・オプションです
[-p pid]	pidにより特定のRCAHの状態情報を削除します
[-n rcah_no]	RCAH番号により特定のRCAHの状態情報を削除します
[-k shmkey]	RCAが使用する共有メモリーのキー値の情報を削除します

## 2.14. rcastat

RCAはTmaxシステムではクライアント・プロセスとして管理されるため、Tmaxシステム管理ツールのtmadminでモニタリングできます。TmaxではRCAをモニタリングするrcastatという別途のツールを提供しています。このツールを使って管理者は現在のRCAの設定内容を確認したり、現在RCAに接続しているクライアントの数などをモニタリングすることができます。

- 使用方法

```
rcastat [-h] [-p pid] [-n rcah_no] [-k shmkey]
```

項目	説明
[-h]	コマンドのヘルプ・オプションです
[-p pid]	pidにより特定のRCAHの状態情報を出力します。  pidを指定していない場合、RCAで動作しているすべてのRCAHの情報を出力します
[-n rcah_no]	RCAH番号により特定のRCAHの状態情報を出力します。  rcah_noを指定していない場合、RCAで動作しているすべてのRCAHの情報を出力します
[-k shmkey]	RCAが使用する共有メモリー・キー値を設定します。  共有メモリー・キー値を指定しない場合は、ユーザー・システム環境変数に設定したRCA_SHMKEY値を読み込みます。システム環境変数にも設定していない場合は、デフォルト値の74565が使用されます。誤ったキー値を入力した場合には、正しい情報を出力できません

- 例

```
-----  
rca_dir: /user/tmax/server/  
rca_mode: Local  
rca_port: 8123  
rcal_name: rca  
rcal_pid: 1722  
rcah_name: rcah  
shmkey: 74565, shmsize: 4832  
#rcah: 2, #thread per rcah: 60  
-----  
  
rcah_no    pid    #client  
-----  
0          1724    0
```



1	1723	0
-----		

出力項目名	説明
rca_dir	RCAH実行ファイルと通信用パイプが生成されるディレクトリーです
rca_mode	現在使用されているモード情報です
rca_port	使用可能なポート番号です
rcal_name	RCALの名前です
rcal_pid	RCALのpidです
rcah_name	RCAHの名前です
shmkey	RCAで使用している共有メモリー・キー値です
shmsize	共有メモリーのサイズです
#rcah	使用されているRCAの数です
#thread per rcah	1つのRCAH当たり起動されているスレッド数です
rcah_no	RCA内部で管理される番号であり、それぞれのRCAHごとに番号が振られます
pid	RCAに起動されているRCAH別のpidです
#client	クライアントの数です

以下は、コマンドを使って照会したシステムの設定情報です。

```
$ rcastat
rcastat: RCA_SHMKEY env is not set, using default shmkey (74565)
```

## 2.15. sdlc

サーバーおよびクライアント間のデータ通信に使用する方式はいくつかありますが、その中で構造体(Struct)を使用する場合は、該当構造体をTmaxシステムで認識できる必要があります。sdlcは、こうした構造体を定義したファイルをコンパイルするコマンドです。

**SDL**(Structure Data Language)は、Tmaxで決めた標準構造体データ形式です。これは、異機種ノード間の通信時、Integer、Float、Doubleなどのデータ型と関連して以下のような問題点があります。

- Type Length
- Machine Type - Big / Little Endian
- Alignment

SDLは、このような問題を解決するために1つの標準型を決めたものであり、Integer、Float、Doubleの通信をサポートし、自由な通信タイプをサポートします。

クライアント・プログラムとサーバー・プログラムで使用される構造体は同一である必要があります。構造体  
 が変更された場合、クライアントとサーバー別に、sdlcコマンドによって構造体が再コンパイルされる必要が  
 あり、新しく作成された<構造体ファイル名\_sdl.c>は、サーバー・プログラムと一緒に再コンパイルされる必  
 要があります。使用される構造体ファイル名は<.s>形式のファイルで、結果は<構造体ファイル名\_sdl.c>形  
 式です。構造体ファイルには、サーバーおよびクライアント間の通信に使用されるバッファ型構造体が  
 定義されます。構造体名は通信バッファでtpalloc()、tpcall()、tpacall()などに使用されます。

sdlcコマンドで生成されたファイルは、クライアントのSDLFILEという環境変数に必ず登録します。

## ● 使用方法

```
$ sdlc [-c] [-h ヘッダー・ファイル名] {-i 構造体ファイル名} [-o 結果名]
        [-s] [-v ビュー定義ファイル] [-32] [-4vb|-4dp] [-f] [-V]
```

項目	説明
[-c]	<p>クライアントのために使用されるオプションです。クライアント・プログラムで使用する構造体ファイルをsdlcコマンドでコンパイルすると、構造体の各データが標準データ型に変換され、必要な情報のバイナリ形式のファイルが生成されます。</p> <p>&lt;構造体ファイル名.sdl&gt;は、クライアント・プログラムの実行時に標準通信型でデータが送受信されるのに使用されます。</p> <p>[-c]オプションを使用した場合、複数の構造体ファイルを一緒にコンパイルできます。(例: sdlc -c -i *.sまたはsdlc -c -i demo.s sam.s abc.s)生成されるsdlファイル名は&lt;demo.sdl&gt;で、別名を使用するには[-o]オプションを利用してファイル名を指定します。</p> <p>[-c]オプションを使用しない場合、sdlcは構造体ファイルをコンパイルし、構造体の各フィールドを標準通信型のSDL形式に合わせて暗号化(encoding:ノード・データ型を標準データ型に変換)、復号化(decoding:標準データ型をノード・データ型に変換)するプログラムが生成されます。サーバー・プログラムと一緒にコンパイルされ、標準SDL形式のデータ型に通信が行われる&lt;構造体ファイル名_sdl.c&gt;ファイルが生成されます</p>
[-h ヘッダー・ファイル名]	[-h]オプションなしでコンパイルすると、sdlヘッダー・ファイル名は<構造体ファイル名_sdl.h>です。ヘッダー・ファイル名を変更する場合に使用します
{-i 構造体ファイル名}	<p>サーバー/クライアント・プログラムで使用する構造体を定義したファイルを設定します。</p> <p>必須オプションで、パスと一緒に指定できます。構造体名は16文字まで設定できます</p>
[-o 結果名]	構造体ファイルが1つの場合、[-o]オプションなしでコンパイルすると、sdlファイル名は<構造体ファイル名.sdl>です。ファイル名を別名に変更する場合に使用します

項目	説明
[-s]	<p>サーバー・プログラムで使用されるオプションで、構造体定義バイナリ・ファイルを作成します。構造体定義バイナリ・ファイルは、[-c]オプションを使用した場合に生成されるファイルと同じです。</p> <p>構造体ファイルをコンパイルし、構造体の各フィールドを標準通信型のSDL形式に合わせて暗号化、復号化するプログラムが生成されます。生成されるファイル名は&lt;構造体ファイル名_sdl.c&gt;で、このファイルはサーバー・プログラムと一緒にコンパイルする必要があります。[-c]オプションとは排他的に使用され、[-c]オプションが使用されない場合、デフォルト値として使用されます</p>
[-v ビュー定義ファイル]	<p>構造体バッファとフィールド・キー・バッファ間で形式変換を行う場合にこのオプションを使用します。構造体定義バイナリ・ファイルを生成する必要があります。構造体バッファのデータをフィールド・キー・バッファに保存してサービスを要請するか、その逆にフィールド・キー・バッファの内容を構造体バッファに保存してサービスを要請できます。</p> <p>使用される関数には<b>fbftos()</b>と<b>fbstof()</b>があります。詳細説明は例を参照してください</p>
[-32]	64Bit環境で32Bitライブラリーを使用するには、構造体ファイルを32Bitでコンパイルします
[-4vb]	Visual Basicインターフェース・ファイルを作成します
[-4dp]	Delphiインターフェース・ファイルを作成します
[-f]	[-4vb]または[-4dp]オプションと一緒に使用する必要があり、Visual BasicまたはDelphiインターフェース・ファイル名を指定します
[-V]	実行ファイルのバージョンを確認できます

- 適用環境

UNIX運用システムとMS-DOS運用システムでサポートされます。

- 例

– VIEW定義ファイルの構造は以下のとおりです。

<demo.v>

```
VIEW demo
#type  Cname  fldkey count  flag size null
String Demodata      INPUT  5    -    20    ""
Int     Num          INTDATA  5    -    -     0
END
```

項目	説明
type	構造体内のメンバー変数のデータ型です
Cname	メンバー変数名です
fldkey	指定したメンバー変数とマッピングされるフィールド・キーです
count	構造体で保存できる最大のフィールド順序です
flag	現在は使用していません
size	文字列型の場合の配列サイズです
null	初期化時の値です

構造体定義バイナリ・ファイルは、以下のようなコマンドで作成できます。

```
$ sdlc -c -v demo.s -o tmax.sdl
```

- 以下は、サーバーのために現行ディレクトリーにある<demo.s>構造体ファイルをsdlcでコンパイルするコマンドの例です。結果として、<demo.sdl>と<demo\_sdl.c>というファイルが生成されます。<demo\_sdl.c>はサーバー・プログラムの作成に使用されます。

```
$ sdlc -i demo.s
```

- 以下は、クライアントのために現行ディレクトリーにある<demo.s>構造体ファイルをsdlcでコンパイルするコマンドの例です。結果として、バイナリ・データ・ファイルの<demo.sdl>というファイルが作成されます。

```
$ sdlc -c -i demo.s
```

## 2.16. svcrcpt

Tmaxシステムの運用時、サービス実行に関連したログ記録を分析して出力するコマンドです。出力内容は、実行されたサーバーおよびサービスの名前と回数、平均サービス実行時間であり、ユーザーが指定する時間範囲内で1時間単位で出力が可能です。サービス・ログは、サーバー別にCLOPTセクションに[-l]オプションを設定すると、5分間隔で\$ULOGDIRに<svclog.mmddyyyy>という名前のファイルが保存されます。

### ● 使用方法

```
$ svcrcpt [-T | -N] [-s svcname] [-v svrname] [-x] [-S] [-d mm:dd] [-c column]
          [-f hh:mm:ss] [-t hh:mm:ss] [-V] -i logfile
```

項目	説明
[-T   (-N)]	<ul style="list-style-type: none"> <li>– [-T]: サービスの総実行時間の順に結果をソートします</li> <li>– [-N]: サービスの実行回数を基にソートします(デフォルト値)</li> </ul>

項目	説明
[ -s <i>svcname</i> ]	出力するサービス名を設定します
[ -v <i>svrname</i> ]	出力するサーバー名を設定します
[ -x ]	サーバー・プログラム名、最大・最小処理時間など、より細かい内容を照会します
[ -S ]	サービス要約を照会します
[ -d <i>mm:dd</i> ]	サービス・ログの分析を行う日付を入力します
[ -c <i>column</i> ]	出力するデフォルト列を入力した数字に変更します(デフォルト値: 5)
[ -f <i>hh:mm:ss</i> ]	分析を開始する時間帯を入力します
[ -t <i>hh:mm:ss</i> ]	分析を終了する時間帯を入力します
[ -V ]	実行ファイルのバージョンを照会します
-i <i>logfile</i>	分析するログファイル名を入力します

- 適用環境

Tmaxがインストールされている運用システム環境で使用できます。

- 例

- 以下は、<svclog.04152002>というログファイルを分析し、1時間単位で結果を表示する例です。

```
$ svcrcpt -i svclog.04152002
```

- 以下は、<svclog.04012002>というログファイルの4月15日午前に実行されたaccountというサービスの実行回数および平均実行時間を1時間単位で出力する例です。

```
$ svcrcpt -s account -d 04:15 -f 09:00:00 -t 12:00:00 -i svclog.04012002
```

## 2.17. tdlclean

runディレクトリーの旧バージョンのライブラリー・ファイルや不要なファイルを整理するコマンドです。特に、[-m]オプションあるいは[-M]オプションを使用すると、共有メモリーで指定した動的モジュールが完全に削除されます。

- 使用方法

```
$ tdlclean [-p TDLルート・ディレクトリー・パス] [-m ライブラリー名] [-M 関数名] [-b]
           [-d yyyyymmddhhmi] [-D "n hour|day" [-N 個数] [-v | -V] [-h]
```

項目	説明
[ -p TDLルート・ディレクトリー・パス ]	TDLルート・ディレクトリーは、 <b>\$(TDLDIR)</b> または <b>\$(TMAXDIR)</b> を使用します。  ルート・パスを別途指定して実行する場合、[-p]オプションを使用してパスを指定できます
[ -m ライブラリー名 ]	TDL共有メモリーで指定したライブラリーとrunディレクトリーのファイルを削除します
[ -M 関数名 ]	共有メモリーで指定した関数のみ削除し、runディレクトリーの関連ファイルは削除しません。ただし、VERSION=1と設定された場合、[-m]オプションと同一動作します
[ -b ]	TDL環境ファイル(tdl.cfg)にBACKUPパラメータが指定されていても、共有メモリー・ファイルのバックアップを実行しません。  [-m]または[-M]オプションと一緒に使用します
[ -d yyyymmddhhmi ]	指定時間(yyyymmddhhmi)以前の旧バージョンのライブラリー・ファイルをすべて削除します。  [-m]または[-M]オプションと一緒にには使用できません
[ -D “n hour day” ]	nは、時間(hour)または日(day)を設定します。指定時間または指定日以前の旧バージョンのライブラリー・ファイルをすべて削除します。  [-m]または[-M]オプションと一緒にには使用できません
[ -N 個数 ]	指定した個数(number)分の旧バージョンのライブラリー・ファイルを残し、それ以外の旧バージョンのファイルをすべて削除します。  [-m]または[-M]オプションと一緒にには使用できません
[ -v   -V ]	バージョン情報を出力します
[ -h ]	使用方法を出力します

#### ● 例

- 以下は、旧バージョンのライブラリー・ファイルを削除する例です。

```
$ tdlclean
```

- 以下は、2009年2月1日00時00分以前の旧バージョンのライブラリー・ファイルを削除する例です。

```
$ tdlclean -d 200902010000
```

- 以下は、5日以前の旧バージョンのライブラリー・ファイルを削除する例です。

```
$ tdlclean -D "5 day"
```

- 以下は、mylibraryをTDL共有メモリーから完全に削除し、旧バージョンのファイルも削除する例です。

```
$ tdlclean -m mylibrary
```

## 2.18. tdlinit

TDL共有メモリーおよび動的モジュールを初期化するコマンドです。tdlinitコマンドは、インストール時に1回のみ実行し、Tmaxを起動前に実行する必要があります。マルチノード環境の場合、マスター・ノードでのみ実行可能です。

### • 使用方法

```
$ tdlinit [-p TDLルート・ディレクトリーのパス] [-x エクスポート関数抽出スクリプト・ファイルのパス]
          [-f] [-b] [-B バックアップ・ファイルのパス] [-i] [-v | -V] [-h]
```

項目	説明
[-p TDLルート・ディレクトリーのパス]	TDLルート・ディレクトリーは、 <code>\${TDLDIR}</code> または <code>\${TMAXDIR}</code> を使用します。ルート・パスを別途指定して実行する場合、[-p]オプションを使用してパスを指定できます
[-x エクスポート関数抽出スクリプト・ファイルのパス]	エクスポート関数の抽出のためのスクリプト・ファイルのパスを指定します
[-f]	共有メモリーがすでに存在する場合、強制的に初期化します
[-b]	バックアップ・ファイルから共有メモリーをリカバリーします
[-B バックアップ・ファイルのパス]	指定されたバックアップ・ファイルから共有メモリーをリカバリーします
[-i]	バックアップのリカバリー後、runディレクトリーをチェックします。[-b]または[-B]オプションと一緒に使用します
[-v   -V]	バージョン情報を出力します
[-h]	使用方法を出力します

### • 例

- 以下は、TDL環境ファイル(tdl.cfg)を参照し、共有メモリーおよびモジュールを初期化する例です。

```
$ tdlinit
```

- 以下は、装備不具合の場合や再度開始する場合、バックアップ・ファイルから共有メモリーをリカバリーする例です。

```
$ tdlinit -b
```

- 参考

VERSION=4に設定した場合には、エクスポート関数を抽出するためのスクリプト・ファイルが必要です。

```
/* Example of exp file */
/* dlib.exp */

#! dlib.so
TmaxSoft::Airplain
Car
```

## 2.19. tdlnm

VERSION=2以上で指定したライブラリーに対して自動エクスポートされる関数の一覧を照会するコマンドです。

- 使用方法

```
$ tdlnm [-p TDLルート・ディレクトリーのパス] [-x exportエクスポート関数抽出スクリプト・ファイルのパス]
        [-m ライブラリー名] [-v | -V] [-h]
```

項目	説明
[-p TDLルート・ディレクトリーのパス]	TDLルート・ディレクトリーは、 <code>#{TDLDIR}</code> または <code>#{TMAXDIR}</code> を使用します。ルート・パスを別途指定して実行する場合、[-p]オプションを使用してパスを指定できます
[-x エクスポート関数抽出スクリプト・ファイルのパス]	エクスポート関数の抽出のためのスクリプト・ファイルのパスを指定します
[-m ライブラリー名]	自動エクスポート関数一覧照会のためのライブラリー名を指定します
[-v   -V]	バージョン情報を出力します
[-h]	使用方法を出力します

- 例

以下は、mylibraryファイルのエクスポート関数一覧を照会する例です。

```
$ tdlnm -m mylibrary
```



## 2.20. tdlrm

TDLを今後使用しない場合、共有メモリーを完全に削除するためのコマンドです。tdlrmコマンドを行う場合、**tdlcall()**を使用できません。

- 使用方法

```
$ tdlrm [-p TDLルート・ディレクトリーのパス] [-v | -V] [-h]
```

項目	説明
[-p TDLルート・ディレクトリーのパス]	TDLルート・ディレクトリーは、 <b>\${TDLDIR}</b> または <b>\${TMAXDIR}</b> を使用します。ルート・パスを別途指定して実行する場合、[-p]オプションを使用してパスを指定できます
[-v   -V]	バージョン情報を出力します
[-h]	使用方法を出力します

- 例

以下は、TDL共有メモリーを削除する例です。

```
$ tdlrm
```

## 2.21. tdlseqno

特定のモジュールと関数のシーケンス番号を照会します。

- 使用方法

```
$ tdlseqno [-m module|-M library [-p tdlmdir]] [-V] [-h]
```

項目	説明
[-p ディレクトリー・パス]	TDLルート・ディレクトリーは、 <b>\${TDLDIR}</b> または <b>\${TMAXDIR}</b> を使用します。 ルート・パスを別途指定して実行する場合、[-p]オプションを使ってパスを指定できます
[-M library]	当該ライブラリー名を指定します
[-m module]	当該関数名を指定します
[-V]	バージョン情報を出力します
[-h]	使用方法を出力します

- 例

- `${TDLDIR}`または`${TMAXDIR}`の当該モジュールと関数のシーケンス番号を照会する例です。

```
$ tdlseqno -m module1 -M library1
```

- 特定のディレクトリー下の当該モジュールと関数のシーケンス番号を照会する例です。

```
$ tdlseqno -m module1 -M library1 -p dirname
```

## 2.22. tdlshm

TDL共有メモリー情報を照会し、統計モニタリングの有効化およびモジュールの有効化を設定するコマンドです。

### ● 使用方法

```
$ tdlshm [-p TDLルート・ディレクトリーのパス] [-r] [-S] [-n ノード名] [-m 関数名]
          [-M ライブラリー名] [-C] [-c start_index end_index] [-s e|d|r]
          [-u e|d] [-I 最小衝突回数] [-v | -V] [-h]
```

項目	説明
<code>[-p TDLルート・ディレクトリーのパス]</code>	TDLルート・ディレクトリーは、 <code>\${TDLDIR}</code> または <code>\${TMAXDIR}</code> を使用します。ルート・パスを別途指定して実行する場合、 <code>[-p]</code> オプションを使用してパスを指定できます
<code>[-r]</code>	マルチノード環境でアップデート同期化中のモジュール情報を照会します
<code>[-S]</code>	動的モジュール統計情報を照会します。統計情報は、各モジュール別の実行時間AVG/MIN/MAXとCPU AVG/MIN/MAXが照会されます
<code>[-n ノード名]</code>	マルチノード環境で照会するノードを指定します
<code>[-m 関数名]</code>	照会する関数名を指定します
<code>[-M ライブラリー名]</code>	照会するライブラリー名を指定します
<code>[-C]</code>	全体モジュールに対して <code>dlopen</code> 、 <code>dlsym</code> を実行します
<code>[-c start_index end_index]</code>	指定した索引範囲のモジュールに対して <code>dlopen</code> 、 <code>dlsym</code> を実行します
<code>[-s e d r]</code>	動的モジュール統計収集を設定します – e: 有効化 – d: 無効化 – r: 初期化
<code>[-u e d]</code>	動的モジュールを設定します。 – e: 有効化

項目	説明
	<p>– d: 無効化</p> <p>1度に1つのモジュールのみ設定可能なため、必ず[-m]または[-M]オプションと一緒に使用します。</p> <p>– VERSION=1、VERSION=2の場合、モジュールは全体ライブラリーで一意であることを保証する必要があるため、[-m]オプションのみ指定します（関数名が全体的に一意である必要があります）</p> <p>– VERSION=3でライブラリーが異なるのであれば、関数名は重複しても構わないため、[-m]オプションと[-M]オプションは必ず入力します。[-m]オプションなしで[-M]オプションのみを使用してはいけません</p>
[-l 最小衝突回数] (大文字i)	<p>モジュールごとにハッシュ衝突によるバケットを検索するための比較回数を出力します。</p> <p>0以上の値を指定でき、衝突回数が指定した回数以上のモジュールに対してのみ出力します。</p> <p>[-p]、[-m]、[-M]、[-r]オプションと一緒に使用できます</p>
[-v   -V]	バージョン情報を出力します
[-h]	使用方法を出力します

## ● 例

- 以下は、tdlshdの基本的な使用例です。

```
$ tdlshd -S TDLDIR = /home/jeffry/tmax LOGDIR = /home/jeffry/tmax/log/dlog
- BACKUP = /home/jeffry/tmax/log/dlog/tdl.bak VERSION = 2, SHMKEY = 0x90000,
- IPCPERM = 0750 MAXMODULES = 256, CURMODULES = 3, Global SEQNO = 45e27d28,
- MONITOR = Y MODE = SINGLE, DOMAINID = 1 Index = 125, Funcname = myfunction1,
- Libname = myfunction1, Seqno = 45e27d28, Active = Y Count = 0 SVC: Avg = 0.000,

- MinTime = 0.000, Maxtime = 0.000 CPU: Avg = 0.000, MinTime = 0.000,
- Maxtime = 0.000 Index = 126, Funcname = myfunction2, Libname = myfunction2,
- Seqno = 45e27d28, Active = Y Count = 0 SVC: Avg = 0.000, MinTime = 0.000,
- Maxtime = 0.000 CPU: Avg = 0.000, MinTime = 0.000, Maxtime = 0.000 Index =
127,
- Funcname = myfunction3, Libname = myfunction3, Seqno = 45e27d28, Active = Y
- Count = 0 SVC: Avg = 0.000, MinTime = 0.000, Maxtime = 0.000 CPU: Avg = 0.000,

- MinTime = 0.000, Maxtime = 0.000 $ tdlshd TDLDIR = /home/jeffry/tmax LOGDIR =
/home/jeffry/tmax/log/dlog
- BACKUP = /home/jeffry/tmax/log/dlog/tdl.bak VERSION = 2, SHMKEY = 0x90000,
- IPCPERM = 0750 MAXMODULES = 256, CURMODULES = 3, Global SEQNO = 45e27d28,
```

```

- MONITOR = Y MODE = SINGLE, DOMAINID = 1 Index = 125, Funcname = myfunction1,
- Libname = myfunction1, Seqno = 45e27d28, Active = Y Index = 126,
- Funcname = myfunction2, Libname = myfunction2, Seqno = 45e27d28, Active = Y,
- Index = 127, Funcname = myfunction3, Libname = myfunction3, Seqno = 45e27d28,

Active = Y

```

- 以下は、TDL統計情報を照会する例です。

```

$ tdlshm -S TDLDIR = /home/jeffry/tmax LOGDIR = /home/jeffry/tmax/log/dlog
- BACKUP = /home/jeffry/tmax/log/dlog/tdl.bak VERSION = 2, SHMKEY = 0x90000,
- IPCPERM = 0750 MAXMODULES = 256, CURMODULES = 3, Global SEQNO = 45e27d28,
- MONITOR = Y MODE = SINGLE, DOMAINID = 1 Index = 125, Funcname = myfunction1,
- Libname = myfunction1, Seqno = 45e27d28, Active = Y Count = 0 SVC: Avg = 0.000,

- MinTime = 0.000, Maxtime = 0.000 CPU: Avg = 0.000, MinTime = 0.000,
- Maxtime = 0.000 Index = 126, Funcname = myfunction2, Libname = myfunction2,
- Seqno = 45e27d28, Active = Y Count = 0 SVC: Avg = 0.000, MinTime = 0.000,
- Maxtime = 0.000 CPU: Avg = 0.000, MinTime = 0.000, Maxtime = 0.000 Index =
127,
- Funcname = myfunction3, Libname = myfunction3, Seqno = 45e27d28, Active = Y
- Count = 0 SVC: Avg = 0.000, MinTime = 0.000, Maxtime = 0.000 CPU: Avg = 0.000,

- MinTime = 0.000, Maxtime = 0.000

```

- 以下は、[-s]または[-u]オプションの使用例です。

```

$ tdlshm -s r
$ tdlshm -s e -m myfunction
$ tdlshm -s d -m myfunction
$ tdlshm -u e -m myfunction
$ tdlshm -u d -m myfunction

```

- 以下は、[-C]または[-c]オプションの使用例です。

```

$ tdlshm -C
$ tdlshm -c 0 1024
$ tdlshm -c 1024

```

## 2.23. tdlsync

TDL共有メモリーとバックアップ・ファイルの同期化を実行するコマンドで、自動バックアップを使用しない場合に必要な管理ツールです。

- 使用方法

```
$ tdlsync [-p TDLルート・ディレクトリーのパス] [-B バックアップ・ファイルのパス] [-v | -V] [-h]
```

項目	説明
[-p TDLルート・ディレクトリーのパス]	TDLルート・ディレクトリーは <code>\${TDLDIR}</code> または <code>\${TMAXDIR}</code> を使用します。ルート・パスを別途指定して実行する場合、[-p]オプションを使用してパスを指定できます
[-B バックアップ・ファイルのパス]	バックアップ・ファイルのパスを指定します
[-v   -V]	バージョン情報を出力します
[-h]	使用方法を出力します

- 例

以下は、TDL環境ファイル(tdl.cfg)のBACKUPパラメータに指定されたファイルで共有メモリーをバックアップする例です。

```
$ tdlsync
```

## 2.24. tdltrace

TDLの環境設定情報と統計情報を照会します。

- 使用方法

```
$ tdltrace [-p ディレクトリー・パス] [-P PID] [-V] [-h] [-c]
```

項目	説明
[-p ディレクトリー・パス]	TDLルート・ディレクトリーは、 <code>\${TDLDIR}</code> または <code>\${TMAXDIR}</code> を使用します。ルート・パスを別途指定して実行する場合、[-p]オプションを使ってパスを指定できます
[-P PID]	特定のサーバー・プロセスの情報を照会します
[-V]	バージョン情報を出力します
[-h]	使用方法を出力します
[-c]	ゾンビ・プロセスを探して関連情報を削除します

- 例

```
$ tdltrace
```

```
$ tdltrace -P 20113
```

```
$ tdltrace -P 20113 -p dirname
```

```
$ tdltrace -c
```

## 2.25. tdlupdate

指定した動的モジュールをアップデートするコマンドです。[-m]オプションでライブラリー名を必ず指定します。指定したライブラリーがすでに登録されている場合、指定したライブラリーをアップデートします。まだ登録されていない場合、新規追加します。マルチノード環境では、マスター／スレーブの両ノードで実行が可能です。

- 使用方法

```
$ tdlupdate [-p TDLルート・ディレクトリーのパス] [-x エクスポート関数抽出スクリプト・ファイルのパス] [-f]
            [-m ライブラリー名] [-b] [-c] [-l ファイル名] [-v | -V] [-h] [-r ディレクトリー・パス]
```

項目	説明
[-p TDLルート・ディレクトリーのパス]	TDLルート・ディレクトリーは、 <b>\$(TDLDIR)</b> または <b>\$(TMAXDIR)</b> を使用します。 ルート・パスを別途指定して実行する場合、[-p]オプションを使用してパスを指定できます
[-x エクスポート関数抽出スクリプト・ファイルのパス]	エクスポート関数の抽出のためのスクリプト・ファイルのパスを指定します
[-f]	VERSION=2で、別のライブラリーに関数名がすでに存在しているとしても強制的にアップデートします
[-m ライブラリー名]	アップデートするライブラリー名を指定します
[-b]	TDL環境ファイル(tdl.cfg)にBACKUPパラメータが指定されていても、共有メモリー・ファイルのバックアップを実行しません
[-c]	マルチノード環境でノード間のバージョン同期化を実行します
[-l ファイル名]	アップデート一覧をファイルで作成してアップデートを実行します。モジュールの区切り子はコンマ(,)、または<Enter>です
[-v   -V]	バージョン情報を出力します
[-h]	使用方法を出力します
[-r ディレクトリー・パス]	tdlupdateを実行するリモート・ノードのTDLDIRを指定します。 [-p]オプションを使用する場合、2つ以上のTDLルート・ディレクトリーを持ちますが、それを指定するために使用します

- 例

- 以下は、<mylibrary.so>ファイルをアップデートする例です。

```
$ tdlupdate -m mylibrary
```

- 以下は、複数のモジュールをアップデートする例です。区切り子はコンマ(,)であり、空白なしで作成する必要があります。個数の制限はなく、マルチノードの場合は1024個に制限します。

```
$ tdlupdate -m mylibrary,mylibrary2,mylibrary3
```

- 以下は、アップデートするリストをファイルに作成して、アップデートを実行する例です。ファイル内に指定するモジュールの数に制限はなく、マルチノードの場合は1024個に制限します。

```
$ tdlupdate -l update.list
```

- update.listの作成方法です。以下のように、コンマ(,)や<Enter>キーを区切り子として使用して作成します。

```
mylibrary,mylibrary2
mylibrary3
```

- 以下は、マルチノード環境で共有メモリーの整合性に問題があるときに、同期化を実行する例です。

```
$ tdlupdate -c
```

## ● 参考

VERSION=4に設定した場合には、エクスポート関数を抽出するためのスクリプト・ファイルが必要です。

```
/* Example of exp file */
/* dlib.exp */

#! dlib.so
TmaxSoft::Airplain
Car
```

## 2.26. tencrypt

SVRGROUPセクションのOPENINFOフィールドの構文を暗号化するユーティリティーです。OPENINFOでは、DBの接続IDとパスワードが照会されます。

DB管理者とTP管理者を分離してセキュリティを運用する必要がある状況では、cflを実行すると、DB管理者が必要になることがあるので不便が発生することがあります。tencryptを使用すれば、DB管理者がOPENINFOの値を暗号化してTP管理者に渡し、DB情報が漏れることなく、TP管理者とDB管理者を運用することができます。

tencryptはcflで暗号化した解釈可能な構文を出力する機能であり、以下はその使用方法です。

- 使用方法

```
$ tencrypt [-e 構文 [-d]] [-g]
```

項目	説明
-e 構文 [-d]	<p>暗号化する構文を入力します。</p> <pre>tencrypt -e ORACLE_XA+Acc=P/scott/tiger+SesTm=6</pre> <p>暗号化を使用すると、以下のように画面に出力されます。</p> <pre>Please insert the following text in the configuration file "OPENINFO" field. [@@QEAzMWU4Y0lOYXd2Y2lkSzdpTW1lL0F4U0dlNCswekkxSFlaMjJweld OTERaM2ptVT0jQEBvNVJJPVUE9PQ==@@]</pre> <p>OPENINFOフィールドに[ ]内の内容をコピーします。</p> <pre>OPENINFO = "@@QEAzMWU4Y0lOYXd2Y2lkSzdpTW1lL0F4U0dlNCswekkx SFlaMjJweldOTERaM2ptVT0jQEBvNVJJPVUE9PQ==@"</pre> <p>-dオプションと一緒に使用すると、復号化された結果も一緒に出力します</p>
-g	<p>SEEDを使用するために、すべてのモジュールおよびノードで共通して使用する秘密鍵が保存されているファイルが生成されます。暗号化・復号化のためには、秘密鍵ファイルの作成が必要です。秘密鍵を変更すると、暗号化プロセスを再実行しなければなりません。マルチノードに適用する場合には、同じ秘密鍵が適用されるように、生成された秘密鍵ファイルの内容をコピーして他のノードにも適用しなければなりません。SEEDではない既存の暗号化方式では、秘密鍵ファイルを作成しません</p>

- 適用環境

tencryptだけ別途にコピーして使用することができます。

---

## 参考

cflコマンドについての詳細は「[2.1. cfl](#)」を参照してください。

---



## 2.27. tmaxadmin

Tmaxシステム管理を行うコマンドです。Tmaxシステムの動的な管理のために、コマンド・インタプリタ形式で提供されるモニター・プログラムです。Tmaxシステムが使用する共有メモリーの情報を読み込み、動作中のシステムの環境設定内容やサーバー・プロセスの動作状態、またはサービス状態などを把握できます。

- 使用方法

```
$ tmaxadmin [-l] [-s|m] [-h] [-f [Config File]] [-n [Node Name]]
              [-v] [-V] [-p] [-t]
```

項目	説明
[-l]	racdコマンドを使用して複数ノードで構成されているシステムを中央で集中管理する場合、ローカル・ノードのみを管理するために使用します。それぞれのノードは、このオプションで自身のシステムのみを管理できます
[-s]	読み取り専用モードでそれぞれのtmaxadminツールを10個まで使用できるオプションです。このオプションを使用した場合、環境を動的に変更できません(デフォルト値)
[-m]	マスター・モードで動的に環境を変更できます。1名のユーザーのみがマスター・モードを使用することを推奨します。1名以上のユーザーが環境を変更した場合、深刻なシステム問題が発生する可能性があります。環境ファイルは変更せず、決められたマニュアルに従います
[-h]	コマンドのヘルプ・オプションです
[-f [Config File]]	指定したバイナリ・ファイルを管理します。環境ファイルをデフォルト値のtmconfigではない別名でバイナリ・ファイルを作成した場合、tmaxadminを実行時にバイナリ・ファイル名を指定します
[-n [Node Name]]	特定ノードに対するモニタリングが可能となるようにします。マルチノード環境でtmaxadminを実行する場合にこのオプションを設定すると、指定されたノードの情報のみを確認するため、より利便的にシステムを管理できます
[-v]	Tmaxのバージョンを確認します。このオプションを設定すると、Tmaxの起動と関係なく、どの位置からでもバージョン確認が可能です
[-V]	実行ファイルのバージョンを確認できます
[-p]	st-p、st-s、si、ci、cfgコマンドが実行時に結果が画面単位で出力されるようにします (more機能)
[-t]	コマンドの実行開始および終了時間を出力します。このオプションについての詳細は、下の説明を参照してください

-tオプションを与えてtmaxadminを実行する場合、以下の形式で時間を出力します。

```
[TIME][タイプ] : hh:MM:ss:millisec
```

タイプ	説明
START	コンソールでコマンドを実行時に開始時間を出力します
END	コンソールでコマンドの実行が終了した時に終了時間を出力します (リモート・ノードのコマンドまで終了した後の時間です)
R_END	コンソールでコマンドを実行時に、リモート・ノードの実行が終了した時に終了時間 (tmadminを実行したローカル時間であり、リモート・ノードの時間ではありません) を出力します
RP_START	tmadminのrepeatコマンドを使用して実行する場合、1件のコマンドを実行する開始時間を出力します
RP_END	tmadminのrepeatコマンドを使用して実行する場合、1件のコマンドが終了した時の終了時間を出力します (リモート・ノードのコマンドまで実行した後の時間です)

- 適用環境

Tmaxシステムがインストールされている運用システム環境で使用できます。

- 例

以下は、tmadminプログラムが実行状態であることを表示する例です。

```
$ tmadmin
```

tmadminを実行すると、以下のメッセージと一緒にプロンプトが現れます。

```
--- Welcome to Tmax Admin (Type "quit" to leave) --- $$1 (tmadm):
```

## 参考

cfl、tmboot、tmdownコマンドについての詳細は、それぞれ[「2.1. cfl」](#)、[「2.31. tmboot」](#)、[「2.33. tmdown」](#)を参照してください。

## tmadminツールで利用できるコマンド

以下は、tmadminを実行後に使用できるコマンドです。

- 環境情報の照会

コマンド	説明
tmaxinfo(ti)	Tmaxシステム情報を確認します
history(hist)	以前に保存されたコマンドを照会します
config(cfg)	環境設定内容を照会します

- 動作状態情報の照会

コマンド	説明
stat(st)	プロセスおよびサービス状態の統計を照会します。ワイルドカード(Wild card)を使用できます
gwinfo	GATEWAYセクションに設定したゲートウェイのチャンネル状態を確認します
txgwinfo(txgwi)	Tmaxゲートウェイの情報を確認します
nontxgwinfo	Tmax Nontランザクション・ゲートウェイの情報を確認します
jgwinfo	JEUSゲートウェイの情報を確認します
ajgwinfo	JEUS Asyncゲートウェイの情報を確認します
wsgwinfo	Webサービス・ゲートウェイの情報を確認します
smtrc	GIDを使用して該当サービスの実行状態を照会します
clhinfo	マルチノード環境でCLH間の接続状態情報を確認します
tmmsinfo	マルチノード環境でTMM間の接続状態情報を確認します
repeat(r)	コマンドを反復します
clientinfo(ci)	接続クライアントを確認します
svrinfo(si)	サーバー情報を確認します。ワイルドカード(Wild card)を使用できます
txquery(txq)	トランザクションの処理情報を確認します
rqstat(rqs)	RQの状態を確認するか、ディスク・キューに蓄積されているサービスを処理します

- 運用管理

コマンド	説明
suspend(sp)	動作中のサーバー・プロセスを中止します
resume(rs)	中止されたサーバー・プロセスを再開します
advertise / unadvertise	特定サービス名をadvertise/unadvertiseします
restat	特定サーバー・プロセス、またはすべてのサーバー・プロセスの統計情報を初期化します
rebootsvr(rbs)	サーバー・プログラムを交替します
cfgadd(ca)	動的でサービスを追加します
set	現在設定されている環境設定値を動的に変更します
qpurge(qp)	キューに蓄積したアプリケーションを削除します
discon(ds)	接続中のクライアントを強制的に解除します
logstart / logend	ロギングを開始または終了します
chtrc	トレース管理を行います

コマンド	説明
chlog	TMM、CLH、特定サーバーのログレベルをランタイム中に動的に変更します
txcommit / txrollback	トランザクション処理中に障害が発生した場合、コミットまたはロールバックを再発行して該当トランザクションを終了する機能です
wsgwreload	Webサービス・ゲートウェイのサービス情報、設定変更を適用します
restart	サーバー・プロセスを再起動します

- その他

コマンド	説明
!	直前に使用したコマンドを反復します
quit(q)	tmadminを終了します
help(h)	使用可能なオプション一覧を照会します
nodeset(ns)	マルチノード環境で特定ノードに対する情報のみを取得する場合に使用します
nodeunset(nus)	マルチノード環境で特定ノードに対する情報のみを取得する設定を解除します
tmd	仮想のクライアント・エミュレーターとしてサービス・プログラムの有効性を検討する場合、クライアント・プログラムを別途作成せず、このユーティリティを使用します

## 参考

各コマンドの使用方法、処理結果の詳細については、『*Tmax 運用ガイド*』の「5.1 tmadmin」を参照してください。

## 2.28. tmapm

Tmaxシステムでサービス・タイムアウトはシグアラムを使用して動作します。tmapmは、サービスのシグアラムを再定義したり、シグアラムが使用できないサービスにサービス・タイムアウトを提供するサーバーです。

tmapmはUSCサーバーであり、環境設定ファイルのSERVERセクションに以下のように設定します。

- 使用方法

```
*SERVER
tmapm      CLOPT = [ -i sec ]
                [-r sec ]
                [-c command ]
                [-l [ 0 | 1 | 2 ]]
```

項目	説明
[ -i sec ]	サービス・タイムアウトをチェックする周期(秒)です。周期が短いほど、システムの負荷が増加します。小数点も設定できます
[ -r sec ]	サービス・タイム以降コマンドを実行するまでの制限時間(秒)です。整数型で設定します
[ -c command ]	サービスが実行時間(SCVTIME + コマンド実行制限時間)までRUNNING状態である場合に実行するコマンドです。シェル・コマンドやスクリプトを使用できます
[ -l [ 0   1   2 ] ]	ユーザー・ログのレベルです。デフォルト値は0であり、数値が高いほど、より詳細な情報を出力します

[-c]オプションで設定したコマンドを実行する場合、以下のようにパラメータを渡します。

User Command	PID	SVRNAME	SVCNAME[Elapse Time]
kill.sh	9659	svr2	TOUPPER[5]

- 例

以下は、tmapmの設定例です。

```
*SERVER
tmapm      SVGNAME = svg,
           SVRTYPE = UCS,
           CLOPT = "-o ulog -- -i 5 -r 1 -c kill.sh -l 1"
```

## 2.29. tmaxlibver

Tmaxライブラリーのバージョン情報を照会するコマンドです。UNIX用のライブラリーを使用する場合、該当ライブラリーのみではTmaxのバージョンを確認できません。Tmax v5.0 SP1以降では、tmaxlibverというコマンドを使用してTmaxライブラリーのバージョンを確認できます。

CFLAGS環境変数を設定した場合、tmaxlibverの実行時にCFLAGSに設定されたコンパイル・オプションを使用します。

- 使用方法

```
$ tmaxlibver {-l filename} {-d | -s} [-6] [-L directory] [-o arg] [-h]
```

項目	説明
{ -l filename }	照会するライブラリー名を指定します
{ -d   -s }	ライブラリーのdynamic(-d)またはstatic(-s)の可否を設定します

項目	説明
[-6]	ライブラリーが64ビットの場合に設定するオプションです。設定していない場合、32ビットで動作します。-6が設定され、[-L]オプションが設定されていない場合、\${TMAXDIR}/lib64ディレクトリーに位置したライブラリーを自動参照します
[-L <i>directory</i> ]	照会するライブラリーが位置する絶対パスまたは相対パスを入力します。指定していない場合、[-6]オプションに従ってデフォルト・パスが変わります
[-o <i>arg</i> ]	ユーティリティーを実行時に生成される一時ファイルのパスを設定しない場合、ユーティリティーは「/tmp/tmaxlibver_stub」という名前でファイルを作成し、実行を終了時に削除します
[-h]	コマンドのヘルプ・オプションです

- 例

```
$ tmaxlibver -l libsvr.a -s -6
libsvr.a for TMAX Version 5.0 SP #1 64bit binary for AIX 5L
```

#### 注

DB stub、TCacheなどの一部ライブラリーでは、バージョン情報の照会機能を使用できません。

## 2.30. tmaxtrace

Tmaxアプリケーション管理者や開発者がアプリケーションのランタイム・トレースを行えるコマンドです。ランタイム・トレースは、アプリケーションを実行している間の操作記録を残すトレース・ポイントの概念に基づいています。トレース・ポイントの例として、tpcall()のようなatmi関数の開始や終了、トランザクションの開始などが挙げられます。

以下は、トレース・ポイントの発生時に現れる現象です。

1. トレース・ポイントが有効であるかを確認するためにフィルターが適用されます。トレース・ポイントが有効である場合、トレース・レコードが受信ファイルに記録され、最後にアクションが起きます。アクションは、プロセスを終了(abort)させたり、システム・コールを呼び出したりするなど、様々です。これは選択オプションです。
2. **Filter**、**Receiver**、**Trigger**項目は、下で説明するスペックに従って定義されます。これらの項目は、**TMAX\_TRACE**環境変数を宣言した場合に定義することができ、クライアントとサーバーの両方に設定できます。

実行中のプロセスのスペックを変更するには、**tmaxadmin**ツールの**chtrc**コマンドを使用します。

3. TMAX\_TRACEのスペックのうち、Triggerオプションをdyeに設定した場合、該当プロセスの要求を受信したサーバーのプロセスもランタイム・トレースが実行されます。クライアントでdyeを設定した場合は、該当サービスを実行するプロセスがatmi categoryになり、ランタイム・トレースが実行されます。
4. クライアントでTMAX\_TRACEを使用する場合は、クライアント環境設定ファイル(tmax.env)または環境変数にULOGPFXを必ず指定します。下記のように指定した場合、該当ディレクトリーに<clilog.日付>ファイルが生成され、このファイルにログが蓄積されます。

```
ULOGPFX=/data1/tmax50/client/clilog
```

以下は、TMAX\_TRACEの使用方法についての説明です。

#### ● 使用方法

```
TMAX_TRACE=filter-spec:receiver-spec[:trigger-spec]
```

– filter-spec : category

特定のカテゴリに対してランタイム・トレースを実行するかどうかを決定します。すべてのカテゴリを表現するためにアスタリスク(\*)を設定できます。filter-specに値を設定していない場合、カテゴリが何も選択されていないことを意味します。

以下は、カテゴリの一覧です。

カテゴリ	説明
atmi	ユーザーが呼び出したATMIおよびTXインターフェース(tpまたはtxで始まる関数)を呼び出した場合に、ランタイム・トレースが実行されます
iatmi	ユーザーが呼び出したATMIおよびTXインターフェース以外に、インターフェースの内部で呼び出されたインターフェースまでもランタイム・トレースが実行されます
xa	すべてのXAインターフェースを呼び出した場合、ランタイム・トレースが実行されます
trace	Tmaxトレースの設定情報を表示します
gqs	ユーザーが呼び出したGQインターフェース(gqで始まる関数)を呼び出した場合、ランタイム・トレースが実行されます
tdl	ユーザーが呼び出したTDLインターフェース(tdlで始まる関数)を呼び出した場合、ランタイム・トレースが実行されます

カテゴリを設定する際、複数の項目をパイプライン(|)区切り子で指定することができます。

```
atmi|xa|tdl
```

– receiver-spec : [/regular-expression/]receiver

レシーバーは、トレース・レコードが送信されるファイルです。現在レシーバーは、ulogのみが使用されます。ulogが設定された場合、ユーザー・ログにランタイム・トレースが実行されます。regular-expression

が使用された場合、filter-specに適用されるトレース・ポイントを制限できます。receiver-specを設定していない場合、トレース・レコードも記録されません。

項目	説明
trace record	レシーバーに「process-name.pid.hhmmss: TMAXTRACE:cc data」の形式で記録されます。  ccは、カテゴリーによって異なります。 <ul style="list-style-type: none"><li>– atmiの場合: at</li><li>– iatmiの場合: dt</li><li>– xaの場合: xa</li><li>– traceの場合: tr</li></ul>

- trigger-spec : [/regular-expression/]action

トリガーは、トレース・レコードがレシーバーに送信された後のアクションを設定するオプション項目です。

以下は、アクションに設定できる動作についての説明です。

項目	説明
abort	abort()を呼び出し、プロセスを終了させます
ulog(message)	ユーザー・ログにメッセージを書き込みます
system(command)	system()を使用してコマンドを実行します (Windowsではサポートされていません)
trace(trace-spec)	trace-specでTmaxトレースのスペックを変更します
dye	メッセージをdyeingします
undyed	メッセージをdyeingしません
sleep(seconds)	設定した秒の間待機します。Windowsではサポートされていません

- 例

- 以下は、atmiインターフェースを呼び出した場合、すべてランタイム・トレースが実行される例です。

```
export TMAX_TRACE=atmi:ulog
```

- 以下は、atmi/txインターフェースのうち、tpcall()を呼び出した場合にのみ、ランタイム・トレースが実行される例です。

```
export TMAX_TRACE=atmi:/tpcall/ulog
```



- 以下は、atmi/txインターフェースが呼び出された場合、abort()を使用して該当サーバー・プロセスを終了する例です。

```
export TMAX_TRACE=atmi:uolog:abort
```

- 以下は、atmi/txインターフェースを呼び出した場合、すべてランタイム・トレースが実行されますが、tpend()を呼び出した場合には、該当プロセスはabort()によって終了される例です。

```
export TMAX_TRACE=atmi:uolog:/tpend/abort
```

- 以下は、すべてのカテゴリーに対してランタイム・トレースが実行される例です。

```
export TMAX_TRACE=*:uolog:dye
```

- 以下は、atmi/txインターフェースを呼び出した後に、trace specが「\*:uolog:dye」に修正される例です。

```
export TMAX_TRACE=atmi:uolog:trace(*:uolog:dye)
```

- 以下は、tpalloc()の呼び出し後に、trace specが「\*:uolog:dye」に修正される例です。

```
export TMAX_TRACE=atmi:uolog:/tpalloc/trace(*:uolog:dye)
```

- 以下は、「TMAX\_TRACE=atmi:uolog:dye」と同じ例です。

```
export TMAX_TRACE=on
```

- 以下は、tmaxtraceを使用しない例です。(デフォルト値)

```
export TMAX_TRACE=off
```

## 2.31. tmboot

Tmaxシステムの全体または一部分を実行させるコマンドです。Tmax環境ファイルをベースにシステムを実行します。オプションなしで実行した場合、あるいは[-f]オプションのみを使用した場合、すべてのTmax管理プロセスとTmax環境ファイルのSERVERセクションに登録されているすべてのサーバー・プロセスを実行させます。NODEセクションに登録されているすべてのノードでTmax管理プロセスのTMM、CLL、CLHプロセスが順番に実行されます。

SVRGROUPセクションにOPENINFO項目が登録されているサーバー・グループが存在する場合、各サーバー・モジュール別にTMSNANEとMINTMS項目を参照し、TMSプロセスが実行されます。Tmax管理プロセスは、ノード別に定義されたTMAXDIRディレクトリー下位のbinディレクトリーで実行されます。

Tmax管理プロセスの生成後、SERVERセクションのすべての応用サーバー・プロセスが実行されます。応用サーバー・プロセスは、SERVERセクションに登録されている順番に実行されます。tmbootコマンドは実

行されたサーバー・プロセスの初期化を実行後に、次のサーバー・プロセスを実行させます。プロセスの初期化は、tpsvrinit()を使用して進めます。すべてのサーバー・プロセスが初期化を終えるまでは、次のプロセスが実行されません。tmbootコマンドは、各応用サーバー・プロセスをそれらのMIN項目に定義された個数の分だけ実行させます。MIN項目が定義されていない場合、デフォルト値は1個です。

tmbootコマンドは、SERVERセクションのサーバーに対して、**CLOPT**、**MIN**、**MAX**項目の値を使用します。サーバー・プロセスの実行時、tmbootコマンドによって使用されるサーバーのbootパラメータです。サーバーの残りの項目は、サーバーの実行後にシステムによって実行されるruntimeパラメータです。設定情報は、ソース設定ファイルのSERVERセクションを参照してください。

すべての応用サーバー・プロセスは、動作するノードに定義されているAPPDIRディレクトリーで実行されます。

#### ● 使用方法

```
$tmboot [-A] [-b] [-c] [-f バイナリTmax環境ファイル名]
        [-g servergroup_name [-a]] [-h] [-i] [-V]
        [-n node_name] [-o clopt_string] [-q RQ svg_name]
        [-s server_name [-k count] [-a]]
        [-S server_name [-a]] [-t TMS_name[-k all]]
        [-B TRB_node] [-T] [-w] [-d boot_time] [-D]
        [-e clh | cas | tlm] [-k] [-R]
```

項目	説明
[-A]	Tmax環境ファイルの <b>SERVER</b> セクションに定義されたすべての応用サーバー・プロセスを実行します
[-b]	バックアップで指定されたサーバー・プロセスを任意で起動させる際に使用します
[-c]	CLHプロセスをもう1つ実行させるオプションです。  現在動作中のCLHプロセス数がTmax環境ファイルに定義された <b>MAXCLH</b> 値を超えない範囲内でのみ使用可能です
[-f Tmax ]	参照するバイナリTmax環境ファイル(ソース設定ファイルをcflでコンパイルした結果)をパスと一緒に指定する必要があります。  [-f]オプションが省略された場合、デフォルトとしてTMAXDIRディレクトリー下位のconfigディレクトリーで <b>tmconfig</b> ファイルを参照します
[-g servergroup_name [-a]]	指定されたサーバー・グループに存在するサーバー・プロセスを実行するオプションです。  使用されるサーバー・グループ名は、Tmax環境ファイル内の <b>SVRGROUP</b> セクションに登録します。[-a]オプションと一緒に使用する場合、サーバーの起動をTMBOOTプロセスではなく、TMMプロセスに委任します

項目	説明
[ -h ]	コマンドのヘルプ・オプションです
[ -i ]	特定のサーバーがmaxまで達しても、起動を停止せずに起動されていない残りのサーバーを起動する場合に使用します
[ -V ]	実行ファイルのバージョンを確認できます
[ -n <i>node_name</i> ]	指定されたノードに存在するサーバー・プロセスを実行させるオプションです。ノード名はTmax環境ファイル内の <b>NODE</b> セクションに事前に登録されている必要があります
[ -o <i>clopt_string</i> ]	CLOPT stringを追加します
[ -q <i>RQ svg_name</i> ]	RQSを始動させます
[ -s <i>server_name</i> [-k <i>count</i> ] [-a] ]	<p>指定されたサーバー・プロセスのみを実行させます。</p> <p>使用されるサーバー・プロセス名は、Tmax環境ファイル内の<b>SVRGROUP</b>セクションに事前に定義されている必要があります。</p> <p>[-k]オプションと一緒に使用してサーバー・プロセス数を指定できます。サーバー・プロセス数は、現在実行されている数を含み、SERVERセクションのMAX項目に定義された数を超えてはいけません。</p> <p>[-k]オプションを省略した場合、該当サーバー・プロセスは1つのみ実行されます。[-a]オプションと一緒に使用する場合、サーバーの起動をTMBOOTプロセスではなく、TMMプロセスに委任します</p>
[ -S <i>server_name</i> ]	<p>指定されたサーバー・プロセスをMINの数だけ実行します。</p> <p>[-a]オプションと一緒に使用する場合、サーバーの起動をTMBOOTプロセスではなく、TMMプロセスに委任します</p>
[ -t <i>TMS_name</i> [-k all] ]	<p>指定されたTMSプロセスをもう1つ実行します。</p> <p>Tmax環境ファイルに定義された<b>MAXTMS</b>値を超えない場合にのみ可能です。</p> <p>[-k all]は、トランザクション・リカバリー機能を使用する場合に必要なオプションです。グループ全体を終了後に起動する場合、リカバリーが実行されるため、リカバリーはTMSグループ単位で可能です。特定名をもつTMS全体を起動/終了する場合、このオプションを使用できます</p>
[ -B <i>TRB node</i> ]	TRBノードを起動させます
[ -T ]	Tmaxシステムのプロセス(TMM、CLL、CLH、TMS)だけを実行させます
[ -w ]	オプションがない場合は、登録されているサーバー・プロセスを同時に起動させます。この場合、OSによっては同時にリソースを作成できず、サーバー・プロセスが正常に起動しない場合が発生します。この問題を解決するために、プロセスを1つずつ起動させ、正常に起動させるオプションです。

項目	説明
	<ul style="list-style-type: none"> <li>– サーバー・プロセスがTMMIに接続時のLOCK使用条件(LOCK   NOLOCK)</li> <li>– サーバー・プロセスが起動時のWAIT条件(NO-WAIT   FINITE-WAIT) <ul style="list-style-type: none"> <li>• 「-d -1000000 (1sec)」と同一効果を持ちます</li> <li>• [-d]オプションがない場合にのみ意味があります</li> <li>• [-d]オプションが使用された場合、[-w]オプションは無視されます</li> </ul> </li> </ul>
[ -d boot_time]	<p>一度に多くのサーバー・プロセスを起動させた場合、CLHでの登録要求が集中します。その際に生じる問題を解決するために、サーバー・プロセスの起動にかかる総時間を指定して、登録間隔を調節できます</p> <p>(デフォルト値: LOCK、NO-WAIT、単位: usec)</p> <ul style="list-style-type: none"> <li>– サーバー・プロセスがTMMIに接続時のLOCK使用条件(LOCK   NOLOCK)</li> <li>– サーバー・プロセス起動時のWAIT条件(NO-WAIT   FINITE-WAIT) <ul style="list-style-type: none"> <li>• -d val &lt; 0 : LOCK,  VAL  FINITE-WAIT *</li> <li>• -d val = 0 : NO-LOCK, NO-WAIT</li> <li>• -d val &gt; 0 : NO-LOCK,  VAL  FINITE-WAIT</li> <li>• 基本的に[-d]オプションのvalが0でない場合、絶対値( VAL )を使用し、単位はusecです</li> </ul> </li> <li>– FINITE-WAITの場合、 VAL 値は各プロセスごとに最大のWAIT時間です (全体プロセスの総WAIT時間ではない)</li> <li>– サーバー・プロセスがシグナルを与えた場合、WAITは解除されます。つまり、 VAL 時間になっていなくても、シグナルを受けると次のプロセスの起動を行います。VALが負数の場合はLOCKを使用します。このオプションが使用された場合、[-w]オプションは無視されます</li> </ul>
[ -D]	[-d]オプションとほぼ似ていますが、finite-waitのときにシグナルが来ても VAL までは必ず待機します
[ -e clh   cas   tlm ]	<p>Tmaxエンジン・プロセスのうち、CLH、CAS、TLMを起動するためのオプションです。tmbootの実行時に問題が発生した場合や、killによってエンジン・プロセスが異常終了した場合に使用します。</p> <p>tmdownコマンドでは、このオプションを提供していません</p> <ul style="list-style-type: none"> <li>– clh : CLHを起動します</li> </ul>

項目	説明
	<ul style="list-style-type: none"> <li>– cas : CASを起動します</li> <li>– tlm: TLMを起動します</li> </ul>
[ -R ]	リモート・シェル(rshまたはremsh)を使用して、リモート・ノードのTmaxシステムを起動させる場合、[-R]オプションを使用します

## ● 適用環境

Tmaxシステムがインストールされている運用システム環境で使用できます。

## ● 例

- 以下は、TMAXDIRディレクトリー下位のconfigディレクトリーにあるtmconfigファイルを参照し、Tmaxプロセスと応用サーバー・プロセスをすべて実行する例です。

```
$ tmboot
```

- 以下は、tms\_nameで生成されたすべてのTMSが起動する例です。

```
$ tmboot -t tms_name -k all
```

- 以下は、サーバー・グループの設定オプションを使用した例です。互いに異なるサーバー・グループでtms\_nameを同一に使用する場合、以下のように、[-g]オプションで該当TMSが属しているサーバー・グループ名を指定します。そうでない場合、該当TMS名をもつ最初のサーバー・グループのTMSが起動します。

```
$ tmboot -t tms_name -k all -g svgname
```

- 以下は、tmconfig環境ファイルを参照し、SERVERセクションに定義されたすべての応用サーバー・プロセスを実行する例です。

```
$ tmboot -A
```

- 以下は、/user1/tmax/conディレクトリーのexconfig環境ファイルを参照し、NODEセクションに登録されているcosmoノードに存在する応用サーバー・プロセスを実行する例です。

```
$ tmboot -n cosmo -f /user1/tmax/con/exconfig
```

- 以下は、tmconfig2環境ファイルを参照し、svr1プロセスを5つ実行する例です。

```
$ tmboot -s svr1 -k 5 -f tmconfig2
```

---

## 参考

cfl、tmboot、tmdownコマンドについての詳細は、それぞれ[「2.1. cfl」](#)、[「2.31. tmboot」](#)、[「2.33. tmdown」](#)を参照してください。

---

## tmbootコマンドを実行時のtmconfigパス参照

tmbootコマンドでTmaxを起動する場合、バイナリ環境ファイル\${TMAXDIR}/config/tmconfigを\${TMAXDIR}/path/tmconfigにコピー後、\${TMAXDIR}/pathディレクトリーにある環境ファイルを使用します。

しかし、すでにTmaxエンジンが起動している状況でtmboot -Sまたは-sを使用して特定サーバーのみを起動しようとした場合にも\${TMAXDIR}/config/tmconfigを参照した場合、以下のような環境で問題になることがあります。

- 環境ファイル

<既存の運用環境ファイル>

```
*SVRGROUP
svg1      NODENAME = "tmaxh4"
*SERVER
svr1      SVGNAME = svg1
svr2      SVGNAME = svg1
*SERVICE
TOUPPER1  SVRNAME = svr1
TOUPPER2  SVRNAME = svr2
```

<運用中に変更された環境ファイル>

```
*SVRGROUP
svg1      NODENAME = "tmaxh4"
*SERVER
svr1      SVGNAME = svg1
svr3      SVGNAME = svg1
svr2      SVGNAME = svg1
*SERVICE
TOUPPER1  SVRNAME = svr1
TOUPPER3  SVRNAME = svr3
TOUPPER2  SVRNAME = svr2
```

以下のように、運用中に変更された環境ファイルをCFLで再コンパイルします。

```
$ cfl -i node1.m
```

新規追加されたサーバーを起動します。

```
$ tmboot -S svr3
```

運用中の環境で環境ファイルを変更後、tmboot-Sを実行しようとした場合、以下のようなエラーが発生することがあります。

```
(E) BOOT3007 maxsvr (1) is over for svr(svr3:svr2): nodeno = 0, svri = 5, cur = 1, ksvr = 1 [BOOT0015]
```

運用環境でCFLを実行した場合、\${TMAXDIR}/config/tmconfigは変更された内容が適用されます。しかし、実際の共有メモリには変更前の\${TMAXDIR}/path/tmconfigと同一構成されているため、tmboot-Sで新規追加したサーバー・プロセスが起動時、実際にはすでに実行中のサーバー・プロセスを追加で起動します。Tmaxエンジンが起動している状況でのCFLは許容されませんが、このミスによるエラーが発生した場合、該当エラーはデバッグが難しいです。

したがって、Tmaxエンジンの動作中に、tmboot [-S]、[-s]、[-g]、[-q]、[-t]、[-A]などのオプションで各サーバーを起動する場合に参照するtmconfigのパスは、\${TMAXDIR}/config/tmconfigではなく、\${TMAXDIR}/path/tmconfigです。ただし、エンジンを起動する場合は\${TMAXDIR}/path/tmconfigを参照します。

```
$ cfl -i new_config.m -o tmchg  
$ tadmin : cfgadd -I tmchg  
$ tmboot -S new_svr -f tmchg
```

---

#### 注

tmbootコマンドを実行時、[-f]オプションを使用して特定バイナリ環境ファイルを指定した場合、\${TMAXDIR}/config/tmconfigを参照してサーバーを起動します。動的にサーバーを追加した場合は、必ず[-f]オプションで特定環境ファイルを指定し、\${TMAXDIR}/configの変更されたバイナリ環境ファイルを参照するようにします。

---

## 2.32. tmd

サーバー・プログラムをテストするために、クライアントをシミュレーションするコマンドです。プログラマーは、直接クライアント・プログラムを作成しなくても、簡単にサーバー・プログラムの動作状態を確認できます。

---

#### 参考

tmdコマンドは、Windows NTおよびWindows 2000環境では動作しません。

---

tmdコマンドでサポートするバッファ・タイプは、STRING、CARRAY、FIELD、構造体バッファです。CARRAYバッファの場合、印刷可能なデータに対してサポートします。STRINGやCARRAYデータは空白をデータの終わりとしみなすため、継続したデータを表示するためには、二重引用符(" ")でデータを処理します。構造体バッファの場合、単一構造体に対してサポートしており、送受信構造体タイプが一致する必要

があります。すなわち、受信したバッファ・タイプと同じバッファを tpreturn()関数内のパラメータで使  
います。

そうでない場合、以下のようなエラー・メッセージが出力されます。

```
(E) 3004 not supported output type
```

## ● 使用方法

```
$ tmd [-X] [-T 時間(秒)] [-t 時間(秒)] [-s] [-V] [-i file_name] [-l size] [-q]
```

項目	説明
[-X]	トランザクションの使用有無を指定します。フラグが指定された場合、与えられたサービスをトランザクションと見なして処理します
[-T 時間(秒)]	トランザクション・タイムアウトを指定します。tx_set_transaction_timeout(時間)と同じ機能を実行します
[-t 時間(秒)]	ブロック・タイムアウトを指定します。tpset_timeout(時間)と同じ機能を実行します
[-s]	tpstart情報をコンソールで入力を受けられるようにし、ユーザーの情報がシステムに渡されるようにする機能を実行します。オプションを設定した場合、tmdコマンド実行時、domain passwordとuser name、user passwordの入力を受け付けます
[-V]	実行ファイルのバージョンを確認できます
[-i file_name]	呼び出すサービスを定義したファイルです
[-l size]	コマンドにより入力されるデータの最大サイズを指定します(デフォルト値: 1024 バイト)
[-q]	引用符(' ')が含まれた文字列型のデータを処理できます。文字列データに含まれた「\」を「"」に認識して処理します

## ● 例

- 以下は、tmdコマンドを使用した例です。

```
$ tmd -i input_data
$ tmd -i tmdTest
$ tmd -i tmd.sh -s
```

- 以下は、フィールド・キー・バッファを使用する例です。フィールド・キー・バッファはアスタリスク(\*)で開始し、1行目は、関数名、サービス名、バッファ・タイプによって構成されます。次の行には、フィールド名とデータが並び、サービス実行の意味で最後の行に改行を入れます。

```
Input File ( input_data ) 作成
*tpcall BR_ADD FIELD
```



```
BRANCH_ID      1
LAST_ACCT      9999
LAST_TELLER    99
ADDRESS        "25 Powell St. San Francisco, CA 94188"
PHONE          415-753-9000
(Newline)
```

- STRINGバッファを使用する場合は、バッファ・タイプにFIELDの代わりにSTRINGを入力します。

<tmdTestファイルの作成例>

```
*tpcall TOUPPER STRING
abc
(Newline)
```

## 2.33. tmdown

Tmaxシステム全体、あるいは一部分を終了させるコマンドです。tmdownコマンドはTmax環境ファイルに基づいてTmaxシステムを終了させるため、基本的に[-f]オプションを使用し、参照するバイナリTmax環境ファイル(ソース設定ファイル)をcflコマンドでコンパイルした結果をパスと一緒に指定する必要があります。[-f]オプションが省かれた場合、デフォルト値としてTMAXDIRディレクトリー下位のconfigディレクトリーにあるtmconfigファイルを参照します。

[-f]以外のオプションが使用されていない場合、tmdownコマンドはTmaxのすべての管理プロセスとTmax環境ファイルのSERVERセクションに登録されているすべてのサーバー・プロセスを終了させます。そして、Tmaxシステムと関連したIPCリソースを削除します。終了の順序は、以下のとおりです。

1. SERVERセクションに登録されている応用サーバー・プロセスが終了します。
2. サーバー・グループ別にTMSプロセスが動作中の場合は、TMSプロセスが終了します。
3. Tmaxシステム管理プロセスが終了します。プロセスの終了は、**CLH**、**CLL**、**TMM**の順に行われます。これが一般的なシステム管理プロセスの終了順序ですが、CLHのMIN値が1でない場合は、CLLがCLHより先に終了することもあります。

バックアップで起動されたサーバーに動的に登録されているサービスがある場合に、このサービスがすべてダウンして、動的に登録されているサービスが共有メモリーから消える場合を仮定します。この場合、障害復旧後(バックアップ・サーバーがすべて終了し、正常ノードのサーバーが再起動している状態)、バックアップ・ノードに接続したクライアントに対してnamingサービスを提供できなくなります。したがって、バックアップ・サーバーの動的サービスは、該当サーバーがすべて終了しても、共有メモリーから削除されないように処理します。

- 使用方法

```
$tmdown [-A] [-f バイナリTmax環境ファイル名] [-g servergroup_name] [-h] [-V][-i]
        [-n node_name] [-p server_num] [-q RQ svg_name]
        [-s server_name [-k count]] [-S server_name] [-t TMS_name[-k all]]
        [-w wait_time] [-B Nodename][-R] [-y]
```

項目	説明
[-A]	すべての応用サーバー・プロセスを終了させます
[-f バイナリTmax環境ファイル名]	システム終了時に参照するバイナリTmax環境ファイルの名前を指定する項目です。ファイル名を指定していない場合、デフォルトでTMAXDIRディレクトリ下位のconfigディレクトリーにある <b>tmconfig</b> ファイルを参照します
[-g servergroup_name]	指定されたサーバー・グループに属するサーバー・プロセスを終了します
[-h]	コマンドのヘルプ・オプションです
[-V]	実行ファイルのバージョンを確認できます
[-i]	tmdownコマンドを即時実行します。基本的にtmdownコマンドは該当アプリケーションをすべて終了して実行されますが、[-i](immediately)オプションによる終了は、現在実行中のアプリケーションをすべて中断するため、慎重に使用する必要があります
[-n node_name]	指定されたノードに存在するすべてのサーバー・プロセスを終了させます。ノード名は、Tmax環境ファイルの <b>NODE</b> セクションに登録されている必要があります
[-p server_num]	指定されたサーバー・プロセスを終了させます。[-s]オプションによる終了とは異なり、tmadminで「st-p」コマンドで確認できるプロセス番号(spr_no)を使用して特定プロセスを終了させます
[-q RQ svg_name]	RQSを終了させます
[-s server_name [-k count]]	指定されたサーバー・プロセス1つのみを終了させます。使用されるサーバー・プロセス名は、Tmax環境ファイル内のSERVERセクションにあらかじめ登録されている必要があります
[-S server_name]	指定されたサーバー・プロセスをすべて終了させます。1つ以上のプロセスが指定されている場合、該当するすべてのプロセスを終了させます
[-t TMS_name [-k all]]	指定されたTMSプロセスを1つのみ終了させます。  [-k all]は、トランザクション・リカバリー機能を使用する場合に必要なオプションです。グループ全体を終了後に起動する場合、リカバリーが実行されるため、リカバリーはTMSグループ単位で可能です。特定名をもつTMS全体を起動/終了する場合、このオプションを使用できます
[-w wait_time]	wait_timeに指定された時間が過ぎた場合、tmdownコマンドを実行します
[-B Nodename]	TRBノードを終了させます
[-R]	Rolling Down時に使用します

項目	説明
[ -y ]	tmdown時に、終了の可否(y   n)を聞かずに即終了させます

● 例

- 以下は、TMAXDIRディレクトリー下位のconfigディレクトリーにあるtmconfigファイルを参照し、全体Tmaxシステムを終了する例です。つまり、Tmax管理プロセスと応用サーバー・プロセスをすべて終了します。

```
$tmdown
```

- 以下は、名前がtms\_nameのすべてのTMSが終了する例です。

```
$tmdown -t tms_name -k all
```

- 以下は、サーバー・グループの設定オプションを使用した例です。互いに異なるサーバー・グループでtms\_nameを同一に使用する場合、上記のように、[-g]オプションで該当TMSが属しているサーバー・グループ名を指定します。

そうでない場合、該当TMS名をもつ最初のサーバー・グループのTMSが終了します。

```
$tmdown -t tms_name -k all -g svgname
```

- 以下は、tmconfig2環境ファイルを参照し、全体Tmaxシステムを終了する例です。

```
$tmdown -f tmconfig2
```

- 以下は、tmconfig環境ファイルを参照し、svr1という応用サーバー・プロセスをすべて終了する例です。

```
$tmdown -S svr1
```

- 以下は、tmconfig環境ファイルを参照し、svr1という応用サーバー・プロセスをすべて強制終了する例です。svr1のうち、特定サーバー・プロセスのサービスが終了していない場合、[-i]オプションを使用して強制終了します。

```
$tmdown -S svr1 -i
```

- 以下は、tmconfig環境ファイルを参照し、<spr\_no>のサーバー・プロセスのみ強制終了する例です。該当サーバーのサーバー・プロセスが複数の場合、特定サーバーがloopingの場合、該当プロセスのみ強制終了します。

```
$tmdown -k <spr_no> -i
```

- 以下は、/user1/tmax/conディレクトリーのexconfig環境ファイルを参照し、NODEセクションに登録されているcosmoノードに存在する応用サーバー・プロセスを終了する例です。

```
$tmdown -n cosmo -f /user1/tmax/con/exconfig
```

- 以下は、tmconfig2環境ファイルを参照し、動作中のsvr1プロセス1つのみを終了する例です。

```
$tmdown -s svr1 -f tmconfig2
```

## Rolling Down機能

既存バージョンでは、クライアントの要求を処理していたTmaxシステムが異常終了した場合、現在処理中の要求に対してのみ応答を処理して渡した後、キューに蓄積している要求に対してTPECLOSEエラーを渡していました。しかしTmax v5.0からは、Tmaxエンジンをダウンさせる前に、要求されたすべてのクライアントに対して正常に応答するRolling Down機能を提供します。

- 使用方法

```
$ tmdown -R -n node_name
```

項目	説明
-n node_name	終了されるノード名を指定します

- 適用環境

Tmaxシステムがインストールされている運用システム環境で使用できます。

- 例

以下は、NODEAとNODEBがマルチノード(またはマルチドメイン)で構成されており、総100個のクライアントがNODEAに接続されていると仮定した場合の処理過程を説明する例です。

- NODEAのTmaxシステムを終了させる場合

```
$tmdown -R -n NODEA
```

1. NODEAのCLLは、クライアントからのListenポートを遮断します。
2. NODEAで既存処理されていた要求に対する処理の完了後、クライアントに処理結果を渡します。
3. キューに蓄積している要求に対しては、TMAX\_BACKUP\_ADDRと設定されたNODEBに要求を送ります。
4. NODEAのTmaxシステムが終了します。
5. NODEBでは、NODEAから受信した要求を処理後、初めに該当リクエストを要求したクライアントに処理結果を直接渡します。

6. NODEAに接続されているすべてのクライアントは、正常に応答を受信します。

– NODEBのTmaxシステムを終了させる場合

```
$tmdown -R -n NODEB
```

1. 100個のクライアント要求を、NODEAのCLHが約50:50でNODEAとNODEBに分配されます。
2. tmdown -R -n NODEBで、NODEBのTmaxシステムを終了させます。
3. NODEBのCLLは、クライアントからListenポートを遮断します。
4. NODEBで既存処理されていた要求に対しては、処理を完了します。
5. クライアントはNODEAに接続されている状況のため、該当処理結果をNODEAのCLHに渡し、NODEAのCLHはクライアントに処理結果を渡します。
6. NODEBのキューに蓄積している要求に対しては、TMAX\_BACKUP\_ADDRと設定されたNODEAに要求を送ります。
7. NODEBのTmaxシステムが終了します。
8. NODEAでは、NODEBから受信した要求を処理後、初めに該当リクエストを要求したクライアントに処理結果を渡します。
9. NODEAに接続されているすべてのクライアントは、正常に応答を受信します。100個のクライアントがすべて正常に応答を受信する必要があります。

---

#### 注

NODEAの要求をNODEBが処理するためには、NODEAに接続したクライアントのTMAX\_BACKUP\_ADDR、TMAX\_BACKUP\_PORTがNODEBに設定されている必要があります。そうでない場合、NODEAのTmaxシステムが終了した瞬間、未処理のクライアント要求に対してTPESYSTEMエラーを渡します。

---

## 2.34. tmmbfgen

テキストのサービス情報ファイルを、サービス情報バイナリ・ファイルで作成するコマンドです。ユーザーが作成したサービス情報ファイルをそのままWebサービス・ゲートウェイとxwsdlgenで使うことはできないため、tmmbfgenコマンドで新規ファイルを作成する必要があります。tmmbfgenコマンドによって作成されたファ

イルを、サービス情報バイナリ・ファイルといいます。tmmbfgenコマンドを実行すると、文法をチェックし、パラメータのタイプ・チェックによって、Webサービス・ゲートウェイで参照する前にあらかじめ有効性のチェックができます。また、テキスト文書を複数に分割して管理できます。

- 使用方法

```
$ tmmbfgen [-r text_file,] [-i text_file] [-d svc] -o binary_file
```

項目	説明
[-r text_file]	サービス情報ファイル(text)リストを入力します
[-i text_file]	サービス情報ファイル(text)を入力します
[-d svc]	削除するサービス名リストを入力します
-o binary_file	作成する(変更する)サービス情報バイナリ・ファイル(default = sample)です

- 例

- 以下は、新規サービス情報バイナリ・ファイルを作成する例です。サンプルを作成し、sample.txtとsample2.txtに定義されたサービス情報をサンプルに入力します。

```
$ tmmbfgen -r sample.txt,sample2.txt -o sample
```

- 以下は、既存のサービス情報バイナリ・ファイルにサービス情報を追加する例です。既存のサンプル・ファイルに、sample.txtに定義されたサービス情報を追加します。同じサービスがある場合、ファイルが入れ替わります。

```
$ tmmbfgen -i sample.txt -o sample
```

- 以下は、既存のサービス情報バイナリで特定サービスを削除する例です。sampleファイルからサービスSVC1、SVC2の情報を削除します。

```
$ tmmbfgen -d SVC1,SVC2 -o sample
```

---

## 参考

untmmbfenコマンドについての詳細は[「2.39. untmmbfgen」](#)を参照してください。

---

## 2.35. tmsnmpd

標準SNMPプロトコルに基づいて、Tmaxの構成および性能情報を照会するコマンドです。照会が可能となるようにSNMP Agentのtmsnmpdが追加されました。

CNMPプロトコルは、基本的にUDP161番を使用するため、このポートを使用するにはルート権限が必要です。1024番以降のポートを使用する場合、一般ユーザーのアカウントでも使用可能です。Tmax SNMP Agentは、tmbootコマンドとは無関係にtmsnmpdコマンドを実行する必要があります。racdコマンドと同じようにtmboot/tmdownコマンドと無関係に動作します。tmsnmpdコマンドを実行前に、環境変数でTMAXDIRが指定されている必要があります。

- 使用方法

```
$ tmsnmpd [[-h | -d| -f | -n node_name | -i | -l | -t| -p| -V] [[プロトコル:][IP:][PORT]]
```

項目	説明
[-h]	使用方法を照会します
[-d]	デバッグ・モードで動作します
[-f]	config_fileで該当位置のconfigファイルを使用します。設定していない場合、\${TMAXDIR}/snmp/tmsnmpd.confを使用します
[-n node_name]	マルチノード環境でノード名を指定します
[-i]	env_fileで該当位置のenv_fileファイルを使用します
[-l]	環境ファイル内で適用するラベルを指定します
[-t]	TMMとの接続チェック周期を設定します
[-p]	性能情報の収集周期を設定します
[-V]	バージョン情報を照会します
[プロトコル]	TCP、UDPの中から指定します（デフォルト値: UDP）
[IP]	モニタリングするIPを指定します
[PORT]	SNMPユーティリティーが接続時に使用するtmsnmpdのリスン・ポートを指定します

- 例

tmsnmpdコマンドの実行時にroot権限がない場合、以下のようにポートを指定します。または、tmsnmpd.confでagentAddressで指定できます。

```
$ tmsnmpd 9999
```

---

## 参考

tmmbfgenコマンドについての詳細は「[2.34. tmmbfgen](#)」を参照してください。

---

## Tmax SNMP MIB

ユーザーは、Tmaxサーバーの属性および統計情報に接続できます。これは、Tmax MIBに定義されています。MIBは、標準SNMPの一部です。MIBについては[www.ietf.org](http://www.ietf.org)を参照してください。

- MIBブラウジング

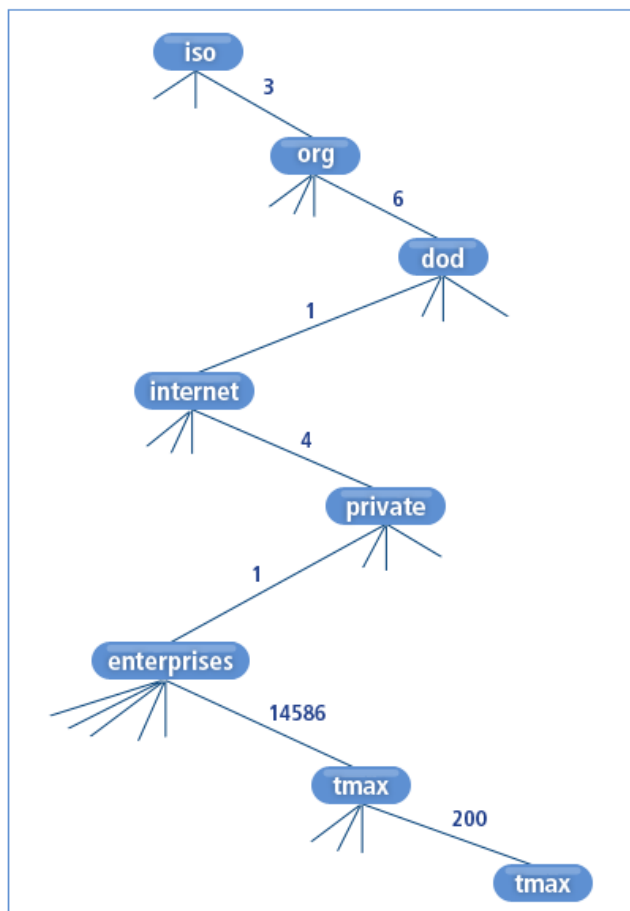
Tmax MIBファイルは、`${TMAXDIR}/snmp/mib/TMAXSOFT-TMAX-MIB.mib`に存在する必要があります。直接MIBファイルを確認することができ、3rd-party MIBブラウザを使用できます。TmaxではMIBブラウザを提供しませんが、SNMPユーティリティー・ベンダーの大部分がMIBブラウザを提供します。

- オブジェクト識別(OID)

OIDは、管理されるオブジェクトを区別するための整数を示したものです。OIDツリー構造を使用して、オブジェクトのパスを定義します。OIDはMIBファイルで定義されます。Tmax SNMP Agentに特定のOID値をもつSNMPパケットを送った場合、OIDと一致する情報を取得できます。

以下は、Tmax MIBのOIDツリーを示します。TmaxSoftのEnterprise OIDは1.3.6.1.4.1.14586です。すべてのTmaxの属性値の接頭語は1.3.6.1.4.1.14586.200です。

**[図 2.2] Tmax SNMPのOID**





- 注意事項

現行バージョンでは、SNMP v1、v2cのみをサポートします。構成および性能に対する照会のみ可能です（読み取り専用機能のみサポート）。SNMPユーティリティー（snmpget、snmpwalkなど）は、該当OSベンダーでサポートを受けるか、NET-SNMP(<http://net-snmp.sourceforge.net/>)などを使用することもできます。

NET-SNMPの簡単な使用例は以下のとおりです。

```
$ nmpwalk -v 2c -c tmax 192.168.1.100:9999 1.3.6.1.4.1.14586.200
```

## 環境設定

- 環境設定ファイルの位置

SNMP Agent(tmsnmpd)を使用するには、**`\${TMAXDIR}/snmp/tmsnmpd.conf`**環境設定ファイルが必要です。

- 環境設定方法

SNMPユーティリティー（snmpget、snmpwalkなど）を使用して情報を照会したり値を設定するなどの通信のためのcommunity stringをサポートします。Tmax v4.0 SP1以上では、rocommunity設定のみサポートします。IPv6通信環境では「rocommunity6」に設定します。

```
rocommunity <string>
```

- 環境設定の例

コミュニティの名前は「tmax」、プロトコルはUDP、ポートは「9999」であり、IPv4とIPv6を同時にオープンしたい場合、`\${TMAXDIR}/snmp/tmsnmpd.conf`ファイルに以下のように設定してtmsnmpdを起動します。

```
agentAddress udp:9999, udp6:9999
rocommunity tmax
rocommunity6 tmax
```

## 2.36. tperr

Tmaxのエラー番号とエラー・タイプを利用し、エラーに関する詳細情報を照会できるようにするコマンドです。Tmaxシステムの運用中に発生するエラーに対して、簡単に原因を見付け出し、解決できるようにします。

- 使用方法

```
$ tperr {-t error type} {-e errno number} [-f filename] [-h] [-V]
```

項目	説明
{ -t <i>error type</i> }	詳細については下の表を参照してください
{ -t <i>error type</i> }	<p>エラー・タイプを設定します。モジュール名を指定するオプションで、必須オプションです。</p> <p>以下は、エラー・タイプについての説明です。</p> <ul style="list-style-type: none"> <li>– TPE : Tmax APIエラーです</li> <li>– ADM : Tmax管理ツール(tmadmin)エラーです</li> <li>– BOOT : Tmax起動ツール(tmboot)エラーです</li> <li>– CAS : CAS(Client Authentication Server)エラーです</li> <li>– CFL : CFL(Tmax Configuration File Compiler)エラーです</li> <li>– CLH : CLH(Client Handler)エラーです</li> <li>– CLI : CLI(Client Library)エラーです(libcli.a、tmax.dll)</li> <li>– CLL : CLL(Client Listener)エラーです</li> <li>– DOWN : Tmax終了ツール(tmdown)エラーです</li> <li>– FDLC : FDLC(FDL File Compiler)エラーです</li> <li>– GST : サービス表作成ツール(gst)エラーです</li> <li>– MKPW : パスワード管理ツール(mkpw)エラーです</li> <li>– RAC : RACD(Remote Access Control Daemon)エラーです</li> <li>– RQS : RQS(Reliable Queue Server)エラーです</li> <li>– SDLC : SDLC(SDL File Compiler)エラーです</li> <li>– SVR : サーバー・ライブラリー・エラーです(libsvr.a)</li> <li>– TCPGW : カスタムTCP/IPゲートウェイ・エラーです(libtcpgw.a)</li> <li>– TMD : サーバー・アプリケーション・テスト・ツール・エラーです</li> <li>– TMGW : カスタムTCP/IPゲートウェイ・エラーです(libtcpgw.a)</li> <li>– TMM : Tmax管理(tmm)エラーです</li> <li>– TMS : TMSライブラリー・エラーです(libtms.a)</li> </ul>
{ -e <i>errno number</i> }	エラー番号を指定するオプションで、必須です
[ -f <i>filename</i> ]	エラー・メッセージに関する内容が保存されているテキスト・ファイルです。指定していない場合、\${TMAXDIR}/bin/_tmax_errnoというテキスト・ファイルが適用され、このファイルは基本的に提供されます

項目	説明
[-h]	コマンドのヘルプ・オプションです
[-V]	実行ファイルのバージョンを確認できます

- 適用環境

Tmaxシステムがインストールされている運用システム環境で使用できます。

- 例

```
(E) BOOT3008 server(svr000) is not in config [BOOT0022]

$ tperr -t BOOT -e 3008
[BOOT3008] : server (svr_name) is not in config.
Type : ERROR
Description : You specified invalid server name.
Action : Check the configuration file for valid server names.
TPEBADDESC (2)

$ tperr -t TPE ?e 2
[TPE 2] : TPEBADDESC
Cause : invalid call descriptor.
Solution : After confirm whether cd is valid, must call again tpstart().
```

---

### 参考

エラー・メッセージの詳細については、『Tmax メッセージリファレンスガイド』を参照してください。

---

## 2.37. twagent

Tmax Web Agentと接続するデーモン・プロセスを起動するためのコマンドです。

- 使用方法

```
$ twagent -m 環境ファイル [-v|-V] [-d] [-p sec] [ -l filename ]
```

項目	説明
-m 環境ファイル	twagentを起動するために環境設定ファイルを指定します
[-v -V]	バージョン情報を出力します
[-d]	デバッグ・モードを有効にします
[-p sec]	TMMと接続を試みる周期(秒)を設定します
[-l filename]	ログファイルを指定します

- 例

```
$ twagent -m sample.m -p 10
$ twagent -m sample.m -d
$ twagent -m sample.m
```

## 2.38. unconf

テキスト形式のTmax環境ファイルをコンパイルして作成されたtmconfig(バイナリTmax環境ファイル)を、また逆に分析し、テキスト形式の環境ファイルに変換するコマンドです。環境ファイルをコンパイルして起動後、ミスでテキスト環境ファイルを削除してしまった場合に有効に使用できます。

また、複数のテキスト環境ファイルを作成後システムを運用中に、現在動作している環境を確認する場合にも有効に使用できます。また、tmadminツールのcfgaddコマンドを使用してサービスを動的に追加する際にも有効に使用できます。cfgaddコマンドについての詳細は、『Tmax 運用ガイド』の「cfgadd」部分を参照してください。

- 使用方法

```
$ unconf [-i バイナリTmax環境ファイル名] [-o テキストTmax環境ファイル名] [-h] [-V]
```

項目	説明
[-i バイナリTmax環境ファイル名]	逆に分析してテキスト・ファイルに変換するバイナリTmax環境ファイル名を設定します。パスと一緒に指定でき、パスが指定されていない場合、現行ディレクトリーに位置しているバイナリ環境ファイルが参照されます。  オプションが省略された場合、デフォルトでtmconfigという名前のファイルを参照し、テキスト環境ファイルが生成されます
[-o テキストTmax環境ファイル名]	バイナリTmax環境ファイルを逆に分析して変換するテキスト形式のTmax環境ファイル名の指定に使用されるオプションで、必須です。  パスと一緒に指定でき、パスが指定されていない場合、デフォルトで現行ディレクトリーにテキスト環境ファイルが生成されます
[-h]	コマンドのヘルプ・オプションです
[-V]	実行ファイルのバージョンを確認できます

- 適用環境

Tmaxシステムがインストールされている運用システム環境で使用できます。

- 例

- 以下は、現在ディレクトリーにある「tmconfig」というバイナリTmax環境ファイルを参照し、/user1/tmax/tempディレクトリーに「basic.m」というテキスト形式のTmax環境ファイルを作成する例です。

```
$ uncfl -o /user1/tmax/temp/basic.m
```

- 以下は、/user1/tmax/binディレクトリーにある「tmconfig」というバイナリTmax環境ファイルを参照し、現行ディレクトリーに「basic.m」というテキスト形式のTmax環境ファイルを作成する例です。

```
$uncfl -i /user1/tmax/bin/tmconfig -o basic.m
```

---

## 参考

cflコマンドについての詳細は「[2.1. cfl](#)」を参照してください。

---

## 2.39. untmmbfgen

サービス情報バイナリ・ファイルをサービス情報ファイル(テキスト)に変換するコマンドです。tmmbfgenコマンドの詳細内容は「[2.34. tmmbfgen](#)」を参照してください。

- 使用方法

```
$ untmmbfgen -i binary_meta_file -o text_file
```

項目	説明
-i <i>binary_meta_file</i>	サービス情報バイナリ・ファイルを指定します
--o <i>text_file</i>	生成されるサービス情報ファイル(テキスト)を指定します

- 例

以下は、ユーザーがサービス情報バイナリ・ファイルを確認できるよう、untmmbfgenコマンドを使用してテキスト形式のファイルに変換する例です。

```
$ untmmbfgen -i sample -o unsample.txt
```

## 2.40. xwsdlgen

Webサービスのスペックのうち、明細書の役割をするWSDL文書を作成するコマンドです。現在WSDL文書は、1.1、2.0が存在します。xwsdlgenは、Webサービス・ゲートウェイ環境設定ファイルとサービス情報バイナリ・ファイルの入力を受けて、WSDL文書を作成します。

- 使用方法

```
$ xwsdlgen [options] -g wsgw_config_file -m binary_meta_file -o wsdl_file
```

– [options]

項目	説明
-w <i>version</i>	– 0 : WSDL1.1 (デフォルト値) – 1 : WSDL2.0
-b <i>binding_style</i>	– 0 : rpc (デフォルト値) – 1 : document

– 入力項目

項目	説明
-g <i>wsgw_config_file</i>	Webサービス・ゲートウェイの環境設定ファイルを設定します
-m <i>binary_meta_file</i>	サービス情報バイナリ・ファイルを設定します
-o <i>wsdl_file</i>	作成するWSDLファイルを設定します

## 第3章 関数

本章では、Tmaxで使用可能な関数について説明します。エラーコードの詳細については、『Tmax アプリケーション開発ガイド』を参照してください。

### 3.1. サーバー/クライアント関数

#### 3.1.1. gettperrno

Tmaxシステムの呼び出し時に、サーバーとクライアントで設定されたエラー・コード(errno)を返す関数です。

- プロトタイプ

```
#include <atmi.h>
int gettperrno(void)
```

- 戻り値

エラー・コード(errno)を返します。エラー・コードについての詳細内容は、[「付録 A. エラー別対応方法」](#)または『Tmax アプリケーション開発ガイド』を参照してください。

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
void main(int argc, char* argv[])
{
    int ret, usrrerrno=0;
    char *sndbuf, *rcvbuf;
    ret=tpstart((TPSTART_T*)NULL);
    if (ret==-1){
        usrrerrno=gettperrno();
        printf("error no : %d\n", usrrerrno);
    }
    data process...
    tpend();
}
```

- 関連関数

gettpurcode()

### 3.1.2. tgstrerror

TmaxGrid APIを使用時に発生するtgerrnoに該当する番号のメッセージを出力します。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
char *tgstrerror(int tgerrno)
```

- パラメータ

パラメータ	説明
tgerrno	出力するエラー番号です

- 戻り値

戻り値	説明
エラー・メッセージ	エラーコードのメッセージがある場合です

### 3.1.3. tmax\_chk\_conn

クライアントの接続状態をチェックする関数です。tpstartの実行可否チェック、ソケット状態のチェック、メッセージ転送によって接続状態をチェックする役割をします。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_chk_conn (int sec)
```

- パラメータ

パラメータ	説明
sec	以下の3種類の値の中から1つを指定できます – 負数: <b>tpstart()</b> の実行可否をチェックします – 0: ソケットの状態をチェックします。 <b>tpalivechk()</b> と同じ役割をします – 正数: メッセージの転送によって接続状態をチェックします

- 戻り値



戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

## ● エラー

tmax\_chk\_conn()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEPROTO]	まだtpstart()が実行されていない場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

## ● 例

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <usrinc/atmi.h>
#include <usrinc/fbuf.h>
#include <usrinc/tmaxapi.h>
#include "../fdl/demo_fdl.h"

main(int argc, char *argv[])
{
    FBUF    *sndbuf, *rcvbuf;
    int      fc, fg, ret, i, chkno;
    char      sndata[30], rcvdata[30];
    char      input[10], outdata[10];
    long      sndlen, rcvlen;
    int Count = 1;

    if (argc != 2)
        error processing...

    if ( (ret = tmaxreadenv( "tmax.env", "TMAX" )) == -1 )
        error processing...

    /* tpstartの実行可否のチェック */
    /**
    chkno = tmax_chk_conn(-1);
    printf("chkno = %d\n", chkno);
    if(chkno < 0)
```

```

        {
            printf("tpstart is not started\n");
            printf("errno = %d\tsterror=%s\n", tperrno, tpsterror(tperrno)
);
        }
        else if(chkno == 0)
        {
            printf("tpstart is started\n");
        }
        ***/

        if (tpstart((TPSTART_T *)NULL) == -1)
            error processing...
        if ((sndbuf = (FBUF *)tpalloc("FIELD", NULL, 0)) == NULL)
            error processing...
        if ((rcvbuf = (FBUF *)tpalloc("FIELD", NULL, 0)) == NULL)
            error processing...
        /* ソケット状態のチェック */
        /*****
        chkno = tmax_chk_conn(0);
        printf("chkno = %d\n", chkno);
        if(chkno < 0)
        {
            printf("The situation of socket is bad\n");
            printf("errno = %d\tsterror=%s\n", tperrno, tpsterror(tperrno)
);
        }
        else if(chkno == 0)
        {
            printf("The situation of socket is good\n");
        }
        *****/
        printf("My Count = %d\n", Count++);
        strcpy(input, "INPUT");
        strcpy(sndata, argv[1]);
        fc = fbput(sndbuf, fbget_fldkey(input), sndata, 0);
        if (tpcall("FDLTOUTPER", (char *)sndbuf, 0, (char **)&rcvbuf,
            (long *)&rcvlen, TPNOFLAGS) == -1)
            error processing...
        /* メッセージの転送によって接続状態をチェック */
        chkno = tmax_chk_conn(1);
        printf("chkno = %d\n", chkno);
        if(chkno < 0)
        {
            printf("The situation of connection(Message Transfer) is bad\n");

            printf("errno = %d\tsterror=%s\n", tperrno, tpsterror(tperrno)

```

```

);
    }
    else if(chkno == 0)
    {
        printf("The situation of connection(Message Transfer) is good\n");

    }

    strcpy(outdata, "OUTPUT");
    fg = fbget(rcvbuf, fbget_fldkey(outdata), rcvdata, 0);
    fbprint(rcvbuf);
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
}

```

- 関連関数

tpalivechk()

### 3.1.4. tmax\_gq\_count

GQに保存されたデータ数を返す関数です。

- プロトタイプ

```

#include <tmaxapi.h>
int tmax_gq_count (void)

```

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tmax\_gq\_count()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPENOENT]	現行セッションのためのSQが存在しない場合です

### 3.1.5. tmax\_gq\_get

GQからデータを取得する関数です。キーを指定した場合、該当キーのデータを取得します。基本的に、tmax\_gq\_get()を実行するとキューからデータが削除されます。データを保存するには、TPSQ\_PEEKフラグを設定する必要があります。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_gq_get(char *key, long keylenl, char **data, long *lenl, long flagsl)
```

- パラメータ

パラメータ	説明
key	GQから取得するデータのためのキー値を保存するバッファです
keylenl	キー・バッファのサイズを指定します(単位: バイト)
data	GQから取得するデータ・バッファのポインターです。tpalloc()で割り当てられたバッファのポインターである必要があります
lenl	GQから取得したデータ・バッファのサイズが保存されます(単位: バイト)
flagsl	TPSQ_PEEKが指定できます。このフラグを指定した場合、データをキューから削除しません

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tmax\_gq\_get()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEMATCH]	指定したキーのデータが存在しない場合です

### 3.1.6. tmax\_gq\_getkeylist

GQのキー・リストを取得する関数です。キー・リストは、SQ\_KEYLIST\_Tタイプのハンドルが返されます。各キーについての情報は、tmax\_keylist\_count()、tmax\_keylist\_count()、tmax\_keylist\_free()関数を使用して確認できます。キー・リストはバイト単位で、ASCIIコード値を比較して低い値順にソートされます。

キー値を指定した場合、該当キー値より大きいキー・リスト、もしくは同一のキー・リストがソートされて照会されます。キー値をNULLと指定した場合、全体キー・リストが照会されます。

- プロトタイプ

```
#include <tmaxapi.h>
SQ_KEYLIST_T tmax_gq_getkeylist(char *key, long keylen1)
```

- パラメータ

パラメータ	説明
key	キー値を保存するバッファです
keylen1	キー・バッファのサイズを指定します(単位: バイト)

- 戻り値

戻り値	説明
SQ_KEYLIST_T	関数の呼び出しに成功した場合です
NULL	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tmax\_gq\_getkeylist()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEMATCH]	指定したキーより大きいキー、もしくは同一のキーが存在しない場合です

## 3.1.7. tmax\_gq\_keygen

システム・キーを作成および取得する関数です。システム・キーのサイズは、SQ\_SYSKEY\_SIZE(16バイト)です。バッファ・サイズはシステム・キーのサイズより大きい値か、同一の値を割り当てる必要があります。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_gq_keygen(char *key, long keylen1)
```

- パラメータ

パラメータ	説明
key	生成されたシステム・キー値が保存されるバッファです。バッファのサイズは、SQ_SYSKEY_SIZE(16バイト)より大きい値を割り当てる必要があります

パラメータ	説明
keylenl	キー・バッファのサイズを指定します。必ずSQ_SYSKEY_SIZE(16バイト)より大きい値を指定します

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tmax\_gq\_keygen()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	入力したキーが正しくない場合です

### 3.1.8. tmax\_gq\_purge

GQのデータを削除する関数です。キーを指定した場合は該当キーのデータを削除し、キーを指定していない場合はすべてのデータを削除します。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_gq_purge(char *key, long keylenl)
```

- パラメータ

パラメータ	説明
key	GQから削除するデータのためのキー値を保存するバッファです。NULLの場合、GQのすべてのデータを削除します
keylenl	キー・バッファのサイズを指定します(単位: バイト)

- 戻り値

戻り値	説明
削除されたデータ数	関数の呼び出しに成功した場合です
-1	関数の呼び出しに成功した場合です。tperrnoにエラーコードが設定されます

- エラー

tmax\_gq\_purge()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	入力したキーが正しくない場合です
[TPENOENT]	現行セッションのためのSQが存在しない場合です

## 3.1.9. tmax\_gq\_put

データをGQに保存する関数で、キーとデータ値を渡します。tmax\_gq\_keygen()関数を使用して生成されたシステム・キーを使用するには、TPSQ\_SYSKEYフラグを使用します。システム・キーを作成して保存するには、TPSQ\_KEYGENフラグを使用します。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_gq_put(char *key, long keylenl, char *data, long lenl, long flagsl)
```

- パラメータ

パラメータ	説明
key	GQに保存するデータのためのキー値を保存するバッファです
keylenl	キー・バッファのサイズを指定します(単位: バイト)
data	GQに保存するデータを保存しているバッファです。tpalloc()で割り当てられたバッファのポインタである必要があります
lenl	GQに保存するデータ・バッファのサイズを指定します(単位: バイト)
flagsl	TPSQ_UPDATE、TPSQ_SYSKEY、TPSQ_KEYGENが指定できます。  以下は、flagslに設定可能な値についての説明です <ul style="list-style-type: none"> <li>– TPSQ_UPDATE キー値が同じ場合、アップデートします。このフラグが設定されていない場合、TPEMATCHエラーが返されます</li> <li>– TPSQ_SYSKEY システム・キーを使用する場合に設定します</li> <li>– TPSQ_KEYGEN システム・キーを自動作成して保存します。このフラグを使用する場合、keyはSQ_SYSKEY_SIZEサイズを割り当てる必要があります(keylenl =</li> </ul>

パラメータ	説明
	SQ_SYSKEY_SIZE)。正常に実行した場合、keyに生成されたキー値が保存されます

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tmax\_gq\_put()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEMATCH]	使用されたキー値がすでに存在する場合です
[TPELIMIT]	GQの最大キー数を超えた場合です

## 3.1.10. tmax\_grid\_create

キーを作成します。キーを作成するときに、キー・タイプを指定する必要があります。

作成したノードに対してenqueue、dequeue、またはLock、Unlockを使用するには、TG\_QUEUEまたはTG\_LOCKタイプを指定する必要があります。当該タイプで作成されたノードには、enqueue、dequeue、Lock、Unlock APIを使用する方法でのみ子ノードを作成することができます。さらに、これらのタイプは常にTG\_PERSISTENTタイプで作成されます。TG\_TEMPORARYタイプで作成されたノードの子ノードを作成するときは、TG\_PERSISTENTタイプで作成してはなりません。

---

### 注

TG\_TEMPORARYタイプのノードを削除すると、すべての子ノードが一緒に削除されるので注意が必要です。

---

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_grid_create(char *key, int keylen, int flags)
```

- パラメータ



パラメータ	説明
key	作成するキーの名前です。127バイトを超えることはできません
keylen	キーのサイズです
flags	動作方式を指定します <ul style="list-style-type: none"> <li>– TG_PERSISTENT クライアントが終了しても削除されません</li> <li>– TG_TEMPORARY クライアントが終了する場合やタイムアウトが過ぎた場合に自動で削除されます</li> <li>– TG_QUEUE enqueue、dequeueを使用するキーを作成します</li> <li>– TG_LOCK Lock、Unlockを使用するキーを作成します</li> </ul>

- 戻り値

戻り値	説明
$\geq 0$	関数の呼び出しに成功した場合です
$< 0$	関数の呼び出しに失敗した場合です。tgerrnoにエラーコードが設定されます

- エラー

tmax\_grid\_create()が正常に実行されなかった場合、tgerrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TGEINVAL]	キーがNULLであるか、key_len $\leq 0$ または128バイト以上である場合、あるいはフラグに適切でない値が指定された場合に発生します
[TGEOS]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TGENOMEM]	メモリーが不足して割り当てに失敗した場合に発生します
[TGEKEY]	キーを正常に認識できない場合に発生します
[TGEMAXCHL]	環境設定のMax Child制限に達して子キーを作成できなくなった場合に発生します
[TGEPROTO]	TG_QUEUEまたはTG_LOCKタイプのキーの子キーを作成した場合に発生します

エラーコード	説明
[TGESHMFULL]	TmaxGridの共有メモリー空間が不足したり、あるいは環境設定のTGMAXの最大値までキーが作成されてキーが作成できなくなった場合に発生します
[TGEDUPKEY]	キーの名前が重複する場合は
[TGECONF]	TmaxGridが設定されていない場合は
[TGEENABLE]	TmaxGridが動作しない場合は

### 3.1.11. tmax\_grid\_create2

キーを作成します。キーを作成するときに、値を入力し、Watcherと一緒に登録することができます。詳細については、[tmax\\_grid\\_create](#)、[tmax\\_grid\\_set](#)、[tmax\\_grid\\_set\\_watcher](#) APIを参照してください。データがNULLでない場合に値を入力します。コールバック関数がNULLでない場合にWatcherが設定されます。

#### ● プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_grid_create2(char *key, int keylen, char *data, int len,
TG_WATHCER_CALLBACK callback, void *args, int flags)
```

#### ● パラメータ

パラメータ	説明
key	作成するキーの名前です。127バイトを超えることはできません
keylen	キーのサイズです
data	値です。tpallocで割り当てられたポインターです。NULLの場合は、値を保存せずにキーだけ作成します
len	データのサイズです
callback	Watcherを登録後、イベントが発生したときに呼び出すコールバック関数のポインターです
args	コールバック関数を呼び出す場合に渡す引数のポインターです
flags	動作方式を指定します <ul style="list-style-type: none"> <li>– TG_PERSISTENT クライアントが終了しても削除されません</li> <li>– TG_TEMPORARY クライアントが終了する場合、自動で削除されます</li> <li>– TG_QUEUE</li> </ul>

パラメータ	説明
	enqueue、dequeueを使用するキーを作成します
	– TG_LOCK
	Lock、Unlockを使用するキーを作成します
	– TG_WATCHER_PERSISTENT
	Watcherを登録した場合、クライアントに通知を送り続けます。キーが削除されたら、Watcherは自動で削除されます
	– TG_WATCHER_ONCE
	Watcherを登録した場合、一度だけ通知し、以降は通知を送りません。もし情報を継続して受信したい場合は、Watcherを再登録する必要があります
	– TG_EVENT_DELETE_SELF
	自身のキーが削除されたときにイベントを受信します。コールバック関数で当該イベントを受信した場合、登録されたWatcherは削除されるので、Watcherを再登録する必要があります
	– TG_EVENT_CREATE_CHILD
	子キーが作成されたときにイベントを受信します
	– TG_EVENT_SET_SELF
	自身のキーにデータが設定されたときにイベントを受信します
	– TG_EVENT_GET_SELF
	自身のキーにデータが取得されたときにイベントを受信します

- 戻り値

戻り値	説明
$\geq 0$	関数の呼び出しに成功した場合です
$< 0$	関数の呼び出しに失敗した場合です。tgerrnoにエラー・コードが設定されます

- エラー

tmax\_grid\_create2()が正常に実行されなかった場合、tgerrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TGEINVAL]	キーがNULLであるか、key_len $\leq 0$ または128バイト以上である場合、あるいはフラグに適切でない値が指定された場合に発生します

エラーコード	説明
[TGEOS]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TGENOMEM]	メモリーが不足して割り当てに失敗した場合に発生します
[TGEKEY]	キーを正常に認識できない場合に発生します
[TGEMAXCHL]	環境設定のMax Child制限に達して子キーを作成できなくなった場合に発生します
[TGPROTO]	TG_QUEUEまたはTG_LOCKタイプのキーの子キーを作成した場合に発生します
[TGESHMFULL]	TmaxGridの共有メモリー空間が不足したり、あるいは環境設定のTGMAXの最大値までキーが作成されてキーが作成できなくなった場合に発生します
[TGEDUPKEY]	キーの名前が重複する場合です
[TGECONF]	TmaxGridが設定されていない場合です
[TGEENABLE]	TmaxGridが動作しない場合です

### 3.1.12. tmax\_grid\_destroy

キーを削除します。値も一緒に削除されます。基本的に子キーが存在すれば削除されませんが、フラグを設定すると、persistent typeに限って子キーまで一緒に削除することができます。またキーが削除されるときに、Watcherが登録されている場合には、設定されたイベントが通知された後Watcherも一緒に削除されます。

#### ● プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_grid_destroy(char *key, int keylen, int flags)
```

#### ● パラメータ

パラメータ	説明
key	作成するキーの名前です。127バイトを超えることはできません
keylen	キーのサイズです
flags	動作方式を指定します – TG_CHILDREN - 子キーも一緒に削除します

#### ● 戻り値

戻り値	説明
$\geq 0$	関数の呼び出しに成功した場合です
$< 0$	関数の呼び出しに失敗した場合です。tgerrnoにエラーコードが設定されます

- エラー

tmax\_grid\_destroy()が正常に実行されなかった場合、tgerrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TGEINVAL]	キーがNULLであるか、key_len $\leq 0$ または128バイト以上である場合、あるいはフラグに適切でない値が指定された場合に発生します
[TGEOS]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TGENOMEM]	メモリーが不足して割り当てに失敗した場合に発生します
[TGEKEY]	キーを正常に認識できない場合に発生します
[TGESHMFULL]	TmaxGridの共有メモリー空間が不足したり、あるいは環境設定のGQMAXの最大値までキーが作成されてキーが作成できなくなった場合に発生します
[TGECONF]	TmaxGridが設定されていない場合です
[TGEENABLE]	TmaxGridが動作しない場合です
[TGEDELCHL]	子キーが存在して削除に失敗した場合です
[TGEPROTO]	子キーにTG_QUEUEまたはTG_LOCKタイプのノードが存在するか、これらのタイプの子キーを削除する場合に発生します
[TGENOKEY]	該当するキーがない場合です

### 3.1.13. tmax\_grid\_set

キーに値を入力します。キーに値が存在する場合は、既存の値は削除され、新しい値に更新されます。キーにWatcherが登録されている場合、既存の値が削除されるときにイベントが発生しません。TG\_EVENT\_SET\_SELFイベントが登録されている場合は、イベントを通知します。TG\_QUEUEまたはTG\_LOCKタイプのキーにも値を入力できます。しかし、このタイプの子キーには値を入力できません。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_grid_set(char *key, int keylen, int type, void *data, int len, int flags)
```

- パラメータ

パラメータ	説明
key	作成するキーの名前です。127バイトを超えることはできません
keylen	キーのサイズです
type	キー値のタイプです。  tpallocで割り当てたデータの場合は0を入力し、プリミティブ型の場合は以下を入力します。  – TMAX_GRID_INT  – TMAX_GRID_FLOAT  – TMAX_GRID_LONG  – TMAX_GRID_SHORT  – TMAX_GRID_DOUBLE  – TMAX_GRID_STRING  プリミティブ型を使用する場合は、tpallocを実行しません
data	入力する値です。tpallocで割り当てたポインターまたは実際のプリミティブ型データです
len	データのサイズです
flags	使用しません

- 戻り値

戻り値	説明
$\geq 0$	関数の呼び出しに成功した場合です
$< 0$	関数の呼び出しに失敗した場合です。tgerrnoにエラーコードが設定されます

- エラー

tmax\_grid\_set()が正常に実行されなかった場合、tgerrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TGEINVAL]	キーがNULLであるか、key_len $\leq 0$ または128バイト以上である場合、あるいはフラグに適切でない値が指定された場合に発生します
[TGEOS]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TGENOMEM]	メモリーが不足して割り当てに失敗した場合に発生します
[TGEKEY]	キーを正常に認識できない場合に発生します

エラーコード	説明
[TGEPROTO]	TG_QUEUEまたはTG_LOCKタイプのノードの子ノードにデータを入力する場合に発生します
[TGESHMFULL]	TmaxGridの共有メモリー空間が不足したり、あるいは環境設定のTGMAXの最大値までキーが作成されてキーが作成できなくなった場合に発生します
[TGEITYPE]	データが正しくないSDLまたはFDLである場合です
[TGECONF]	TmaxGridが設定されていない場合です
[TGEENABLE]	TmaxGridが動作しない場合です

### 3.1.14. tmax\_grid\_get

キーの値を取得し、当該キーの値を削除します。値を削除せずに取得するには、フラグを指定します。キーを削除するには、[tmax\\_grid\\_destroy](#)関数を呼び出します。

TG\_QUEUEまたはTG\_LOCKタイプの子キーの場合は値を取得できません。

#### ● プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_grid_get(char *key, int keylen, int *type, char **data, int *len, int flags)
```

#### ● パラメータ

パラメータ	説明
key	作成するキーの名前です。127バイトを超えることはできません
keylen	キーのサイズです
type	関数の実行後、データのタイプを戻します
data	関数の実行後、値を戻します。tpallocまたはfballocであらかじめ割り当てる必要があります
len	関数の実行後、値のサイズを割り当てます
flags	動作方式を指定します  – TG_GET_PEEK  データを取得し、当該ノードのデータを削除しません

#### ● 戻り値

戻り値	説明
$\geq 0$	関数の呼び出しに成功した場合です
$< 0$	関数の呼び出しに失敗した場合です。tgerrnoにエラーコードが設定されます

- エラー

tmax\_grid\_get()が正常に実行されなかった場合、tgerrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TGEINVAL]	キーがNULLであるか、key_len $\leq 0$ または128バイト以上である場合、あるいはフラグに適切でない値が指定された場合に発生します
[TGEOS]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TGENOMEM]	メモリーが不足して割り当てに失敗した場合に発生します
[TGEKEY]	キーを正常に認識できない場合に発生します
[TGECONF]	TmaxGridが設定されていない場合です
[TGEENABLE]	TmaxGridが動作しない場合です
[TGEOTYPE]	data tpallocで割り当てていない場合や、取得したデータが正しくないSDLまたはFDLタイプである場合に発生します

### 3.1.15. tmax\_grid\_is\_exist

キーが存在するかチェックします。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_grid_is_exist(char *key, int keylen, int flags)
```

- パラメータ

パラメータ	説明
key	チェックするキーの名前です。127バイトを超えることはできません
keylen	キーのサイズです
flags	使用しません

- 戻り値



戻り値	説明
0	キーが存在しない場合です
1	キーだけ存在し、値は存在しない場合です
2	キーと値が両方存在する場合です
< 0	関数の呼び出しに失敗した場合です。tgermnolにエラーコードが設定されます

- エラー

tmax\_grid\_is\_exist()が正常に実行されなかった場合、tgermnolに以下の値のいずれかが設定されます。

エラーコード	説明
[TGEINVAL]	キーがNULLであるか、key_len <=0または128バイト以上である場合、あるいはフラグに適切でない値が指定された場合に発生します
[TGEOS]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TGENOMEM]	メモリーが不足して割り当てに失敗した場合に発生します
[TGEKEY]	キーを正常に認識できない場合に発生します
[TGECONF]	TmaxGridが設定されていない場合です
[TGEENABLE ]	TmaxGridが動作しない場合です

### 3.1.16. tmax\_grid\_count

キーの全体数を照会します。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_grid_count(int flags)
```

- パラメータ

パラメータ	説明
flags	使用しません

- 戻り値

戻り値	説明
>= 0	関数の呼び出しに成功した場合です。戻り値はキーの全体数です

戻り値	説明
< 0	関数の呼び出しに失敗した場合です。tgererrnoにエラーコードが設定されます

- エラー

tmax\_grid\_count()が正常に実行されなかった場合、tgererrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TGEOS]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TGENOMEM]	メモリーが不足して割り当てに失敗した場合に発生します
[TGECONF]	TmaxGridが設定されていない場合です
[TGEENABLE ]	TmaxGridが動作しない場合です

### 3.1.17. tmax\_grid\_lock

キーの名前でロックを実行します。他のクライアントまたはサーバーで同じキー名ですでにロックを実行している場合は、これらのプロセスでアンロックが行われ、自分の番が回ってくるまでタイムアウトで指定した時間の間待機します。ロックに成功した場合は、必ずアンロックする必要があります。もしプログラムがアンロックせずに終了すると、自動でアンロック処理が行われ、次のプロセスがロックを取得します。キーは必ずTG\_LOCKで作成する必要があり、TG\_LOCKで作成されたキーはデータを取得、設定することができません。キーを作成せずにキー名でロックを実行した場合は、自動で当該名前のキーが作成されます。ロックを実行すると、自動で振られる番号を持つ子キーが作成されます。この子キーはアンロックが呼び出されるときに削除されます。キーの名前はサイズを小さく指定することを推奨します。

1つのプロセス内で同じキー名で再帰的にロックを呼び出すことができます。そのような場合には、必ず同じ回数でアンロックを実行する必要があります。またその場合には、単一の子キーが作成されます。共有ロックを作成するには、フラグをTG\_SHARED\_LOCKIに設定します。共有ロックを作成する場合、同時に複数のノードがロックを取得できます。以前の子ノードがアンロックされると、以降連続する子ノードが共有ロックを使用する場合、そのすべての子ノードがロックを取得します。その後、すべての共有ロックがアンロックされると、待機中のTG\_EXCLUSIVE\_LOCKIに設定された子ノードがロックを取得します。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_grid_lock(char *key, int keylen, int timeout, int flags)
```

- パラメータ

パラメータ	説明
key	作成するキーの名前です。127バイトを超えることはできません

パラメータ	説明
keylen	キーのサイズです
timeout	ロックを取得できる待機時間です(単位：秒)
flags	TG_EXCLUSIVE_LOCK、TG_SHARED_LOCKを設定します

- 戻り値

戻り値	説明
>= 0	関数の呼び出しに成功した場合です
< 0	関数の呼び出しに失敗した場合です。tgerrnoにエラーコードが設定されます

- エラー

tmax\_grid\_lock()が正常に実行されなかった場合、tgerrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TGEINVAL]	キーがNULLであるか、key_len <=0または128バイト以上である場合、あるいはフラグに適切でない値が指定された場合に発生します
[TGEOS]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TGENOMEM]	メモリーが不足して割り当てに失敗した場合に発生します
[TGEKEY]	キーを正常に認識できない場合に発生します
[TGEMAXCHL]	環境設定のMax Child制限に達して子キーを作成できなくなった場合に発生します
[TGEPROTO]	キーの名前がTG_LOCKで作成されたキーでない場合に発生します
[TGESHMFULL]	環境設定のTGMAXの最大値までノードが作成され、ロックを行うための子ノードを作成できなくなった場合に発生します
[TGEMAXCHL]	環境設定のTGMAX_CHILD制限に達してロックを行うための子ノードを作成できなくなった場合に発生します
[TGETIME]	タイムアウトの間ロックの要求に失敗した場合です
[TGECONF]	TmaxGridが設定されていない場合です
[TGEENABLE ]	TmaxGridが動作しない場合です

### 3.1.18. tmax\_grid\_unlock

キーの名前でロックを解除します。次のロックを実行できます。同じキー名で再帰的にロックを実行した場合は、必ずアンロックを同じ回数実行する必要があります。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_grid_unlock(char *key, int keylen, int flags)
```

- パラメータ

パラメータ	説明
key	ロックするキーの名前です。127バイトを超えることはできません
keylen	キーのサイズです
flags	使用しません

- 戻り値

戻り値	説明
>= 0	関数の呼び出しに成功した場合は
< 0	関数の呼び出しに失敗した場合は。tgererrnoにエラーコードが設定されます

- エラー

tmax\_grid\_unlock()が正常に実行されなかった場合、tgererrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TGEINVAL]	キーがNULLであるか、key_len <=0または128バイト以上である場合、あるいはフラグに適切でない値が指定された場合に発生します
[TGEOS]	Tmaxシステムにエラーが発生した場合は。詳細情報はログファイルに記録されます
[TGENOMEM]	メモリーが不足して割り当てに失敗した場合に発生します
[TGEKEY]	キーを正常に認識できない場合に発生します
[TGECONF]	TmaxGridが設定されていない場合は
[TGEENABLE]	TmaxGridが動作しない場合は
[TGENOKEY]	ロックを実行時に設定したキーが存在しない場合は

## 3.1.19. tmax\_grid\_set\_watcher

指定したキーのイベントが発生したときに呼び出す関数を登録します。flagsは、Bitwise OR演算で複数の設定を同時に適用できます。

Watcherが設定された状態で再設定を行うと、既存のイベント設定は新しい値に代替されます。コールバックをNULLに指定すると、Watcherを解除し、当該キーに対してそれ以上イベントを受信しません。

コールバック関数で受信できるイベントの種類は以下のとおりです。

イベント	説明
TG_EVENT_DELETE_CHILD	下のフラグと同じです
TG_EVENT_DELETE_SELF	下のフラグと同じです
TG_EVENT_CREATE_CHILD	下のフラグと同じです
TG_EVENT_SET_SELF	下のフラグと同じです
TG_EVENT_GET_SELF	下のフラグと同じです
TG_EVENT_RECOVERED	GQサーバーに障害が発生し、復旧されたことを表します。このイベントを受信した場合、ユーザーはWatcherを再登録する必要があります

#### ● プロトタイプ

```
#include <usrinc/tmaxapi.h>
int  tmax_grid_set_watcher(char *key, int keylen, TG_WATHCER_CALLBACK callback,
void *args, int flags)
```

#### ● パラメータ

パラメータ	説明
key	キーの名前です。127バイトを超えることはできません
keylen	キーのサイズです
callback	キーのイベントが発生したとき、イベントを処理する関数のポインターです。ユーザーは「int CALLBACK(char *key, int keylen, int event_type, void *args);」タイプで関数を実装します。それぞれのキーごとに別々のコールバック関数を指定することができます
args	当該キーのイベントが発生してコールバック関数が呼び出されたときに一緒に渡されるユーザー定義引数です
flags	動作方式を指定します – TG_WATCHER_PERSISTENT  Watcherを登録した後、イベントが発生した場合に、クライアントに通知を送り続けます  – TG_WATCHER_ONCE

パラメータ	説明
	<p>Watcherを登録した後、一度だけ通知し、それ以降は通知を送りません。通知をさらに受信したい場合は、コールバック関数を呼び出した後にWatcherを再登録する必要があります</p> <p>– TG_EVENT_DELETE_SELF</p> <p>自身のキーが削除されたときにイベントを受信します。コールバック関数で当該イベントを受信すると、登録されたWatcherが削除されるので、Watcherを再登録する必要があります</p> <p>– TG_EVENT_CREATE_CHILD</p> <p>子キーが作成されたときにイベントを受信します</p> <p>– TG_EVENT_SET_SELF</p> <p>自身のキーにtmax_grid_set APIによりデータが設定されたときにイベントを受信します。またtmax_grid_enqueue APIによりデータが入力されたときにもイベントを受信します</p> <p>– TG_EVENT_GET_SELF</p> <p>自身のキーのデータが取得されたときにイベントを受信します。tmax_grid_get APIを呼び出すときに、フラグにTG_GET_PEEKを設定するとイベントは発生しません</p> <p><b>[注意]</b></p> <p>TG_WATCHER_PERSISTENTとTG_WATCHER_ONCEは単独で使うことができません。必ずその他のイベントとBitwise OR演算で組み合わせて適用する必要があります</p>

- 戻り値

戻り値	説明
$\geq 0$	関数の呼び出しに成功した場合です
$< 0$	関数の呼び出しに失敗した場合です。tgerrnoにエラーコードが設定されます

- エラー

tmax\_grid\_set\_watcher()が正常に実行されなかった場合、tgerrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TGEINVAL]	キーがNULLであるか、key_len <=0または128バイト以上である場合、あるいはフラグに適切でない値が指定された場合に発生します
[TGEOS]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TGENOMEM]	メモリーが不足して割り当てに失敗した場合に発生します
[TGENOKEY]	keyに指定した名前で作成されたキーが存在しない場合や、Watcherを解除するときにkeyの名前で指定しなかった場合に発生します
[TGECONF]	TmaxGridが設定されていない場合です
[TGEENABLE]	TmaxGridが動作しない場合です

### 3.1.20. tmax\_grid\_wait\_watcher

イベントが発生するまでタイムアウトに指定した時間の間待機します。1つのイベントを受信すると、受信したイベントのノードに指定されたコールバック関数が呼び出され、戻されます。

#### ● プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_grid_wait_watcher(int timeout, int flags)
```

#### ● パラメータ

パラメータ	説明
timeout	待機する時間です(単位：秒)
flags	使用しないパラメータです

#### ● 戻り値

戻り値	説明
>= 0	関数の呼び出しに成功した場合です
< 0	関数の呼び出しに失敗した場合です。tgerrnoにエラーコードが設定されます

#### ● エラー

tmax\_grid\_wait\_watcher()が正常に実行されなかった場合、tgerrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TGEOS]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TGENOMEM]	メモリーが不足して割り当てに失敗した場合に発生します
[TGETIME]	イベントが発生しなかった場合に発生します

### 3.1.21. tmax\_grid\_enqueue

キーの名前でデータを入力します。当該キーの子キーの最後の番号でデータが追加されます。キーの名前は必ずTG\_QUEUEタイプで作成する必要があります。キーを作成していない場合は、自動でキーを作成してデータを入力します。データの入力順序は保証されます。[tmax\\_grid\\_get](#)、[tmax\\_grid\\_set](#) APIで直接子キーのデータにアクセスすることは許容されません。しかし、キーの名前ではアクセスが許容されます。

#### ● プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_grid_enqueue(char *key, int keylen, int type, void *data, int len, int flags)
```

#### ● パラメータ

パラメータ	説明
key	作成するキーの名前です。127バイトを超えることはできません
keylen	キーのサイズです
type	<p>キー値のタイプです。</p> <p>tpallocで割り当てたデータの場合は0を入力し、プリミティブ型の場合は以下を入力します。</p> <ul style="list-style-type: none"> <li>– TMAX_GRID_INT</li> <li>– TMAX_GRID_FLOAT</li> <li>– TMAX_GRID_LONG</li> <li>– TMAX_GRID_SHORT</li> <li>– TMAX_GRID_DOUBLE</li> <li>– TMAX_GRID_STRING</li> </ul> <p>プリミティブ型を使用する場合はtpallocを実行しません</p>



パラメータ	説明
data	キーに入力する値です。tpallocまたはfballocで割り当てたデータのみ入力できます。あらかじめデータを割り当ててする必要があります
len	データのサイズです
flags	動作方式を指定します  – TG_TEMPORARY  クライアントが終了すると、自動で削除します。「/test」というキーでenqueueをした場合、enqueueの対象となる「/test/0」だけ削除され、親ノードの「/test」は削除されません

- 戻り値

戻り値	説明
>= 0	関数の呼び出しに成功した場合です
< 0	関数の呼び出しに失敗した場合です。tgerrnoにエラーコードが設定されます

- エラー

tmax\_grid\_enqueue()が正常に実行されなかった場合、tgerrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TGEINVAL]	キーがNULLであるか、key_len <=0または128バイト以上である場合、あるいはフラグに適切でない値が指定された場合に発生します
[TGEOS]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TGENOMEM]	メモリーが不足して割り当てに失敗した場合に発生します
[TGEKEY]	キーを正常に認識できない場合に発生します
[TGEMAXCHL]	環境設定のMax Child制限に達して子キーを作成できなくなった場合に発生します
[TGPROTO]	キーの名前がTG_QUEUEタイプで作成されたノードでない場合に発生します
[TGESHMFULL]	TmaxGridの共有メモリー空間が不足したり、あるいは環境設定のGQMAXの最大値までキーが作成されてキーが作成できなくなった場合に発生します
[TGEDUPKEY]	キーの名前が重複する場合です
[TGECONF]	TmaxGridが設定されていない場合です
[TGEENABLE]	TmaxGridが動作しない場合です
[TGEITYPE]	データが正しくないSDLまたはFDLである場合です

## 3.1.22. tmax\_grid\_dequeue

指定したキーの子キーの最初の番号の値を取得します。当該子キーは削除されます。[tmax\\_grid\\_get](#)、[tmax\\_grid\\_set](#) APIで子キーの値にアクセスすることは許容されません。しかし、キーの名前ではアクセスが許容されます。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_grid_dequeue(char *key, int keylen, int *type, char **data, int *len,
int flags)
```

- パラメータ

パラメータ	説明
key	キーの名前です。127バイトを超えることはできません
keylen	キーのサイズです
type	関数を実行した後、データのタイプを戻します
data	キーで取得する値のバッファです(tpallocまたはfballocで割り当てられたデータのみ可能です。あらかじめデータを割り当てる必要があります)。取得したデータがバッファのサイズより大きい場合は、内部で再度割り当てられます
len	ノードで取得するデータのサイズが保存されます
flags	使用しません

- 戻り値

戻り値	説明
>= 0	関数の呼び出しに成功した場合です
< 0	関数の呼び出しに失敗した場合です。tgererrnoにエラーコードが設定されます

- エラー

tmax\_grid\_dequeue()が正常に実行されなかった場合、tgererrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TGEINVAL]	キーがNULLであるか、key_len <=0または128バイト以上である場合、あるいはフラグに適切でない値が指定された場合に発生します
[TGEOS]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TGENOMEM]	メモリーが不足して割り当てに失敗した場合に発生します
[TGEKEY]	キーを正常に認識できない場合に発生します

エラーコード	説明
[TGENOKEY]	指定したキーがない場合です
[TGENODATA]	入力したデータが存在しない場合です
[TGECONF]	TmaxGridが設定されていない場合です
[TGEENABLE ]	TmaxGridが動作しない場合です
[TGEOTYPE ]	data tpallocでデータを割り当てていない場合や、取得したデータが正しくないSDLまたはFDLデータである場合です

### 3.1.23. tmax\_grid\_get\_children

キーの名前で子キーの名前リストを照会します。子キーの情報を含めているTG\_KEYLIST\_Tハンドラーを返します。失敗した場合は、NULLが返されます。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
TG_KEYLIST_T tmax_grid_get_children(char *key, int keylen, int *child_count)
```

- パラメータ

パラメータ	説明
key	キーの名前です。127バイトを超えることはできません
keylen	キーのサイズです
child_count	動作方式を指定します

- 戻り値

戻り値	説明
NULLでない場合	TG_KEYLIST_Tでキーの名前リストの開始ポインターを返します。使用しなくなった場合は、freeを実行する必要があります
NULLの場合	関数の呼び出しに失敗した場合です。tgerrnoにエラーコードが設定されます

- エラー

tmax\_grid\_get\_children()が正常に実行されなかった場合、tgerrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TGEINVAL]	キーがNULLであるか、key_len <=0または128バイト以上である場合、あるいはフラグに適切でない値が指定された場合に発生します
[TGEOS]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TGENOMEM]	メモリーが不足して割り当てに失敗した場合に発生します
[TGEKEY]	キーを正常に認識できない場合に発生します
[TGECONF]	TmaxGridが設定されていない場合です
[TGEENABLE]	TmaxGridが動作しない場合です
[TGENOKEY]	指定したキーがない場合です

### 3.1.24. tmax\_grid\_get\_child\_with\_index

子キーの情報を含めているTG\_KEYLIST\_Tハンドラーでnth番目のキーの情報をTG\_KEYINFO\_T構造体に保存します。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_grid_get_child_with_index(TG_KEYLIST_T keylist, int nth,
                                   TG_KEYINFO_T * keyinfo)
```

- パラメータ

パラメータ	説明
keylist	tmax_grid_get_children()を呼び出したときに戻されるハンドラーです
nth	参照する子キーの番号です
keyinfo	子ノードの情報を保存する構造体。割り当てられたバッファを使用する必要があります

- 戻り値

戻り値	説明
>= 0	関数の呼び出しに成功した場合です
< 0	関数の呼び出しに失敗した場合です。tgerrnoにエラーコードが設定されます

- エラー

tmax\_grid\_get\_child\_with\_index()が正常に実行されなかった場合、tgerrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TGEINVAL]	keylistがNULLであるか、nth < 0またはkeyinfoがNULLの場合に発生します
[TGEOS]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TGENOMEM]	メモリーが不足して割り当てに失敗した場合に発生します
[TGEITYPE]	正しくないTG_KEYLIST_Tハンドラーの値が入力された場合です
[TGELIMIT]	nthが子ノードの数より大きい場合です

### 3.1.25. tmax\_grid\_keylist\_free

子キーの情報を含めているTG\_KEYLIST\_Tハンドラーのリソースを解除します。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_grid_keylist_free(TG_KEYLIST_T keylist)
```

- パラメータ

パラメータ	説明
keylist	tmax_grid_get_children()を呼び出したときに戻されるハンドラーです

- 戻り値

戻り値	説明
>= 0	関数の呼び出しに成功した場合です
< 0	関数の呼び出しに失敗した場合です。tgerrnoにエラーコードが設定されます

- エラー

tmax\_grid\_keylist\_free()が正常に実行されなかった場合、tgerrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TGEINVAL]	keylistがNULLの場合です
[TGEOS]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます

エラーコード	説明
[TGEITYPE]	正しくないTG_KEYLIST_Tハンドラーの値が入力された場合に発生します

### 3.1.26. tmax\_get\_sessionid

現行セッションのIDを返す関数です。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_get_sessionid(void)
```

- 戻り値

戻り値	説明
-1ではない値	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tmax\_get\_sessionid()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEPROTO]	セッションが終了している場合です

### 3.1.27. tmax\_keylist\_count

keylistハンドルからキー・リストの数を返す関数です。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_keylist_count(SQ_KEYLIST_T keylist)
```

- パラメータ

パラメータ	説明
keylist	tmax_sq_getkeylist()またはtmax_gri_getkeylist()から渡されたハンドルです

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tmax\_keylist\_count()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEBADDESC]	keylistハンドルが正しくない場合です

## 3.1.28. tmax\_keylist\_free

keylistハンドルのメモリーやその他のリソースを解除する関数です。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_keylist_free(SQ_KEYLIST_T keylist)
```

- パラメータ

パラメータ	説明
keylist	tmax_sq_getkeylist()またはtmax_gq_getkeylist()から渡されたハンドルです

- 戻り値

戻り値	説明
-1ではない値	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tmax\_keylist\_free()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEBADDESC]	keylistハンドルが正しくない場合です

## 3.1.29. tmax\_keylist\_getakey

keylistハンドルからn番目のキー情報を取得する関数です。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_keylist_getakey(SQ_KEYLIST_T keylist, int nth, SQ_KEYINFO_T *keyinfo)
```

- パラメータ

パラメータ	説明
keylist	<b>tmax_sq_getkeylist()</b> または <b>tmax_gq_getkeylist()</b> から渡されたハンドルです
nth	keylistハンドルからn番目のキーです。nは0からtmax_keylist_count() -1の間の値である必要があります
keyinfo	n番目のキー情報が保存される構造体です。詳細は、下の内容を参照してください

以下は、keyinfoについての説明です。

```
Structure keyinfo {
    long keylen
    long datalen
    time_t starttime
    char *key
};
```

メンバー	説明
long keylen	キーのサイズです(単位: バイト)
long datalen	データのサイズです(単位: バイト)
time_t starttime	データが保存またはアップデートされた時間です
char *key	キー値を保存しているバッファです

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tmax\_keylist\_getakey()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。



エラーコード	説明
[TPEBADDESC]	keylistハンドルが正しくない場合です
[TPELIMIT]	nthの範囲を超えた場合です

### 3.1.30. tmax\_sq\_count

現行SQに保存されているデータ数を返す関数です。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_sq_count(void)
```

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tmax\_sq\_count()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPENOENT]	現行セッションのためのSQが存在しない場合です

### 3.1.31. tmax\_sq\_get

データをセッション・キューに保存する関数です。SQでデータを取得するためにキーを指定した場合、該当キーのデータを取得します。基本的に、tmax\_sq\_get()を実行するとキューからデータが削除されます。データを保存するには、**TPSQ\_PEEK**フラグを設定する必要があります。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_sq_get(char *key, long keylen1, char **data, long *len1, long flags1)
```

- パラメータ

パラメータ	説明
key	SQから取得するデータのためのキー値を保存するバッファです
keylenl	キー・バッファのサイズを指定します(単位: バイト)
data	SQから取得するデータ・バッファのポインターです。tpalloc()で割り当てられたバッファのポインターである必要があります
lenl	SQから取得したデータ・バッファのサイズが保存されます(単位: バイト)
flagsl	TPSQ_PEEKを指定できます。このフラグを指定した場合、データをキューから削除しません

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tmax\_sq\_get()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPENOENT]	現行セッションのためのSQが存在しない場合です
[TPEMATCH]	指定したキーのデータが存在しない場合です

### 3.1.32. tmax\_sq\_getkeylist

現行セッションSQのキー・リストを取得する関数です。キー・リストは、SQ\_KEYLIST\_Tタイプのハンドルが返され、各キーについての情報は、tmax\_keylist\_count()、tmax\_keylist\_getakey()、tmax\_keylist\_free()関数を使用して確認できます。

キー・リストはバイト単位で、ASCIIコード値を比較して低い値順にソートされます。キー値を指定した場合、該当キー値より大きいキー・リスト、もしくは同一のキー・リストがソートされて照会されます。キー値をNULLと指定した場合、全体キー・リストが照会されます。

- プロトタイプ

```
#include <tmaxapi.h>
SQ_KEYLIST_T tmax_sq_getkeylist(char *key, long keylenl)
```

- パラメータ

パラメータ	説明
key	キー値を保存するバッファです
keylenl	キー・バッファのサイズを指定します(単位: バイト)

- 戻り値

戻り値	説明
SQ_KEYLIST_T	関数の呼び出しに成功した場合です
NULL	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tmax\_sq\_getkeylist()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPENOENT]	現行セッションのためのSQが存在しない場合です
[TPEMATCH]	指定したキーより大きいキー、もしくは同一のキーが存在しない場合です

### 3.1.33. tmax\_sq\_keygen

システム・キーを作成および取得する関数です。システム・キーのサイズは、SQ\_SYSKEY\_SIZE(16バイト)です。バッファ・サイズはシステム・キーのサイズより大きい値か、同一の値を割り当てる必要があります。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_sq_keygen(char *key, long keylenl)
```

- パラメータ

パラメータ	説明
key	生成されたシステム・キー値が保存されるバッファです。バッファのサイズは、SQ_SYSKEY_SIZE(16バイト)より大きい値を割り当てる必要があります
keylenl	キー・バッファのサイズを指定します。必ずSQ_SYSKEY_SIZE(16バイト)より大きい値を指定します

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です

戻り値	説明
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tmax\_sq\_keygen()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPENOENT]	現行セッションのためのSQが存在しない場合です

### 3.1.34. tmax\_sq\_purge

SQのデータを削除する関数です。キーを指定した場合は該当キーのデータを削除し、キーを指定していない場合はすべてのデータを削除します。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_sq_purge(char *key, long keylenl)
```

- パラメータ

パラメータ	説明
key	SQから削除するデータのためのキー値を保存するバッファです。NULLの場合、現行セッションSQのすべてのデータを削除します
keylenl	キー・バッファのサイズを指定します(単位: バイト)

- 戻り値

戻り値	説明
削除されたデータ数	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tmax\_sq\_purge()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPENOENT]	現行セッションのためのSQが存在しない場合です

### 3.1.35. tmax\_sq\_put

サーバーのデータをSQに保存する関数です。キーとデータ値を渡す必要があります。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_sq_put(char *key, long keylenl, char *data, long lenl, long flagsl)
```

- パラメータ

パラメータ	説明
key	SQに保存するデータのためのキー値を保存するバッファです
keylenl	キー・バッファのサイズを指定します(単位: バイト)
data	SQに保存するデータを保存しているバッファです。 <b>tpalloc()</b> で割り当てられたバッファのポインターである必要があります
lenl	SQに保存するデータ・バッファのサイズを指定します(単位: バイト)
flagsl	TPSQ_UPDATE、TPSQ_SYSKEY、TPSQ_KEYGENを指定できます
flagsl	以下は、flagslに設定される値についての説明です  – TPSQ_UPDATE  キー値が同じ場合、アップデートします。フラグが設定されていない場合、TPEMATCHエラーが返されます  – TPSQ_SYSKEY  <b>tmax_sq_keygen()</b> により生成されたシステム・キーを使用する場合に設定します  – TPSQ_KEYGEN  システム・キーを自動で作成して保存します。キーはSQ_SYSKEY_SIZEのサイズで割り当てる必要があります(keylenl = SQ_SYSKEY_SIZE)。正常に実行した場合、keyに生成されたキー値が保存されます

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tmax\_sq\_put()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPENOENT]	現行セッションのためのSQが存在しない場合に発生します
[TPEMATCH]	使用されたキー値がすでに存在する場合に発生します
[TPELIMIT]	SQの最大数を超えているか、またはSQの最大キー数を超えている場合に発生します

### 3.1.36. tmaxlastsvc

サーバーとクライアントで一番最後に実行されたサービスを照会する関数です。エラーが発生したサービス名または最後にルーチンを実行したサービス名を返します。

**tpforward()**や**tpreply()**などを使用するには、ユーザーが呼び出していたサービス以外の複数のサービス・ルーチンを実行します。複数のサービス・ルーチンを実行中にエラーが発生した場合、どのようなサービス・ルーチンでエラーが発生したのかユーザーは分かりません。

#### ● プロトタイプ

```
#include <tmaxapi.h>
char *tmaxlastsvc(char *idata, char *odata, long flags)
```

#### ● パラメータ

パラメータ	説明
idata, odata	tpcallで使用していたsend/rcvバッファーを使用します
flags	現行バージョンではサポートしていませんが、TPNOFLAGSに設定します

#### ● 戻り値

戻り値	説明
内部バッファー・ポインター	関数の呼び出しに成功した場合です
NULL	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

#### ● エラー

tmaxlastsvc()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です

- 例

```
#include <usrinc/atmi.h>
main(int argc, char *argv[])
{
    ...
    if(tpcall("TOUPPER", sndbuf, 0, &rcvbuf, &rcvlen, TPNOFLAGS)==-1){
        memset(svc, NULL, 16);
        strcpy((char *)svc, tmaxlastsvc(sndbuf, rcvbuf, TPNOFLAGS));
        printf("servicename = %s\n", svc);
        error processing
    }
    ...
}
```

### 3.1.37. tmaxreadenv

サーバーとクライアントで、接続するシステムの情報を読み込み、環境変数に新規値を設定する関数です。

Tmaxシステムと接続するには、いくつかの環境変数をシステムに登録する必要があります。登録された環境変数を参照し、**tpstart()**関数を使用してTmaxシステムと接続します。環境変数は、UNIXの場合、**cs**hは<cshrc>、**k**shは<.profile>に定義します。**DOS**の場合は<autoexec.bat>ファイルに定義します。

接続するシステムが2つ以上の場合、環境変数に2つのシステムに関する情報を登録できません。そのためクライアントは、状況によって接続するシステムを変更して使用します。Tmaxシステムと接続するための情報を環境変数に設定するため、Tmaxシステムに接続前に実行される必要があります。

---

#### 参考

環境変数をファイルに登録する方法は、『Tmax 運用ガイド』を参照してください。

---

- プロトタイプ

```
#include <tmaxapi.h>
int tmaxreadenv (char *file, char *label)
```

- パラメータ

パラメータ	説明
file	接続するシステムの環境情報が保存されているファイルの名前です。ファイルはテキスト形式で、一定の形式に合わせて登録されている必要があります
label	ファイル内に登録されている環境情報の記述子です。2つ以上のシステム情報を1つのファイルに登録する場合、それぞれのシステムを区別できません

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	ファイルまたはラベルが存在しない場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

void main(int argc, char* argv[])
{
    tmaxreadenv("tmax.env", "tmax"); /* tmax.envの tmaxセクションで環境変数を取得
    */
    tpstart((TPSTART_T *)NULL);
    data process...
    tpend();
}
```

- 関連関数

tpstart()

## 3.1.38. tp\_sleep

サーバーとクライアントからのデータの到着を秒単位で待機する関数です。最大sec時間の間待機し、その間にデータが到着した場合は即時に返します。

- プロトタイプ

```
#include <tmaxapi.h>
int tp_sleep (int sec)
```

- パラメータ

パラメータ	説明
sec	待機する時間を正の整数値で入力します(単位：秒)

- 戻り値



戻り値	説明
0	sec時間までにデータが到着していない場合です
-1	エラーが発生した場合です。tperrnoにエラーコードが設定されます
1	clhからイベントが発生した場合です。tpacallに対する応答や非要求メッセージなどが入ってきた場合です
2	tmmからイベントが発生した場合です。clhからのイベントは発生していません。tpschedule()を呼び出してtmmイベントを処理する必要があります
3	clhとtmmの両方からイベントが発生した場合です。tpschedule()を呼び出してtmmイベントを処理する必要があります

- エラー

tp\_sleep()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 関連関数

tpacall(), tpbroadcast(), tpgetrply(), tp\_usleep(), tpsleep()

## 3.1.39. tp\_usleep

サーバーとクライアントからのデータの到着を100万分の1秒単位で待機する関数です。最大sec時間の間待機し、その間にデータが到着した場合は即時に返します。

- プロトタイプ

```
#include <tmaxapi.h>
int tp_usleep (int usec)
```

- パラメータ

パラメータ	説明
usec	待機する時間を正の整数値で入力します(単位 : 1000000(100万)分の1秒)

- 戻り値

戻り値	説明
0	usec時間までにデータが到着していない場合です
-1	エラーが発生した場合です。tperrnoにエラーコードが設定されます
1	clhからイベントが発生した場合です。tpacallに対する応答や非要求メッセージなどが入ってきた場合です
2	tmmからイベントが発生した場合です。clhからイベントは発生していません。tpschedule()を呼び出してtmmイベントを処理する必要があります
3	clhとtmmの両方からイベントが発生した場合です。tpschedule()を呼び出してtmmイベントを処理する必要があります

### ● エラー

tp\_usleep()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

### ● 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

void main(int argc, char *argv[])
{
    int ret, cd;
    char *buf;
    long revent;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf=tpalloc("STRING", NULL, 0);
    if (buf==NULL) {error processing}
    data process...

    cd=tpconnect("SERVICE", buf, 0, TPRECVONLY);
    if (cd<0) { error processing }
    while(1)
    {
        ret=tp_usleep(2000000);
```

```

        if (ret<0) {error processing }
        if (ret==0) printf("waited 2sec..\n");
        else {
            printf("data received!\n");
            break;
        }
    }
    ret=tprecv(cd, &buf, &len, TPNOFLAGS, &revent);
    if (ret<0) { error processing }
    data process....
    tpend();
}

```

- 関連関数

tmax\_get\_db\_username(), tmax\_get\_db\_tnsname()

## 3.1.40. tpabort

サーバーとクライアントでグローバル・トランザクションをロールバックする関数で、**tx\_rollback()**関数と同じ機能を実行します。この関数はTuxedoで使用した関数をTmaxシステムにそのまま適用する場合に使用します。Tuxedoで開発されたプログラムを変更することなくTmaxで変換できるようにサポートします。

- プロトタイプ

```

#include <tuxfml.h>
int tpabort (long flags)

```

- パラメータ

パラメータ	説明
flags	現行バージョンではサポートしていませんが、TPNOFLAGSに設定します

- 戻り値

「[3.1.107. tx\\_rollback](#)」を参照してください。

- エラー

「[3.1.107. tx\\_rollback](#)」を参照してください。

- 例

```

#include <stdio.h>
#include <usrinc/atmi.h>
#include <tuxinc/tuxfml.h>

void main(int argc, char *argv[])
{
    char *buf;
    int ret;
    long len, revent;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf=tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }

    data process...
    ret = tpbegin(5, TPNOFLAGS);
    if (ret < 0){ error processing }

    ret=tpcall("SERVICE",(char *)buf,strlen(buf),(char **)&buf,&len,TPNOFLAGS);

    if (ret == -1)
    {
        ret = tpabort(TPNOFLAGS);
        if (ret < 0){ error processing }
        tpfree(buf);
        tpend();
        exit(1);
    }
    ret = tpcommit(TPNOFLAGS);
    if (ret < 0){ error processing }
    tpfree(buf);
    tpend();
}

```

- 関連関数

tx\_rollback()

### 3.1.41. tpacall

サーバーとクライアントで非同期サービスの要求を送信する関数です。**svc**で指定されたサービスに、サービス要求メッセージを送信します。非同期通信でメッセージを送信後、結果を受信するまで待機せず、すぐに

返されます。**tpgetrply()**を使用して応答を受信することができます。あるいは、**tpcancel()**を使用して応答をキャンセルすることもできます。

tx\_beginを呼び出してから、tx\_timeが過ぎた後、tpacallはflagにTPNOTRAN | TPNOREPLYを設定した呼び出しだけ成功し、それ以外はすべてTPETIMEエラーを発生させます。

#### ● プロトタイプ

```
# include <atmi.h>
int tpacall (char *svc, char *data, long len, long flags)
```

#### ● パラメータ

パラメータ	説明
svc	呼び出されるサービス名です。Tmax応用サーバー・プログラムで提供されるものである必要があります
data	NULL値でない場合、tpalloc()で割り当てられたバッファのポインターである必要があります
len	送信されるデータ長です – dataが指すバッファが、長さを指定する必要がないタイプ (STRING、STRUCT、X_COMMON、X_C_TYPE) の場合、lenは無視され、一般的に0が使用されます – dataが指すバッファが、長さを指定する必要があるタイプ (X_OCTET、CARRAY、MULTI STRUCTURE) の場合、lenは0になれません – dataがNULLの場合、lenは無視され、データがない状態でサービス要求が送信されます。dataのタイプ(type)とサブタイプ(subtype)は、svcがサポートするタイプである必要があります。サービス要求がトランザクション・モードで送信された場合、該当応答は必ず受信されます
flags	呼び出し時に使用されるオプションであり、呼び出しモードを指定します  flagsに使用可能な値は以下のとおりです – TPBLOCK  フラグなしでtpacall()が使用されると、svcに呼び出されたサービスがない場合、もしくは正しくない結果が返された場合、正常な結果が返されます。tpgetrply()の呼び出し時にエラーが返されます。TPBLOCKフラグを使用してtpacall()を呼び出した場合、サービス状態が正常かどうかを確認できます – TPNOTRAN  トランザクション・モードで、svcがトランザクションをサポートしないサービスである場合は、tpacall()を呼び出すときに、flagsをTPNOTRANに設定する必要があります。tpacall()の呼び出し元が、トランザクション・モードでTPNOTRANを設定してsvcサービスを要求した場合、svcサービスはトランザクション・モードから除外され

パラメータ	説明
	<p>て実行されます。トランザクション・モードでtpacall()を呼び出すとき、TPNOTRANが設定されていても、トランザクション・タイムアウト(timeout)の影響を受けます。つまり、トランザクション・タイムアウトが過ぎた後のTPNOTRANを適用したtpacallも呼び出さずに失敗するという意味です。例外として、TPNOTRAN TPNOREPLYを適用したtpacallは、呼び出しを許容します。TPNOTRANで呼び出されたサービスが失敗した場合、呼び出し元のトランザクションには影響を及ぼしません</p> <p>– TPNOREPLY</p> <p>tpacall()で送信したサービス要求は、応答を待たずに即時返します。結果は、後でtpacall()で返した記述子を利用してtpgetrply()で結果を受信します。flagsをTPNOREPLYに設定した場合、サービスに対する応答を受けないように設定されます。TPNOREPLYを設定した場合、tpacall()はサービスが正常に呼び出されたときに0を返します。関数の呼び出し元がトランザクション・モードにある場合、TPNOREPLYを使用するためには、TPNOTRANフラグと一緒に設定する必要があります。またサービス状態が正常かどうかをチェックするためには、TPBLOCKと一緒に設定する必要があります。TPBLOCKを設定していない場合は、サービスがNRDYの場合でもエラーを返しません</p> <p>– TPNOBLOCK</p> <p>内部バッファが送信メッセージでいっぱいになったときのようなブロッキング(blocking)状況になった場合に、サービス要求が失敗するように設定するフラグです。TPNOBLOCKフラグが設定されていない状態でtpacall()が呼び出され、ブロッキング状況が発生すると、ブロッキング状況が解除されるか、タイムアウト(トランザクション・タイムアウト、またはブロック・タイムアウト)が発生するまで関数の呼び出し元は待機します</p> <p>– TPNOTIME</p> <p>関数の呼び出し元がブロック・タイムアウトを無視し、応答が受信されるまで無限に待機するように設定するフラグです。トランザクション・タイムアウト内でtpacall()を実行した場合、トランザクション・タイムアウトが適用されます</p> <p>– TPSIGRSTRT</p> <p>シグナル(signal)割り込みを受け入れる場合に使用します。内部でシグナル割り込みが発生し、システム関数の呼び出しが妨害されたとき、システム関数の呼び出しが再実行されます。TPSIGRSTRTが設定されていない状態でシグナル割り込みが発生した場合、関数は失敗し、tperrnoにTPGOTSIGが設定されます</p>

- 戻り値

戻り値	説明
記述子(descriptor)	関数の呼び出しに成功した場合です。返された記述子は、送信されたサービス要求に対する応答の受信に使用されます
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

## ● エラー

tpacall()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、svcがNULLである場合や、データがtpalloc()で割り当てられていないバッファを指す場合、フラグが有効でない場合に発生します
[TPENOENT]	svcというサービスが存在しないため、サービスを要求できない場合です
[TPEITYPE]	dataのタイプおよびサブタイプが、svcがサポートしていないタイプです。構造体の場合、使用された構造体がSDLFILEファイルに宣言されていない場合に発生します
[TPELIMIT]	処理されていない非同期性サービス要求数が限界に達したため、呼び出し元のサービス要求が送信されなかった場合に発生します
[TPETIME]	タイムアウトが発生した場合です。関数の呼び出し元がトランザクション・モードの場合は、トランザクション・タイムアウトが発生したことを意味します。トランザクションはロールバックされます。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、TPNOTIMEやTPNOBLOCKがすべて設定されていない状態ではブロック・タイムアウトが発生します。トランザクション・タイムアウトが発生した場合、トランザクションがロールバックされるまで新規サービスの要求を送信したり、まだ受信していない応答を待機することはすべて[TPETIME]エラーとして処理されます
[TPEBLOCK]	TPNOBLOCKが設定された状態でブロッキング状況が発生した場合です
[TPGOTSIG]	TPSIGRSTRTが設定されていない状態でシグナルを受信した場合です
[TPEPROTO]	トランザクション・モードでTPNOREPLYサービスの呼び出し時に、TPNOTRANフラグを設定していないなどの不適切な状況で発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です
[TPEOS]	運用システムにエラーが発生した場合です

## ● 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
void main(int argc, char *argv[])
{
    char *buf;
    int ret,cd;
```

```

long len;

ret=tpstart((TPSTART_T *)NULL);
if (ret<0) { error processing }

buf = (char *)tpalloc("CARRAY", NULL, 20);
if (buf==NULL) {error processing }
data process....

cd = tpacall("SERVICE", sndbuf, 20, TPNOTIME);
if (cd<0) {error processing }
data process...

ret=tpgetrply(&cd, (char **)&buf, &len, TPNOTIME);
if (ret<0) { error processing }
data process....

tpfree((char *)buf);
tpend();
}

```

- 関連関数

tpalloc(), tpcall(), tpcancel(), tpgetrply()

### 3.1.42. tpacallsvg

COUSINIにまとめられたマルチサーバー・グループ環境で、特定サーバー・グループに属するサービスを指定し、非同期型の通信でサービスを要求する関数です。特定サーバー・グループを指定してサービスを呼び出す以外は、tpacall()と同じ動作をします。

- プロトタイプ

```

#include <usrinc/tmaxapi.h>
int tpacallsvg (int svgno, char *svc, char *data, long lenl, long flagsl)

```

- パラメータ

svgnoを除いた他のパラメータは「[3.1.41. tpacall](#)」を参照してください。

パラメータ	説明
svgno	<p>svgnoには、呼び出すサービスが属しているサーバー・グループの番号を指定します。サーバー・グループの番号は、<b>tpgetsvglist()</b>を使用して確認できます。</p> <p>-1を設定した場合、既存のtpcall()と同じ動作をします。</p>



パラメータ	説明
	サービス内で他のサービス呼び出す場合、 <b>tpgetmysvgno()</b> を使用し、現在自身に属するサーバー・グループの番号を確認後、同一サーバー・グループに属するサービスを呼び出すこともできます

- 戻り値

戻り値	説明
記述子	関数の呼び出しに成功した場合です (client descriptor)
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

エラーについての詳細内容は[「3.1.41. tpacall」](#)を参照してください。

- 例

#### <クライアント・プログラム>

```
#include <usrinc/tmaxapi.h>
#include "../fdl/demo_fdl.h"

int main(int argc, char *argv[])
{
    FBUF    *sndbuf, *rcvbuf;
    int      ret, I, cd;
    char     snddata[30], rcvdata[30];
    long     sndlen, rcvlen;
    struct svglist *svg_list;
    char     svc[32];

    ret = tmaxreadenv( "tmax.env", "TMAX" );
    ret = tpstart((TPSTART_T *)NULL);
    sndbuf = (FBUF *)tpalloc("FIELD", NULL, 0);
    rcvbuf = (FBUF *)tpalloc("FIELD", NULL, 0);
    strcpy(snddata, argv[1]);

    ret = fbput(sndbuf, INPUT, snddata, 0);
    strcpy(svc, "FDLToupper");
    svg_list = (struct svglist *)tpgetsvglist(svc, 0);

    for(i=0; i< svg_list->ns_entry; i++)
    {
        printf("\n");
    }
}
```

```

printf(" >>> tpcallsvg ( %d th svg )\n", i+1);
strcpy(sndata, argv[1]);
ret = fbput(sndbuf, INPUT, sndata, 0);
cd = tpacallsvg(svg_list->s_list[i], svc, (char *)sndbuf, 0, 0);
if(cd < 0)
{
    printf("tpacall failed! errno = %d[%s]\n", tperrno,
tpstrerror(tperrno));
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
    exit(1);
}
ret = tpgetrply(&cd, (char **)&rcvbuf, (long *)&rcvlen, 0);
ret = fbget(rcvbuf, OUTPUT, rcvdata, 0);
fbprint(rcvbuf);
}
tpfree((char *)sndbuf);
tpfree((char *)rcvbuf);
tpend();
return 0;
}

```

- 関連関数

tpacall(), tpcallsvg(), tpgetsvglist(), tpgetmysvgno()

### 3.1.43. tpalivechk

クライアントの接続状態をチェックする関数で、ソケットの状態をチェックする役割をします。**tmx\_chk\_conn()**関数でパラメータを0に設定した場合と同じ役割をします。

- プロトタイプ

```

#include <usrinc/tmaxapi.h>
int tpalivechk(void)

```

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpalivechk()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEPROTO]	まだtpstart()が実行されていない場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <usrinc/atmi.h>
#include <usrinc/fbuf.h>
#include <usrinc/tmaxapi.h>
#include "../fdl/demo_fdl.h"

main(int argc, char *argv[])
{
    FBUF      *sndbuf, *rcvbuf;
    int        chkno;
    char       snddata[30], rcvdata[30];
    char       input[10], outdata[10];
    long       sndlen, rcvlen;

    if ( (ret = tmaxreadenv( "tmax.env", "TMAX" )) == -1 ){
        printf( "tmax read env failed\n" );
        exit(1);
    }
    if (tpstart((TPSTART_T *)NULL) == -1)
        error processing...
    if ((sndbuf = (FBUF *)tpalloc("FIELD", NULL, 0)) == NULL)
        error processing...
    if ((rcvbuf = (FBUF *)tpalloc("FIELD", NULL, 0)) == NULL)
        error processing...
    /* socketの状態チェック */
    chkno = tpalivechk();
    printf("chkno = %d\n", chkno);
    if(chkno < 0)
    {
        printf("The situation of socket is bad\n");
        printf("errno = %d\tstrerror=%s\n", tperrno, tpstrerror(tperrno) );
    }
    else if(chkno == 0)
```

```

{
    printf("The situation of socket is good\n");
}
tpcall...
data process...
tpend();
}

```

- 関連関数

tmax\_chk\_conn()

## 3.1.44. tpalloc

サーバーとクライアントで型付きバッファを割り当てる関数です。Cライブラリーのmalloc()、realloc()、free()とは一緒に使用できません。たとえば、tpalloc()で割り当てられたバッファをfree()で除去できません。この場合は**tpfree()**を使用します。

tpalloc()は、typeで指定されたタイプのバッファを割り当て、それについてのポインターを返します。タイプによって、subtypeとsizeは選択で指定できます。一部のバッファ・タイプは使用前に初期化が必要なため、tpalloc()はバッファを割り当て後、これを初期化して返します。そのため、呼び出し元に返されたバッファは即時使用可能です。初期化に失敗した場合、tpalloc()はバッファを0に初期化できず、割り当てられたバッファは除去(free)されます。

- プロトタイプ

```

# include <atmi.h>
char * tpalloc (char *type, char *subtype, long size)

```

- パラメータ

パラメータ	説明
type	バッファ・タイプとして、以下のうち1つを指定します <ul style="list-style-type: none"> <li>– STRING: NULLで終わる文字列のデータ送信時に使用します</li> <li>– CARRAY, X_OCTET: 長さが指定された文字型のデータ転送時に使用します</li> <li>– STRUCT, X_C_TYPE: C言語構造体タイプのデータ転送時に使用します</li> <li>– X_COMMON: char、int、longのみ可能なC言語構造体の場合に使用します</li> <li>– FDL(FIELD/バッファ): データを識別子と識別子に該当する値の形式で保存する場合に使用します</li> </ul>

パラメータ	説明
subtype	<p>typeがSTRUCT、X_C_TYPE、X_COMMONの場合、必ずsubtypeを指定します。typeの最初の8バイトとsubtypeの最初の16バイトのみが使用されます。指定された長さを超過して使用した場合、超過した長さは無視されます。指定されたバッファ・タイプがサブタイプを使用しない場合、subtypeは無視され、一般的にNULLが使用されます。</p> <p>割り当てられたバッファ・サイズはデフォルト・サイズ(1024バイト)以上です</p>
size	<p>バッファ・サイズです。CARRAYとX_OCTETでは必ず指定する必要があり、それ以外の場合は省略可能です。0と指定した場合、各バッファのデフォルト・サイズが使用されます。</p> <p>STRING、STRUCT、X_C_TYPE、X_COMMONのデフォルト・サイズは1024バイトです。CARRAYのデフォルト・サイズは0ですが、バッファを割り当てる場合は必ず0より大きい値を指定します</p>

- 戻り値

戻り値	説明
バッファ・ポインタ	関数の呼び出しに成功した場合です。適切な型付きバッファのポインタを返します
NULL	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpalloc()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、NULL形式の場合に発生します
[TPENOENT]	認識できないタイプ、あるいはサブタイプです。STRUCTバッファ・タイプの場合、サブタイプが(構造体のタグ名)SDLFILEに存在しない場合に発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	メモリの割り当てを受けられない運用システムにエラーが発生した場合です
[TPEOTYPE]	要求されているサーバーのデータ型が構造体バッファであるにもかかわらず、該当サーバーのコンパイル時に構造体ファイルと一緒にコンパイルされなかった場合に発生します

- 例

```

#include <stdio.h>
#include <usrinc/atmi.h>
#include "../sdl/demo.s"

void main(int argc, char *argv[])
{
    int ret;
    struct data *buf;
    long len;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret<0) { error processing }
    buf=(struct data *)tpalloc("STRUCT", "data",0);
    data process....

    ret=tpcall("SERVICE", (char *)sndbuf, 0, (char **)&rcvbuf, &len, TPNOFLAGS);

    if (ret<0) {error processing }
    data process....
    tpfree((char *)buf);
    tpend();
}

```

- 関連関数

tpfree(), tprealloc(), tptypes()

## 3.1.45. tpbegin

**tx\_set\_transaction\_timeout()**と**tx\_begin()**の機能を一度に実行できる関数です。Tuxedoで使用するものと同じ形式であるため、Tuxedo用に開発されたアプリケーションを変換しなくても使用できます。関数の機能は、tx\_begin()と同じです。

- プロトタイプ

```
int tpbegin(unsigned long timeout, long flags);
```

- パラメータ

パラメータ	説明
timeout	<b>tx_set_transaction_timeout()</b> 関数に設定する値と同じ意味をもちます。トランザクション・タイムアウト時間を入力します(単位：秒)
flags	現行バージョンではサポートしていませんが、TPNOFLAGSに設定します

- 戻り値

[「3.1.110. tx\\_set\\_transaction\\_timeout」](#)と[「3.1.104. tx\\_begin」](#)を参照してください。

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <tuxinc/tuxfml.h>
void main(int argc, char *argv[])
{
    char *buf;
    int ret;
    long len, revent;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }
    buf=tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }

    data process...
    ret = tpbegin(5, TPNOFLAGS);
    if (ret < 0){ error processing }

    ret = tpcall("SERVICE", (char *)buf, strlen(buf),
                (char **)&buf, &len, TPNOFLAGS);
    if (ret == -1)
    {
        ret = tpabort(TPNOFLAGS);
        if (ret < 0){ error processing }
        tpfree(buf);
        tpend();
        exit(1);
    }
    ret = tpcommit(TPNOFLAGS);
    if (ret < 0){ error processing }
    tpfree(buf);
    tpend();
}
```

- 関連関数

tx\_set\_transaction\_timeout(), tx\_begin()

### 3.1.46. tpbroadcast

Tmaxシステムに登録されているクライアントに要求していないメッセージを送信する関数です。メッセージ送信可能なクライアントは、**tpstart()**でTmaxシステムにすでに接続されている必要があります。この際、クライアントの名前とフラグを正しく定義します。非要求メッセージを受信するには、非要求メッセージを受信するという情報を与える必要があります。tpstart()を使用する場合、TPSTART\_T構造体のフラグ値をTPUNSOL\_POLLやTPUNSOL\_HNDに設定しなければ、非要求メッセージを受信できません。

#### ● プロトタイプ

```
# include <atmi.h>

int tpbroadcast (char *nodename, char *usrname, char *cltname,
                char *data, long len, long flags)
```

#### ● パラメータ

パラメータ	説明
nodename, usrname, cltname	対象クライアントの選択に使用される論理的な名前で、63文字以内で指定します。名前の指定には、疑問符(?)やアスタリスク(*)などのワイルドカード(wildcards)を使用できます。また、NULL値を使用できますが、これはすべてのクライアントに対応するワイルドカードで動作します。長さが0のstring引数は、長が0のクライアント名にのみ対応します。  cltnameで使用する名前は、クライアントがtpstart()を使用してTmaxシステムに初めて接続する際に登録するクライアント名です
data	以前にtpalloc()によって割り当てられたバッファを必ず使用します
len	送信するデータ長です  – dataが指すバッファが、特別な長さ明示が不要なタイプ(STRING、STRUCT、X_COMMON、X_C_TYPE)の場合、lenは無視され、0が使用されます  – dataがNULLの場合もlenは無視されます
flags	以下は、flagsで使用可能な値についての説明です  – TPNOBLOCK  内部バッファが送信メッセージでいっぱいになったときのようにブロッキング状況になった場合、要求は送信されません  – TPNOTIME  関数の呼び出し元がブロック・タイムアウトを無視し、応答が受信されるまで無限に待機します。トランザクション・タイムアウト内でtpbroadcast()を実行した場合、トランザクション・タイムアウトが適用されます



パラメータ	説明
	<p>– TPSIGRSTRT</p> <p>シグナル割り込みを受け入れる場合に使用します。内部でシグナル割り込みが発生し、システム関数の呼び出しが妨害されたときに、システム関数の呼び出しが再実行されます。TPSIGRSTRTフラグが設定されていない状態でシグナル割り込みが発生した場合、関数は失敗し、tperrnoにTPGOTSIGが設定されます</p>

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpbroadcast()が正常に実行されなかった場合、いかなるメッセージもクライアントに送信されず、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、識別子が長すぎる場合や、フラグが有効でない場合に発生します。nodenameを正しく使用しなかった場合、tpbroadcast()が失敗し、TPEINVALを返します。しかし、usernameやcltnameが正しくない場合、どこにもメッセージが渡されず、成功したものとして実行されます
[TPETIME]	TPNOBLOCKやTPNOTIMEが設定されていない状態でブロック・タイムアウトが発生した場合です
[TPEBLOCK]	TPNOBLOCKが設定された状態でブロッキング状況が発生した場合です
[TPGOTSIG]	TPSIGRSTRTが設定されていない状態でシグナルを受信した場合です
[TPEPROTO]	tpbroadcast()が不適切な状況で呼び出された場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
void main(int argc, char *argv[])
{
    int ret;
    char *buf;
```

```

TPSTART_T *tpinfo;

tpinfo = (TPSTART_T *)tpalloc("TPSTART", NULL, sizeof(TPSTART_T));
if (tpinfo==NULL) { error processing }
strcpy(tpinfo->cltname, "cli1");
strcpy(tpinfo->usrname, "navis");
ret=tpstart(tpinfo);
if (ret==-1) { error processing }

buf=tpalloc("STRING", NULL, 0);
if (buf==NULL) { error processing }
data process...

tpbroadcast("tmax", NULL, NULL, buf, 0, TPNOFLAGS);
data process....

tpfree(buf);
tpend();
}

```

- 関連関数

tpalloc(), tpend(), tpstart()

## 3.1.47. tpcall

サーバーとクライアントの同期型のサービス要求を送受信する関数です。同期型通信で**svc**に指定されたサービスにサービス要求を送信し、その応答を受信します。**tpcall()**を呼び出した後、引き続き**tpgetrply()**を呼び出した場合と同一に処理されます。

tx\_beginを呼び出してから、tx\_timeが過ぎた後、tpcallはflagにTPNOTRAN | TPNOREPLYを設定した呼び出しだけ成功し、それ以外はすべてTPETIMEエラーを発生させます。

- プロトタイプ

```

# include <atmi.h>
int tpcall (char *svc, char *idata, long ilen, char **odata, long *olen,
            long flags)

```

- パラメータ

パラメータ	説明
svc	呼び出されるサービス名です。Tmax応用サーバー・プログラムで提供しているものである必要があります

パラメータ	説明
idata	サービス要求のデータに対するポインターです。以前にtpalloc()によって割り当てられたバッファである必要があります。idataのタイプ(type)とサブタイプ(subtype)は、svcがサポートするタイプである必要があります
ilen	<p>送信するデータ長です</p> <ul style="list-style-type: none"> <li>– idataが指すバッファが、特に長さを指定する必要がないタイプ (STRING、STRUCT、X_COMMON、X_C_TYPE) の場合、ilenは無視され、デフォルトで0が使用されます</li> <li>– idataが指すバッファが、長さを指定する必要があるタイプ (X_OCTET、CARRAY、MULTI STRUCTURE) の場合、ilenは0になれません</li> <li>– idataがNULLの場合、ilenは無視されます</li> </ul>
*odata	<p>*odataは、受信する応答バッファのポインターです。バッファは*olenで返された長さ分の応答を受信します。*odataは、必ず以前にtpalloc()によって割り当てられたバッファである必要があります。</p> <p>同一バッファが送信と受信の役割を両方共行う場合、*odataはidataのアドレスで設定される必要があります。応答バッファのサイズ変更可否を決定するには、tpcall()が完了する前に、*odataで割り当てられた応答バッファのサイズと受信した*olenを比較します。受信した*olenの方が大きい場合、割り当てられた応答バッファのサイズが増加します。そうでない場合、サイズは変更されません。</p> <p>idataと*odataで同一バッファが使用されてtpcall()が呼び出された場合、*odataが変更されたら、idataが指すアドレスは今後有効ではありません。*odataは受信データが大きくて変更されることがあり、その他の理由によっても変更されることがあります。</p> <p>*olenが0と返された場合、いかなるデータも受信されず、*odataと*odataが指すバッファも何の変化もありません。*odataやolenがNULLになるのはエラーです</p>
*olen	*odataに返される応答の長さです
flags	<p>呼び出し時に使用されるオプションです。通信方式を指定します。</p> <p>flagsで使用可能な値は以下のとおりです</p> <ul style="list-style-type: none"> <li>– TPNOTRAN</li> </ul> <p>tpcall()の呼び出し元がトランザクション・モードでこのフラグを設定してsvcサービスを要求した場合、svcサービスはトランザクション・モードから除外されて実行されます。トランザクション・モードで、svcがトランザクションをサポートしないサービスである場合、tpcall()を呼び出すときは、フラグを必ずTPNOTRANに設定する必要があります。トランザクション・モード内でtpcall()を呼び出したとき、TPNOTRANが設定されていても、トランザクション・タイムアウト(timeout)の影響を受けます。つ</p>

パラメータ	説明
	<p>まり、トランザクション・タイムアウトが過ぎた後のTPNOTRANを適用したtpcallも、サービスを呼び出さずに失敗処理するという意味です。TPNOTRANで呼び出されたサービスが失敗した場合、呼び出し元のトランザクションには影響を及ぼしません</p> <p>– TPNOCHANGE</p> <p>TPNOBLOCKフラグを設定した状態で、内部バッファが送信メッセージでいっぱいになったときのようなブロッキング状況になった場合、サービス要求は失敗します。TPNOCHANGEはtpcall()のTx部分にのみ適用されます。TPNOBLOCKフラグを設定せずにtpcall()を呼び出したときにブロッキング状況が発生すると、関数の呼び出し元はブロッキング状況が解除されるか、タイムアウト(トランザクション・タイムアウト、またはブロック・タイムアウト)が発生するまで待機します</p> <p>– TPNOTIME</p> <p>関数の呼び出し元がブロック・タイムアウトを無視し、応答を受信するまで無限に待機します。トランザクション・タイムアウト内でtpcall()を実行した場合、トランザクション・タイムアウトが適用されます</p> <p>– TPSIGRSTRT</p> <p>シグナル割り込みを受け入れる場合に使用します。内部でシグナル割り込みが発生し、システム関数の呼び出しが妨害されたとき、システム関数の呼び出しが再実行されます。TPSIGRSTRTフラグが設定されていない状態でシグナル割り込みが発生した場合、関数は失敗し、tperrnoにTPGOTSIGが設定されます</p>

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpcall()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効ではないか、フラグが有効でない場合です。たとえば、svcがNULLであるか、dataがtpalloc()で割り当てられていないバッファを指します
[TPENOENT]	svcというサービスが存在しないため、サービスを要求できない場合です
[TPEITYPE]	dataのタイプおよびサブタイプが、svcがサポートしていないタイプです

エラーコード	説明
[TPEOTYPE]	<p>受信された応答バッファのタイプあるいはサブタイプが、呼び出し元が認識できないタイプです。フラグがTPNOCHANGEと設定されているにもかかわらず、*odataが指すバッファのタイプおよびサブタイプが、受信された応答バッファのタイプおよびサブタイプと一致しない場合、*odataの内容と*olenはすべて変更されません。</p> <p>呼び出し元がトランザクション・モードでサービスを要求した場合、そのトランザクションは応答が無視されるためロールバックされます</p>
[TPETRAN]	トランザクション・サービスの呼び出し時に、データベースに問題が発生してxa_starが失敗した場合です
[TPETIME]	<p>タイムアウトが発生した場合です。関数の呼び出し元がトランザクション・モードの場合はトランザクション・タイムアウトが発生し、そのトランザクションはロールバックされます。</p> <p>トランザクション・モードではなく、TPNOTIMEとTPNOBLOCKのどちらも指定されていない場合、ブロック・タイムアウトが発生します。この2つの場合に、*odataの内容と*olenは変更されません。</p> <p>トランザクション・タイムアウトが発生した場合、新規サービス要求を送信したり応答を待機したりすることは、トランザクションがロールバックされるまで[TPETIME]エラーで失敗します</p>
[TPESVCFAIL]	<p>サービス要求に対する応答を送信するサービス・ルーチンでアプリケーション・エラーが発生し、TPFAILでtpreturn()を呼び出した場合です。サービス要求が受信されたら、その内容は*odataが指すバッファを通じて使用できます。</p> <p>トランザクション・タイムアウトが発生し、トランザクションがロールバックされる前に他の通信が実行されることがあります。そういった通信は正常に処理される場合もあり、失敗する場合もあります。通信が正常に実行されるには、TPNOTRANが設定されている必要があります。呼び出し元のトランザクション・モードで実行された作業は、トランザクションの完了時にすべてロールバックされます</p>
[TPESVCERR]	<p>サービス・ルーチン実行中、あるいはtpreturn()(たとえば、誤った引数が送信された場合)実行中にエラーが発生した場合です。エラーが発生すると、いかなる応答データも返さず、*odataの内容または*olenもすべて変更されません。</p> <p>tpallocで生成されていないバッファを使用した場合や、割り当てられたバッファのTmaxヘッダーが正しくないポインター(memcpyなど)の影響を受けた場合、あるいはtpacallまたはtpconnectのcdで返した場合、Recvモードでサービスが有効でない対話型の内容のときに発生します。ただし、tpreturnを実行した場合、クライアントはTPESVCERRを受信します。</p>

エラーコード	説明
	<p>クライアントが強制的に対話を解除し、サービス・プログラムがTPEV_DISCOMNを受信するのと同じTPEV_DISCOMNイベントが発生した場合、クライアント・サービスのtpreturnにTPESVCERR tperrnoが転送されます。</p> <p>関数の呼び出し元がトランザクション・モードの場合、トランザクション・タイムアウトが発生する前に他の通信が実行されることがあります。そういった通信は正常に処理される場合もあり、または失敗する場合があります。通信が正常に実行されるには、TPNOTRANが設定されている必要があります。呼び出し元のトランザクション・モードで実行された作業は、トランザクションの完了時にすべてロールバックされます。</p> <p>Tmax環境ファイルにサービス別にSVCTIMEOUTを設定できます。サービスの実行時間が該当時間を超えた場合、サービスは実行を停止し、TPESVCERRを返します。SVCTIMEOUTが発生した場合、tpsvctimeout()を呼び出しますが、この関数内でバッファ解除やロギング作業など、業務別に適切な作業を行うことができます</p>
[TPEBLOCK]	TPNOBLOCKが設定された状態でブロッキング状況が発生した場合です
[TPGOTSIG]	TPSIGRSTRTが設定されていない状態でシグナルが受信された場合です
[TPEPROTO]	tpcall()が不適切な状況で呼び出された場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

#### ● 例

```
#include <stdio.h>
#include <usrinc/atmi.h>

void main(int argc, char *argv[])
{
    int ret;
    char *sndbuf, *rcvbuf;
    long sndlen, rcvlen;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    sndbuf = (char *)tpalloc("CARRAY", NULL, 20);
    if (sndbuf==NULL) {error processing };

    rcvbuf = (char *)tpalloc("CARRAY", NULL, 20);
    if (rcvbuf==NULL) {error processing };

    data process....
```

```

    sndbuf=strlen(sndbuf);
    ret=tpcall("SERVICE", sndbuf, sndlen, &rcvbuf, &rcvlen, TPNOCHANGE);
    if (ret==-1) { error processing }

    data process....

    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
}

```

- 関連関数

tpalloc(), tpacall(), tpgetrply(), tpreturn()

## 3.1.48. tpcallsvg

特定サーバー・グループに属しているサービスをサーバーとクライアントで呼び出す関数です。COUSINIにまとめられたマルチサーバー・グループ環境で特定サーバー・グループに属するサービスを指定し、同期型通信でサービス要求を送信し、これに対する応答を受信できます。特定サーバー・グループを指定してサービスを呼び出すこと以外は、tpcall()と同一の動作をします。

サービス内で他のサービスを呼び出す場合、**tpgetmysvgno()**を使用して現在自身が属するサーバー・グループの番号を確認後、同じサーバー・グループに属するサービスを呼び出すこともできます。

- プロトタイプ

```

#include <usrinc/tmaxapi.h>
int tpcallsvg (int svgno, char *svc, char *idata, long ilenl, char **odata,
               long *olen, long flagsl)

```

- パラメータ

svgno以外のパラメータは「[3.1.47. tpcall](#)」を参照してください。

パラメータ	説明
svgno	<p>呼び出すサービスが属するサーバー・グループの番号を指定します。サーバー・グループの番号はtpgetsvglist()とサーバー・グループのシリアル番号を確認できます。</p> <p>1に設定した場合、tpcall()と同じ動作をします</p>

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

エラーについての詳細内容は[「3.1.47. tpcall」](#)を参照してください。

- 例

<クライアント・プログラム>

```
#include <usrinc/tmaxapi.h>
#include "../fdl/demo_fdl.h"

int main(int argc, char *argv[])
{
    FBUF    *sndbuf, *rcvbuf;
    int     ret, i;
    char    sndata[30],    rcvdata[30];
    long    sndlen, rcvlen;
    struct svglist *svg_list;
    char    svc[32];

    ret = tmaxreadenv( "tmax.env", "TMAX" );
    ret = tpstart((TPSTART_T *)NULL);
    sndbuf = (FBUF *)tpalloc("FIELD", NULL, 0);
    rcvbuf = (FBUF *)tpalloc("FIELD", NULL, 0);
    strcpy(sndata, argv[1]);
    ret = fbput(sndbuf, INPUT, sndata, 0);
    strcpy(svc, "FDLToupper");
    svg_list = (struct svglist *)tpgetsvglist(svc, 0);

    for(i=0; i< svg_list->ns_entry; i++)
    {
        printf("\n");
        printf(" >>> tpcallsvg ( %d th svg )\n", i+1);
        strcpy(sndata, argv[1]);
        ret = fbput(sndbuf, INPUT, sndata, 0);
        if (tpcallsvg(svg_list->s_list[i], svc, (char *)sndbuf, 0,
            (char **)&rcvbuf, &rcvlen, 0) == -1)
        {
            printf("tpcall failed! errno = %d[%s]\n", tperrno, tpstrerror(tperrno));

            tpfree((char *)sndbuf);
            tpfree((char *)rcvbuf);
        }
    }
}
```



```

        tpend();
        exit(1);
    }
    ret = fbget(rcvbuf, OUTPUT, rcvdata, 0);
    fbprint(rcvbuf);
}
tpfree((char *)sndbuf);
tpfree((char *)rcvbuf);
tpend();
return 0;
}

```

- 関連関数

tpcall(), tpacallsvg(), tpgetsvglist(), tpgetmysvgno()

## 3.1.49. tpcancel

サーバーとクライアントの応答をキャンセルする関数です。**tpacall()**が返した呼び出し記述子であるcdをキャンセルします。

- プロトタイプ

```

#include <atmi.h>
int tpcancel (int cd)

```

- パラメータ

パラメータ	説明
cd	tpacall()が返した呼び出し記述子で、キャンセル対象を設定します。グローバル・トランザクション(global transaction)と関連したサービスはキャンセルできません。サービス応答が正常にキャンセルされた場合、cdは無効化され、cdを通じて受信した応答もすべて無視されます

- 戻り値

戻り値	説明
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpcancel()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEBADDESC]	cdが有効でない記述子です
[TPETRAN]	cdが呼び出し元のグローバル・トランザクションに関連しています。cdは有効で、呼び出し元の現在のトランザクションは影響を受けません
[TPEPROTO]	tpcancel()が不適切な状況で呼び出された場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合は。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合は

## ● 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
void main(int argc, char *argv[])
{
    char *test[2];
    int ret, i, cd[2];
    long len;

    if (argc != 4) { error processing }
    ret=tpstart((TPSTART_T *)NULL;
    if (ret==-1) { error processing }

    for (i=0; i<3; i++)
    {
        test[i] = tpalloc("STRING",NULL,0);
        if (test[i])==NULL} { error processing }
        strcpy(test[i],argv[i+1]);
        cd[i]=tpacall("SERVICE", test[i], 0, TPNOTIME);
    }

    ret=tpcancel(cd[1]);          /* 2番目の応答をキャンセルします */
    if (ret==-1) { error processing }
    for (i=0; i<3; i++)
    {
        ret=tpgetrply(&cd[i], (char **)&test[i], &len, TPNOTIME)
        if (ret==-1) printf("Can't rcv data from service of %d\n",cd[i]);
        else prtinf("%dth rcv data : %s\n", I+1, test[I]);
        tpfree(test[I]);
    }
    tpend();
}
```

## ● 関連関数

tpacall()

## 3.1.50. tpcommit

サーバーとクライアントでグローバル・トランザクションをコミットする関数です。**tx\_commit()**と同一機能を実行します。

tpcommit()は、Tuxedoで使った関数をTmaxシステムにそのまま適用するために使用する関数です。Tuxedoで開発されたプログラムを変更することなく、Tmaxに変更できるようサポートします。

- プロトタイプ

```
#include <tuxfml.h>
int tpcommit (long flags)
```

- パラメータ

パラメータ	説明
flags	意味のないパラメータです。TPNOFLAGSに設定します

- 戻り値

[「3.1.105. tx\\_commit」](#)を参照してください

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <tuxinc/tuxfml.h>

void main(int argc, char *argv[])
{
    char *buf;
    int ret;
    long len, revent;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf=tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }

    data process...
    ret = tpbegin(5, TPNOFLAGS);
    if (ret < 0){ error processing }
```

```

ret=tpcall("SERVICE",(char *)buf,strlen(buf),(char **)&buf,&len,TPNOFLAGS);

if (ret == -1)
{
    ret = tpabort(TPNOFLAGS);
    if (ret < 0){ error processing }
    tpfree(buf);
    tpend();
    exit(1);
}
ret = tpcommit(TPNOFLAGS);
if (ret < 0){ error processing }

tpfree(buf);
tpend();
}

```

- 関連関数

tx\_commit()

## 3.1.51. tpconnect

サーバーとクライアントでプログラムが対話型サービスsvcと通信を接続する関数です。通信は、同時に受信または送信のみ可能なハーフ・デュプレックス(half-duplex)形式です。接続を設定する過程で、関数の呼び出し元はサービス・ルーチンにデータを渡すことができます。対話型サービスは、TPSVCINFO構造体を通じてdataとlenを受信するため、tpconnect()に渡されたデータを受信するために**tprecv()**を呼び出す必要があります。

- プロトタイプ

```

# include <atmi.h>
int tpconnect (char *svc, char *data, long len, long flags)

```

- パラメータ

パラメータ	説明
svc	対話型サービスのサービス名を指定します
data	呼び出し元がデータを渡す場合、tpalloc()によって以前割り当てられたバッファである必要があります。dataのタイプ(type)とサブタイプ(subtype)は、svcがサポートするタイプである必要があります
len	渡すデータ長を指定します

パラメータ	説明
	<ul style="list-style-type: none"> <li>– dataが指すバッファが、特別な長さ明示が不要なタイプ (STRING、STRUCT、X_COMMON、X_C_TYPE) の場合、lenは無視され、0が使用されます</li> <li>– dataが指すバッファが、長さを指定する必要があるタイプ (X_OCTET、CARRAY、MULTI STRUCTURE) の場合、lenは0になれません</li> <li>– dataがNULLの場合、lenは無視されます。この場合、データは何も対話型サービスに残しません</li> </ul>
flags	<p>flagsで使用可能な値は以下のとおりです</p> <ul style="list-style-type: none"> <li>– TPNOTRAN <p>tpconnect()の呼び出し元がトランザクション・モードでこのフラグを設定してsvcサービスを要求した場合、svcサービスはトランザクション・モードから除外されて実行されます。トランザクション・モードで、svcがトランザクションをサポートしないサービスである場合、tpconnect()を呼び出すときは、フラグをTPNOTRANに設定する必要があります。トランザクション・モード内でtpconnect()を呼び出したとき、TPNOTRANが設定されていても、トランザクション・タイムアウトの影響を受けます。TPNOTRANで呼び出されたサービスが失敗した場合、呼び出し元のトランザクションには影響を及ぼしません</p> </li> <li>– TPSENDONLY <p>接続が完了した後、関数の呼び出し元は最初データの送信のみが可能で、要求されたサービスのみ行えるように設定するフラグです。呼び出し元が最初に通信制御権をもちます。TPSENDONLYまたはTPRECVONLYのいずれかを必ず指定する必要があります</p> </li> <li>– TPRECVONLY <p>接続が完了した後、関数の呼び出し元はデータの受信のみが可能で、要求されたサービスが最初にデータの送信を開始できるようにするフラグです。つまり、要求されたサービスが最初に通信制御権をもちます。TPSENDONLYまたはTPRECVONLYのいずれかを必ず指定する必要があります</p> </li> <li>– TPNOTIME <p>関数の呼び出し元がブロック・タイムアウトを無視し、応答を受信するまで無限に待機します。トランザクション・タイムアウト内でtpconnect()を実行した場合、トランザクション・タイムアウトが適用されます</p> </li> <li>– TPSIGRSTRT <p>シグナル割り込みを受け入れる場合に使用します。内部でシグナル割り込みが発生し、システム関数の呼び出しが妨害されたとき、システム関数の呼び出しが再実</p> </li> </ul>

パラメータ	説明
	行されます。TPSIGRSTRTフラグが設定されていない状態でシグナル割り込みが発生した場合、関数は失敗し、tperrnoにTPGOTSIGが設定されます

- 戻り値

戻り値	説明
記述子(descriptor)	関数の呼び出しに成功した場合です。以降、接続のために参照される記述子を返します
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpconnect()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。特別な言及がなければ、関数の呼び出しに失敗した場合、呼び出し元のトランザクションに影響を及ぼしません。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。  たとえば、svcとdataがNULLであれば、指示するバッファがtpalloc()で割り当てられていない場合や、フラグがTPSENDONLYまたはTPRECVONLYに指定されていない場合、またはフラグが有効でない場合に発生します
[TPENOENT]	svcというサービスが存在しないため、サービスを要求できない場合に発生します
[TPEITYPE]	dataのタイプおよびサブタイプが、svcがサポートしていないタイプおよびサブタイプである場合に発生します
[TPELIMIT]	接続数が制限に達したため、サービスを要求できない場合です
[TPETRAN]	トランザクション・サービスの呼び出し時に、データベースに問題が発生してxa_startが失敗した場合です
[TPETIME]	タイムアウトが発生した場合です。関数の呼び出し元がトランザクション・モードの場合、トランザクション・タイムアウトが発生し、そのトランザクションはロールバックされます。関数の呼び出し元がトランザクション・モードではなく、かつTPNOTIMEとTPNOBLOCKのどちらも指定されていない場合は、ブロック・タイムアウトが発生しました。トランザクション・タイムアウトが発生した場合、新規サービス要求はトランザクションがロールバックされるまで[TPETIME]エラーとなります
[TPEBLOCK]	TPNOBLOCKが設定された状態でブロッキング状況が発生した場合です
[TPGOTSIG]	TPSIGRSTRTが設定されていない状態でシグナルが受信された場合です
[TPEPROTO]	tpconnect()が不適切な状況で呼び出された場合です

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>

void main(int argc, char *argv[])
{
    char *buf;
    int ret, cd;
    long len, revent;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf=tpalloc("STRING",NULL,0);
    if (buf=NULL) { error procesing }

    data process ....

    cd = tpconnect("SERVICE",sndbuf,0,TPRECVONLY);
    if (cd==-1) { error processing }
    data process....

    ret=tprecv(cd, &buf, &len, TPNOFLAGS, revent);
    if (ret==-1) { error processing }
    tpfree(buf);
    tpend();
}
```

- 関連関数

tpalloc(), tpdiscon(), tprecv(), tpsend()

### 3.1.52. tpdeq

サーバーとクライアントでRQからデータをロードする関数です。**tpenq()**を使用してサービスの要求結果を受信するか、サービス名をNULLにして保存したデータを読み取ります。tpenq()の呼び出し時、flagsに

TPNOREPLYを設定した場合、サービスは結果を受信できません。そのため、トランザクション・モードでtpdeq()を実行中にエラーが発生してもトランザクションは影響を及ぼしません。

## ● プロトタイプ

```
# include <tmaxapi.h>
int tpdeq (char *qname, char *svc, char **data, long *len, long flags)
```

## ● パラメータ

パラメータ	説明
qname	データを保存するRQの名前です。configファイルに登録された名前である必要があります
svc	tpenq()を呼び出す場合、svcに渡した名前である必要があります。サービス名でtpenq()を呼び出した場合、サービスが自動要求され、結果がRQに保存されます。サービス結果を受信するには、tpdeq()も同一サービス名を入力する必要があります。  サービス名をNULLでtpenq()を呼び出した場合、tpdeq()も同一サービス名を入力します。サービス名がNULLの場合、サービス名に関係なく、キューに蓄積しているすべてのデータを1つずつロードできます。tpenq()の場合、エラーやシステム障害によってフェイル・キューに保存されたデータをtpdeq()するには、svcに_rq_sub_queue_name[TMAX_FAIL_QUEUE]を与え、deqする必要があります
*data	tpalloc()によって割り当てられたバッファに対するポインターです。関数が正常に返された場合、*dataは受信されたデータが保存されます
len	tpdeq()が正常に受信したデータ長です。tpdeq()は、必要であれば応答内容が指定されたバッファに受信されるようバッファのサイズを増加させます。  lenは*dataのデータ長です。*dataは受信データが大きくて変更されることもあり、それ以外の理由によって変更されることもあります。lenが呼び出し前のバッファの総サイズより大きい場合、lenが該当バッファの新規サイズになります。lenが0と返された場合、いかなるデータも受信されず、*dataとlenが指示するバッファすべて何も変化はありません。  *dataやlenがNULLになるのはエラーです
flags	flagsで使用可能な値は以下のとおりです  – TPRQS  RQからデータを取得時に使用されます。replyキューからサービス結果を取得するために設定されます  – TPFUNC



パラメータ	説明
	<p>サービスごとにRQデータを管理するときに使用されます。TPFUNCフラグが設定されていない場合、最初のtpenq()により保存されたデータが除去されます。データの保存前に除去が必要な場合、tpenq()の呼び出し時にTPFUNCと一緒に設定します</p> <p>– TPBLOCK</p> <p>tpdeq()を呼び出したとき、ブロック・タイムアウト時間の間、メッセージが来るまで待機します</p> <p>– TPNOTIME</p> <p>TPBLOCKと一緒に使用された場合、ブロック・タイムアウト時間に関係なく、応答が来るまで待機します</p> <p>– 0(zero)</p> <p>自身が接続したクライアントのバッファからデータを取得時に使用します</p>

- 戻り値

戻り値	説明
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpdeq()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、データがtpalloc()で割り当てられていないバッファを指す場合、あるいはflagsが有効でない場合に発生します
[TPGOTSIG]	TPSIGRSTRTが設定されていない状態でシグナルが受信された場合です
[TPEMATCH]	サービス名が正しくない場合や除去するデータがない場合、あるいは該当条件を満たす除去データが見つからなかった場合に発生します
[TPENOENT]	存在しないqnameが使用された場合です
[TPEPROTO]	tpdeq()が不適切な状況で呼び出された場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```

#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

void main(int argc, char *argv[])
{
    int ret;
    char *buf;
    long len;
    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf = (char *)tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
    data process....

    ret=tpenq("RQ", "SERVICE", (char *)buf, strlen(buf), TPRS);
    if (ret==-1) { error processing }
    data process....

    ret=tpdeq("RQ", "SERVICE", (char **)&buf, (long *)&len, TPRS);
    if (ret==-1) { error processing }
    data process....

    tpfree(buf);
    tpend();
}

```

- 関連関数

tpenq(), tpqstat()

### 3.1.53. tpdeq\_ctl

サーバーとクライアントでトランザクションをサポートし、RQからデータをロードする関数です。**tpenq\_ctl()**を使用してサービスを要求した結果、あるいはサービス名をNULLにして保存したデータを読み取る関数です。**tpenq\_ctl()**を呼び出したとき、flagsにTPNOREPLYを設定したサービスは結果を受信できません。**tpdnq\_ctl()**関数は、トランザクション・モードで実行した場合、データをRQからロードする間トランザクションで処理されます。2回以上の**tpdep\_ctl()**を1つのトランザクションに結合した場合、すべてのデータがRQからロード中にエラーが発生するとロールバックされます。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tpdeq_ctl(char *qname, char *svc, TMQCTL *ctl, char **data,
              long *len, long flags);
```

#### ● パラメータ

パラメータ	説明
qname	データを保存するRQの名前です。環境ファイルに登録された名前である必要があります
svc	<p>tpenq_ctl()を呼び出す場合、svcに渡した名前である必要があります。</p> <p>サービス名でtpenq_ctl()を呼び出した場合、サービスが自動要求され、結果がRQに保存されます。サービス結果を受信するには、tpdeq_ctl()も同一サービス名を入力します。サービス名をNULLでtpenq_ctl()を呼び出した場合、tpdeq_ctl()も同一サービス名を入力します。サービス名がNULLの場合、サービス名に関係なく、キューに蓄積しているすべてのデータを1つずつdeqできます。</p> <p>tpenq_ctl()関数がエラーやシステム障害によってフェイル・キューに保存されたデータをtpdeq_ctl()するには、svcに_rq_sub_queue_name [TMAX_FAIL_QUEUE]を与え、deqする必要があります</p>
ctl	<p>– * flags</p> <ul style="list-style-type: none"> <li>• TPRQS_AUTOACK : deqするとき、別途にackを与えなくてもRQSから削除されるようにします</li> <li>• TPRQS_NOAUTOACK : deqするとき、RQSから削除されないようにします</li> <li>• TPRQS_ACK_SUCCESS : ctlに該当するq要素をackを与えて削除します</li> <li>• TPRQS_ACK_FAIL : deqするとき、RQSから削除されないようにします</li> </ul> <p>– * exp_time</p> <p>deqした後、与えられた時間が過ぎても応答がない場合は、q要素が再びdeqできる状態に戻ります(単位: 秒)</p>
data	以前にtpalloc()によって割り当てられたバッファに対するポインターである必要があります。tpdeq_ctl()が正常に実行された場合、受信されたデータが保存されます
len	<p>tpdeq_ctl()が正常に受信したデータ長です。必要な場合、応答内容が指定されたバッファに受信されるようにバッファのサイズを増加させます。</p> <p>*dataは、受信データが大きくて変更されることもあり、それ以外の理由によって変更されることもあります。lenが呼び出し前のバッファの総サイズより大きい場合、lenが該当バッファの新規サイズになります。lenが0と返された場合、いかなるデータ</p>

パラメータ	説明
	も受信されず、*dataとlenが指示するバッファへすべて何も変化はありません。*dataやlenがNULLになるのはエラーです
flags	<p>データ処理タイプを設定します。</p> <p>以下は、flagsで使用可能な値についての説明です</p> <ul style="list-style-type: none"> <li>– TPRQS <p>RQからデータを取得時に使用されます。replyキューからサービス結果を取得するために設定されます</p> </li> <li>– TPFUNC <p>サービスごとにRQデータを管理するときに使用されます。TPFUNCフラグが設定されていない場合、最初のtpenq()により保存されたデータが除去されます。データの保存前に除去が必要な場合、tpenq()の呼び出し時にTPFUNCと一緒に設定します</p> </li> <li>– TPBLOCK <p>tpdeq_ctl()を呼び出したとき、ブロック・タイムアウト時間の間、メッセージが来るまで待機します</p> </li> <li>– TPNOTIME <p>TPBLOCKと一緒に使用された場合、ブロック・タイムアウト時間に関係なく、応答が来るまで待機します</p> </li> <li>– 0(zero) <p>自身が接続したクライアントのバッファからデータを取得時に使用します</p> </li> </ul>

- 戻り値

戻り値	説明
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tpdeq\_ctl()が正常処理されていない場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、データがtpalloc()で割り当てられていないバッファを指す場合、あるいはflagsが有効でない場合に発生します

エラーコード	説明
[TPGOTSIG]	TPSIGRSTRTが設定されていない状態でシグナルが受信された場合です
[TPEMATCH]	サービス名が正しくない場合です。あるいは、該当条件を満たすデータが見つからない場合です
[TPENOENT]	存在しないqnameが使用された場合です
[TPEPROTO]	tpdeq_ctl()が不適切な状況で呼び出された場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

main(int argc, char *argv[])
{
    char    *sndbuf, *rcvbuf;
    long    rcvlen, sndlen, revent;
    int ret, i, cd;
    TMQCTL  *ctl;

    ctl = (TMQCTL*)malloc(sizeof(TMQCTL));
    memset(ctl, 0, sizeof(TMQCTL));
    if (argc != 2) {
        printf("Usage: toupper_rq string\n");
        exit(1);
    }

    if ( (ret = tmaxreadenv( "tmax.env", "TMAX" )) == -1 ){
        printf( "tmax read env failed\n" );
        exit(1);
    }

    if (tpstart((TPSTART_T *)NULL) == -1){
        printf("tpstart failed\n");
        exit(1);
    }

    if ((sndbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL) {
```

```

    printf("sendbuf alloc failed !\n");
    tpend();
    exit(1);
}

if ((rcvbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL) {
    printf("recvbuf alloc failed !\n");
    tpfree((char *)sndbuf);
    tpend();
    exit(1);
}

ret = tx_begin();
if(ret < 0)
{
    fprintf(stderr, "tx_begin() fail\n");
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    exit(1);
}

strcpy(sndbuf, argv[1]);
cd = tpdeq_ctl("txrq1", "TOUPPER", ctl, &rcvbuf, &rcvlen, TPRS );
if (cd < 0)
{
    printf("tpdeq failed [%s]\n", tpstrerror(tperrno) );
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
    exit(1);
}

cd = tpdeq_ctl("txrq1", "TOUPPER", ctl, &rcvbuf, &rcvlen, TPRS );
if (cd < 0)
{
    printf("tpdeq failed [%s]\n", tpstrerror(tperrno) );
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
    exit(1);
}

data process....
ret = tx_commit();
if(ret < 0)
{
    fprintf(stderr, "tx_commit() fail\n");

```

```

        tpfree((char *)sndbuf);
        tpfree((char *)rcvbuf);
        tx_rollback();
        exit(1);
    }
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
}

```

- 関連関数

tpenq\_ctl(), tpqstat()

### 3.1.54. tpdicon

サーバーとクライアントで対話型通信の接続を終了する関数です。tpconnect()で接続したサービスの場合、極めて例外的な場合に接続を即時終了し、接続された相手方にTPEV\_DISCONIMMイベントを発生させます。

tpdicon()は、対話型通信を開始した側でのみ呼び出すことができ、記述子を提供したサービスでは呼び出すことができません。対話型サービスと通信するプログラムは、対話型通信を終了できます。正しい結果を保証するには、サービス側からtpreturn()で終了するのが望ましいです。

tpdicon()は、接続を強制的に終了します。強制終了した場合、目的地に届いていない一部のデータは失われることがあります。tpdicon()は、接続された相手方プログラムが呼び出し元のトランザクションに関わっている状況で呼び出されることもあります。この場合トランザクションはキャンセルされ、データは失われることがあります。tpdicon()を呼び出す関数の呼び出し元が通信制御権をもつ必要はありません。

- プロトタイプ

```

#include <atmi.h>
int tpdicon (int cd)

```

- パラメータ

パラメータ	説明
cd	tpconnect()で返した参照記述子です

- 戻り値

戻り値	説明
-1	関数の呼び出しにエラーが発生した場合です。tperrnoにエラーコードが設定されます

- エラー

tpdiscon()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEBADDESC]	cdが有効でない場合です。あるいは、すでに対話型サービスで使用されている記述子です
[TPETIME]	タイムアウトが発生した場合です。cdは有効でなくなります
[TPEPROTO]	tpdiscon()が不適切な状況で呼び出された場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <stdlib.h>
#include <usrinc/atmi.h>

void main(int argc, char *argv[])
{
    int ret, cd;
    char *buf;
    long len, revent;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf=tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }

    data process....
    cd=tpconnect("SERVICE",buf,0,TPRECVONLY);

    if (cd==-1) { error processing }
    data process....
    ret=tprecv(cd, (char **)&buf, &len, TPNOFLAGS, &revent);
    if (ret==-1 && revent != TPEV_SENDOONLY && revent != TPEV_SVCSUCC)
    { error processing }
    printf("received data = %s\n", buf);
    if (atoi(buf)>90) {
        ret=tpdiscon(cd);
        if (ret==-1) {error processing }
        tpfree(buf);
        tpend();
    }
}
```



```

        exit(1);
    }

    data process....
    tpfree(buf);
    tpend();
}

```

- 関連関数

tpconnect(), tprecv(), tpreturn(), tpsend()

### 3.1.55. tpenq

サーバーとクライアントでRQにデータを保存する関数です。Tmaxシステムは、システムの障害やエラーによるサービス不可能な状態でも、RQに保存されたデータは整合性を保証できます。RQにデータを保存しておき、様々な状況でシステムがダウンして、復旧後に再度実行された場合、まだ処理できていないデータを引き続き処理できます。

**tpcall()**や**tpacall()**でサービスを要求した場合、該当サービスが実行するデータが累積していると、サービスを要求したデータも待機(waiting)します。この際、システムの障害やエラーによってシステムがダウンした場合、待機中のデータは失われます。こういった問題点を補完し、データの整合性を保証できるように、**tpenq()**はサービスを要求する場合、データをRQに保存します。トランザクション・モードで実行しても、トランザクション・モードから除外されるため、トランザクション・モードで関数を実行中にエラーが発生してもトランザクションは影響を及ぼしません。

- プロトタイプ

```

# include <tmaxapi.h>
int tpenq (char *qname, char *svc, char *data, long len, long flags)

```

- パラメータ

パラメータ	説明
qname	qnameは、データを保存するRQの名前です。configファイルに登録された名前である必要があります
svc	データをRQに保存し、svc名がNULLでない場合、即時サービスを要求します。  svc名がNULLの場合、データはRQに保存され、サービスは実行されません。この場合、後にtpdeq()を使用してサービスを要求する必要があります。svcで命令されたサービスがない場合、またはサービスを実行して処理結果を受信していない状態でシステム障害が発生した場合、このデータは内部的にフェイル・キューに保存されます。データは <b>tpdeq()</b> でサービスを再要求するか、エラー処理をする必要があります

パラメータ	説明
data	NULL値の場合を除き、必ずtpalloc()で割り当てられたバッファに対するポインターである必要があります。dataのタイプとサブタイプは、svcがサポートするタイプである必要があります
len	送信するデータ長です <ul style="list-style-type: none"> <li>dataが指すバッファが、特別な長さ明示が不要なタイプ (STRING、STRUCT、X_COMMON、X_C_TYPE) の場合、lenは無視され、0が使用されます</li> <li>dataが指すバッファが、長さを指定する必要があるタイプ (X_OCTET、CARRAY、MULTI STRUCTURE) の場合、lenは0になれません</li> <li>dataがNULLの場合、lenは無視され、データなしでサービス要求が送信されます</li> </ul>
flags	flagsに使用可能な値は以下のとおりです <ul style="list-style-type: none"> <li>TPRQS <p>svcがNULLでない場合、svcに指定されたサービスを要求し、処理結果をRQに保存します。サービスの処理結果は、<b>tpdeq()</b>を利用して受信します。svcがNULLの場合、データはRQに保存され、サービスは実行しません</p> </li> <li>TPNOREPLY <p>svcがNULLでない場合、svcに指定されたサービスを要求しますが、処理結果はRQに保存しません。svcがNULLの場合、データはRQに保存され、サービスは実行しません</p> </li> <li>TPFUNC <p>サービスごとにRQデータを管理するときに使用されます。TPFUNCフラグが設定されていない場合、最初のtpenq()により保存されたデータが除去されます。データの保存前に除去が必要な場合、tpenq()の呼び出し時にTPFUNCと一緒に設定します</p> </li> <li>0(zero) <p>サービス処理結果をRQに保存せず、関数の呼び出し元が接続されているTmaxシステムのクライアント・バッファに保存します。サービスはRQを使用して要求しますが、結果はtpcall()のように関数の呼び出し元が接続したクライアントのバッファから取得時に使用します。0(zero)フラグが設定された場合、後で処理結果を受信するためには、tpdeq()の呼び出し時にflagsに0(zero)を設定します</p> </li> </ul>

- 戻り値

戻り値	説明
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpenq()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、データがtpalloc()で割り当てられていないバッファを指す場合、あるいはflagsが有効でない場合に発生します
[TPENOENT]	存在しないqnameが使用された場合です
[TPEQFULL]	持続的なサービス結果のため指定されたキューのサイズを超えた場合に発生します
[TPGOTSIG]	TPSIGRSTRTが設定されていない状態でシグナルが受信された場合です
[TPEPROTO]	tpenq()が不適切な状況で呼び出された場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

void main(int argc, char *argv[])
{
    int ret;
    char *buf;
    long len;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf = (char *)tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
    data process....

    ret=tpenq("RQ", "SERVICE", (char *)buf, strlen(buf), TPRQS);
    if (ret==-1) { error processing }
    data process....

    ret=tpdeq("RQ", "SERVICE", (char **)&buf, (long *)&len, TPRQS);
    if (ret==-1) { error processing }
    data process....
    tpfree(buf);
    tpend();
}
```

- 関連関数

tpdeq(), tpqstat()

### 3.1.56. tpenq\_ctl

サーバーとクライアントでトランザクションをサポートし、RQデータを保存する関数です。Tmaxシステムは、システムの障害やエラーによってサービス不可能な状態でも、RQに保存されたデータは整合性を保証できます。RQにデータを保存しておき、様々な状況でシステムがダウンして、復旧以降に再度実行された場合、まだ処理できていないデータを引き続き処理できるようにします。

**tpcall()**や**tpacall()**でサービスを要求した場合、該当サービスが実行するデータが累積していると、サービスを要求したデータも待機(waiting)します。この際、システムの障害やエラーによってシステムがダウンした場合、待機中のデータは失われます。こういった問題点を補完し、データの整合性を保証できるように、**tpenq\_ctl()**はサービスを要求する場合、データをRQに保存します。

**tpenq\_ctl()**関数は、トランザクション・モードで実行した場合、データをRQに保存するまでトランザクションで処理されます。2回以上の**tpenq\_ctl()**を1つのトランザクションに結合した場合、すべてのデータがRQに保存されるまで**tpdeq\_ctl()**を使用してロードすることはできず、保存中にエラーが発生するとロールバックされます。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tpenq_ctl(char *qname, char *svc, TMQCTL *ctl, char *data, long len,
              long flags);
```

- パラメータ

パラメータ	説明
qname	データを保存するRQの名前です。環境ファイルに登録された名前である必要があります
svc	データをRQに保存し、svc名がNULLでない場合は即時サービスを要求します。  svc名がNULLの場合、データはRQに保存され、サービスは実行されません。この場合、後に <b>tpdeq_ctl()</b> を使用してサービスを要求する必要があります。svcで命令されたサービスがない場合、あるいはサービスを実行して処理結果を受信していない状態でシステム障害が発生した場合、このデータは内部的にフェイル・キューに保存されます。このデータも、 <b>tpdeq_ctl()</b> でサービスを再要求するか、エラー処理する必要があります
ctl	TMQCTLのdet_timeを使用して一定時間を設定した場合、その時間以降にsvc callします。  deq_timeの設定時は、現在の時間にdelay_timeを設定する必要があります。

パラメータ	説明
	<p>delay_timeに3を設定したい場合、「cur_time + 3」のように設定します。TMQCTLの機能のうち、現在はdeq_timeのみサポートします。deq_timeはトランザクションと一緒に使用できません。</p> <p>flags項目にTPRQS_NON_PERSISTENTを設定する場合、メッセージをファイルに書き込まずにメモリーにのみ書き込みます</p>
data	NULLの場合を除いて、必ずtpalloc()で割り当てられたバッファに対するポインターである必要があります。dataのタイプとサブタイプは、svcがサポートするタイプである必要があります
len	<p>送信するデータ長です</p> <ul style="list-style-type: none"> <li>dataが指すバッファが特別な長さ明示が不要なタイプ (STRING、STRUCT、X_COMMON、X_C_TYPE) の場合、lenは無視され、0が使用されます</li> <li>dataが指すバッファが、長さを指定する必要があるタイプ (X_OCTET、CARRAY、MULTI STRUCTURE) の場合、lenは0になれません</li> <li>dataがNULLの場合、lenは無視され、データなしでサービス要求が送信されます</li> </ul>
flags	<p>データ処理タイプを設定します。</p> <p>以下は、flagsに設定可能な値についての説明です</p> <ul style="list-style-type: none"> <li>TPRQS <p>svcがNULLでない場合、svcに指定されたサービスを要求し、処理結果をRQに保存します。サービスの処理結果は、tpdeq_ctl()関数を利用して受信します。svcがNULLの場合、データはRQに保存され、サービスは実行しません</p> </li> <li>TPNOREPLY <p>svcがNULLでない場合、svcに指定されたサービスを要求しますが、処理結果はRQに保存しません。svcがNULLの場合、データはRQに保存され、サービスは実行しません</p> </li> <li>TPFUNC <p>サービスごとにRQデータを管理するときに使用されます。TPFUNCフラグが設定されていない場合、最初のtpenq_ctl()により保存されたデータが除去されます。データの保存前に除去が必要な場合、tpenq_ctl()の呼び出し時にTPFUNCと一緒に設定します</p> </li> <li>0(zero) <p>サービス処理結果をRQに保存せず、関数の呼び出し元が接続されているTmaxシステムのクライアント・バッファに保存します。サービスはRQを使用して要求しますが、結果はtpcall()のように関数の呼び出し元が接続したクライアントのバッ</p> </li> </ul>

パラメータ	説明
	ファークから取得時に使用します。処理結果を受信するには、tpdeq_ctl()の呼び出し時にflagsに0(zero)を設定します

- 戻り値

戻り値	説明
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpenq\_ctl()が正常処理されていない場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、dataがtpalloc()で割り当てられていないバッファを指す場合、あるいはflagsが有効でない場合に発生します
[TPENOENT]	存在しないqnameが使用された場合です
[TPEQFULL]	持続的なサービス結果のため指定されたキューのサイズを超えた場合に発生します
[TPGOTSIG]	TPSIGRSTRTが設定されていない状態でシグナルが受信された場合です
[TPEPROTO]	tpenq_ctl()が不適切な状況で呼び出された場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

main(int argc, char *argv[])
{
    char    *sndbuf, *rcvbuf;
    long    rcvlen, sndlen, revent;
    int ret, i, cd;
    TMQCTL  *ctl;
    long t;
```

```

ctl = (TMQCTL*)malloc(sizeof(TMQCTL));
memset(ctl, 0, sizeof(TMQCTL));
time(&t);
ctl->deq_time = (int)t + 3;

if (argc != 2) {
    printf("Usage: toupper string\n");
    exit(1);
}
if ( (ret = tmaxreadenv( "tmax.env", "TMAX" )) == -1 ){
    printf( "tmax read env failed\n" );
    exit(1);
}
if (tpstart((TPSTART_T *)NULL) == -1){
    printf("tpstart failed\n");
    exit(1);
}
if ((sndbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL) {
    printf("sndbuf alloc failed !\n");
    tpend();
    exit(1);
}
if ((rcvbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL) {
    printf("rcvbuf alloc failed !\n");
    tpfree((char *)sndbuf);
    tpend();
    exit(1);
}
ret = tx_begin();
if(ret < 0)
{
    fprintf(stderr, "tx_begin() fail\n");
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    exit(1);
}

strcpy(sndbuf, argv[1]);
cd = tpenq_ctl("txrq1", "TOUPPER", ctl, (char *)sndbuf, 0, TPRS );
if (cd < 0) {
    printf("tpenq failed [%s]\n", tpstrerror(tperrno));
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
    exit(1);
}
cd = tpenq_ctl("txrq1", "TOUPPER", ctl, (char *)sndbuf, 0, TPRS );

```

```

    if (cd < 0) {
        printf("tpenq failed [%s]\n", tpstrerror(tperrno));
        tpfree((char *)sndbuf);
        tpfree((char *)rcvbuf);
        tpend();
        exit(1);
    }
    data process....
    ret = tx_commit();
    if(ret < 0)
    {
        fprintf(stderr, "tx_commit() fail\n");
        tpfree((char *)sndbuf);
        tpfree((char *)rcvbuf);
        tx_rollback();
        exit(1);
    }

    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);

    tpend();
}

```

- 関連関数

tpdeq\_ctl(), tpqstat()

## 3.1.57. tperrordetail

サーバーとクライアントでTmaxシステムの呼び出し時に発生したエラーの詳細情報を取得する際に使用する関数です。エラーの深刻度を測定時に使用されます。こういった場合、クライアントが適切な措置をとることで、エラーに対して迅速な対処ができます。システム段階のエラーが発生した場合は、管理者にエラーの修正を要求し、アプリケーション段階で発生したエラーの場合は、開発者に問題解決を要求できます。

- プロトタイプ

```

#include <atmi.h>
int tperrordetail(int errno)

```

- パラメータ

パラメータ	説明
errno	現在は使用されていません。エラーコードの情報はtperrnoで判断します



- 戻り値

戻り値	説明
1	アプリケーション段階のエラーが発生した場合です
2	システム段階のエラーが発生した場合です
-1	認識できないエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
void main(int argc, char *argv[])
{
    int ret;
    char *buf;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    data process...

    ret = tpcall("SERVICE", sndbuf, 20, &rcvbuf, &rcvlen, TPNOCHANGE);
    if (ret == -1){
        fprintf(stderr, "tpcall fail,,,[%d][%s]\n",
                tperrordetail(tperrno), tpstrerror(tperrno));
        error processing
    }

    tpfree(buf);
    tpend();
}
```

## 3.1.58. tpextsvcinfn

サーバーとクライアントで**tpdeq()**を使用してRQからデータを読み込んだ場合、該当データについての詳細情報を提供する関数です。

- プロトタイプ

```
# include <tmaxapi.h>
int tpextsvcinfn (char *data, char *svc, , int *type, int *errcode )
```

- パラメータ

パラメータ	説明
data	tpalloc()で割り当てられ、tpdeq()を使用してRQから読み込んだデータが保存されているポインターです
svc	該当データのサービス名を取得するためのポインターです
type	<p>該当データの処理結果を表します。</p> <p>以下は、typeに設定可能な値についての説明です</p> <ul style="list-style-type: none"> <li>– TPREQ(0) <p>tpenq()の実行時、2番目のパラメータとしてNULLを指定した場合、tpenqが正常になった際、typeにTPREQが設定されます</p> </li> <li>– TPFAIL(1) <p>tpenq()の実行時、2番目のパラメータとしてサービス名を指定した場合、サービスでtpreturnの最初のパラメータとしてTPFAILが呼び出された際、typeにTPFAILが設定されます</p> </li> <li>– TPSUCCESS(2) <p>tpenq()の実行時、2番目のパラメータとしてサービス名を指定した場合、サービスでtpreturnの最初のパラメータとしてTPSUCCESSが呼び出された際、typeにTPSUCCESSが設定されます</p> </li> <li>– TPERR(-1) <p>tpenq()が失敗してフェイル・キューに送信された場合、typeにTPERRが設定されます</p> </li> </ul>
errcode	エラーが発生した場合、該当するエラーコード値が保存されます

- 戻り値

戻り値	説明
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpextsvcname()が正常処理されていない場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、dataまたはsvcがNULLです
[TPEITYPE]	引数が有効でない場合です。たとえば、dataがRQから取得したdataでない場合に発生します

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です。

- 関連関数

tpenq(), tpdeq(), tpextsvcname()

### 3.1.59. tpextsvcname

サーバーとクライアントで**tpdeq()**を使用してRQからデータを読み込んだ場合、該当データのサービス名を確認する際に使用する関数です。tpextsvcname()は、\_Failキューに保存されているデータをtpdeq()で読み込んだ場合に使用します。

- プロトタイプ

```
# include <tmaxapi.h>
int tpextsvcname (char *data, char *svc)
```

- パラメータ

パラメータ	説明
data	tpdeq()を使用してRQから読み込んだデータが保存されているポインターです。tpalloc()で割り当てられます
svc	該当データのサービス名を取得するためのポインターです

- 戻り値

戻り値	説明
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpextsvcname()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、dataにtpalloc()で割り当てられていないバッファが渡された場合に発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます

エラーコード	説明
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>
void main(int argc, char *argv[])
{
    int ret;
    char *buf, *svc_name;
    long len;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf = (char *)tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
    data process....

    ret=tpenq("RQ", "SERVICE", (char *)buf, strlen(buf), TPRQS);
    if (ret==-1) { error processing }
    data process....

    ret=tpdeq("RQ", "SERVICE", (char **)&buf, (long *)&len, TPRQS);
    if (ret==-1) { error processing }

    ret=tpextsvcname(buf, svc_name);
    if (ret==-1) { error processing }
    printf("svc name : %s      ",svc_name);
    data process....

    tpfree(buf);
    tpend();
}
```

- 関連関数

tpenq(), tpdeq()

## 3.1.60. tpfree

サーバーとクライアントで型付きバッファ(typed buffer)に割り当てられたメモリーを解除する関数です。以前に**tpalloc()**や**tprealloc()**で取得されたバッファを解除します。

- プロトタイプ

```
# include <atmi.h>
void tpfree(char *ptr)
```

- パラメータ

パラメータ	説明
ptr	tpalloc()やtprealloc()で取得されたバッファに対するポインターです。  ptrがNULLの場合、何も起こりません。ptrが型付きバッファを指していない場合、もしくはtpfree()ですでに解除されている領域を指している場合、その結果を知ることはいけません。サービス・ルーチン内部でptrがサービス・ルーチンに渡されたバッファの場合、tpfree()はバッファを解除せず、そのまま返します。バッファを除去する過程で、一部のバッファ・タイプは関連データや状態情報を解除する必要があります。tpfree()は、バッファを解除する前にこれに関連する情報も除去します。  tpfree()が実行された場合、ptrはXATMIルーチンにパラメータで渡すことができず、いかなる方式も使用できません

- 戻り値

tpfree()は、関数の呼び出し元にいかなる値も返しません。

- 例

```
#include <usrinc/atmi.h>
#include <stdio.h>
#include "../sdl/demo.s"

void main(int argc, char *argv[])
{
    int ret;
    struct data *buf;
    char *message, *message2;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf=(struct data *)tpalloc("STRUCT", "data",0);
    if (buf==NULL) { error processing }
```

```

message=tpalloc("STRING", NULL, 0);
if (message==NULL) { error processing }

message2=tpalloc("CARRAY", NULL, 20);
if (message==NULL) { error processing }

data process....
tpfree((char *)buf);
tpfree(message);
tpfree((char *)message2);
tpend();
}

```

- 関連関数

tpalloc(), tprealloc()

---

#### 注

tpfree()は、Cライブラリーのmalloc()、realloc()、またはfree()と一緒に使用できません。tpalloc()で割り当てられたバッファはfree()で解除できません。

---

## 3.1.61. tpget\_timeout

サーバーとクライアントでブロック・タイムアウト時間を返す関数です。サーバーに設定されているサービス制限時間、現在設定されているブロック・タイムアウト時間を確認する際に使用されます。

**tpset\_timeout()**でサービス制限時間を設定した場合、**tpget\_timeout()**で確認できます。**tpset\_timeout()**関数でブロック・タイムアウトを設定していない場合、Tmax環境ファイルに設定された**BLOCKTIME**に設定された値を確認できます。

- プロトタイプ

```

#include <tmaxapi.h>
int  tpget_timeout(void)

```

- 戻り値

tpget\_timeout()は、現在設定されているブロック・タイムアウトを秒単位で返し、エラーは返しません。

- 例

```

#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

```

```

void main(int argc, char *argv[])
{
    int ret;
    char *sndbuf, *rcvbuf;
    long sndlen, rcvlen;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    sndbuf = (char *)tpalloc("CARRAY", NULL, 20);
    if (sndbuf==NULL) {error processing };
    rcvbuf = (char *)tpalloc("CARRAY", NULL, 20);
    if (rcvbuf==NULL) {error processing };
    data process....
    sndbuf=strlen(sndbuf);
    ret=tpset_timeout(4);
    if (ret==-1) { error processing }

    ret=tpget_timeout();
    printf("block time = %d\n", sec);

    ret=tpcall("SERVICE", sndbuf, sndlen, &rcvbuf, &rcvlen, TPNOCHANGE);
    if (ret==-1) { error processing }
    data process....
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
}

```

### 3.1.62. tpgetactivesvr

サーバーとクライアントで現在アクティブなサーバーの一覧を照会する関数です。パラメータで指定したノードで、現在アクティブなサーバー・リストを取得できます。

- プロトタイプ

```

#include <tmaxapi.h>
int tpgetactivesvr(char *nodename, char **outbufp);

```

- パラメータ

パラメータ	説明
nodename	アクティブなサーバーをリストで確認する場合の該当ノードの名前です。

パラメータ	説明
	<p>クライアントでnodenameをNULLで呼び出した場合、現在接続しているノードのサーバー・リストを渡します。サーバーでnodenameをNULLで呼び出した場合、サーバーが起動しているノードのサーバー・リストを渡します。</p> <p>サーバー・リストは、ユーザーがパラメータで入力したポインターのoutbufpに保存されます</p>
outbufp	<p>受信される応答バッファに対するポインターです。サーバー・リストが保存されます。</p> <p>サーバー・リストは、tpgetactivesvr()内でtpalloc()で作成したCARRAYタイプのバッファです。使用後は、必ずtpfree()で除去します。</p> <p>サーバー・リストは、NULLで区分される文字列の集まりです。たとえば、該当ノードにsvr10とsvr111という2つのサーバーがある場合、サーバー・リストの内容は、{'s', 'v', 'r', '1', '0', NULL, 's', 'v', 'r', '1', '1', '1', NULL }です</p>

- 戻り値

戻り値	説明
サーバー数	関数の呼び出しに成功した場合です。アクティブなサーバーの数を返します
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpgetclid()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	設定された引数が正しくない場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。クライアントでは、ほとんどがネットワーク・エラーです
[TPEOS]	運用システムにエラーが発生した場合です。ほとんどがメモリー不足によるエラーです

- 例

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <usrinc/atmi.h>
```



```

main(int argc, char *argv[])
{
    char    *sndbuf, *rcvbuf, *data;
    long    rcvlen, sndlen;
    int      ret, n, i;
    char nodename[35];

    if (argc != 2) {
        printf("Usage: toupper string\n");
        exit(1);
    }
    if ( (ret = tmaxreadenv( "tmax.env", "TMAX" )) == -1 ){
        printf( "tmax read env failed\n" );
        exit(1);
    }

    if (tpstart((TPSTART_T *)NULL) == -1){
        printf("tpstart failed\n");
        exit(1);
    }

    if ((rcvbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL) {
        printf("rcvbuf alloc failed !\n");
        tpfree((char *)sndbuf);
        tpend();
        exit(1);
    }

    strcpy(nodename, "starbj");
    if((n = tpgetactivesvr(nodename, &rcvbuf)) < 0) {
        printf("getactivesvr failed\n");
        tpfree((char *)rcvbuf);
        tpend();
        exit(1);
    }
    printf("Total %d servers\n", n);
    data = rcvbuf;
    for (i = 0; i < n; i++) {
        printf("ACTIVE[%s]\n", data);
        data += strlen(data) + 1;
    }
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
}

```

## 3.1.63. tpgetcliaddr

Tmaxシステムに接続しているクライアントのうち、clidに該当するクライアントのIPアドレスとポート番号を取得する関数です。

### 参考

この関数は、IPv6プロトコル環境では使用できません。IPv6環境では、「[3.1.64. tpgetcliaddr\\_ipv6](#)」関数を使用します。

### ● プロトタイプ

```
#include <tmaxapi.h>
int tpgetcliaddr(int clid, int *ip, int *port, long flags);
```

### ● パラメータ

パラメータ	説明
clid	照会するclidです
ip	該当クライアントの情報が載ります。IPアドレスはsockaddr_in構造体にあるs_addrフィールドのため、dot形式に変えるためにはinet_ntoa()を使用する必要があります
port	該当クライアントの情報が渡されます
flags	現行バージョンではサポートしていませんが、TPNOFLAGSに設定します

### ● 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

### ● エラー

tpgetcliaddr()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。NULL形式が渡された場合に発生します
[TPEITYPE]	IPv6プロトコル環境で呼び出す場合に発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます

### ● 例

```

#include <stdio.h>
#include <usrinc/atmi.h>

TOUPPER(TPSVCINFO *msg)
{
    int          ret;
    int          clid;
    int          port;
    int*         ip;
    ...
    clid = tpgetclid();
    printf("clid = %d\n", clid);

    ret = tpgetcliaddr(clid, ip, &port, 0);
    if(ret < 0)
        error routine..

    printf("ip = %s, port = %d\n", inet_ntoa(ip), port);
    tpreturn(TPSUCCESS,0,(char *)msg->data, 0,0);
}

```

- 関連関数

tpgetclid()

### 3.1.64. tpgetcliaddr\_ipv6

Tmaxシステムに接続されたクライアントのうちclidに該当するクライアントのIPアドレスとポート番号を取得する関数であり、IPv6環境で使します。

- プロトタイプ

```

#include <tmaxapi.h>
#include <arpa/inet.h>
int tpgetcliaddr_ipv6(int clid, struct sockaddr_storage *saddr, long flags);

```

- パラメータ

パラメータ	説明
clid	照会するclidです
ipaddr	当該クライアントの情報が格納されます。関数の呼出しに成功すれば、IPアドレスとポート番号が設定されます。sockaddr_storage構造体のss_familyメンバーを使用してIPプロトコルのバージョンを確認することができます。IPアドレスを文字列に変更

パラメータ	説明
	するには、inet_ntop()を利用します。正しいポート番号を取得するには、ntohs()を利用します
flags	現行バージョンではサポートしていませんが、TPNOFLAGSに設定します

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpgetcliaddr\_ipv6()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合であり、NULL形式が渡された場合に発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <arpa/inet.h>

TOUPPER(TPSVCINFO *msg)
{
    int          ret;
    int          cliid;
    int          portno;
    struct sockaddr_storage saddr;
    struct sockaddr_in *cli_sin4;
    struct sockaddr_in6 *cli_sin6;
    char ipaddrbuf[INET6_ADDRSTRLEN];
    const char *ipaddr = NULL;
    ...
    cliid = tpgetclid();
    printf("clid = %d\n", cliid);

    ret = tpgetcliaddr_ipv6(cliid, &ipaddr, 0);
    if(ret < 0)
        error routine..
}
```

```

        if (saddr.ss_family == AF_INET) {
            cli_sin4 = (struct sockaddr_in *)&saddr;
            ipaddr = inet_ntop(AF_INET, &(cli_sin4->sin_addr), ipaddrbuf,
sizeof(ipaddrbuf));
            portno = ntohs(cli_sin4->sin_port);
        } else if (saddr.ss_family == AF_INET6) {
            cli_sin6 = (struct sockaddr_in *)&saddr;
            ipaddr = inet_ntop(AF_INET6, &(cli_sin6->sin6_addr), ipaddrbuf,
sizeof(ipaddrbuf));
            portno = ntohs(cli_sin6->sin6_port);
        }
        if (ipaddr == NULL)
            ipaddr = "unknown";

        printf("ip = %s, port = %d\n", ipaddr, portno);
        tpreturn(TPSUCCESS,0,(char *)msg->data, 0,0);
    }

```

- 関連関数

tpgetclid()

### 3.1.65. tpgetctxt

関数を呼び出すスレッドに現在設定されているコンテキストのIDを最初のパラメータとして返す関数です。

マルチスレッド、マルチコンテキスト・サーバーでは、スレッドに設定されたコンテキストが有効である場合は、1以上の値を取得し、コンテキストが設定されていないか有効でない場合には、TPNULLCONTEXT(-2)を取得します。コンテキストは、サービス・スレッドでサービス要求を実行している場合にのみ有効です。サービス要求がすべて処理され、tpreturn()が呼び出された場合は、該当コンテキストは無効になり、ユーザー作成スレッドではコンテキストを使用できなくなります。

tpgetctxt関数は、クライアントとサーバー・プログラムで作成方法が異なりますので、以下でサーバーとクライアントの例を別々に説明します。

---

#### 参考

マルチスレッド、マルチコンテキスト・サーバーでは、シングル・コンテキストをサポートしません。

---

- プロトタイプ

```

#include <usrinc/atmi.h>
int tpgetctxt(int *ctxtid, long flags)

```

- パラメータ

パラメータ	説明
ctxtid	関数を呼び出した時点のコンテキストを取得します – multicontextの場合: 1より大きい値を取得します – singlecontextの場合: 0を取得します
flags	現行バージョンではサポートしていませんが、TPNOFLAGSに設定します

- 戻り値

戻り値	説明
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpgetctxt()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	設定された引数が正しくない場合です。たとえば、最初のパラメータがポインタ値であったり、2番目のパラメータに0以外の値が設定された場合に発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログ・ファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例 - クライアント・プログラム

```
int newContext()
{
    int i;
    int id;
    i = tpstart(tpinfo);
    if (i < 0)
    {
        printf("\t[newContext]tpstart fail[%d][%s]\n", tperrno, tpstrerror(tperrno));

        tpfree((char *)tpinfo);
        return -1;
    }
    i = tpgetctxt(&id, TPNOFLAGS);
    if (i < 0)
    {
        printf("\t[newContext]tpgetctxt
```

```

fail[%d][%s]\n", tperrno, tpstrerror(tperrno));
    return -1;
}
return id;
}

```

## ● 例 - サーバー・プログラム

```

typedef param {
    int ctxtid;
    TPSVCINFO *svcinfo;
} param_t;

MSERVICE(TPSVCINFO *svcinfo)
{
    pthread_t tid;
    param_t param;

    printf("MSERVICE service is started!");
    tpgetctxt(&param.ctxtid, TPNOFLAGS);
    param.svcinfo = svcinfo;

    pthread_create(&tid, NULL, THREAD_ROUTINE, &param);
    pthread_join(tid, NULL);

    printf("MSERVICE service is finished!");
    tpreturn(TPSUCCESS, 0, svcinfo->data, 0L, TPNOFLAGS);
}

void *THREAD_ROUTINE(void *arg)
{
    param_t *param;
    TPSVCINFO *svcinfo;

    param = (param_t *)arg;
    svcinfo = param->svcinfo;

    if (tpsetctxt(param->ctxtid, TPNOFLAGS) == -1) {
        printf("tpsetctxt(%d) failed, [tperrno:%d]", param->ctxtid, tperrno);
        return NULL;
    }

    tpcall("MTOUPPER", sndbuf, 0, &rcvbuf, &rcvlen, TPNOFLAGS);

    if (tpsetctxt(TPNULLCONTEXT, TPNOFLAGS) == -1) {
        printf("tpsetctxt(TPNULLCONTEXT) failed, [tperrno:%d]", tperrno);
        return NULL;
    }
}

```

```

    }

    return NULL;
}

```

- 関連関数

tpsetctxt()

### 3.1.66. tpgetenv

サーバーとクライアントでnameという名前で登録されている環境変数の値を返す関数です。一般的に、サーバーで利用できる**getenv()**と同一機能を実行します。Windowsクライアントでは、**autoexec.bat**ファイルに適用された値、**tmaxreadenv()**によって読み込まれたファイルに適用された値、またはWindows NT、Windows 2000では環境変数に設定されている値を返します。サーバーでは、**シェル環境ファイル**または**envfile**に設定された値を返します。

- プロトタイプ

```

#include <tmaxapi.h>
char *tpgetenv(char *name)

```

- パラメータ

パラメータ	説明
name	登録された環境変数名を設定します

- 戻り値

戻り値	説明
NULL	環境変数が存在しない場合です
ポインタ	環境変数が存在しない場合です。設定された環境変数の値に対するポインタを返します

- 例

```

#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

main(int argc, char *argv[])
{
    char *TMAXD, *SDLFI;

```



```

TMAXD=tpgetenv("TMAXDIR");
SDLFI=tpgetenv("SDLFILE");
printf("tmaxdir : %s\nsdlfile : %s\n",TMAXD,SDLFI);
}

```

- 関連関数

tpputenv()

## 3.1.67. tpgetlev

サーバーとクライアントで使用する関数です。トランザクション・モードであるかどうかを確認します。

- プロトタイプ

```

#include <tuxatmi.h>
int tpgetlev(void)

```

- 戻り値

戻り値	説明
1	トランザクション・モードです
0	トランザクション・モードではありません

- 例

```

#include <stdio.h>
#include <usrinc/atmi.h>
#include "../sdl/demo.s"

EXEC SQL include sqlca.h;
EXEC SQL begin declare section;
struct emp *buf;
EXEC SQL end declare section;

DELETE(TPSVCINFO *msg)
{
    struct emp *buf;
    buf = (struct emp *)msg->data;
    data process...
    if (tpgetlev()) printf("transaction mode\n");
    else printf(« nontransaction mode\n »);
    EXEC SQL DELETE FROM emp
    WHERE empno = :buf->empno;
}

```

```
data process...
    tpreturn(TPSUCCESS,0,buf, strlen(buf), 0);
}
```

### 3.1.68. tpgetpeername

サーバーとクライアントで接続されている相手方のソケット・アドレスを取得する関数です。Tmaxシステムに接続が完了後、相手方(ノード)のソケット・アドレスを返します。

● プロトタイプ

```
#include <tmaxapi.h>
int tpgetpeername(struct sockaddr *name, int *namelen)
```

● パラメータ

パラメータ	説明
name	アドレスが保存された構造体です。IPv6プロトコル環境では、sockaddr_in6構造体を使用してアドレス情報を確認します。sockaddr_storage構造体を使用すると、IPv4とIPv6の両方の環境で使用できます
namelen	関数を呼び出す前に、nameで渡す構造体のサイズに初期化する必要があります。リターンに成功した場合には、実際にnameに割り当てられた構造体のサイズが保存されます

● 戻り値

戻り値	説明
ソケットアドレス	関数の呼び出しに成功した場合です。相手側のソケット・アドレスが返されます
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

● エラー

tpgetpeername()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPINVAL]	引数が有効でない場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログ・ファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

エラーコード	説明
[TPEITYPE]	namelen引数値のname構造体のサイズが、実際に保存される構造体のサイズより小さい場合です

## ● 例

```
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>
...
DELETE(TPSVCINFO *msg)
{
    struct sockaddr_in cli;
    char ipAddr[16];
    int cli_len, ret;

    data process...
    memset((char *)&cli, 0, sizeof(cli));
    ret = tpgetpeername((struct sockaddr *)&cli, &cli_len);
    if (ret == -1){ error processing }
    else{
        memcpy(ipAddr, inet_ntoa(cli.sin_addr), 16);
    }
    printf("ip = %s , port = %d\n", ipAddr, cli.sin_port);
    data process...

    tpreturn(TPSUCCESS, 0, 0, 0, 0);
}
```

## ● 例 - IPv6プロトコル環境

```
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>
...
DELETE(TPSVCINFO *msg)
{
    struct sockaddr_storage cli_saddr;
    struct sockaddr_in *cli_sin4;
    struct sockaddr_in6 *cli_sin6;
    char ipaddrbuf[INET6_ADDRSTRLEN];
    const char *ipaddr = NULL;
```

```

int cli_len, ret;
int portno;

data process...
memset((char *)&cli_saddr, 0, sizeof(cli_saddr));
cli_len = sizeof(cli_saddr);
ret = tpgetpeername((struct sockaddr *)&cli_saddr, &cli_len);
if (ret == -1) {
    error processing
}
else {
    if (cli_saddr.ss_family == AF_INET) {
        cli_sin4 = (struct sockaddr_in *)&cli_saddr;
        ipaddr = inet_ntop(AF_INET, &(cli_sin4->sin_addr), ipaddrbuf,
                           sizeof(ipaddrbuf));
        portno = ntohs(cli_sin4->sin_port);
    } else if (cli_saddr.ss_family == AF_INET6) {
        cli_sin6 = (struct sockaddr_in *)&cli_saddr;
        ipaddr = inet_ntop(AF_INET6, &(cli_sin6->sin6_addr), ipaddrbuf,
                           sizeof(ipaddrbuf));
        portno = ntohs(cli_sin6->sin6_port);
    }
    if (ipaddr == NULL)
        ipaddr = "unknown";
}
printf("ip = %s , port = %d\n", ipaddr, portno);
data process...

tpreturn(TPSUCCESS, 0, 0, 0, 0);
}

```

- 関連関数

tpgetpeer\_ipaddr(), tpgetsockname(), tpstart()

### 3.1.69. tpgetrcahseqno

Tmax APIのtpgetsvrseqno APIと同様に、RCAHプロセス番号を返す関数です。この番号はRCALが順次に与えた番号です。

- プロトタイプ

```

#include <ucs.h>
int tpgetrcahseqno()

```

- 戻り値

戻り値	説明
0	RCAHプロセスの順序番号です

### 3.1.70. tpgetrcainfo

RCAHのスレッド情報を表示する関数です。

- プロトタイプ

```
#include <ucs.h>
void * tpgetrcainfo()
```

- 戻り値

RCAヘッダー・ファイルのRCAINFOに対するポインターを返します。

### 3.1.71. tpgetrply

サーバーとクライアントで非同期的に要求したサービスに対する応答を受信する関数です。**tpacall()**で要求したサービスに対する応答を受信します。

- プロトタイプ

```
# include <atmi.h>
int tpgetrply(int *cd, char **data, long *len, long flags)
```

- パラメータ

パラメータ	説明
cd	tpacall()によって返された呼び出し記述子を指します。一般的に、cdと一致する応答が受信されるか、あるいはタイムアウトが発生するまで待機します。一般的に、cdは応答の受信後、無効になります
*data	以前にtpalloc()によって割り当てられたバッファに対するポインターである必要があります
len	tpgetrply()が正常に受信したデータ長です。必要な場合、応答内容が指定されたバッファに受信されるように、バッファのサイズを増加させます。  *dataは、受信データが大きくて変更されることもあり、それ以外の理由によって変更されることもあります。lenが呼び出し前のバッファの総サイズより大きい場合、len

パラメータ	説明
	<p>が該当バッファの新規サイズになります。lenが0と返された場合、いかなるデータも受信されず、*dataとlenが指示するバッファすべて何も変化はありません。</p> <p>*dataやlenがNULLになるのはエラーです</p>
flags	<p>flagsで使用可能な値は以下のとおりです</p> <ul style="list-style-type: none"> <li>– TPGETANY <p>入力値として設定したcdを無視し、これとは関係なく、受信可能な応答を返すようにします。cdは、返された応答に対する呼び出し記述子になります。応答が何もない場合、一般的にtpgetrply()は応答が到着するまで待機します。TPGETANYが設定されていない場合、特に言及がない限り、*cdは無効化される点に留意します。TPGETANYが設定されている場合、cdはエラーが発生した応答に対する記述子になります。応答が返される前にエラーが発生した場合、cdは0になります。特に言及がない場合、呼び出し元のトランザクションに影響を及ぼしません</p> </li> <li>– TPNOCHANGE <p>*dataが指すバッファのタイプは変更できません。受信された応答バッファと*dataが指すバッファのタイプが異なる場合、*dataのバッファ・タイプの受信者が認識できる限度内で受信された応答バッファのタイプに変更されます。TPNOCHANGEフラグが設定されている場合、変更はできません。受信された応答バッファのタイプおよびサブタイプは、*dataが指すバッファのタイプおよびサブタイプと一致する必要があります</p> </li> <li>– TPNOBLOCK <p>応答が到着するまで待機しません。受信可能な応答がある場合は返します。TPNOBLOCKフラグが指定されていない状態で受信可能な応答がない場合、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・タイムアウトやブロック・タイムアウト)が発生するまで関数の呼び出し元は待機します</p> </li> <li>– TPNOTIME <p>関数の呼び出し元がブロック・タイムアウトを無視し、応答が受信されるまで無限に待機します。トランザクション・モードでtpgetrply()を実行した場合、トランザクション・タイムアウトが適用されます</p> </li> <li>– TPSIGRSTRT <p>シグナル割り込みを受け入れる場合に使用します。システム関数の呼び出しが妨害されたとき、システム関数の呼び出しが再実行されます。TPSIGRSTRTフラグが設定されていない状態でシグナル割り込みが発生した場合、関数は失敗し、tperrnoにTPGOTSIGが設定されます</p> </li> </ul>

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です。tpreturn()で渡されるtpurcodeグローバル変数は、tpgetrply()が正常に返された場合、あるいはtperrnoが[TPESVCFAIL]の場合、アプリケーションで定義した値をもちます
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpgetrply()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、cdやdata、*data、lenなどがNULLである場合や、flagsが有効でない場合に発生します。cdがNULLでない場合は、エラー発生後もcdは有効であり、それに対する応答を待ち続けます
[TPEBADDESC]	cdが有効でない記述子である場合です
[TPEOTYPE]	受信された応答のタイプまたはサブタイプは、呼び出し元を認識できないタイプです。  flagsはTPNOCHANGEに設定されていますが、*dataのタイプおよびサブタイプが、サービスが送信した応答のタイプと一致しません。*dataの内容と*lenはすべて変更されません。応答が呼び出し元のトランザクション・モードで受信された場合、そのトランザクションは応答が無視されるため、ロールバックされます
[TPETIME]	タイムアウトが発生した場合です。関数の呼び出し元がトランザクション・モードの場合、トランザクション・タイムアウトが発生し、そのトランザクションはロールバックされます。関数の呼び出し元がトランザクション・モードではなく、TPNOTIMEとTPNOBLOCKのどちらも指定されていない場合は、ブロック・タイムアウトが発生します。この2つの場合、*dataの内容と*lenは変更されません。トランザクション・タイムアウトが発生した場合、新規サービスの要求を送信することや、応答を待機することは、トランザクションがロールバックされるまで[TPETIME]エラーとなります
[TPESVCFAIL]	サービス要求に対する応答を送信するサービス・ルーチンが、アプリケーション・エラーが発生したため、TPFAILでtpreturn()を呼び出した場合です。サービス応答が受信されたら、その内容は*dataが指すバッファにより使用できます。関数の呼び出し元がトランザクション・モードである場合、そのトランザクションはロールバックされます。  トランザクション・タイムアウトが発生するまでは、トランザクションがロールバックされる前に他の通信が実行されることがあります。そういった通信は正常に処理されることもあり、または失敗することもあります。通信が正常に実行されるには、TPNOTRANが設定されている必要があります。呼び出し元のトランザクション・モードで実行された作業は、トランザクションの完了時にすべてロールバックされます

エラーコード	説明
[TPEBLOCK]	TPNOBLOCKが設定された状態でブロッキング状況が発生した場合です。記述子(cd)は有効です
[TPGOTSIG]	TPSIGRSTRTが設定されていない状態でシグナルが受信された場合です
[TPEPROTO]	tpgetrply()が不適切な状況で呼び出された場合です
[TPETRAN]	トランザクション・サービスの呼び出し時に、データベースに問題が発生してxa_startが失敗した場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
void main(int argc, char *argv[])
{
    int ret;
    long len;
    char *buf;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf = (char *)tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
    data process ...

    cd = tpacall("SERVICE", buf, 0, TPNOFLAGS);
    if (cd==-1) { error procesing }
    data process....

    ret=tpgetrply(&cd, &buf, &len, TPNOTIME);
    if (ret==-1) { error processing }
    data process....

    tpfree(buf);
    tpend();
}
```

- 関連関数

tpacall(), tpalloc(), tpreturn()



## 3.1.72. tpgetsockname

サーバーとクライアントでTmaxシステム内部で使用するソケット・アドレスを取得する関数です。

- プロトタイプ

```
#include <tmaxapi.h>

int tpgetsockname (struct sockaddr *name, int *namelen)
```

- パラメータ

パラメータ	説明
name	ソケット・アドレスを取得するメモリーのアドレスです。IPv6プロトコル環境では、sockaddr_in6構造体を使用してアドレス情報を確認します。sockaddr_storage構造体を使用すると、IPv4とIPv6の両方の環境で使用できます
namelen	関数を呼び出す前に、nameで渡す構造体のサイズに初期化する必要があります。リターンに成功した場合には、実際にnameに割り当てられた構造体のサイズが保存されます

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpgetsockname()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です
[TPEITYPE]	namelen引数値のname構造体のサイズが、実際に保存される構造体のサイズより小さい場合です

- 例

```
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

```

#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>
...
DELETE(TPSVCINFO *msg)
{
    ...
    struct sockaddr_in cli;
    char ipAddr[16];
    int cli_len, ret, i;
    ...
    memset((char *)&cli, 0, sizeof(cli));

    ret = tpgetsockname((struct sockaddr *)&cli_saddr, &cli_len);
    if (ret == -1){
        error processing
    }
    else{
        memcpy(ipAddr, inet_ntoa(cli.sin_addr), 16);
    }
    printf("ip = %s , port = %d\n", ipAddr, cli.sin_port);
    ...
}

```

- 例 - IPv6プロトコル環境

```

#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>
...
DELETE(TPSVCINFO *msg)
{
    struct sockaddr_storage cli_saddr;
    struct sockaddr_in *cli_sin4;
    struct sockaddr_in6 *cli_sin6;
    char ipaddrbuf[INET6_ADDRSTRLEN];
    const char *ipaddr = NULL;
    int cli_len, ret;
    int portno;

    data process...
    memset((char *)&cli_saddr, 0, sizeof(cli_saddr));
    cli_len = sizeof(cli_saddr);

```

```

ret = tpgetsockname((struct sockaddr *)&cli_saddr, &cli_len);
if (ret == -1) {
    error processing
}
else {
    if (cli_saddr.ss_family == AF_INET) {
        cli_sin4 = (struct sockaddr_in *)&cli_saddr;
        ipaddr = inet_ntop(AF_INET, &(cli_sin4->sin_addr), ipaddrbuf,
sizeof(ipaddrbuf));
        portno = ntohs(cli_sin4->sin_port);
    } else if (cli_saddr.ss_family == AF_INET6) {
        cli_sin6 = (struct sockaddr_in *)&cli_saddr;
        ipaddr = inet_ntop(AF_INET6, &(cli_sin6->sin6_addr),
ipaddrbuf, sizeof(ipaddrbuf));
        portno = ntohs(cli_sin6->sin6_port);
    }
    if (ipaddr == NULL)
        ipaddr = "unknown";
}
printf("ip = %s , port = %d\n", ipaddr, portno);
...
}

```

- 関連関数

tpgetpeer\_ipaddr(), tpgetpeername(), tpstart()

### 3.1.73. tpgetsprlist

サーバー・プロセス単位で呼び出しを行うために、該当サービスが属しているサーバー・プロセスの索引を取得する関数です。該当サービスが属するサーバー・リストのstarti、endiを提供します。

- プロトタイプ

```

#include <usrinc/tmaxapi.h>
int tpgetsprlist(char *svc, int svgn, int *starti, int *endi, long flags);

```

- パラメータ

パラメータ	説明
svc	呼び出すサービスを設定します
svgno	該当サービスが属しているサーバーグループの番号です。 <b>tpgetsvglist()</b> を呼び出すか、 <b>tadmin</b> の <b>cfg -g</b> を使用して確認できます
starti	該当サービスが属するサーバーの最初のプロセスの索引番号です
endi	最後のプロセスの索引番号です。たとえば、startiが36、endiが40の場合、プロセスのシリアル番号は36から40までになります。サーバーのMIN値が5、MAX値が10の場合、 <b>tpgetsprlist()</b> を実行すると、MAX値である10個の索引を取り出します
flagsl	現行バージョンではサポートしていませんが、TPNOFLAGSか0に設定します

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です。starti、endiを取得できます
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpgetsprlist()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です
[TPEOS]	運用システムにエラーが発生した場合です
[TPEBLOCK]	ブロッキング状況が発生した場合です

- 例

```
main(int argc, char *argv[])
{
    int      ret;
    int      starti = 0, endi = 0;

    if (argc != 2) {
        printf("Usage: argv[1] string\n");
        exit(1);
    }
    if ((ret = tmaxreadenv( "tmax.env", "TMAX" )) == -1 ){
        printf("<%-15s> tmaxreadenv fail [%s]", __FILE__, tpstrerror(tperrno));

        exit(1);
    }
}
```

```

    if (tpstart((TPSTART_T *)NULL) == -1){
        printf("<%-15s> tpstart fail [%s]", __FILE__, tpstrerror(tperrno));
        exit(1);
    }

    /* tpgetsprlist in client */
    ret = tpgetsprlist(argv[1], 2, &starti, &endi, 0);
    if (ret < 0) {
        printf("<%-15s> tpgetsprlist fail [%s]", __FILE__, tpstrerror(tperrno));

        exit(1);
    }
    else {
        printf("Received Message : starti[%d], endi[%d]\n", starti, endi);
    }
    tpend();
}

```

### 3.1.74. tpgetsvglist

サーバーとクライアントで該当サービスが属しているサーバー・グループと、該当サーバー・グループのCOUSINIに設定されているサーバー・グループについての情報を提供する関数です。返した構造体には、サーバー・グループ数とサーバー・グループのシリアル番号の配列が保存されています。

- プロトタイプ

```

#include <tmaxapi.h>
struct svglist* tpgetsvglist(char *svc, long flags)

```

- パラメータ

パラメータ	説明
svc	確認するサービス名です
flags	現行バージョンではサポートしていませんが、TPNOFLAGSに設定します

- 戻り値

戻り値	説明
0以上の整数	関数の呼び出しに成功した場合です。サーバープロセスのシリアル番号に該当する0以上の整数値を返します
NULL	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

エラーが発生した場合はNULLを返し、状況に該当する値がtperrnoに設定されます。そうでない場合、サービスの呼び出しに失敗したサーバー・グループ・リストをstruct svglistに設定して返します。

返されたstruct svglistの内容は以下のとおりです。

```
struct svglist {
    int ns_entry;
    int nf_entry;
    int *s_list;
    int *f_list;
};
```

メンバー	説明
ns_entry	該当サービスが属するサーバー・グループの数です
nf_entry	失敗したサーバー・グループの数です
s_list	サーバー・グループのシリアル番号の配列に対するポインターです。nf_entryとf_listは、他の用途で使用され、それぞれ0とNULL値をもちます
f_list	失敗したサーバー・グループのシリアル番号の配列です

- エラー

tpgetsvglist()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です。メモリーの割り当て失敗などがこれに該当します

## 3.1.75. tpgprio

サーバーとクライアントで要求を受けたサービスの優先順位を表示する関数です。クライアントのtpgprio()は最後に送信した要求の優先順位を返し、サーバーのtpgprio()は最後に受信された要求の優先順位を返します。

tpgprio()は、tpcall()が呼び出された後に呼び出され、受信した要求の優先順位を返します。また、要求を受けたサービスの優先順位を確認するために、サービス・ルーチン内で呼び出されることがあります。優先順位は、要求されたメッセージのサービス優先順位を表す値です。値は0から100までで、各サービスのデフォルト値は50です。tpsprio()とtpgprio()を使用して、要求されたメッセージのサービス順位を管理できます。

- プロトタイプ

```
#include <tuxfatmi.h>
int tpgprio (void)
```

- 戻り値

戻り値	説明
優先順位	関数の呼び出しに成功した場合です。サービスの優先順位を返します
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tpgprio()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPENOENT]	tpgprio()が呼び出された後も受信された要求がない場合や、要求のない対話型サービス状態で呼び出された場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

SERVICE(TPSVCINFO *msg)
{
    int ret, prio;
    prio = tpgprio();
    if (prio== -1) { error processing }
    if (prio > 50) tpforward("SERVICE1", msg->data, msg->len, 0);
    else tpforward("SERVICE2", msg->data, msg->len, 0);
}
```

- 関連関数

tpacall(), tpcall(), tpsprio()

## 3.1.76. tpmcall

サーバーとクライアントで、COUSINにまとめられたすべてのサーバー・グループ・サーバーのサービス呼び出す関数です。COUSINにまとめられた複数のサーバー・グループがあるTmax環境では、**tpcall()**を使用した場合、Tmaxのルーティング方式によって1つのサーバー・グループに属するサービス呼び出します。しかし、tpmcall()を使用した場合、COUSINにまとめられたサーバー・グループの該当サービスをすべて呼び出すため、クライアント側ではmulticastingを実行します。

tpmcall()を使用した場合、内部的にCOUSINに属するサーバー・グループのサービスで、tpacall(..., TPNOPLY)を実行したのと同じ方式で動作します。tpmcall()はトランザクションのカテゴリに入りません。

### ● プロトタイプ

```
#include <tmaxapi.h>
struct svglist *tpmcall(char *qname, char *svc, char *data, long len, long flags)
```

### ● パラメータ

パラメータ	説明
qname	qnameを設定した場合、RQを経てサービスを呼び出します。現行バージョンではサポートしていません
svc, data, len	tpcall()で使用される1番目～3番目のパラメータと同一です
flags	現行バージョンではサポートしていませんが、TPNOFLAGSに設定します

### ● 戻り値

エラーが発生した場合はNULLを返し、状況に該当する値がtperrnoに設定されます。そうでない場合、サービスの呼び出しに失敗したサーバー・グループ・リストをstruct svglistに設定して返します。

返したstruct svglistの内容は以下のとおりです。

```
struct svglist {
    int ns_entry;
    int nf_entry;
    int *s_list;
    int *f_list;
};
```

メンバー	説明
ns_entry	tpmcall()に成功したサーバー・グループ数です
nf_entry	tpmcall()に失敗したサーバー・グループ数です
s_list	tpmcall()に成功したサーバー・グループのシリアル番号の配列です。nf_entryとf_listは他の用途で使用され、それぞれ0とNULL値をもちます
f_list	tpmcall()に失敗したサーバー・グループのシリアル番号の配列です



- エラー

tpmcall()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、svcがNULLである場合や、dataがtpalloc()で割り当てられていないバッファを指す場合、フラグが有効でない場合に発生します
[TPENOENT]	設定したqnameがTmaxに登録されていない場合です
[TPEITYPE]	dataのタイプおよびサブタイプが、svcがサポートしていないタイプです
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です。メモリの割り当て失敗などがこれに該当します

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>

main(int argc, char *argv[])
{
    char    *sndbuf;
    int     ret;
    ...

    ret = tpstart((TPSTART_T *)NULL);
    if (ret == -1){ error processing }

    sndbuf = (char *)tpalloc("STRING", NULL, 0);
    if (sndbuf == NULL){ error processing }

    ret = tpmcall(NULL, "TPMCALL", sndbuf, 0, TPNOFLAGS);
    if (ret == NULL){ error processing }
    ...
}
```

- 関連関数

tpcall(), tpgetsvglist(), tpmcallx()

## 3.1.77. tpmcallx

tpmcallx()は、既存のtpmcall()の拡張機能の提供を目的とする関数です。従来とは異なり、COUSINサーバー・グループのサービスから応答をすべて受けるまで待機します。また、応答の成功の可否が保存されるsvglist構造体にr\_listが追加され、flagsに項目が追加されました。

### ● プロトタイプ

```
#include <usrinc/tmaxapi.h>
struct svglistx* tpmcallx(char *svc, char *data, long len, long flags)
```

### ● パラメータ

パラメータ	説明
svc	呼び出すサービス名です。Tmax応答サーバー・プログラムで提供されているものである必要があります
data	サービス要求のデータに対するポインターです。以前にtpalloc()によって割り当てられたバッファーである必要があります。dataのタイプとサブタイプは、svcがサポートするタイプである必要があります
len	送信するデータ長です  – dataが指すバッファーが特に長さの指定が不要なタイプ (STRING、STRUCT、X_COMMON、X_C_TYPE) である場合、lenは無視され、0が使用されます  – dataが指すバッファーが、長さを指定する必要があるタイプ (X_OCTET、CARRAY、MULTI STRUCTURE) の場合、lenは0になれません  – dataがNULLの場合、lenは無視されます
flags	呼び出し時に使用されるオプションです。通信方式を指定します  flagsで使用可能な値は以下のとおりです  – TPNOREPLY  送信のみ実行します。このフラグを設定した場合、既存のtpmcall()と似た動作をします  – TPBLOCK  送信がCLHまで正常に渡されたかどうかを確認します  – TPNOTIME  受信時、ブロック・タイムアウト時間を無視し、無限に待機します

### ● 戻り値

エラーが発生した場合はNULLを返し、状況に該当する値がtperrnoに設定されます。そうでない場合、サービスの呼び出しに失敗したサーバー・グループ・リストをstruct svglistxに設定して返します。

返したstruct svglistxの内容は以下のとおりです。

```
struct svglistx {
    int ns_entry; /* number of entries of s_list */
    int nf_entry; /* number of entries of f_list */
    int nr_entry; /* number of entries of r_list */
    int *s_list; /* list of server group numbers */
    int *f_list; /* list of tperrno of each server group */
    int *r_list; /* list of tpurcode of each server group */
};
```

メンバー	説明
ns_entry	tpmcallx()に成功したサーバー・グループ数です
nf_entry	tpmcallx()に失敗したサーバー・グループ数です
nr_entry	r_listが設定されたサーバー・グループ数です
s_list	tpmcallx()に成功したサーバー・グループのシリアル番号の配列です
f_list	tpmcallx()に失敗したサーバー・グループのシリアル番号の配列です
r_list	tpurcodeが設定されたサーバー・グループのシリアル番号の配列です

## ● エラー

tpmcall()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、svcがNULLである場合や、dataがtpalloc()で割り当てられていないバッファを指す場合、フラグが有効でない場合に発生します
[TPENOENT]	設定したqnameがTmaxに登録されていない場合です
[TPEITYPE]	dataのタイプおよびサブタイプが、svcがサポートしていないタイプです
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です。メモリーの割り当て失敗などがこれに該当します

## ● 例

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
```

```

#include <string.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>
main(int argc, char *argv[])
{
    char *sndbuf, *rcvbuf;
    long rcvlen, sndlen;
    struct svglistx svglist;
    struct svglistx *psvglist;
    int ret, i;

    psvglist = &svglist;
    if (argc != 2) {
        printf("Usage: toupper string\n");
        exit(1);
    }

    if ( (ret = tmaxreadenv( "tmax.env","TMAX" )) == -1 ){
        printf( "tmax read env failed\n" );
        exit(1);
    }

    if (tpstart((TPSTART_T *)NULL) == -1){
        printf("tpstart failed[%s]\n",tpstrerror(tperrno));
        exit(1);
    }

    if ((sndbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL)
    {
        printf("sendbuf alloc failed !\n");
        tpend();
        exit(1);
    }

    if ((rcvbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL)
    {
        printf("recvbuf alloc failed !\n");
        tpfree((char *)sndbuf);
        tpend();
        exit(1);
    }

    strcpy(sndbuf, argv[1]);
    psvglist = tpmcallx("TOUPPER", sndbuf, 0, TPBLOCK);
    if(psvglist == NULL)
    {
        printf("tpmcall is failed[%s]\n",

```

```

        tpstrerror(tperrno));
        tpfree((char *)sndbuf);
        tpfree((char *)rcvbuf);
        tpend();
        exit(1);
    }

    printf("send data: %s\n", sndbuf);
    printf("ns_entry = %d\n", psvglst->ns_entry);
    printf("nf_entry = %d\n", psvglst->nf_entry);
    printf("nr_entry = %d\n", psvglst->nr_entry);

    for(i=0; i<psvglst->ns_entry; i++)
        printf("psvglst->s_list[%d] = %d\n", i, psvglst->s_list[i]);
    for(i=0; i<psvglst->nf_entry; i++)
        printf("psvglst->f_list[%d] = %d\n", i, psvglst->f_list[i]);
    for(i=0; i<psvglst->nr_entry; i++)
        printf("psvglst->r_list[%d] = %d\n", i, psvglst->r_list[i]);

    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
}

```

- 関連関数

tpcall(), tpgetsvglst(), tpmcall()

## 3.1.78. tpnotify

サーバーで指定されているクライアントに非要求メッセージを送信する関数です。

- プロトタイプ

```

#include <atmi.h>
int tpnotify(CLIENTID *id, char *data, long len, long flags);

```

- パラメータ

パラメータ	説明
id	TPSVCINFO構造体に保存されているクライアントIDに対するポインターです
data	以前にtpalloc()によって割り当てられたバッファである必要があります
len	送信するバッファ長です

パラメータ	説明
	<ul style="list-style-type: none"> <li>– dataの長さの指定が不要なバッファを指す場合、lenは無視され、0が使用されます</li> <li>– dataの長さの指定が必要なバッファを指す場合、lenは0になれません</li> <li>– dataがNULLの場合、lenは無視されます</li> </ul>
flags	<p>flagsで使用可能な値は以下のとおりです</p> <ul style="list-style-type: none"> <li>– TPACK 以前にtpalloc()によって割り当てられたバッファである必要があります</li> <li>– TPNOBLOCK データが到着するまで待機しません。受信可能なデータがある場合、これを返します。TPNOBLOCKフラグが指定されず、受信可能なデータがない場合、呼び出し元はデータが到着するまで待機します</li> <li>– TPNOTIME 関数の呼び出し元がブロック・タイムアウトを無視し、応答が受信されるまで無限に待機します。しかし、トランザクション・タイムアウト内でWinTmaxSend()を実行した場合、トランザクション・タイムアウトが適用されます</li> </ul>

#### ● 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

#### ● エラー

tpnotify()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です
[TPENOENT]	対象クライアントが存在しない場合、あるいは非要求メッセージ・ハンドラが設定されていない場合に発生します
[TPETIME]	タイムアウトが発生しました。関数の呼び出し元がトランザクション・モードの場合、トランザクション・タイムアウトが発生し、そのトランザクションはロールバックされます。関数の呼び出し元がトランザクション・モードではなく、TPNOTIMEとTPNOBLOCKのどちらも指定されていない場合は、ブロック・タイムアウトが発生します

エラーコード	説明
[TPEPROTO]	tpnotify()が不適切な状況で呼び出された場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/fbuf.h>
#include "../fdl/demo_fdl.h"

TOUPPER(TPSVCINFO* msg) {
    int i, n;
    char    rdata[100];
    FBUF    *stdata;

    stdata=(FBUF *)msg->data;
    memset(rdata, 0x00, sizeof(rdata));
    if (fbget(stdata, INPUT, rdata, 0) == -1){
        printf("fbget failed errno = %d\n", ferror);
    }

    for (i = 0; i < msg->len; i++) {
        rdata[i] = toupper(rdata[i]);
    }
    if (fbput(stdata, OUTPUT, rdata, 0) == -1)
        printf("fbput failed\n");

    i = tpnotify(&(msg->cltid), (char*)stdata, 0, TPACK);
    printf("i= %d\n", i);
    tpreturn(TPSUCCESS, 0, (char *)stdata, 0, 0);
}
```

## 3.1.79. tppost

サーバーとクライアントで特定イベントを発生させ、メッセージを渡す関数です。tppost()は、名前がeventnamのイベントにtppsubscribe()で登録されたすべてのクライアントとサーバー・プロセスにイベントの発生を通知し、必要な場合はメッセージを渡します。

- プロトタイプ

```
# include <tmaxapi.h>
int tppost(char *eventname, char *data, long len, long flags)
```

## ● パラメータ

パラメータ	説明
eventname	NULLで終わる63文字以内の文字列で発生するイベントの名前を意味します。ワイルドカードやpartial-matchingなどはサポートしていません
data	送信メッセージに対するポインターです。tpalloc()によって割り当てられたバッファである必要があります
len	送信するバッファ長です <ul style="list-style-type: none"> <li>dataの長さの指定が不要なバッファを指す場合、lenは無視され、0が使用されます</li> <li>dataの長さの指定が必要なバッファを指す場合、lenは0になれません</li> <li>dataがNULLの場合、lenは無視されます</li> </ul>
flags	現在はTPNOTIMEのみ意味があります

## ● 戻り値

戻り値	説明
正数	関数の呼び出しに成功した場合です。
負数	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

## ● エラー

tppost()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。クライアントでは、ネットワーク・エラーが最も多いです
[TPEOS]	OSレベルのエラーが発生した場合です。メモリの割り当てなどの問題を含みます
[TPEINVAL]	設定された引数が正しくない場合です
[TPENOENT]	tpsubscribe()の場合、TPEVCTL構造体のqname、あるいはsvclに該当するRQがサーバーが存在しない場合です
[TPEPROTO]	クライアントでtpsubscribe()を実行した場合に、ctlがNULLでない場合や、サーバーで実行する場合に、ctlがNULLである場合です
[TPETIME]	タイムアウトが発生した場合です



- 関連関数

tpsetunsol(), tpsetunsol\_flag(), tpgetunsol(), tpacall(), tpenq()

## 3.1.80. tpputenv

サーバーとクライアントで環境変数値を再設定する関数です。stringで入力された「名前=値」を環境変数に適用します。既存の環境変数が存在する場合は修正し、環境変数が存在しない場合は新規追加します。

サーバーで実行可能な**putenv()**と同じ機能を実行する関数です。クライアントでは、**autoexec.bat**ファイルに適用された値を修正、あるいは新規追加します。サーバーでは、それぞれのシェル環境ファイルに適用された値を修正したり、あるいは存在する値がなければ追加します。

- プロトタイプ

```
#include <tmaxapi.h>
int tpputenv(char *string)
```

- パラメータ

パラメータ	説明
string	環境変数に設定する値です(「名前=値」)

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
負数	関数の呼び出しに失敗した場合です。該当環境変数を適用できる十分なメモリを確保できない場合に発生します

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

void main(int argc, char *argv[])
{
    int ret;
    char *SDLFI;

    SDLFI=tpgetenv("SDLFILE");
    if (SDLFI=NULL) ret=tpputenv("SDLFILE=/tmax/sample/sdl/test.sdl");
    if (ret<0) { error processing }
```

```

SDLFI=tpgetenv("SDLFILE");
printf ("tmax sdlfile : %s\n",SDLFI);
}

```

- 関連関数

tpgetenv()

### 3.1.81. tpqstat

サーバーとクライアントで使用され、RQに保存されているデータの統計を要求する関数です。RQは、内部的に\_fail queue、\_request queue、\_reply queueの3部で構成されています。フラグ値を使用して、各キューに保存されたデータ統計を求めることができます。

- プロトタイプ

```

#include <tmaxapi.h>
int tpqstat (char *qname, long flags)

```

- パラメータ

パラメータ	説明
qname	Tmax環境ファイルに登録されたRQ名を表します
flags	<p>対象データのタイプです。</p> <p>flagsで使用可能な値は以下のとおりです</p> <ul style="list-style-type: none"> <li>– 0(TMAX_ANY_QUEUE) : _fail queue、_request queue、_reply queueのデータ統計を求める際に使用します</li> <li>– 1(TMAX_FAIL_QUEUE) : _fail queueのデータ統計を求める際に使用します</li> <li>– 2(TMAX_REQ_QUEUE) : _request queueのデータ統計を求める際に使用します</li> <li>– 3(TMAX_RPLY_QUEUE) : _reply queueのデータ統計を求める際に使用します</li> </ul>

- 戻り値

戻り値	説明
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpqstat()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、qnameがNULLの場合や、qnameに該当するキューがない場合、flagsが有効でない場合に発生します
[TPEPROTO]	tpqstat()が不適切な状況で呼び出された場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

void main(int argc, char *argv[])
{
    int ret, i;
    char *buf;

    if (argc!=2) { error processing }
    ret=tpstart((TPSTART_T *)NULL);
    if (ret== -1) {error processing }

    buf = (char *)tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
    strcpy(buf, argv[1]);

    data process...

    ret=tpenq("RQ", NULL, (char *)buf, strlen(buf), TPRS);
    if (ret== -1) {error processing }
    printf("qstat :");
    for (i=0;i<4;i++) {
        ret=tpqstat("rq", i);
        if (ret== -1) {error processing }
        printf("  %d",ret);          /* qstat :  1  0  0  1 */
    }
    printf("\n");
    data process...

    ret=tpenq("RQ", "SERVICE", (char *)buf, strlen(buf), TPRS);
```

```
if (ret==-1) {error processing }
printf("qstat :");

for (i=0;i<4;i++) {
    ret=tpqstat("rq", i);
    if (ret==-1) {error processing }
    printf("  %d",ret);          /* qstat :  2  0  0  2 */

}
printf("\n");
tpfree((char *)buf);
tpend();
}
```

● 関連関数

tpenq(), tpdeq()

3.1.82. tpqsvcstat

サーバーとクライアントで使用され、RQに保存されているデータのうち、指定したサービスの統計を要求する関数です。RQに保存されているデータの統計を要求します。RQは、内部的に\_fail queue、\_request queue、\_reply queueの3部で構成されています。フラグ値を使用して、各キューに保存されたデータ統計を求めることができます。

● プロトタイプ

```
# include <tmaxapi.h>
int tpqsvcstat (char *qname, char * svc, long flags )
```

● パラメータ

パラメータ	説明
qname	Tmax環境ファイルに登録されたRQ名を設定します
svc	統計情報を取得するサービス名です。NULLの場合、tpqstat()と同一の意味をもちます
flags	
flags	データ統計のタイプを設定します。  以下は、flagsに設定可能な値についての説明です  – 0(TMAX_ANY_QUEUE) : _fail queue, _request queue, _reply queue のデータ統計を求める際に使用します

パラメータ	説明
	<ul style="list-style-type: none"> <li>– 1(TMAX_FAIL_QUEUE) : _fail queueのデータ統計を求める際に使用します</li> <li>– 2(TMAX_REQ_QUEUE) : _request queueのデータ統計を求める際に使用します</li> <li>– 3(TMAX_RPLY_QUEUE) : _reply queueのデータ統計を求める際に使用します</li> </ul>

- 戻り値

戻り値	説明
-1以外の値	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpqscvstat()が正常処理されていない場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、qnameがNULLの場合や、qnameに該当するキューがない場合、タイプが有効でない場合に発生します
[TPEPROTO]	tpqscvstat()が不適切な状況で呼び出された場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 関連関数

tpenq(), tpdeq(), tpqstat()

### 3.1.83. tprealloc

サーバーとクライアントで型付きバッファの再割り当てを行う関数です。ptrが指すバッファのサイズをバイトで再割り当てし、バッファが変更された場合、新規バッファに対するポインタを返します。

tpalloc()と同じようにバッファのサイズは基本サイズ(1024バイト)以上です。バッファのタイプは、再割り当てされた後も同一に維持されます。関数が正常に返された場合、返されたポインタがバッファを参照するために使用され、ptrは使用できなくなります。再割り当てされたバッファのサイズが縮小された場合、本来のptrの内容は保証できません。

一部バッファのタイプは、使用前に初期化する必要があります。tprealloc()は、バッファの再割り当て後に再初期化して返します。そのため、呼び出し元に返されたバッファは、即時使用可能です。バッファが再初期化に失敗した場合、tprealloc()はNULLを返し、ptrが指示するバッファの内容は有効ではありません。

#### ● プロトタイプ

```
# include <atmi.h>
char * tprealloc (char *ptr, long size)
```

#### ● パラメータ

パラメータ	説明
ptr	割り当てるバッファに対するポインターです
size	割り当てるバッファのサイズです

#### ● 戻り値

戻り値	説明
バッファ・ポインター	関数の呼び出しに成功した場合です。適切な型付きバッファに対するポインターを返します
NULL	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

#### ● エラー

tprealloc()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、ptrが指すバッファが、tpalloc()で割り当てられたものでない場合に発生します
[TPEPROTO]	tprealloc()が不適切な状況で呼び出された場合です
[TPENOENT]	ptrが指すバッファがtpalloc()で割り当てられていないバッファです
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

#### ● 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
void main(int argc, char *argv[])
{
```

```

char *buf;
buf=tpalloc("STRING",NULL,10);
if (buf==NULL) { error processing }
buf=tprealloc(buf,20); /* ok */
if (buf==NULL) { error processing }

buf="test";
buf=tprealloc(buf,30); /*error : TPEINVAL */
if (buf==NULL) { error processing }
}

```

- 関連関数

tpalloc(), tpfree(), tptypes()

---

注

Cライブラリーの**malloc()**、**realloc()**または**free()**と一緒に使用できません。たとえば、tprealloc()で割り当てられたバッファをfree()で解除できません。

---

### 3.1.84. tprecv

サーバーとクライアントで対話型通信を行う場合、メッセージを受信する関数です。対話型通信で接続された相手方のプログラムから送信されたデータを受信するために使用されます。tprecv()は、クライアントあるいはサーバーのうち、通信制御権をもたないプログラムでのみ使用されます。

- プロトタイプ

```

# include <atmi.h>
int tprecv (int cd, char **data, long *len, long flags, long *revent)

```

- パラメータ

パラメータ	説明
cd	データを受信する接続を指定します。 <b>tpconnect()</b> やTPSVCINFO媒介変数のうち1つから変換された記述子です
data	tpalloc()によって、以前割り当てられたバッファに対するポインター・アドレスです。関数が正常に返された場合、*dataは受信されたデータを指します
len	データの長さです。  lenが呼び出し前のバッファの総サイズより大きい場合、バッファの新規サイズはlenになります。

パラメータ	説明
	<p>lenが0の場合、いかなるデータも受信されず、*dataも、あるいは*dataが指すバッファも変化はありません。*dataかlenがNULLの場合、dataはエラーになります</p>
flags	<p>flagsで使用可能な値は以下のとおりです</p> <ul style="list-style-type: none"> <li>– TPNOCHANGE <p>受信された応答バッファと*dataが指すバッファのタイプが異なる場合、*dataのバッファ・タイプは受信者が認識できる限度内で受信された応答バッファのタイプに変更されます。しかし、このフラグが設定された場合、*dataが指すバッファのタイプは変更できません。受信された応答バッファのタイプおよびサブタイプは、*dataが指すバッファのタイプおよびサブタイプと一致する必要があります</p> </li> <li>– TPNOBLOCK <p>データが到着するまで待機しません。受信可能なデータがある場合、これを返します。TPNOBLOCKフラグが指定されず、受信可能なデータがない場合、呼び出し元はデータが到着するまで待機します</p> </li> <li>– TPNOTIME <p>関数の呼び出し元がブロック・タイムアウトを無視し、応答が受信されるまで無限に待機します。しかし、トランザクション・タイムアウト内でtprecv()を実行した場合は、トランザクション・タイムアウトが適用されます</p> </li> <li>– TPSIGRSTRT <p>シグナル割り込みを受け入れる場合に使用します。内部でシグナル割り込みが発生し、システム関数の呼び出しが妨害された際、システム関数の呼び出しが再実行されます。TPSIGRSTRTフラグが設定されていない状態でシグナル割り込みが発生した場合、関数は失敗し、tperrnoにTPGOTSIGが設定されます</p> </li> </ul>
revent	<p>reventに返されるイベント・タイプは以下のとおりです</p> <ul style="list-style-type: none"> <li>– TPEV_DISCONIMM <p>対話従属者が受信するこのイベントは、対話開始者がtpdiscon()で対話型接続を強制で終了したことを意味します。またこのイベントは、サーバー、ノード、ネットワーク障害などのような通信エラーによって接続が切断された場合、対話開始者あるいは従属者に返されることがあります。これは、強制的な接続解除の通知であるため、転送中のデータは失われることもあります。2つのプログラムが同一トランザクションに参加している場合、そのトランザクションはロールバックされます。対話型通信に使用されていた記述子(cd)は、無効になります</p> </li> <li>– TPEV_SENDOONLY</li> </ul>



パラメータ	説明
	<p>接続された相手のプログラム側で通信制御権を放棄しました。TPEV_SENDOONLYイベントの受信者はデータを送信することはできませんが、データの受信者が制御権を譲渡するまでは、いかなるデータも受信できません</p> <p>– TPEV_SVCERR</p> <p>対話開始者が受信するこのイベントは、対話従属者がtpreturn()の実行中にエラーが発生したことを通知します。たとえば、tpreturn()に誤ったパラメータが渡されたり、サービスが他の従属者と接続を維持している間にtpreturn()が呼び出されることがあります。この場合、リターン・コードやデータの一部は使用が不可能です。対話型接続が終了し、cdは以降無効になります。このイベントが受信者のトランザクション・プロセスで発生した場合、そのトランザクションはロールバックされます</p> <p>– TPEV_SVCFAIL</p> <p>対話開始者が受信するこのイベントは、相手側の対話従属者サービスがアプリケーション上の失敗でサービスを終了したことを通知します。TPFAILでtpreturn()を呼び出しました。対話従属者サービスがtpreturn()を呼び出した際に通信制御権をもっていた場合、サービスは接続された相手側にデータを渡すことができます。サービスが終了すると同時にサーバー・プロセスは対話型接続を終了します。そのため、cdは以降無効になります。このイベントが受信者のトランザクション・プロセスで発生した場合、そのトランザクションはロールバックされます</p> <p>– TPEV_SVCSUCC</p> <p>対話開始者が受信するこのイベントは、相手側の対話従属者サービスが正常に終了したことを通知します。TPSUCCESSでtpreturn()を呼び出しました</p>

#### ● 戻り値

revent値がTREV\_SVCSUCCである場合やTREV\_SVCFAILである場合、tpreturn()で渡されるtpurcodeトランザクション変数は、アプリケーションで定義した値をもちます。そうでない場合は-1を返し、tperrnoにエラー状況に該当する値が設定されます。エラーがない状態でイベントが存在する場合、tprecv()は-1を返し、tperrnoは[TPEEVENT]になります。

#### ● エラー

tprecv()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEBADDESC]	記述子(cd)が有効でない場合です

エラーコード	説明
[TPEBLOCK]	TPNOBLOCKが設定された状態でブロッキング状況が発生した場合です
[TPEEVENT]	イベントが発生し、reventでイベント・タイプを確認できます
[TPEINVAL]	引数が有効でない場合です。たとえば、*dataが指すバッファーがtpalloc()で割り当てられていない場合や、flagsが有効でない場合に発生します
[TPEOS]	運用システムにエラーが発生した場合です
[TPEOTYPE]	<p>入力されたバッファーのタイプあるいはサブタイプが呼び出し元と一致しない場合です。あるいは、flagsはTPNOCHANGEと設定されているが、*dataのタイプおよびサブタイプが、入力されるバッファーのタイプおよびサブタイプと一致しない場合です。この場合、*dataの内容と*lenはすべて何も変化はありません。対話型通信がトランザクションの一部である場合、そのトランザクションは応答が無視されるため、ロールバックされます。</p> <p>エラーが発生した場合、cdに対するポインターは無視され、対話型通信状態は保証できません。したがって、呼び出し元は対話型通信を終了する必要があります</p>
[TPEPROTO]	tprecv()が不適切な状況で呼び出された場合です。たとえば、送信者モードで使った場合に発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPETIME]	<p>タイムアウトが発生した場合です。関数の呼び出し元がトランザクション・モードの場合、トランザクション・タイムアウトが発生し、そのトランザクションはロールバックされます。関数の呼び出し元がトランザクションモードではなく、TPNOTIMEとTPNOBLOCKのどちらも指定されていない場合は、ブロック・タイムアウトが発生します。この2つの場合、*dataの内容とlenは変更されません。トランザクション・タイムアウトが発生した場合、トランザクションがロールバックされるまで、対話型通信でメッセージを送受信したり、新規接続を開始したりすることは、すべて[TPETIME]エラーになります</p>
[TPGOTSIG]	TPSIGRSTRTが設定されていない状態でシグナルが受信された場合です

## ● 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include "../sdl/demo.s"

main(int argc, char* argv[])
{
    int ret, cd;
    struct dat *buf;
    long revent, len;
```

```

    if (argc!=3) {error processing }
    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf=(struct dat *)tpalloc("STRUCT", "dat", 0);
    if (buf==NULL) { error processing }
    strcpy(buf->sdata, argv[1]);
    data process....

    cd=tpconnect("SERVICE", buf, 0, TPSENDONLY);
    if (cd==-1) { error processing }
    strcpy(buf->sdata, argv[2]);
    data process....

    ret=tpsend(cd, buf, 0,TPRECVONLY,&revent);
    if (ret==-1) { error processing }

    ret=tprecv(cd,(char*)&buf,&len,TPNOTIME,&revent);
    if (ret==-1) { error processing }
    data process....

    tpfree(buf);
    tpend();
}

```

- 関連関数

tpalloc(), tpconnect(), tpdiscn(), tpsend()

### 3.1.85. tpreissue

サーバーとクライアントで使用され、該当RQのフェイル・キューにたまっている要求データを、再度リクエスト・キューに入れる関数です。**tpenq()**などでRQを使用したサービスの実行中に、ネットワークの不安定やその他サーバー側のエラーによってサービス実行に失敗し、フェイル・キューにたまったクライアントのサービス要求データを再度リクエスト・キューに入れる関数です。保存されたデータを該当サービスに渡し、結果をリプレイ・キューに保存します。

- プロトタイプ

```

#include <tmaxapi.h>
int tpreissue(char *qname, char *filter, long flags)

```

- パラメータ

パラメータ	説明
qname	Tmaxシステムに登録されたRQの名前です
filter	現在サポートしておらず、NULLと設定します
flags	現行バージョンではサポートしていませんが、TPNOFLAGSに設定します

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpreissue()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、qnameがNULLである場合や、サポートしていないフィルターやフラグに上述した値以外の値を設定した場合に発生します
[TPENOENT]	qnameに該当するRQが存在しない場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。ネットワーク状態が不安定な場合に発生します

- 関連関数

tpenq(), tpdeq()

### 3.1.86. tpremoteconnect

リモート・ホストとTCPで接続する関数です。secが与えられると、その時間の間、接続を試みます。その時間を超えれば、接続エラーが発生します。

- プロトタイプ

```
#include <ucs.h>
int tpremoteconnect(char *host, int portno, int sec)
```

- 戻り値

戻り値	説明
0	ソケット番号です

戻り値	説明
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

### 3.1.87. tpscmt

サーバーとクライアントで使用され、環境ファイルに設定されているトランザクションのコミット方式設定を変更する関数です。

#### ● プロトタイプ

```
# include <tuxatmi.h>
int tpscmt(long flags)
```

#### ● パラメータ

パラメータ	説明
flags	以下の値の中から設定します – 1: トランザクション制御をTP_CMT_LOGGEDに設定します – 2: トランザクション制御をTP_CMT_COMPLETEに設定します

#### ● 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です

#### ● 例

```
#include <stdio.h>
#include <usrinc/tuxatmi.h>

int main(int argc, char *argv[])
{
    ...
    ret = tpstart((TPSTART_T *)NULL);
    if (ret == -1) {
        error processing
    }
    ...
    ret = tpscmt(TP_CMT_COMPLETED);
```

```

    if (ret < 0){
        error processing
    }

    ret = tpbegin();
    if (ret < 0){
        error processing
    }
    ...
    ret= tpcall("SERVICE1",(char*)buf, strlen(buf),(char **)&get,&rlen,TPNOFLAGS);

    if (ret == -1)
    {
        error processing
    }
    ...
    ret= tpcall("SERVICE2",(char*)buf,strlen(buf),(char **)&get,&rlen,TPNOFLAGS);

    if (ret == -1)
    {
        error processing
    }

    ret = tpcommit();
    if (ret < 0) {
        error processing
    }
    ...
}

```

### 3.1.88. tpsend

サーバーとクライアントで使用され、対話型通信で相手方プログラムにデータを送信する関数です。呼び出し元は、通信制御権をもっている必要があります。

- プロトタイプ

```

#include <atmi.h>
int tpsend (int cd, char *data, long len, long flags, long *revent)

```

- パラメータ

パラメータ	説明
cd	データが送信される接続を指定します。 <b>tpconnect()</b> あるいは <b>TPSVCINFO</b> 媒介変数から返される記述子です

パラメータ	説明
data	<b>tpalloc()</b> によって割り当てられたバッファです。アプリケーション・データが何も送信されていない場合（たとえば、いかなるデータも渡さず、通信制御権のみを譲渡する場合）、dataはNULLになります。dataのタイプおよびサブタイプは、接続された相手方が認識できるタイプおよびサブタイプである必要があります
len	送信するバッファ長です。dataの長さの指定が不要なバッファを指す場合、lenは無視され、0が使用されます。dataの長さの指定が必要なバッファを指す場合、lenは0になれません
flags	<p>flagsで使用可能な値は以下のとおりです</p> <ul style="list-style-type: none"> <li>– TPNOBLOCK <p>内部バッファが送信メッセージでいっぱいになったときのようなブロッキング状況が発生した場合、データとイベントは送信されません。TPNOBLOCKフラグが設定されていない状態でtpsend()が呼び出された場合にブロッキング状況が発生すると、呼び出し元はタイムアウト（トランザクション・タイムアウト、またはブロック・タイムアウトのうちいずれか）が発生するか、状況が緩和されるまで待機します</p> </li> <li>– TPNOTIME <p>関数の呼び出し元がブロック・タイムアウトを無視し、応答が受信されるまで無限に待機します。トランザクション・タイムアウト内でtpsend()を使用した場合は、トランザクション・タイムアウトが適用されます</p> </li> <li>– TPRECVONLY <p>呼び出し元がデータを送信後、通信制御権を相手方に譲渡します。呼び出し元は、再度通信制御権を譲渡されるまでtpsend()を呼び出すことはできません。対話の相手方の受信者は、tprecv()でデータを受信しながら、通信制御権をもつことを意味するTPEV_SENDOONLYイベントを受信するようになります。受信者も再度通信制御権を相手方に譲渡するまでtprecv()を呼び出すことはできません</p> </li> <li>– TPSIGRSTRT <p>シグナル割り込みを受け入れる場合に使用します。内部でシグナル割り込みが発生し、システム関数の呼び出しが妨害されたとき、システム関数の呼び出しが再実行されます。TPSIGRSTRTフラグが設定されていない状態でシグナル割り込みが発生した場合、関数は失敗し、tperrnoにTPGOTSIGが設定されます</p> </li> </ul>
revent	<p>記述子のcdに対するイベントが存在する場合、tpsend()は失敗処理され、データは送信されません。イベント・タイプはreventに返されます</p> <p>reventに渡されるイベントは以下のとおりです</p>

パラメータ	説明
	<ul style="list-style-type: none"> <li>– TPEV_DISCONIMM 対話従属者が受信するこのイベントは、対話開始者がtpdiscon()を使用して接続を強制終了したことを意味します。また、このイベントは通信エラー（たとえば、サーバー、ノード、ネットワーク障害など）によって接続が切断したときにも返されます</li> <li>– TREV_SVCERR 対話開始者が受信するこのイベントは、以下のTPEV_SVCFAIL状況以外の場合に、対話従属者が通信制御権がない状態でtpreturn()を実行したことを通知します</li> <li>– TREV_SVCFAIL 対話開始者が受信するこのイベントは、対話従属者が通信制御権がない状態でtpreturn()を実行し、その際tpreturn()はデータが何もない状態でTPFAILで呼び出されました。rvallはTPFAILであり、dataはNULLで実行されたことを通知します</li> </ul>

#### ● 戻り値

戻り値を返す場合、reventがTREV\_SVCFAILの場合、tpurcodeグローバル変数はtpreturn()を呼び出す際に渡されたrcode値になります。

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

#### ● エラー

tpsend()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、dataが指すバッファがtpalloc()で割り当てられていない場合や、flagsが有効でない場合に発生します
[TPEBADDESC]	記述子(cd)が有効でない場合です
[TPETIME]	タイムアウトが発生した場合です。関数の呼び出し元がトランザクション・モードの場合、トランザクション・タイムアウトが発生し、そのトランザクションはロールバックされます。関数の呼び出し元がトランザクションモードではなく、TPNOTIMEとTPNOBLOCKのどちらも指定されていない場合、ブロック・タイムアウトが発生します。この2つの場合、*dataの内容とlenは変更されません。トランザクション・タイ



エラーコード	説明
	ムアウトが発生した場合、トランザクションがロールバックされるまで、対話型通信でメッセージを送受信することや、新規接続を開始することはすべて[TPETIME]エラーとなります
[TPEEVENT]	イベントが発生した場合で、エラーが発生すると、dataは送信されず、イベント・タイプはreventで返されます
[TPEBLOCK]	TPNOBLOCKが設定された状態でブロッキング状況が発生した場合です
[TPGOTSIG]	TPSIGRSTRTが設定されていない状態でシグナルが受信された場合です
[TPEPROTO]	tpsend()が不適切な状況で呼び出された場合です。たとえば、受信者モードで使用する場合に発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

#### ● 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include "../sdl/demo.s"

main(int argc, char* argv[])
{
    int ret, cd;
    struct dat *buf;
    long revent, len;

    if (argc!=3) {error processing }
    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf=(struct dat *)tpalloc("STRUCT", "dat", 0);
    if (buf==NULL) { error processing }
    strcpy(buf->sdata, argv[1]);
    data process....

    cd=tpconnect("SERVICE", buf, 0, TPSENDONLY);
    if (cd==-1) { error processing }
    strcpy(buf->sdata, argv[2]);
    data process....

    ret=tpsend(cd, buf, 0, TPRECVONLY, &revent);
    if (ret==-1) { error processing }
```

```

ret=tprecv(&cd, (char**)&buf, &len, TPNOTIME, &revent);
if (ret==-1) { error processing }
data process....

tpfree(buf);
tpend();
}

```

- 関連関数

tpalloc(), tpconnect(), tpdiscon(), tprecv(), tpreturn()

### 3.1.89. tpsetsvctimeout

サーバーで使用される関数で、サーバーに設定されているサービスのタイムアウト時間を設定します。tpsetsvctimeout()でサービス時間を設定した場合、この関数が呼び出されたときから設定した時間(秒)の間、サービス要求に対する応答を待ちます。設定した時間が過ぎても応答を受信できなかった場合には、タイムアウト・エラーが発生し、要求したサービスに対する応答を待たずにサービス失敗を返します。

- プロトタイプ

```

#include <tmaxapi.h>
int tpsetsvctimeout (int sec, long flags)

```

- パラメータ

パラメータ	説明
sec	サービス・タイムアウト時間を秒単位で設定します
flags	現在は使用されません

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpsetsvctimeout()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <stdlib.h>
#include <usrinc/tmaxapi.h>
#include <usrinc/tdlcall.h>

TOUPPER(TPSVCINFO *msg)
{
    int            i;

    if ( tpsetsvctimeout(10, 0) < 0 )
        ;
    //error handle code
    printf("TOUPPER service is started!\n");
    sleep(15);
    printf("INPUT  : data=%s\n", msg->data);

    for (i = 0; i < msg->len; i++)
        msg->data[i] = toupper(msg->data[i]);
    printf("OUTPUT: data=%s\n", msg->data);
    tpreturn(TPSUCCESS,0,(char *)msg->data, 0,0);
}
```

### 3.1.90. tpset\_timeout

サーバーとクライアントで使用され、サーバーに設定されているサービス制限時間であるブロック・タイムアウト時間を設定する関数です。tpset\_timeout()でサービス制限時間を設定した場合、設定した時間の間、サービス要求に対する応答を待機します。設定した時間が過ぎても応答を受信できなかった場合、タイムアウト・エラーが発生し、要求したサービスに対する応答を待機せず、サービス失敗と返します。

tpset\_timeout()は、該当関数が呼び出された後のサービス要求に適用されます。再度tpset\_timeout()が呼び出されたり、クライアントやサーバー・プロセスが終了するまで関数は有効です。tpset\_timeout()が使用されない場合、ブロック・タイムアウトにTmax環境ファイルに設定された**BLOCKTIME**が適用されます。

- プロトタイプ

```
#include <tmaxapi.h>
int  tpset_timeout (int sec)
```

#### ● パラメータ

パラメータ	説明
sec	ブロック・タイムアウト時間を秒単位で設定します

#### ● 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

#### ● エラー

tpset\_timeout()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

#### ● 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

void main(int argc, char *argv[])
{
    int ret;
    char *sndbuf, *rcvbuf;
    long sndlen, rcvlen;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    sndbuf = (char *)tpalloc("CARRAY", NULL, 20);
    if (sndbuf==NULL) {error processing };
    rcvbuf = (char *)tpalloc("CARRAY", NULL, 20);
    if (rcvbuf==NULL) {error processing };
    data process....
```

```

    sndbuf=strlen(sndbuf);
    ret=tpset_timeout(4);
    if (ret==-1) { error processing }

    ret=tpcall("SERVICE", sndbuf, sndlen, &rcvbuf, &rcvlen, TPNOCHANGE);
    if (ret==-1) { error processing }
    data process....

    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
}

```

### 3.1.91. tpsetfd

ソケットFDをUCSプロセスの外部ソケット・スケジューラーに登録する関数です。UCS方式プロセスを使用したソケットFDを起動する際に使用されます。UCSスケジューラーは、TMM、CLHだけでなく、該当ソケットFDに到着したメッセージも一緒にチェックします。ユーザーが指定したソケットにメッセージが到着した場合、tpschedule()は特に処理をすることなく、正常結果(UCS\_USER\_MSG)を返します。また、どのソケットにメッセージが到着したかを確認するためには、tpsetfd()を使用します。

#### ● プロトタイプ

```

#include <ucs.h>
int tpsetfd (int fd)

```

#### ● パラメータ

パラメータ	説明
fd	登録するソケットFDを設定します

#### ● 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

#### ● エラー

tpsetfd()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

## ● 例

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <usrinc/ucs.h>
...
#define SERV_ADDR "168.126.185.129"
#define SERV_PORT 1500

int fd_read(int, char *, int);
extern int errno;

int usermain(int argc, char *argv[])
{
    ...
    int listen_fd, n, newfd;
    struct sockaddr_in my_addr, child_addr;
    socklen_t child_len;
    buf = tpalloc("STRING", NULL, 0);
    if (buf == NULL){
        error processing
    }

    memset((void *)&my_addr, NULL, sizeof(my_addr));
    memset((void *)&child_addr, NULL, sizeof(child_addr));
    listen_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (listen_fd == -1){
        error processing
    }

    my_addr.sin_family = AF_INET;
    inaddr = inet_addr(SERV_ADDR);
    my_addr.sin_port = htons((unsigned short)SERV_PORT);

    if (inaddr != -1){
        memcpy((char *)&my_addr.sin_addr, (char *)&inaddr, sizeof(inaddr));
    }
    ret = bind(listen_fd, (struct sockaddr *)&my_addr, sizeof(my_addr));
    if (ret == -1){
```

```

        error processing
    }
    ret = listen(listen_fd, 5);
    if (ret == -1){
        error processing
    }

    ret = tpsetfd(listen_fd);
    if (ret == -1){
        error processing
    }
    ...

while(1) {
    n = tpschedule(10);
    ...
    if (n == UCS_USER_MSG){
        if (tpissetfd(listen_fd)) {
            child_len = sizeof(child_addr);
            newfd = accept(listen_fd, &child_addr, &child_len);
            if (newfd == -1){
                error processing
            }

            ret = tpsetfd(newfd);
            if (ret == -1){
                error processing
            }
        }

        if (tpissetfd(newfd)){
            /* ソケットからバッファを読み込みます */
            fd_read(newfd, buf, 1024);
            ret = tpcall("SERVICE", (char *)buf, sizeof(buf), (char **)&buf,
                        (long *)&rlen, TPNOFLAGS);
            if (ret == -1){
                error processing
            }
            ...
            tpclrfd(newfd);
            close(newfd);
        }
        ...
    }
    return 1;
}

```

- 関連関数

tpclrfd(), tpissetfd()

## 3.1.92. tpsleep

サーバーとクライアントで使用され、データが到着するまで待機する関数です。最大タイムアウト時間の間待機し、その間にデータが到着した場合は即時に返します。

- プロトタイプ

```
#include <tmaxapi.h>
int tpsleep (struct timeval *timeout)
```

- パラメータ

timeoutはtimevalという構造体に対するポインターであり、以下のように構成されています (UNIXシステムの<sys/time.h>を参照してください)。

```
struct timeval{
    long    tv_sec      /* seconds unit */
    long    tv_usec     /* 0.000001 (micro seconds) unit */
};
```

- 戻り値

戻り値	説明
正の整数	タイムアウト時間にデータが到着した場合です
0	タイムアウト時間までにデータが到着しなかった場合です
-1	関数の実行中にエラーが発生した場合です。tpernoにエラーコードが設定されます

- エラー

tpsleep()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例



```

#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

void main(int argc, char *argv[])
{
    int ret, cd;
    char *buf;

    struct timeval sl_time, *sleep;
    sl_time.tv_sec =3;
    sl_time.tv_usec=500000;
    sleep=&sl_time;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf = (char *)tpalloc("CARRAY", NULL, 20);
    if (buf==NULL)
        {error processing };
    data process....

    cd=tpacall("SERVICE", buf, 20, TPNOFLAGS);
    if (cd==-1)
        { error processing }

    ret=tpsleap(sleep); /* 3.5秒間待機 */
    if (ret==-1)
        { error processing }
    if (ret==0)
        printf("waited 3.5sec\n");

    ret=tpgetrply(&cd, &buf, &len, TPNOTIME);
    if (ret==-1)
        { error processing }
    data process....

    tpfree((char *)buf);
    tpend();
}

```

- 関連関数

tp\_sleep(), tp\_usleep(), tpacall(), tpbroadcast(), tpgetrply()

## 3.1.93. tpspracall

**tpgetsprlist()**を使用して取得したサーバー・プロセスの索引のうち、特定プロセスにサービスを呼び出す関数です。非同期通信でメッセージを送信後、結果を受信するまで待機せず、即時返します。

結果は、後で**tpgetreply()**を使用して応答を受信することもでき、**tpcancel()**を使用して応答をキャンセルすることもできます。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tpspracall(char *svcname, int spri, char *data, long lenl, long flagsl);
```

- パラメータ

パラメータ	説明
svcname	データを渡すサービスを設定します
spri	データを渡すプロセス番号を設定します
data	サービス要求のデータに対するポインターです。以前tpalloc()によって割り当てられたバッファである必要があります。idataのタイプとサブタイプは、svcがサポートするタイプである必要があります
lenl	送信されるデータ長です
flagsl	現行バージョンではサポートしていませんが、TPNOFLAGSあるいは0に設定します

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpspracall()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPENOENT]	該当索引のサーバー・プロセスが存在しない場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```

....
main(int argc, char *argv[])
{
    char    *sndbuf, *rcvbuf;
    long    sndlen, rcvlen;
    int     ret, cd;
    int     starti = 0, endi = 0, spri = 0;
    if (argc != 2) {
        printf("Usage: argv[1] string\n");
        exit(1);
    }

    if ( (ret = tmaxreadenv( "tmax.env", "TMAX" )) == -1 ){
        printf("<%-15s> tmaxreadenv fail [%s]", __FILE__, tpstrerror(tperrno));

        exit(1);
    }

    if (tpstart((TPSTART_T *)NULL) == -1){
        printf("<%-15s> tpstart fail [%s]", __FILE__, tpstrerror(tperrno));
        exit(1);
    }

    if ((sndbuf = (char*)tpalloc("CARRAY", 0, 0)) == NULL) {
        printf("<%-15s> sndbuf tpalloc fail [%s]", __FILE__, tpstrerror(tperrno));

        exit(1);
    }

    if ((rcvbuf = (char*)tpalloc("CARRAY", 0, 0)) == NULL) {
        printf("<%-15s> sndbuf tpalloc fail [%s]", __FILE__, tpstrerror(tperrno));

        tpfree((char*)sndbuf);
        exit(1);
    }

    /* tpgetsprlist in client */
    ret = tpgetsprlist(argv[1], 2, &starti, &endi, 0);
    if (ret < 0) {
        printf("<%-15s> tpgetsprlist fail [%s]", __FILE__, tpstrerror(tperrno)
);
        exit(1);
    } else {
        /*printf("Received Message from Server :
        starti[%d], endi[%d]\n", starti, endi);*/
    }
    strcpy((char*)sndbuf, "tpspracall test");

```

```

/* tpspracall in client */
for(spri = starti; spri <= endi; spri++) {
    cd = tpspracall(argv[1], spri, sndbuf, strlen(sndbuf), 0);
    if(cd < 0) {
        printf("<%-15s> tpsprcall fail [%s]", __FILE__, tpstrerror(tperrno));

        exit(1);
    } else {
        ret = tpgetrply(&cd, &rcvbuf, &rcvlen, 0);
        if(ret < 0) {
            printf("Received Message from %d spri: tpgetrply fail[%s]\n",
                spri, tpstrerror(tperrno));
        } else {
            printf("Received Message from %d spri: msg[%s], len[%d]\n",
                spri, rcvbuf, rcvlen);
        }
    }
}
tpend();
}

```

### 3.1.94. tpsprio

サーバーとクライアントで使用され、サービス要求の優先順位を設定する関数です。以降、転送されるtpacall()の順番を設定します。設定された順番は、それ以降受信された要求にのみ影響を与えます。順番とは、要求されたメッセージのサービス優先順位を表す値です。順番は0から100までで、各サービスのデフォルト値は50です。

**tpsprio()**と**tpgprio()**を使用して、要求されたメッセージのサービス順位を制御できます。

- プロトタイプ

```

#include <tuxatmi.h>
int tpsprio (int prio, long flags)

```

- パラメータ

パラメータ	説明
prio	-50から50の間の整数を使用できます。prio値と現在の順番を足した値は、次の要求に使用されます
flags	現行バージョンではサポートしていませんが、TPNOFLAGSあるいは0に設定します

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpsprio()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

void main(int argc, char *argv[])
{
    int ret;
    char *buf;
    long len;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf = (char *)tpalloc("STRING", NULL, 0);
    if (buf==NULL) {error processing };
    ret=tpsprio(30,0);
    if (ret==-1) { error processing }

    ret=tpcall("SERVICE", buf, 0, &buf, &len, TPNOFLAGS);
    if (ret==-1) { error processing }

    data process....

    ret=tpsprio(-20,0);
    if (ret==-1) { error processing }

    ret=tpcall("SERVICE", buf, 0, &buf, &len, TPNOFLAGS);
    if (ret==-1) { error processing }
```

```

data process....

tpfree(buf);
tpend();
}

```

- 関連関数

tpacall(), tpcall(), tpgprio()

### 3.1.95. tpsetctx

現行コンテキストを設定する関数です。クライアント・プログラムとサーバー・プログラムで、以下のように作成方法が異なります。

- クライアント・プログラム

関数を使用して1つのクライアントが生成済みの他のコンテキストを現在のクライアントに割り当てることができます。ほとんどのATMI関数は、コンテキスト単位になっています。クライアントの現在のコンテキストを知るためには、tpgetctx()関数を使って返される値を確認します。

クライアントは複数のコンテキストを使用できますが、該当する瞬間にはたった1つのコンテキストのみをもちます。たとえば、context1でtpacall()を実行した場合、別のコンテキストを使用したとしても、tpgetctx()を正常に行うためには、tpgetctx()を実行する時点でcontext1を現在のコンテキストに設定する必要があります。

- サーバー・プログラム

サービス・スレッドは、サービス処理の際に割り当てられたコンテキストを使用しますが、ユーザー作成スレッドは独自のコンテキストが存在しません。ほとんどのATMI関数は、コンテキストが割り当てられるまでは動作しないため、ユーザー作成スレッドは必要な場合、サービス・スレッドのコンテキストを共有して使用しなければなりません。ユーザー作成スレッドは、tpsetctx関数を使用して他のサービス・スレッドのコンテキストを共有することができます。

tpsetctxを呼び出したユーザー作成スレッドは、サービス・スレッドとコンテキスト情報を共有します。たとえば、サービス・スレッドでtpacall()を呼び出した場合、以降ユーザー作成スレッドでtpgetctx()により要求に対する応答を受けることができます。

tpsetctx()関数は、サービス・スレッドでは使用できません。サービス・スレッドは基本的に自身のコンテキストをもち、別のコンテキストに切り替えて使用することはできません。したがって、サービス・スレッドでtpsetctx()関数を呼び出すと、TPEPROTOエラーコードと共にエラーを返します。

ユーザー作成スレッドで、この関数によりサービス・スレッドのコンテキストを共有した場合には、サービス・スレッドがtpreturn()を呼び出す前に、ユーザー作成スレッドがtpsetctx(TPNULLCONTEXT)を呼び出す必要があります。つまり、サービス・スレッドがtpreturn()を呼び出す時点で、他のユーザー作成スレッドが

サービス・スレッドのコンテキストを共有しないように変更しなければなりません。これを守らない場合、`tpreturn()`は失敗し、クライアントに`TPESVCERR`エラーコードを返します。したがって、必ず同期化などにより、これらのスレッド間のプロセスの流れを制御する必要があります。`tpsetctxt()`関数の`ctxtid`パラメータは、サービス・スレッドで`tpgetctxt()`関数を呼び出して取得したコンテキストIDを使用します。

- プロトタイプ

```
#include <usrinc/atmi.h>
int tpsetctxt(int ctxtid, long flags)
```

- パラメータ

パラメータ	説明
ctxtid	関数を呼び出した際の現行コンテキストを設定します。設定可能なコンテキストは、クライアント・プログラムでは、 <code>tpstart()</code> を使用して新規コンテキストが生成されたIDであり、サーバー・プログラムでは、サービス・スレッドのコンテキストIDです。  TPNULLCONTEXTを始め、他の使用可能なコンテキストに設定できますが、TPINVALIDCONTEXTには設定できません
flags	現行バージョンではサポートしていませんが、TPNOFLAGSあるいは0に設定します

- 戻り値

戻り値	説明
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

`tpsetctxt()`が正常に実行されなかった場合、`tperrno`に以下の値のいずれかが設定されます。

エラーコード	説明
[TPEPROTO]	サーバー・プログラムの場合、サービス・スレッドで呼び出した場合や、パラメータとして渡されたコンテキストIDが無効な値である場合など、不適切な状況で関数を呼び出した場合に発生します
[TPEINVAL]	設定されたパラメータが正しくない場合であり、 <code>ctxtid</code> に0またはTPINVALIDCONTEXTが設定された場合や、 <code>flags</code> 値に0以外の値が設定された場合に発生します。  クライアント・プログラムでは、 <code>tpstart()</code> を実行する前にこの関数を呼び出した場合に発生しますが、 <code>tpstart()</code> の呼び出し時にバッファのフラグをTPMULTICONTEXTSに設定していない状態で、この関数を呼び出した場合に発生します
[TPENOENT]	<code>ctxtid</code> に設定された値が設定可能なコンテキストでない場合です

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

#### ● 例 - クライアント・プログラム

```
int altContext(int id)
{
    int i;
    int ret;
    ret = tpsetctxt(id, TPNOFLAGS);
    if (ret < 0)
    {
        printf("\t[altContext]tpsetctxt fail[%d][%s]\n"tperrno,
               tpstrerror(tperrno));
        tpfree((char *)tpinfo);
        return -1;
    }
    return 1;
}
```

#### ● 例 - サーバー・プログラム

```
typedef param {
    int ctxtid;
    TPSVCINFO *svcinfo;
} param_t;

MSERVICE(TPSVCINFO *svcinfo)
{
    pthread_t tid;
    param_t param;

    printf("MSERVICE service is started!");
    tpgetctxt(&param.ctxtid, TPNOFLAGS);
    param.svcinfo = svcinfo;

    pthread_create(&tid, NULL, THREAD_ROUTINE, &param);
    pthread_join(tid, NULL);

    printf("MSERVICE service is finished!");
    tpreturn(TPSUCCESS, 0, svcinfo->data, 0L, TPNOFLAGS);
}
```



```

void *THREAD_ROUTINE(void *arg)
{
    param_t *param;
    TPSVCINFO *svcinfo;

    param = (param_t *)arg;
    svcinfo = param->svcinfo;

    if (tpsetctxt(param->ctxtid, TPNOFLAGS) == -1) {
        printf("tpsetctxt(%d) failed, [tperrno:%d]", param->ctxtid, tperrno);
        return NULL;
    }

    tpccall("MTOUPPER", sndbuf, 0, &rcvbuf, &rcvlen, TPNOFLAGS);

    if (tpsetctxt(TPNULLCONTEXT, TPNOFLAGS) == -1) {
        printf("tpsetctxt(TPNULLCONTEXT) failed, [tperrno:%d]", tperrno);
        return NULL;
    }

    return NULL;
}

```

- 関連関数

tpgetctxt()

## 3.1.96. tpstrerror

サーバーとクライアントで使用され、エラー番号に該当するメッセージを出力する関数です。Tmax関数の使用中にエラーが発生した場合、該当エラーコードはtperrnoというグローバル変数に設定されます。tpstrerror()関数は、tperrnoに設定されたエラーに対するメッセージを出力する関数です。

- プロトタイプ

```

#include <atmi.h>
char *tpstrerror (int tperrno)

```

- パラメータ

パラメータ	説明
tperrno	エラー・メッセージを出力するエラーコードです

- 戻り値

戻り値	説明
エラー・メッセージ	エラーコードに対するメッセージがある場合に返します
NULL	エラーコードに対するメッセージがない場合に返します

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>

void main(int argc, char *argv[])
{
    int ret;
    char buf;
    TPSTART_T *tpinfo;

    tpinfo = (TPSTART_T *)tpalloc("TPSTART", NULL, sizeof(TPSTART_T));
    if (tpinfo==NULL) { error processing }
    strcpy(tpinfo->dompwd, "tuxedo");
    if (tpstart(tpinfo) == -1){
        printf("tpstart fail , err = %s\n", tpstrerror(tperrno));
        exit(1);
    }

    buf = (char *)tpalloc("CARRAY", NULL, 20);
    if (buf==NULL) {error processing };
    data process....

    tpfree((char *) buf);
    tpend();
}
```

## 3.1.97. tpsubqname

サーバーとクライアントで使用され、サブキュー番号に該当するキューの名前を返す関数です。

- プロトタイプ

```
# include <tmaxapi.h>
char *tpsubqname(int type)
```

- パラメータ

[tpqstat\(\)](#)、[tpqsvcstat\(\)](#)などで使用されたパラメータと同じ意味をもちます。サブキュー番号に該当する名前は以下のとおりです。

サブキュー番号	説明
0	RQ_ANY
1	RQ_FAIL
2	RQ_REQ
3	RQ_RPLY

- 戻り値

戻り値	説明
サブキュー名	該当サブキュー番号に該当するサブキューがある場合に返します
NULL	該当サブキュー番号に該当するサブキュー名がない場合に返します

### 3.1.98. tpsubscribe

サーバーとクライアントで使用され、特定イベントの発生時にクライアントやサーバーからのデータ転送を受信するための要求を登録する関数です。eventnameに該当するイベントが発生した場合に通知するようTmaxシステムに登録します。

- プロトタイプ

```
# include <tmaxapi.h>
long tpsubscribe(char *eventname, char *filter, TPEVCTL *ctl, long flags)
```

- パラメータ

パラメータ	説明
eventname	NULLで終わる63文字以内の文字列で、メッセージを受信するイベントの名前を指定します。ワイルドカードやpartial-matchingなどはサポートされず、全体文字列が一致する名前のイベントのみ登録できます
filter	今後の拡張のために予約をしておいたものです。NULLを指定します
ctl	イベントが発生した場合、メッセージを取得する方式を指定する構造体です。tpsubscribe()の主体によって異なる動作をします。クライアントでtpsubscribe()を使用した場合、ctlは常にNULLである必要があり、メッセージはクライアントにunsolicitedデータ形式で渡されます。クライアントはtpsubscribe()あるいはtpgetunsol()などの関数を使用して渡されたデータを処理します
flags	現在はTPNOTIMEのみ意味があります

サーバーでtpsubscribe()を実行する場合、ctlがNULLになってはいけません。下記内容を参照してください。

```

struct tpevctl {
    long ctl_flags;
    long post_flags;
    char svc[XATMI_SERVICE_NAME_LENGTH];
    char qname[RQ_NAME_LENGTH];
};
typedef struct tpevctl TPEVCTL;

```

メンバー	説明
ctl_flags	今後の拡張のために作成されたもので、現在は0に設定する必要があります
post_flags	今後の拡張のために作成されたもので、現在は0に設定する必要があります
qname	qnameを使用する場合、メッセージはtpenq(qname、NULL、data、len、TPNOFLAGS)を使用してRQに保存されます
svc	svcを使用する場合、メッセージはtpacall(svc、data、len、TPNOREPLY)と似た方式でサーバーに渡され、サーバーの戻り値は無視されます。qnameとsvcのいずれか1つだけを使用します

- 戻り値

戻り値	説明
descriptor	関数の呼び出しに成功した場合です。tpunsubscribe()を呼び出すときに使用されるdescriptorを返します
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpsubscribe()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。クライアントでは、ネットワーク・エラーが最も多いです
[TPEOS]	OSレベルで発生するエラーです。メモリーの割り当てなどの問題を含みます
[TPEINVAL]	設定された引数が正しくない場合です
[TPENOENT]	TPEVCTL構造体のqnameやsvcに該当するRQやサーバーが存在しない場合です
[TPEPROTO]	クライアントとサーバーでctlがNULLでない場合です
[TPETIME]	タイムアウトが発生した場合です

- 関連関数

tpsetunsol(), tpsetunsol\_flag(), tpgetunsol(), tpacall(), tpenq()

## 3.1.99. tptypes

サーバーとクライアントで使用され、バッファのタイプおよびサブタイプについての情報を提供する関数です。tptypes()は、ptrでデータ・バッファに対するポインターを入力してもらい、typeとsubtypeにそれぞれバッファのタイプとサブタイプを返します。

- プロトタイプ

```
#include <atmi.h>
long tptypes (char *ptr, char *type, char *subtype )
```

- パラメータ

パラメータ	説明
ptr	必ずtpalloc()で割り当てられたバッファを指します
type, subtype	NULLでない場合、指している文字列にそれぞれバッファのタイプとサブタイプの名前をもちます。サブタイプが存在しない場合、subtypeが指す配列はNULLストリングを含みます。名前が最大長の場合、文字列はNULLで終わりません(typeの最大長は8文字、subtypeの最大長は16文字です)。  typeの先頭の8バイトとsubtypeの先頭の16バイトのみが有効な値をもちます

- 戻り値

戻り値	説明
バッファサイズ	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tptypes()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、ptrが指すバッファがtpalloc()で割り当てられていない場合に発生します
[TPEPROTO]	関数が不適切な状況で呼び出された場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```

#include <stdio.h>
#include <usrinc/atmi.h>
#include "../sdl/demo.s"
main(int argc, char *argv[])
{
    int ret;
    struct sel_o *rcvbuf;
    char type[9], subtype[17];
    long size;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }
    buf=(struct sel_o*)tpalloc("STRUCT","sel_o",0);
    if (buf==NULL) {error processing };

    size =tptypes((char*)buf, type,subtype);
    if (size==-1) {error processing };
    printf ("buf : size %d, type %s, subtype %s\n\n", size, type, subtype);

    /*rcvbuf : size 1024, type STRUCT, subtype sel_o */
    data process...
    tpfree((char *)buf);
    tpend();
}

```

- 関連関数

tpalloc(), tpfree(), tprealloc()

## 3.1.100. tpunsubscribe

サーバーとクライアントで使用され、**tpsubscribe()**で登録した特定イベントに対する要求を解除する関数です。登録されたすべての要求が解除された場合、Tmaxシステムは該当イベントの表を削除します。

- プロトタイプ

```

# include <tmaxapi.h>
int tpunsubscribe(long sd, long flags)

```

- パラメータ

パラメータ	説明
sd	tpsubscribe()で登録時に受け取った戻り値です
flags	現在はTPNOTIMEのみ意味があります

- 戻り値

戻り値	説明
正数	関数の呼び出しに成功した場合です
負数	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpunsubscribe()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。クライアントでは、ネットワーク・エラーが最も多いです
[TPEOS]	OSレベルのエラーが発生した場合です。メモリーの割り当てなどの問題を含みます
[TPEINVAL]	設定された引数が正しくない場合です
[TPETIME]	タイムアウトが発生した場合です

- 関連関数

tpsetunsol(), tpsetunsol\_flag(), tpgetunsol(), tpacall(), tpenq()

### 3.1.101. tuxgetenv

サーバーとクライアントで使用され、環境変数値を返す関数です。サーバーで使用される**getenv()**と同じ機能を行います。クライアントでは**autoexec.bat**ファイルに適用された値を返し、サーバーでは各シェル環境ファイルに適用された値を返します。

- プロトタイプ

```
#include <tuxfml.h>
char *tuxgetenv(char *name)
```

- パラメータ

パラメータ	説明
name	返される環境変数を設定します

- 戻り値

戻り値	説明
環境変数値のポインター	環境変数が存在する場合です
NULL	該当環境変数が存在しない場合です

- 例

```
...
#include <tuxinc/tuxatmi.h>
int main(int argc, char *argv[])
{
    ...
    char hostAddr[20];
    ...
    memset(hostAddr, NULL, sizeof(hostAddr));
    memcpy(hostAddr, tuxgetenv("TMAX_HOST_ADDR"), sizeof(hostAddr));
    printf("host address = %s\n", hostAddr);
    ...
}
```

- 関連関数

tuxputenv()

## 3.1.102. tuxputenv

サーバーとクライアントで使用される関数で、環境変数を適用します。string形式で入力された「名前=値」を環境変数に適用します。設定する環境変数の値がすでに存在する場合、該当環境変数値を変更します。環境変数がない場合は、新しい環境変数を追加します。サーバーで使用する**putenv()**と同じ機能をします。クライアントでは**autoexec.bat**ファイルに適用された値を、サーバーでは各シェル環境ファイルに適用された値をそれぞれ設定します。

- プロトタイプ

```
#include <tuxfml.h>
int tuxputenv(char *string)
```

- パラメータ

パラメータ	説明
string	環境変数に設定する値を入力します

- 戻り値



戻り値	説明
0	関数の呼び出しに成功した場合です
負数	環境変数が十分なメモリー領域を確保できなかった場合です

- 例

```

...
#include <tuxinc/tuxatmi.h>
int main(int argc, char *argv[])
{
    ...
    char hostAddr[20];
    char envVal[20];
    ...
    ret = tuxputenv("A=10");
    if (ret < 0){
        printf("tuxputenv fail...[%d]\n", tperrno);
    }
    memset(envVal, NULL, sizeof(envVal));
    memcpy(envVal, tuxgetenv("A"), sizeof(envVal));
    printf("envVal = %s\n", envVal);
    ...
}

```

- 関連関数

tuxgetenv()

### 3.1.103. tuxreadenv

サーバーとクライアントで使用され、ファイルの環境変数を読み込む関数です。Tmaxシステムに接続するには、環境変数が登録されている必要があります。クライアントは登録された環境変数を参照し、tpstart()を使用してTmaxシステムに接続します。DOSは<autoexec.bat>ファイルに環境変数を定義します。UNIXはcshに<.cshrc>、kshに<.profile>を定義します。2つ以上のシステムに接続する必要がある場合、状況によってシステムを変更します。この場合、該当する2つのシステムに関する情報を環境変数に登録できないため、クライアントは各環境変数をファイル・タイプで登録します。

tuxreadenv()は、接続したシステムに関する情報をファイルから読み込み、環境変数を新規設定する関数です。そのため、Tmaxシステムに接続する前に、システムに関する情報が環境変数に予め登録されている必要があります。したがって、関数はTmaxに接続する前に実行させる必要があります。

- プロトタイプ

```
#include <tuxfml.h>
int tuxreadenv (char *file, char *label)
```

## ● パラメータ

パラメータ	説明
file	システムの接続環境に関する情報が保存されているファイル名を表します。このファイルは、予め作成されたテキスト・フォーマットによって登録されます
label	ファイルに登録された環境設定情報についての記述子です。この値で1つのファイルに2つ以上のシステム情報が登録された場合、各システムを区別できます

## ● 戻り値

戻り値	説明
1	sec秒までデータが受信されなかった場合です
-1	関数の呼び出しに失敗した場合です。tperno!にエラーコードが設定されます

## ● エラー

tuxreadenv()が正常に実行されなかった場合、tperno!に以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	ファイルがまたはラベルが存在しない場合に発生します

## ● 例

```
...
#include <tuxinc/tuxatmi.h>
int main(int argc, char *argv[])
{
    ...
    ret = tuxreadenv( "tmax.env", "TMAX" );
    if (ret == -1){
        error processing
    }
    ret = tpinit((TPINIT *)NULL);
    if (ret == -1){
        error processing
    }
    ...
    ret = tpcall("SERVICE", buf, 0, &buf, &rcvlen, TPNOFLAGS);
    if (ret == -1){
        error processing
    }
}
```

```
    }  
    ...  
    tpterm();  
}
```

- 関連関数

tpstart()

---

**参考**

ファイルに環境変数を登録する方法については『Tmax 運用ガイド』を参照してください。

---

### 3.1.104. tx\_begin

サーバーとクライアントで使用される関数で、呼び出しを実行するとグローバル・トランザクションを開始します。関数の呼び出し元は、トランザクション・モードになります。トランザクションを開始する前に、呼び出しプロセスが**tx\_open()**でリソース・マネージャーに接続されている必要があります。tx\_begin()は、呼び出し元がすでにトランザクション・モードにあるか、tx\_open()が呼び出されていない場合は失敗し、[TX\_PROTOCOL\_ERROR]を返します。

トランザクションが開始すると、呼び出しプロセスは現行トランザクションを完了するために、**tx\_commit()**あるいは**tx\_rollback()**を呼び出します。トランザクションを開始するために必ずしもtx\_begin()を直接呼び出す必要がない、連続(chaining)トランザクションも存在します。詳細については「[3.1.105. tx\\_commit](#)」と「[3.1.107. tx\\_rollback](#)」を参照してください。

- プロトタイプ

```
#include <tx.h>  
int tx_begin (void)
```

- 戻り値

戻り値	説明
TX_OK	関数の呼び出しに成功した場合です
負数	関数の呼び出しに失敗した場合です

- エラー

tx\_begin()が正常に実行されなかった場合、以下のエラーコードを返します。

エラーコード	説明
[TX_OUTSIDE]	現行呼び出しプロセスが外部のグローバル・トランザクションに参与しているため、トランザクション・マネージャーがグローバル・トランザクションを開始できません。それらの作業をすべて完了しなければ、グローバル・トランザクションを開始できません。参与しているトランザクションには影響を与えません
[TX_PROTOCOL_ERROR]	関数が不適切な状況で呼び出された場合です。たとえば、呼び出し元がすでにトランザクション・モードの場合に発生します。現行トランザクションには影響を与えません
[TX_ERROR]	トランザクション・マネージャーまたはリソース・マネージャーがトランザクションを開始中に一時的なエラーが発生した場合です。エラーが返された場合、呼び出し元はトランザクション・モードではなくなります。エラーの原因は製品の特性によって決定されます
[TX_FAIL]	致命的なエラーが発生し、トランザクション・マネージャーやリソース・マネージャーがアプリケーションのための作業を実行できない場合です。エラーが返された場合、呼び出し元はトランザクション・モードではなくなります。エラーの原因は製品の特性によって異なります

#### ● 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tx.h>

void main(int argc, char *argv[])
{
    char *buf;
    int ret,cd;
    long len, revent;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf = (char *)tpalloc("CARRAY", NULL, 20);
    if (buf==NULL) {error processing };

    data process....

    ret=tx_set_transaction_timeout( 5 );
    if (ret<0) { error processing }

    ret=tx_begin();
    if (ret<0) { error processing }

    cd = tpconnect("SERVICE", buf, 20, TPRECVONLY);
    if (cd==-1) { error processing }
```

```

    ret = tprecv(cd, (char **)&buf, &len, TPNOFLAGS, &revent));
    if (ret < 0 && revent != TPEV_SVCSUCC)
        tx_rollback();
    else
        tx_commit();

    data process....

    tpfree((char *)buf);
    tpend();
}

```

- 関連関数

tx\_commit(), tx\_open(), tx\_rollback(), tx\_set\_transaction\_timeout()

### 3.1.105. tx\_commit

サーバーとクライアントで使用され、グローバル・トランザクションをコミットする関数です。

transaction\_controlの特性がTX\_UNCHAINEDの場合、tx\_commit()が返す際に、呼び出し元はトランザクション・モードではなくなります。transaction\_controlの特性がTX\_CHAINEDの場合、tx\_commit()が返す際に、呼び出し元は新規トランザクションのためにトランザクション・モードのまま残ります。transaction\_controlの特性についての詳細は「[3.1.109. tx\\_set\\_transaction\\_control](#)」を参照してください。

- プロトタイプ

```

#include <tx.h>
int tx_commit(void)

```

- 戻り値

戻り値	説明
TX_OK	関数の呼び出しに成功した場合です
負数	関数の呼び出しに失敗した場合、エラーコードを返します

- エラー

tx\_commit()が正常に実行されなかった場合、以下のエラーコードを返します。

エラーコード	説明
[TX_NO_BEGIN]	transaction_controlの特性がTX_CHAINEDの場合にのみ発生するエラーで、トランザクションがコミットされます。このエラーが発生した場合、新規トランザクションは開始できず、呼び出し元はトランザクション・モードではなくなります
[TX_ROLLBACK]	トランザクションがロールバックされた場合です。transaction_controlの特性がTX_CHAINEDの場合、新規トランザクションが開始されます
[TX_ROLLBACK_NO_BEGIN]	transaction_controlの特性がTX_CHAINEDの場合のみ発生するエラーで、トランザクションがロールバックされます。このエラーが発生した場合、新規トランザクションは開始できず、呼び出し元はトランザクション・モードではなくなります
[TX_HAZARD]	エラーのために、トランザクションが一部はコミットされ、一部はロールバックされます。transaction_controlの特性がTX_CHAINEDの場合、新規トランザクションが開始されます
[TX_HAZARD_NO_BEGIN]	transaction_controlの特性がTX_CHAINEDの場合にのみ発生するエラーで、トランザクションが一部コミットもしくはロールバックされます。新規トランザクションは開始できず、呼び出し元はトランザクション・モードではなくなります
[TX_PROTOCOL_ERROR]	関数が不適切な状況で呼び出された場合です。たとえば、呼び出し元がトランザクション・モードでない場合に発生します。トランザクションに関連する呼び出し元の状態は変化がありません
[TX_FAIL]	致命的なエラーが発生した場合で、トランザクション・マネージャーやリソース・マネージャーはアプリケーションのための作業を実行できません。エラーの原因は製品の特性によって異なります。トランザクションに関連する呼び出し元の状態は認識できません

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tx.h>

void main(int argc, char *argv[])
{
    char *buf;
    int ret, cd;
    long len, revent;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf = (char *)tpalloc("CARRAY", NULL, 20);
    if (buf==NULL) {error processing };

    data process....
```

```

ret=tx_set_transaction_timeout( 5 );
if (ret<0) { error processing }

ret=tx_begin();
if (ret<0) { error processing }

cd = tpconnect("SERVICE", buf, 20, TPRECVONLY);
if (cd== -1) { error processing }

ret = tprecv(cd, (char **)&buf, &len, TPNOFLAGS, &revent));
if (ret < 0 && revent != TPEV_SVCSUCC)
    tx_rollback();
else
    tx_commit();

data process....
tpfree((char *)buf);
tpend();
}

```

- 関連関数

tx\_begin(), tx\_set\_commit\_return(), tx\_set\_transaction\_control(), tx\_set\_transaction\_timeout()

### 3.1.106. tx\_info

サーバーとクライアントでグローバル・トランザクション情報を返す関数です。infoが指す構造体を使用してグローバル・トランザクション情報を通知します。また、呼び出し元が現行トランザクション・モードであるかどうかを通知する値を返します。

- プロトタイプ

```

#include <tx.h>
int tx_info (TXINFO *info)

```

- パラメータ

infoがNULLでない場合、infoが指すTXINFO構造体はグローバル・トランザクションに関する情報になります。

TXINFO構造体は、以下のように構成されています。

```

struct TXINFO {
    XID                xid;
    COMMIT_RETURN      when_return;
}

```

```

TRANSACTION_CONTROL    transaction_control;
TRANSACTION_TIMEOUT     transaction_timeout;
TRANSACTION_STATE       transaction_state;
};

```

tx\_info()がトランザクション・モードで呼び出された場合、xidは現行トランザクションのbranch idになり、transaction\_stateは現行トランザクションの状態になります。呼び出し元がトランザクション・モードでない場合、xidはNULL XIDになります。(詳細内容は<tx.h>を参照してください。)

呼び出し元がトランザクション・モードであることとは関係なく、when\_return、transaction\_control、transaction\_timeoutはcommit\_returnの現在の設定とtransaction\_controlの特性、そして秒単位のトランザクション・タイムアウト値を含みます。

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の開始時から使用されます。現行トランザクションが開始後、tx\_set\_transaction\_timeout()の呼び出しによりトランザクション・タイムアウト値を変更することもあり得るため、呼び出し元の現行グローバル・トランザクションのタイムアウト値ではないこともあります。infoがNULLの場合、TXINFO構造体は返されません。

同じグローバル・トランザクション内での継続的なtx\_info()の呼び出しは、同じgtrid(グローバル・トランザクションの記述子)のXID提供を保証します。しかし、必ずしも同じbqual(ローカル・トランザクションの記述子)は保証しません。XIDは同一でないこともあります。

- 戻り値

戻り値	説明
1	呼び出し元がトランザクション・モードの場合、関数の呼び出しに成功した場合は
0	呼び出し元がトランザクション・モードでない場合、関数の呼び出しに成功した場合は
負数	関数の呼び出しに失敗した場合は。エラーコードを返します

- エラー

tx\_info()が正常に実行されなかった場合、以下のエラーコードを返します。

エラーコード	説明
[TX_PROTOCOL_ERROR]	関数が不適切な状況で呼び出された場合は。たとえば、呼び出し元がtx_open()を呼び出していない場合に発生します
[TX_FAIL]	致命的なエラーが発生し、トランザクション・マネージャーがアプリケーションのための作業を実行できない場合は。エラーの原因は製品の特性によって異なります

- 例



```

#include <stdio.h>
#include <string.h>
#include <usrinc/atmi.h>
#include <usrinc/tx.h>
void main(int argc, char *argv[])
{
    int ret;
    long len;
    char *buf;
    TXINFO info;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf = (char *)tpalloc("CARRAY", NULL, 20);
    if (buf==NULL) {error processing };
    data process....
    ret=tx_begin();
    if (ret<0) { error processing }

    ret=tpcall("SERVICE", buf, 20, (char **)&buf, &len, TPNOFLAGS);
    if (ret==-1) { error processing }

    if (tx_info(&info)==1) printf("In transaction \n");
    else printf("Not in transaction \n");

    if (strncmp(buf, "err", 3)==0) tx_rollback();
    else tx_commit();

    data process....
    tpfree((char *)buf);
    tpend();
}

```

- 関連関数

tx\_open(), tx\_set\_commit\_return(), tx\_set\_transaction\_control(), tx\_set\_transaction\_timeout()

### 3.1.107. tx\_rollback

サーバーとクライアントで使用され、グローバル・トランザクションをロールバックする関数です。

transaction\_controlの特性がTX\_UNCHAINEDの場合、tx\_rollback()が返す際に、呼び出し元はトランザクション・モードではなくなります。transaction\_controlの特性がTX\_CHAINEDの場合、tx\_rollback()が返す

際に、呼び出し元は新規トランザクションのためにトランザクション・モードのまま残ります。transaction\_controlの特性についての詳細は「[3.1.109. tx\\_set\\_transaction\\_control](#)」を参照してください。

- プロトタイプ

```
#include <tx.h>
int tx_rollback(void)
```

- 戻り値

戻り値	説明
TX_OK	関数の呼び出しに成功した場合です
負数	関数の呼び出しに失敗した場合です。エラーコードを返します

- エラー

tx\_rollback()が正常に実行されなかった場合、以下のエラーコードを返します。

エラーコード	説明
[TX_NO_BEGIN]	transaction_controlの特性がTX_CHAINEDの場合にのみ発生するエラーで、トランザクションがロールバックされます。  新規トランザクションは開始できず、呼び出し元はトランザクション・モードではなくなります
[TX_MIXED]	トランザクションが一部はコミットされ、一部はロールバックされます。transaction_controlの特性がTX_CHAINEDの場合、新規トランザクションを開始します
[TX_MIXED_NO_BEGIN]	transaction_controlの特性がTX_CHAINEDの場合にのみ発生するエラーで、トランザクションが一部はコミットされ、一部はロールバックされます。新規トランザクションは開始できず、呼び出し元はトランザクション・モードではなくなります
[TX_HAZARD]	エラーによって、トランザクションが一部はコミットされ、一部はロールバックされます。transaction_controlの特性がTX_CHAINEDの場合、新規トランザクションが開始されます
[TX_HAZARD_NO_BEGIN]	transaction_controlの特性がTX_CHAINEDの場合にのみ発生するエラーで、トランザクションが一部はコミットされ、一部はロールバックされます。新規トランザクションは開始できず、呼び出し元はトランザクション・モードではなくなります
[TX_COMMITTED]	トランザクションが独自にコミットされます。transaction_controlの特性がTX_CHAINEDの場合、新規トランザクションが開始されます

エラーコード	説明
[TX_COMMITTED_NO_BEGIN]	transaction_controlの特性がTX_CHAINEDの場合にのみ発生するエラーで、トランザクションが独自にコミットされます。新規トランザクションは開始できず、呼び出し元はトランザクション・モードではなくなります
[TX_PROTOCOL_ERROR]	関数が不適切な状況で呼び出された場合です。たとえば、呼び出し元がトランザクション・モードでない場合に発生します。トランザクションと関連する呼び出し元の状態は変化がありません
[TX_FAIL]	致命的なエラーが発生し、トランザクション・マネージャーやリソース・マネージャーがアプリケーションのための作業を実行できない場合です。エラーの原因は、製品の特性によって異なります。トランザクションと関連する呼び出し元の状態は認識できません

- 例

```

#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tx.h>

void main(int argc, char *argv[])
{
    char *buf;
    int ret,cd;
    long len, revent;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret== -1) { error processing }

    buf = (char *)tpalloc("CARRAY", NULL, 20);
    if (buf==NULL) {error processing };
    data process....
    ret=tx_set_transaction_timeout( 5 );
    if (ret<0) { error processing }

    ret=tx_begin();
    if (ret<0) { error processing }

    cd = tpconnect("SERVICE", buf, 20, TPRECVONLY);
    if (cd== -1) { error processing }
    ret = tprecv(cd, (char **)&buf, &len,TPNOFLAGS, &revent));
    if (ret < 0 && revent != TPEV_SVCSUCC) tx_rollback();
    else tx_commit();
    data process....

    tpfree((char *)buf);

```

```
    tpend();  
}
```

- 関連関数

tx\_begin(), tx\_set\_transaction\_control(), tx\_set\_transaction\_timeout()

### 3.1.108. tx\_set\_commit\_return

サーバーとクライアントでcommit\_returnの特性を設定する関数で、when\_return値で返します。

commit\_returnの特性は、tx\_commit()が関数の呼び出し元に制御権を返す方式を決定します。tx\_set\_commit\_return()は、関数の呼び出し元がトランザクション・モードであることは関係なく、呼び出しが可能です。設定は、tx\_set\_commit\_return()の再呼び出しによって変更されるまで有効です。commit\_returnの特性の初期設定は実行によって異なります。

- プロトタイプ

```
#include <tx.h>  
int tx_set_commit_return (COMMIT_RETURN when_return)
```

- パラメータ

when\_returnで使用可能な値は以下のとおりです。

設定値	説明
TX_COMMIT_DECISION_LOGGED	tx_commit()が、2PC(2 Phase Commit)プロトコルのうち第1段階でロギングされた後、第2段階は完了する前に返します。tx_commit()が呼び出し元により速く応答できますが、トランザクションが独自(heuristic)の結果をもつ危険があります。この場合、呼び出し元はtx_commit()から返されたコードで発生した状況を正確に認識できません。  正常な場合、第1段階でトランザクションをコミットすることにしたトランザクション参加者は、第2段階で正常にコミットされます。しかし、ネットワークやノードの障害が長時間続くなどの異常がある場合、2つの段階の完了ができないこともあり、独自の結果を招くこともあります。トランザクション・マネージャーは、オプションでこの特性をサポートしないよう選択できます。この際、この値をサポートしていないことを表す[TX_NOT_SUPPORTED]値で返します
TX_COMMIT_COMPLETED	2PCプロトコルが完全に終了後に、tx_commit()が返されます。この設定は、トランザクションが独自(heuristic)の結果をもつことになったか、あるいはその可能性を通知するリターン・コードを、tx_commit()の呼び出し元に渡します。トランザクション・マネージャーは、この特性をサポートし

設定値	説明
	ないようオプションで選択でき、この値をサポートしないことを表す [TX_NOT_SUPPORTED]値で返します

#### ● 戻り値

正常に作業が完了した場合、tx\_set\_commit\_return()は負ではない値の[TX\_OK]を返します。

when\_returnが、TX\_COMMIT\_COMPLETEDまたはTX\_COMMIT\_DECISION\_LOGGEDで設定されていない場合、関数は負ではない値で[TX\_NOT\_SUPPORTED]を返し、commit\_returnの特性は現在適用されている値が有効です。トランザクション・マネージャーは、when\_returnを少なくともTX\_COMMIT\_COMPLETEDかTX\_COMMIT\_DECISION\_LOGGEDのうちの1つに設定する必要があります。

関数の呼び出しに失敗した場合、エラーコードを返します。

#### ● エラー

tx\_set\_commit\_return()は、commit\_returnの特性設定の変更なく、負数値のうち1つを返します。

エラーコード	説明
[TX_EINVAL]	when_returnが、TX_COMMIT_COMPLETEDあるいはTX_COMMIT_DECISION_LOGGEDで設定されていない場合です
[TX_PROTOCOL_ERROR]	関数が不適切な状況で呼び出された場合です。たとえば、関数の呼び出し元がまだtx_open()を呼び出していない場合に発生します
[TX_FAIL]	致命的なエラーが発生し、トランザクション・マネージャーがアプリケーションのための作業を実行できない場合です。エラーの原因は、製品の特性によって異なります

#### ● 例

```
#include <usrinc/tx.h>

int main(int argc, char *argv[])
{
    int ret;
    long len;
    char *buf;

    ret = tpstart((TPSTART_T *)NULL);
    if (ret == -1) { error processing }

    buf = (char *)tpalloc("CARRAY", NULL, 20);
    if (buf==NULL) {error processing };
    ret = tx_set_transaction_timeout(5);
```

```

    if (ret < 0){ error processing }

    ret = tx_set_commit_return(TX_COMMIT_COMPLETED);
    if (ret < 0){ error processing }

    ret = tx_begin();
    if (ret < 0){ error processing }

    data process....
    ret = tpcall("SERVICE1", (char *)buf, strlen(buf), (char **)&buf, &len,
                TPNOFLAGS);
    if (ret == -1)
    {
        tx_rollback();
        error processing
    }

    data process...
    ret = tpcall("SERVICE2", (char *)buf, strlen(buf), (char **)&buf, &len,
                TPNOFLAGS);
    if (ret == -1)
    {
        tx_rollback();
        error processing
    }

    data process ....

    ret = tx_commit();
    if (ret < 0) { error processing }

    data process...

    tpfree((char *)buf);
    tpend();
}

```

- 関連関数

tx\_commit(), tx\_open(), tx\_info()

### 3.1.109. tx\_set\_transaction\_control

サーバーとクライアントでtransaction\_controlの特性をcontrol値に設定する関数です。

transaction\_controlの特性は、**tx\_commit()**と**tx\_rollback()**が呼び出し元に返す前に、新規トランザクションを開始するかどうかを決定します。tx\_set\_transaction\_control()は、アプリケーションがトランザクション・モードかどうかとは関係なく、呼び出しが可能です。この設定は、tx\_set\_transaction\_control()の再呼び出しによって変更されるまで有効です。

- プロトタイプ

```
# include <tx.h>
int tx_set_transaction_control(TRANSACTION_CONTROL control)
```

- パラメータ

controlで使用可能な値は以下のとおりです。

設定値	説明
TX_UNCHAINED	tx_commit()とtx_rollback()が関数の呼び出し元に返す前に、新規トランザクションを開始しないようにします。この場合、呼び出し元は、新規トランザクションを開始するためには、tx_begin()を実行する必要があります。transaction_control特性の初期設定値です
TX_CHAINED	tx_commit()とtx_rollback()が呼び出し元に返す前に新規トランザクションを開始します

- 戻り値

戻り値	説明
TX_OK	関数の呼び出しに成功した場合です
負数	関数の呼び出しに失敗した場合です。エラーコードを返します

- エラー

tx\_set\_transaction\_control()は、既存のtransaction\_controlの特性の変更なく、以下のエラーコードを返します。

エラーコード	説明
[TX_EINVAL]	controlのパラメータがTX_UNCHAINEDあるいはTX_CHAINEDと設定されていない場合です
[TX_PROTOCOL_ERROR]	関数が不適切な状況で呼び出された場合です。たとえば、関数の呼び出し元がまだtx_open()を呼び出していない場合に発生します
[TX_FAIL]	致命的なエラーが発生し、トランザクション・マネージャーがアプリケーションのための作業を実行できない場合です。エラーの原因は、製品の特性によって異なります

- 例

```

#include <usrinc/atmi.h>
#include <usrinc/tx.h>

int main(int argc, char *argv[])
{
    int ret;
    long len;
    char *buf;

    ret = tpstart((TPSTART_T *)NULL);
    if (ret == -1) { error processing }

    buf = (char *)tpalloc("CARRAY", NULL, 20);
    if (buf==NULL) {error processing };
    ret = tx_set_transaction_timeout(5);
    if (ret < 0){ error processing }

    ret = tx_set_transaciont_control(TX_UNCHAINED);
    if (ret < 0){ error processing }
    ret = tx_begin();
    if (ret < 0) { error processing }
    data process....

    ret = tpcall("SERVICE1", (char *)buf, strlen(buf), (char **)&buf, &len,
                TPNOFLAGS);
    if (ret == -1)
    {
        tx_rollback();
        error processing
    }

    data process...
    ret = tpcall("SERVICE2", (char *)buf, strlen(buf), (char **)&buf, &len,
                TPNOFLAGS);
    if (ret == -1)
    {
        tx_rollback();
        error processing
    }
    data process ....

    ret = tx_commit();
    if (ret < 0) { error processing }
    data process...

    tpfree((char *)buf);

```



```
    tpend();
}
```

- 関連関数

tx\_begin(), tx\_commit(), tx\_open(), tx\_rollback(), tx\_info()

### 3.1.110. tx\_set\_transaction\_timeout

サーバーとクライアントでtransaction\_timeoutの特性をtimeout値に設定します。設定された値は、トランザクション・タイムアウトの発生前にトランザクションを完了しなければならない時間であり、**tx\_begin()**と**tx\_commit()**、または**tx\_begin()**と**tx\_rollback()**の間の時間です。

tx\_set\_transaction\_timeout()は、関数の呼び出し元がトランザクション・モードかどうかとは関係なく、呼び出しが可能です。tx\_set\_transaction\_timeout()がトランザクション・モードで呼び出された場合、新規タイムアウト値は次のトランザクションから適用されます。

- プロトタイプ

```
# include <tx.h>
int tx_set_transaction_timeout (TRANSACTION_TIMEOUT timeout)
```

- パラメータ

パラメータ	説明
timeout	トランザクション・タイムアウトの発生までに許容された時間を秒単位の数字で指定します。設定値はシステム別に定義されたlong型の最大値まで設定可能です。初期値は0に設定され、タイムアウト制限がないことを意味します

- 戻り値

戻り値	説明
TX_OK	関数の呼び出しに成功した場合です
負数	関数の呼び出しに失敗した場合です。エラーコードを返します

- エラー

tx\_set\_transaction\_timeout()は、既存のtransaction\_timeout値の変更なく、以下のエラーコードを返します。

エラーコード	説明
[TX_EINVAL]	指定されたタイムアウト値が有効でない場合です
[TX_PROTOCOL_ERROR]	関数が不適切な状況で呼び出された場合です。たとえば、呼び出し元がまだtx_open()を呼び出していない場合に発生します
[TX_FAIL]	致命的なエラーが発生し、トランザクション・マネージャーがアプリケーションのための作業を実行できない場合です。エラーの原因は、製品の特性によって異なります

- 例

```
#include <usrinc/atmi.h>
#include <usrinc/tx.h>

void main(int argc, char *argv[])
{
    int ret;
    long len;
    char *buf;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf = (char *)tpalloc("STRING", NULL, 0);
    if (buf==NULL) {error processing };

    ret=tx_set_transaction_timeout( 5 );
    if (ret<0) { error processing }

    ret=tx_begin();
    if (ret<0) { error processing }
    ret = tpcall("SERVICE", (char *)buf, strlen(buf), (char **)&buf, &len,
                TPNOFLAGS);
    if (ret == -1) {
        tx_rollback();
        error processing
    }
    data process ....
    ret = tx_commit();
    if (ret < 0) { error processing }

    data process...
    tpfree((char *)buf);
    tpend();
}
```

- 関連関数

tx\_begin(), tx\_commit(), tx\_open(), tx\_rollback(), tx\_info()

### 3.1.111. ulogsync

**userlog()**を使用し、「ulog」の内容を、ディスクのメモリー・バッファにある<ulog.dat>ファイルに保存する関数です。

- プロトタイプ

```
# include <userlog.h>
int ulogsync(void)
```

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。エラーが発生してもtperrnoに保存される値はありません

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/userlog.h>

void main(int argc, char *argv[])
{
    int ret, cd;
    long len;
    char *buf;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf = (char *)tpalloc("CARRAY", NULL, 20);
    if (buf==NULL) {error processing };

    data process....

    cd = tpacall("SERVICE", (char **)buf, 20, TPNOFLAGS);
    if (cd==-1) { error processing }
```

```

ret=tpgetrply(&cd, &buf, &len, TPNOTIME);
if (ret==-1){ error processing }

ret=userlog(" error : %s\n", tpstrerror(errno));
if (ret==-1) { error processing }
tpfree((char *)buf);

ret=ulogsync();
if (ret==-1) { error processing }

tpend();
}

```

### 3.1.112. userlog

プログラムに必要な情報を<ulog.日付>ファイルに作成する関数です。プログラムの論理エラーを探すために、プログラムの中間にユーザーが必要な情報を保存する際に使用します。

<ulog.日付>ファイルの位置は、設定ファイルのULOGDIR項目に定義したディレクトリーに作成されます。サーバーとクライアント共にこの関数を使用でき、すべて同一ファイルに保存されます。

userlog()関数が呼び出されても、データをファイルに即時保存せず、バッファーに一定量が保存されなければ、実際にファイルに保存されません。ファイルに即時保存するためには、**ulogsync()**を使用します。userlog()関数とUserLog()関数は、C言語で提供しているprintf()関数と同一のフォーマットを使用するため、escape sequence characterをそのまま使用できます。

---

#### 注

userlog()を使用した場合、プログラムで最初のuserlog()関数の呼び出し時、ULOGDIRディレクトリーにログ・ファイルが生成され、それ以降は変わらないことに注意します。

最初に呼び出されたuserlog()によって、この関数がどのディレクトリーにログを残すかが決定されます。したがって、userlog()を1回でも呼び出した後は、クライアントの環境ファイル(ULOGDIR)を変更しても、ファイルの新規作成や位置変更はされません。

---

#### ● プロトタイプ

```

# include <userlog.h>
int userlog(const char *fmt, ...)

```

#### ● パラメータ

パラメータ	説明
fmt	ユーザーが残すログ文字列です

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。エラーが発生してもtperrnoに保存される値はありません

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/userlog.h>

void main(int argc, char *argv[])
{
    int ret, cd;
    long len;
    char *buf;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf = (char *)tpalloc("CARRAY", NULL, 20);
    if (buf==NULL) {error processing };
    data process....

    cd = tpacall("SERVICE", (char **)buf, 20, TPNOFLAGS);
    if (cd==-1) { error processing }

    ret=tpgetrply(&cd, &buf, &len, TPNOTIME);
    if (ret==-1){

    ret=userlog("error no: %d, urcode : %d", tperrno, tpurcode");
    if (ret==-1) { error processing }

    data process..
    tpfree((char *)buf);
    tpend();
}
```

## 3.1.113. UserLog

サーバーとクライアントで使用され、メモリー・バッファーを使用せず、ulogファイルを直接作成する関数です。UserLog()は、呼び出した際に即時データを保存する**ulogsync()**を含みます。用途はuserlog()関数と同じです。バッファー・サイズは8KBに制限されます。

- プロトタイプ

```
# include <userlog.h>
int UserLog (const char *fmt, ...)
```

- パラメータ

パラメータ	説明
fmt	ユーザーが残すログ文字列です

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。エラーが発生してもtperrnoに保存される値はありません

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/userlog.h>

void main(int argc, char *argv[])
{
    int ret;
    long len;
    char *buf, *input;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf = (char *)tpalloc("CARRAY", NULL, 20);
    if (buf==NULL) {error processing };
    data process....

    strcpy(input, buf);
    ret=tpcall("SERVICE", buf, 20, (char **)&buf, &len, TPNOFLAGS);
    if (ret==-1) { error processing }
```

```

data process ...

UserLog("input %s -> output : %s\n",input, buf);

tpfree((char *)buf);
tpend();
}

```

## 3.2. サーバー関数

### 3.2.1. \_tmax\_check\_license

AnyLinkおよびOpenFrameで該当ライセンスが発行されたかどうかをチェックできる関数です。

- プロトタイプ

```
int _tmax_check_license(char *tmaxdir, int subprod);
```

- パラメータ

パラメータ	説明
tmaxdir	現在Tmaxシステムがインストールされているホーム・ディレクトリー(\$TMAXDIR)を設定します
subprod	AnyLinkとOpenFrameのうち、どちらの製品をチェックするかを設定します

以下は、subprodに使用する定数についての定義です。

```

#define SUB_PROD_ANYLINK 0x00200000
#define SUB_PROD_OPENFRAME 0x00400000

```

- 戻り値

戻り値	説明
1	該当ライセンスが発行された状態です
-1	該当ライセンスが発行されていない状態です

## 3.2.2. \_tmax\_event\_handler

SVRTYPEがEVT\_SVRの場合、SLOGの発生時に呼び出されるコールバック関数です。SLOGの発生時、より詳しい情報を照会する場合に使用します。SLOGイベントが発生した場合、該当関数が呼び出され、該当関数のパラメータに詳細情報が入れられます。SVRTYPEがEVT\_SVRのサーバー・プログラム内に定義でき、1つのノードあたり1つのEVT\_SVRタイプのサーバーを定義します。

該当関数が呼び出される頻度は、環境ファイルのNODEセクションのTMMOPTに[-h]オプションを使用し、イベント・ハンドラのログ・レベルを設定します。NODEセクションのTMMOPTの[-h]オプションについての詳細は、『Tmax 運用ガイド』を参照してください。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
int _tmax_event_handler(char *programe, int pid, int tid, char *msg, int flags);
```

- パラメータ

パラメータ	説明
programe	イベントを発生させたプログラム名です
pid	プロセスIDです
tid	スレッドIDです(予備用)
msg	イベント・メッセージです
flags	予備用です。現行バージョンではサポートしていません

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です

- 例

<環境ファイル>

```
*DOMAIN
tmax1      SHMKEY = 78650, MINCLH = 1, MAXCLH = 3,
           TPORTNO = 8650, BLOCKTIME = 30

*NODE
Tmaxh4     TMAXDIR = "/data1/starbj81/tmax",
           APPDIR  = "/data1/starbj81/tmax/appbin",
           PATHDIR = "/data1/starbj81/tmax/path",
```



```

        TLOGDIR = "/data1/starbj81/tmax/log/tlog",
        ULOGDIR = "/data1/starbj81/tmax/log/ulog",
        SLOGDIR = "/data1/starbj81/tmax/log/slog",
        TMMOPT = "-h i", SMSUPPORT = Y, SMTBLSIZE = 1000

*SVRGROUP
svg1          NODENAME = "@HOSTNAME@"

*SERVER
evtsvr        SVGNAME = svg1, SVRTYPE = EVT_SVR

```

### <サーバープログラム>

```

#include <stdio.h>
#include <stdlib.h>
#include <usrinc/atmi.h>
#include <time.h>

int tpsvrinit(int argc, char *argv[])
{
    printf("[EVTHND] started\n");
    return 1;
}

int svrdone()
{
    printf("[EVTHND] stopped\n");
    return 1;
}

int _tmax_event_handler(char *program, int pid, int tid, char *msg, int flags)
{
    time_t t1;
    struct tm *tm;

    time(&t1);
    tm = localtime(&t1);
    printf("[EVTHND] %s.%d.%02d%02d%02d:%s\n", program, pid, tm->tm_hour,
        tm->tm_min, tm->tm_sec, msg);
    return 0;
}

```

### <Makefile.evt>

```

# Server makefile
TARGET = $(COMP_TARGET)
APOBJS = $(TARGET).o

```

```

NSDLOBJ = $(TMAXDIR)/lib64/sdl.o

LIBS     = -lsvrevt -lnodb
OBJJS    = $(APOBJS) $(SVCTOBJ)
SVCTOBJ  = $(TARGET)_svctab.o
CFLAGS   = -Ae +DA2.0W +DD64 +DS2.0 -O -I$(TMAXDIR)

APPDIR   = $(TMAXDIR)/appbin
SVCTDIR  = $(TMAXDIR)/svct
LIBDIR   = $(TMAXDIR)/lib64

#
.SUFFIXES : .c

.c.o:
    $(CC) $(CFLAGS) -c $<

#
# server compile
#

$(TARGET): $(OBJJS)
    $(CC) $(CFLAGS) -L$(LIBDIR) -o $(TARGET) $(OBJJS) $(LIBS) $(NSDLOBJ)
    mv $(TARGET) $(APPDIR)/.
    rm -f $(OBJJS)

$(APOBJS): $(TARGET).c
    $(CC) $(CFLAGS) -c $(TARGET).c

$(SVCTOBJ):
    cp -f $(SVCTDIR)/$(TARGET)_svctab.c .
    touch ./$(TARGET)_svctab.c
    $(CC) $(CFLAGS) -c ./$(TARGET)_svctab.c

#
clean:
    •rm -f *.o core $(APPDIR)/$(TARGET)

```

### 3.2.3. \_tmax\_main

サーバーで使用される関数で、ユーザー・プログラムにmain()が含まれている場合に使用します。サーバー・プロセスの作成時、ユーザー・プログラムにはmain()が含まれてはいけません。ユーザーが作成したライブラリー内にやむを得ずmain()が含まれており、これを修正できない場合に、Tmaxサーバー・ライブラリー内のmain()ルーチン呼び出して処理できるように提供する関数です。

この関数を使用するには、サーバー・プログラム以外に<\*\*\*.c>ファイルを別途1つ作成し、このファイル内にmain()関数を作成します。この関数内で\_tmax\_main()を呼び出した場合、ユーザーが作成したライブラリーにあるmain()が呼び出されず、Tmaxサーバー・ライブラリーにあるmain()関数が呼び出されます。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
int _tmax_main(int argc, char *argv[])
```

- パラメータ

main()関数と同一です。

パラメータ	説明
argc	argvの配列数です
argv	ユーザーがCLOPTセクションに設定したパラメータの一覧です

- 戻り値

整数値で、main()関数と同じです。

- 例

以下は、関数の使用プロセスについての説明です。

1. ユーザーは、以下のようにライブラリーを作成します。

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    printf("my lib main called.\n");
}
```

2. 関数をコンパイル後、<libmysvr.so>ファイルを作成します。

ユーザーが作成したライブラリーのmain()を使用せず、Tmaxサーバー・ライブラリー内のmain()を使用するために<mysvr.c>ファイルを作成し、ファイル内にmain()を作成します。

main()で\_tmax\_main()を呼び出します。現在main()は、ユーザーが作成したライブラリー、Tmaxサーバー・ライブラリー、<mysvr.c>の3つのmain()関数が存在します。しかしTmaxシステムを起動すると、最初にサーバー・プログラムと一緒にコンパイルされた<mysvr.c>のmain()関数が呼び出されます。そのため、関数内で呼び出されたTmaxサーバー・ライブラリー内のmain()関数が呼び出され、ユーザーが作成したサーバー・ライブラリー内のmain()は呼び出されません。

```
#include <stdio.h>
```

```
int main(int argc, char *argv[])
{
    printf("My main is called\n");
    _tmax_main(argc, (char **)argv);
    return 0;
}
```

- 作成された<\*\*\*.c>ファイルをコンパイルし、オブジェクト・ファイルを作成後、サーバー・プログラムと一緒にリンクさせ、1つの実行ファイルを作成します。

### 3.2.4. tmaxadmin

Tmaxシステムの管理ツールのtmaxadminで照会できる統計情報を出力する関数です。プロセス内部で動的に使用が可能です。サーバー・プロセス内部でのみ使用可能な関数です。

- プロトタイプ

```
# include <tmaxadmin.h>
int tmaxadmin(int cmd, void *arg, int opt, long flags);
```

- パラメータ

パラメータ	説明
cmd	以下の項目が設定可能です  TMADM_DISCON, TMADM_CLIINFO, TMADM_QPURGE, TMADM_BOOT, TMADM_DOWN, TMADM_SUSPEND, TMADM_RESUME, TMADM_RESTAT, TMADM_SVC_STAT, TMADM_SPR_STAT, TMADM_SVR_STAT, TMADM_SVC_STAT_EX, TMADM_SPR_STAT_EX, TMADM_TMAX_INFO, TMADM_DOMAIN_CONF, TMADM_NODE_CONF, TMADM_SVG_CONF, TMADM_SVR_CONF, TMADM_SVC_CONF, TMADM_SMTRC
arg	大部分が、各cmdに合う構造体のポインターを指定する必要があります。データ・バッファのサイズとoffsetなどを指定できます
opt	各cmd別にオプション・フラグが設定できます。詳細については、下の表を参照してください
flags	予備用です。現在は使用していません

以下は、cmd別にoptに設定できるオプション・フラグの一覧です。

– TMADM\_DISCON

オプション・フラグ	説明
TPNOFLAGS	クライアントがサービスを要求中は終了しません
TMADM_FFLAG	クライアントがサービスを要求中に終了します

– TMADM\_CLIINFO

オプション・フラグ	説明
TPNOFLAGS	該当ノードのすべてのクライアント情報を渡します
TMADM_SFLAG	該当ノードのすべてのクライアント数を渡します (arg→header.num_left)

– TMADM\_QPURGE

オプション・フラグ	説明
TMADM_SFLAG	サービス・キューをパージします
TMADM_VFLAG	サーバー・キューをパージします

– TMADM\_BOOT(TMADM\_DOWN)

オプション・フラグ	説明
TMADM_SFLAG	arg→args.name1をサーバー名として使用し、該当サーバーを起動/終了します
TMADM_GFLAG	arg→args.name2をサーバー・グループ名として使用し、該当サーバー・グループのサーバーを起動/終了します
TMADM_TFLAG	arg→args.name1をサーバー・グループ名として使用し、該当サーバー・グループのTMSを起動/終了します
TMADM_USFLAG	tmboot/tmdownの -Sオプションと同じです。struct tmadm_bootのargs.countに値を指定しません。TMADM_USFLAGとTMADM_GFLAGを一緒に指定する場合、当該サーバー・グループのサーバーだけ適用されます

以下は、tmadmin()のTMADM\_BOOT、TMADM\_DOWNで使用可能なフラグの組み合わせです。

- 1) TMADM\_SFLAG : -sオプションと同じで、countを指定する必要があります。
- 2) TMADM\_SFLAG | TMADM\_GFLAG : -s -gオプションと同じで、countを指定する必要があります。
- 3) TMADM\_TFLAG : -tオプションと同じで、countを指定する必要があります。
- 4) TMADM\_USFLAG : -Sオプションと同じです。
- 5) TMADM\_USFLAG | TMADM\_GFLAG : -S -gオプションと同じです。

6) 上記のすべての場合 | TMADM\_FFLAG : -iオプションと同じで、TMADM\_DOWNでのみ使用できます。

– TMADM\_SUSPEND(TMADM\_RESUME)

オプション・フラグ	説明
TMADM_SFLAG	サービスを一時停止(suspend)/再開(resume)します
TMADM_VFLAG	サーバーを一時停止(suspend)/再開(resume)します

– TMADM\_RESTAT

オプション・フラグ	説明
TMADM_VFLAG	該当サーバーの統計情報を初期化します
TMADM_AFLAG	すべてのサーバーの統計情報を初期化します

– TMADM\_SVC\_STAT (TMADM\_SPR\_STAT, TMADM\_SVR\_STAT)

オプション	説明
TPNOFLAGS	該当ノードのすべての統計値を渡します
TMADM_SFLAG	arg→header.opt_charに指定された名前に一致する統計値のみ渡します
TMADM_CFLAG	arg→header.opt_intに指定されたCLH番号に該当するCLHの統計値のみ渡します

– TMADM\_SVC\_STAT\_EX (TMADM\_SPR\_STAT\_EX)

オプション	説明
TPNOFLAGS	該当ノードのすべての統計値を渡します。TMADM_SVC_STATの項目にfail_count、error_count、mintime、maxtime項目が追加されました。tmadminでstatコマンドの-xオプションと同じ機能を提供します
TMADM_SFLAG	arg→header.opt_charに指定された名前に一致する統計値のみ渡します
TMADM_CFLAG	arg→header.opt_intに指定されたCLH番号に該当するCLHの統計値のみ渡します

– TMADM\_SMTRC

オプション・フラグ	説明
TPNOFLAGS	tmadm_smtrc構造体を使用します
TMADM_AFLAG	tmadm_smtrcall構造体を使用します。既存情報以外に、spri、reserved、curtime、svctime、ucputime、scputimeなどの情報を追加提供します

- TMADM\_TMAX\_INFO, TMADM\_DOMAIN\_CONF, TMADM\_NODE\_CONF, TMADM\_SVG\_CONF, TMADM\_SVR\_CONF, TMADM\_SVC\_CONF

オプション・フラグ	説明
TPNOFLAGS	TPNOFLAGSのみ使用可能です

- 戻り値

戻り値	説明
0より大きい数	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tadmin()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、NULL形式の場合に発生します。  2番目のパラメータである構造体のうち、tadmin_headerのメンバー変数に入れられる値のsizeとoffset、opt_charが正しくない場合に発生します
[TPENOREADY]	TMADM_SVC_STAT/SPR_STAT/SVR_STATコマンドでTMADM_CFLAGを使用した場合、該当CLHがNOT READY状態の場合に発生します。  TMADM_DISCONコマンドで該当クライアントが存在しない場合や、サービス要求中の状態でありながらTMADM_FFLAGを指定していない場合に発生します
[TPEOS]	運用システムにエラーが発生した場合です。内部バッファの割り当てエラーの場合が多いです
[TPESYSTEM]	Tmaxシステムのエラーです。通信障害の場合が多いです
[TPESVCFAIL]	TMADM_QPURGE、TMADM_BOOT/DOWN、TMADM_SUSPEND/RESUMEコマンドが、TPEINVAL、TPEOS、TPESYSTEM以外のエラーで失敗した場合に発生します

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tadmin.h>

SERVICE(TPSVCINFO *msg)
{
```

```

char cmd, *buf, *sndbuf;
int len;
long sndlen;

cmd = msg->data[0];
switch (cmd)
{
    case TMADM_TMAX_INFO:
        len = tmaxinfo(&buf);
        break;
    case TMADM_SVC_STAT:
        len = svcstat(&buf);
        break;
    case TMADM_SPR_STAT:
        len = sprstat(&buf);
        break;
    case TMADM_QPURGE:
        len = qpurge(&buf);
        break;
    default:
        len = -1;
        break;
}
if (len < 0){ error processing }
sndbuf = (char *)tpalloc("STRING", NULL, len + 1);
if (sndbuf==NULL) {error processing }
memcpy(sndbuf, buf, len);
sndlen = len;
tpreturn(TPSUCCESS, 0, sndbuf, sndlen, TPNOFLAGS);
}

int qpurge(char **buf)
{
    int n;
    /*n=tmadmin( TMADM_QPURGE, "TOUPPER",
    TMADM_SFLAG, TPNOFLAGS);*/
    n = tmadmin(TMADM_QPURGE, "toupper", TMADM_VFLAG, TPNOFLAGS);
    if (n < 0){ error processing }
    ...
}

int tmaxinfo(char **buf)
{
    struct tmadm_tmax_info *info;
    ...
    size = sizeof(struct tmadm_tmax_info) +
    (MAX_NODE - 1) * sizeof (struct tmadm_node_summary);

```



```

    info = (struct tmadm_tmax_info *) malloc(size);
    if (info == NULL){ error processing }
    memset(info, 0x00, size);
    info->header.version = _TMADMIN_VERSION;
    info->header.size = size;

    n = tmadmin(TMADM_TMAX_INFO, info, TPNOFLAGS, TPNOFLAGS);
    if (n < 0){ error processing }
    ...

    /*構造体infoの内容を整理し、*bufに入れます。
       *bufのサイズを返します。*/
}

int svcstat(char **buf)
{
    struct tmadm_svc_stat info;
    struct tmadm_svc_stat *stat;
    ...
    memset(&info, 0x00, sizeof(struct tmadm_svc_stat));
    info.header.version = _TMADMIN_VERSION;
    info.header.size = sizeof(struct tmadm_svc_stat);
    n = tmadmin(TMADM_SVC_STAT, &info, TPNOFLAGS, TPNOFLAGS);
    if (n < 0){ error processing }

    svccount = info.header.num_entry + info.header.num_left;
    size = sizeof(struct tmadm_svc_stat)+svccount - 1}
    sizeof (struct tmadm_svc_stat_body);
    stat = (struct tmadm_svc_stat *) malloc(size);
    if (stat == NULL){ error processing }

    memset(stat, 0x00, size);
    stat->header.version = _TMADMIN_VERSION;
    stat->header.size = size;

    n = tmadmin(TMADM_SVC_STAT, stat, TPNOFLAGS, TPNOFLAGS);
    if (n < 0){ error processing }
    ...

    /*構造体statの内容を整理し、*bufに入れます。
       *bufのサイズを返します。*/
}

int sprstat(char **buf)
{
    struct tmadm_spr_stat info;
    struct tmadm_spr_stat *stat;

```

```

...
memset(&info, 0x00, sizeof(struct tmadm_spr_stat));
info.header.version = _TMADMIN_VERSION;
info.header.size = sizeof(struct tmadm_spr_stat);
n = tmadmin(TMADM_SPR_STAT, &info, TPNOFLAGS, TPNOFLAGS);
if (n < 0){ error processing }

sprcount = info.header.num_entry + info.header.num_left;
size = sizeof(struct tmadm_spr_stat) + (sprcount - 1) *
        sizeof (struct tmadm_spr_stat_body);
stat = (struct tmadm_spr_stat *) malloc(size);
if (stat == NULL){ error processing }

memset(stat, 0x00, size);
stat->header.version = _TMADMIN_VERSION;
stat->header.size = size;

n = tmadmin(TMADM_SPR_STAT, stat, TPNOFLAGS, TPNOFLAGS);
if (n < 0)){ error processing }
...

/*構造体statの内容を整理し、*bufに入れます。
   *bufのサイズを返します。*/
}

```

### 3.2.5. tmax\_get\_db\_passwd

現在Tmaxが接続しているデータベースのusernameのパスワードを照会する関数です。

- プロトタイプ

```

#include <usrinc/tmaxapi.h>
int tmax_get_db_passwd (char *svgrname, char *passwd, int type)

```

- パラメータ

パラメータ	説明
svgrname	パスワードを照会するサーバー・グループの名前を指定します。サーバー・グループは、XAサーバー・グループである必要があります
passwd	結果情報が保存され、返されます。返される情報を保存できるだけの十分なサイズを割り当てる必要があります
type	現在使用しているDBMSを区別するために指定します。  以下のいずれかを指定する必要があります

パラメータ	説明
	<ul style="list-style-type: none"> <li>– ORACLE_TYPE</li> <li>– SYBASE_TYPE</li> <li>– INFORMIX_TYPE</li> <li>– DB2_TYPE</li> </ul>

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tmax\_get\_db\_passwd()が正常に実行されなかった場合、tpernoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、サーバー・グループ名の指定が正しくない場合や、typeをusrinc/tmaxapi.hに定義された値以外の値で定義した場合、あるいはtypeにINFORMIX_TYPEを設定した場合に発生します
[TPEITYPE]	svgnameにNONXAサーバー・グループを指定した場合や、Tmax環境ファイルのOPENINFO情報が正しくない場合にエラーが発生します

- 例

#### <環境ファイルのOPENINFO>

```
svg4      NODENAME = @HOSTNAME@, DBNAME = ORACLE,
OPENINFO = "ORACLE_XA+Acc=P/scott/tiger+SesTm=60+Sqlnet=tmaxil",
TMSNAME = tms_ora
```

#### <サーバー・プログラム>

```
#include <stdio.h>
...
SVC_GETPASSWORD( TPSVCINFO *msg )
{
    char passwd[30];
    printf("\n    => use right..\n");
    ret = tmax_get_db_passwd("svg1", passwd, ORACLE_TYPE);
    if(ret < 0)
```

```

        printf("tmax_get_db_passwd fail[%s]\n",tpstrerror(tperrno));
    else
        printf("\ntmax_get_db_passwd = %s\n", passwd);
    tpreturn( TPSUCCESS, 0, rcvbuf, 0, 0 );
}

```

<結果>

```
tmax_get_db_passwd = tiger
```

- 関連関数

tmax\_get\_db\_username(), tmax\_get\_db\_tnsname()

## 3.2.6. tmax\_get\_db\_tnsname

現在Tmaxが接続しているデータベースのtnsnameを照会する関数です。

- プロトタイプ

```

#include <usrinc/tmaxapi.h>
int tmax_get_db_tnsname (char *svgname, char *tnsname, int type)

```

- パラメータ

パラメータ	説明
svgname	tnsnameを確認するサーバー・グループの名前を指定します。サーバー・グループはXAサーバー・グループである必要があります
tnsname	結果情報が保存され、返されます。返される情報を保存できるだけの十分なサイズを割り当てる必要があります
type	現在使用しているDBMSを区別するための値です。  以下のいずれかを指定します <ul style="list-style-type: none"> <li>– ORACLE_TYPE</li> <li>– SYBASE_TYPE</li> <li>– INFORMIX_TYPE</li> <li>– DB2_TYPE</li> </ul>

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tmax\_get\_db\_tnsname()が正常に実行されなかった場合、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。サーバー・グループ名の指定が正しくない場合や、typeをusrinc/tmaxapi.hに定義された値以外の値で定義した場合、あるいはtypeにINFORMIX_TYPEを設定した場合に発生します
[TPEITYPE]	svgnamにNON-XAサーバー・グループを指定した場合や、Tmax環境ファイルのOPENINFO情報が正しくない場合にエラーが発生します

- 例

<環境ファイルのOPENINFO>

```
svg4      NODENAME = @HOSTNAME@, DBNAME = ORACLE,
          OPENINFO = ORACLE_XA+Acc=P/scott/tiger+SesTm=60+Sqlnet=tmaxil",
          TMSNAME = tms_ora
```

<サーバー・プログラム>

```
#include <stdio.h>
...
SVC_GETTNSNAME( TPSVCINFO *msg )
{
    char tnsname[30];
    printf("\n    => use right..\n");
    ret = tmax_get_db_tnsname("svg1", tnsname, ORACLE_TYPE);
    if(ret < 0)
        printf("tmax_get_db_tnsname fail[%s]\n",tpstrerror(tperrno));
    else
        printf("\ntmax_get_db_tnsname = %s\n", passwd);
    tpreturn( TPSUCCESS, 0, rcvbuf, 0, 0 );
}
```

<結果>

```
tmax_get_db_tnsname = tmaxil
```

- 関連関数

tmax\_get\_db\_username(), tmax\_get\_db\_tnsname()

### 3.2.7. tmax\_get\_db\_username

現在Tmaxが接続しているデータベースのusernameを照会する関数です。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_get_db_username (char *svgname, char *username, int type)
```

- パラメータ

パラメータ	説明
svgname	usernameを確認するサーバー・グループの名前を指定します。サーバー・グループはXAサーバー・グループである必要があります
username	結果情報が保存され、返されます。返される情報を保存できるだけの十分なサイズを割り当てる必要があります
type	現在使用しているDBMSを区別するための値です。  以下のいずれかを指定します  – ORACLE_TYPE  – SYBASE_TYPE  – INFORMIX_TYPE  – DB2_TYPE

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tmax\_get\_db\_username()が正常に実行されなかった場合、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。サーバー・グループ名の指定が正しくない場合や、typeをusrinc/tmaxapi.hに定義された値以外の値で定義した場合、あるいはtypeにINFORMIX_TYPEを設定した場合に発生します
[TPEITYPE]	svgrnameにNONXAサーバー・グループを指定した場合や、Tmax環境ファイルのOPENINFO情報が正しくない場合に発生します

- 例

#### <環境ファイルのOPENINFO>

```
svg4      NODENAME = @HOSTNAME@, DBNAME = ORACLE,
          OPENINFO = ORACLE_XA+Acc=P/scott/tiger+SesTm=60+Sqlnet=tmaxil",
          TMSNAME = tms_ora
```

#### <サーバー・プログラム>

```
#include <stdio.h>
...
SVC_GETUSRNAME( TPSVCINFO *msg )
{
    char username[30];
    printf("\n    => use right..\n");
    ret = tmax_get_db_username("svg1", username, ORACLE_TYPE);
    if(ret < 0)
        printf("tmax_get_db_username fail[%s]\n", tpstrerror(tperrno));
    else
        printf("\ntmax_get_db_username = %s\n", username);
    tpreturn( TPSUCCESS, 0, rcvbuf, 0, 0 );
}
```

#### <結果>

```
tmax_get_db_username = scott
```

- 関連関数

tmax\_get\_db\_passwd(), tmax\_get\_db\_tnsname()

## 3.2.8. tmax\_get\_svccnt

自身が属しているサーバーのサービス数を返す関数です。tmax\_get\_svclist()関数でサービス一覧を取得するためのバッファのサイズの割り当てに使用できます。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_get_svccnt(void)
```

- 戻り値

戻り値	説明
サービス数	関数の呼び出しに成功しました。自身が属しているサービス数を返します

## 3.2.9. tmax\_get\_svclist

自身が属しているサーバーのサービス一覧を取得する関数です。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_get_svclist(char *buf, int bufsize)
```

- パラメータ

パラメータ	説明
buf	以下のサイズで割り当てたバッファに、自身が属しているサーバーがもつサービスの名前を取得します  サービス数(tmax_get_svccnt()の戻り値) x XATMI_SERVICE_NAME_LENGTH
bufsize	以下の値です  サービス数(tmax_get_svccnt()の戻り値) x XATMI_SERVICE_NAME_LENGTH

- 戻り値

戻り値	説明
0	関数が呼び出しに成功した場合です
-1	関数が呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tmax\_get\_svclist()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。



エラーコード	説明
[TPEINVAL]	バッファのサイズが(サービス数×XATMI_SERVICE_NAME_LENGTH)より小さい場合、あるいはNULLの場合に発生します

- 例

#### <サーバー・プログラム>

```
#include <stdio.h>
#include <stdlib.h>
#include <usrinc/tmaxapi.h>

GETSVC(TPSVCINFO *msg)
{
    int i;
    int len, ret, size;
    char *buf;
    char *ptr;

    printf("GETSVC service is started!\n");
    len = tmax_get_svccnt();
    if (len < 0) {
        printf("tmax_get_svccnt fail[%d]\n", tperrno);
        tpreturn(TPFAIL, 0, 0, 0, 0);
    }
    printf("SVCCNT = %d\n", len);

    size = len * XATMI_SERVICE_NAME_LENGTH;
    buf = malloc(size);
    if (buf == NULL) {
        printf("buf is NULL\n");
        tpreturn(TPFAIL, 0, 0, 0, 0);
    }
    ret = tmax_get_svclist(buf, size);
    if (ret < 0) {
        printf("tmax_get_svclist fail[%d]\n", tperrno);
        tpreturn(TPFAIL, 0, 0, 0, 0);
    }
    for (i = 0; i < len; i++) {
        ptr = buf;
        printf("%dth SVC[%s]\n", i, ptr);
        buf += XATMI_SERVICE_NAME_LENGTH;
    }
    free(buf);
    tpreturn(TPSUCCESS, 0, (char *)msg->data, 0, 0);
}
```

## 3.2.10. tmax\_is\_restarted

サーバーでのみ使用できる関数です。Tmax APサーバー・ルーチン内で自身が属しているサーバー・プロセスが異常終了後に再起動したかどうかを判断できるようにする関数です。

以下の場合、関数が再起動されたと判断します。

- APランタイム・エラー（異常終了）
- タイムアウト: tpreturn(TPEXIT)
- tmdown -i -s APを実行後、再起動した場合

以下は、tmax\_is\_restarted関数についての基本的な説明です。

### ● プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_is_restarted(void);
```

### ● 戻り値

戻り値	説明
1	自身が属しているサーバー・プロセスが異常終了後、再起動された場合です
0	再起動されていない一般APサーバーの場合です

### ● 例

<サーバー・プログラム>

```
ISRESTARTED(TPSVCINFO *msg)
{
    int ret;
    ret = tmax_is_restarted()
    if(ret == 1)
        printf("restarted server process\n");
    else
        printf("normal server process\n");
    tpreturn(TPSUCCESS, 0, (char *)NULL, 0, 0);
}
```

## 3.2.11. tmax\_is\_xa

現在自身が属しているサーバーがXAであるか、あるいはNON-XAであるかをチェックする関数です。

### ● プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tmax_is_xa(void);
```

- 戻り値

戻り値	説明
XA_MODE(1)	現在自身が属しているサーバーがXAサーバーである場合です
NONXA_MODE(0)	XAモードでない場合です

- 例

```
#include <stdio.h>
SVC_XA( TPSVCINFO *msg )
{
    ...
    ret = tmax_is_xa();
    if (ret < 0)
        printf("\ntmax_is_xa func fail [%s]\n", tpstrerror(tperrno));

    if(ret == 1) strcpy(mode, « XA_MODE »);
    else if(ret == 0) strcpy(mode, "NONXA_MODE");
    else strcpy(mode, "unknown");
    printf(" => [SVC_XA] result of tmax_is_xa() : %s\n\n", mode);

    tpreturn( TPSUCCESS, 0, rcvbuf, 0, 0 );
}
```

## 3.2.12. tmax\_my\_svrinfo

サーバー・プロセスのシステム設定情報を取得する関数です。tmax\_my\_svrinfo()を使用して取得できる各種情報はサーバー・プロセスの静的情報です。プロセスが動作中の各種状態値を取得するには、**tmadmin()**を使用する必要があります。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_my_svrinfo (TMAXSVRINFO*)
```

- パラメータ

Tmaxシステム環境ファイルに登録されているそれぞれのサーバー・プロセスは、環境ファイルで階層的な情報(ノード名、サーバー・グループ名、サーバー名など)をもっています。また、1つのノードで一意的な値として管理されているサーバー・グループ索引、サーバー索引、サーバー・プロセスのシリアル番号なども確認できます。開発者はこのような値をプロセス管理および区分など、様々な用途で使用できます。

以下は、<tmxapi.h>に設定されたTMAXSVRINFO構造体の定義です。

```
typedef struct {
    int nodeno; /* node index */
    int svgi;   /* server group index; unique in the node */
    int svri;   /* server index; unique in the node */
    int spri;   /* server process index; unique in the node */
    int spr_seqno; /* server process seqno ; unique in the server */
    int min, max; /* min/max server process number */
    int clhi;    /* for RDP only, corresponding CLH id */
    char nodename[TMAX_NAME_SIZE];
    char svgname[TMAX_NAME_SIZE];
    char svrname[TMAX_NAME_SIZE];
    char reserved_char[TMAX_NAME_SIZE];
    /* for more detail use tmadmin API */
} TMAXSVRINFO;
```

メンバー	説明
nodeno	Tmaxシステムで1つのドメイン内で一意の値として管理されているノード索引です
svgi	1つのノードで一意の値として管理されているサーバー・グループ索引です
svri	1つのノードで一意の値として管理されているサーバー索引です
spri	1つのノードで一意の値として管理されているサーバー・プロセス索引です
spr_seqno	サーバー・プロセスのシリアル番号です
min , max	Tmaxシステム環境ファイルに設定されたプロセスの最小数と最大数を表します
clhi	サーバー・プロセスのタイプがRDPの場合のみ該当するフィールドです。該当RDPプロセスに対応するCLH番号を表します
nodename	Tmaxシステム環境ファイルに設定されたノード名です
svgname	Tmaxシステム環境ファイルに設定されたサーバー・グループ名です
svrname	Tmaxシステム環境ファイルに設定されたサーバー名です
reserved_char	現在は使用されておらず、今後使用するための予備用パラメータです

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	サーバー・プロセスの情報を保存する変数が指定されていない場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
```

```
#include <usrinc/tmaxapi.h>

SERVICE(TPSVCINFO *msg)
{
    int n;
    TMAXSVRINFO *info;

    info = (TMAXSVRINFO *)malloc(sizeof(TMAXSVRINFO));
    if (info == NULL) {
        tpreturn(TPFAIL, -1, NULL, 0, TPNOFLAGS);
    }
    n = tmax_my_svrinfo(info);
}
```

- 関連関数

tadmin()

### 3.2.13. tmgetsmgid

サーバーで使用するSysMasterのトレース機能をサポートするための関数で、現在の自身のGIDを取得します。ユーザーはGIDを使用し、システム単位の業務追跡を可能にできます。

関数が正常に実行されるためには、環境ファイルNODEセクションのSMSUPPORT項目が「Y」に設定されている必要があります。tmgetsmgid()関数以外に**tadmin**ツールの(st-p-x)コマンドを使用してサーバー・プロセスのGIDを取得することもできます。

GIDの構造は、以下のとおりです。関数が正常に実行された後、gid→gid1、gid→gid2、gid→seqno項目が保存されてから返されます。

項目	説明
GID1(4バイト)	製品内のクライアント別固有番号(WebtoBの場合cli id)で、domain id、node id、hth #、slot idなどで製品に接続したクライアントを識別するための番号です
GID2 (4バイト)	上位3バイトはseq #、下位1バイトは製品の固有IDで構成されます
SEQNO (4バイト)	上位2バイトは非同期呼び出し時のbranch #に使用され、下位2バイトはすべての呼び出し時にseq #に使用されます

- プロトタイプ

```
#include <usrinc/tmadmin.h>
int tmgetsmgid(tmax_smgid_t *gid);
```

- パラメータ

戻り値	説明
gid	製品内のクライアント別固有番号です

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。gidがNULLに入力された場合に返されます

- 例

<svr02.c : TPNOFLAGS使用>

```
#include <stdio.h>
#include <stdlib.h>
#include <usrinc/atmi.h>
#include <usrinc/tmadmin.h>
#include "../sdl/demo.s"

GETGID(TPSVCINFO *msg)
{
    tmax_smgid_t smgid;
    int ret;
    char *buf;

    buf = (char *)tpalloc("STRING", NULL, 0);
    if(buf == NULL)
        tpreturn(TPFAIL, -1, NULL, 0, 0);
    ret = tmgetsmgid(&smgid);
    memcpy(buf, (char *)&smgid.gid1, 4);
    memcpy(buf+4, (char *)&smgid.gid2, 4);
    memcpy(buf+8, smgid.seqno, 4);
    tpreturn(TPSUCCESS, 0, (char *)buf, strlen(buf), 0);
}

SMTRACE(TPSVCINFO *msg)
{
    struct tmadm_smtrc *smtrc;
    int max = 10, size;
    int gid1, gid2, n, i;
    struct smtrace *ptr;
    char *buf;

    buf = (char *)tpalloc("CARRAY", NULL, 0);
    if(buf == NULL)
```

```

        tpreturn(TPFAIL, -1, NULL, 0, 0);
    ptr = (struct smtrace *)msg->data;
    gid1 = ptr->gid1;
    gid2 = ptr->gid2;

    /*
    printf("SMTRACE start: %x %x\n", gid1, gid2);
    */

    size = sizeof(struct tmadm_smtrc) + (max-1) * sizeof(struct tmadm_smtrc_body);

    smtrc = (struct tmadm_smtrc *)malloc(size);
    if(smtrc == NULL)
    {
        printf("smtrc is null\n");
        tpreturn(TPFAIL, -1, NULL, 0, 0);
    }

    memset(smtrc, 0x00, size);
    smtrc->header.version = _TMADMIN_VERSION;
    smtrc->header.size = size;
    smtrc->header.reserve_int[0] = gid1;
    smtrc->header.reserve_int[1] = gid2;
    n = tmadmin(TMADM_SMTRC, smtrc, TPNOFLAGS, TPNOFLAGS);
    if(n < 0)
    {
        free(smtrc);
        tpreturn(TPFAIL, -1, NULL, 0, 0);
    }
    /*
    printf("smtrc->header.num_entry = %d\n", smtrc->header.num_entry);
    */
    for(i=0; i<smtrc->header.num_entry; i++)
    {
        sprintf(buf, "SMTRACE[%d] : gid[%x-%x-%x] seqno[%x] clhno[%x] status
        [%s] name[%s]\n", i, gid1, gid2, ptr->seqno,
        smtrc->trc[i].seqno,
        smtrc->trc[i].clhno,
        smtrc->trc[i].status,
        smtrc->trc[i].name);
    }

    free(smtrc);
    tpreturn(TPSUCCESS, 0, (char *)buf, strlen(buf), 0);
}

```

### <svr02\_a.c : TMADM\_AFLAG使用>

```
#include <stdio.h>
#include <stdlib.h>
#include <usrinc/atmi.h>
#include <usrinc/tmadmin.h>
#include "../sdl/demo.s"

GETGID_A(TPSVCINFO *msg)
{
    tmax_smgid_t smgid;
    int ret;
    char *buf;

    buf = (char *)tpalloc("STRING", NULL, 0);
    if(buf == NULL)
        tpreturn(TPFAIL, -1, NULL, 0, 0);

    ret = tmgetsmgid(&smgid);

    memcpy(buf, (char *)&smgid.gid1, 4);
    memcpy(buf+4, (char *)&smgid.gid1, 4);
    memcpy(buf+8, smgid.seqno, 4);

    tpreturn(TPSUCCESS, 0, (char *)buf, strlen(buf), 0);
}

SMTRACE_A(TPSVCINFO *msg)
{
    struct tmadm_smtrcall *smtrcall;
    int max = 10, size;
    int gid1, gid2, n, i;
    struct smtrace *ptr;
    char *buf;

    buf = (char *)tpalloc("CARRAY", NULL, 0);
    if(buf == NULL)
        tpreturn(TPFAIL, -1, NULL, 0, 0);

    ptr = (struct smtrace *)msg->data;
    gid1 = ptr->gid1;
    gid2 = ptr->gid2;

    /*
    printf("SMTRACE start: %x %x\n", gid1, gid2);
    */
}
```



```

size = sizeof(struct tmadm_smtrcall) +
      (max-1) * sizeof(struct tmadm_smtrcall_body);
smtrcall = (struct tmadm_smtrcall *)malloc(size);
if(smtrcall == NULL)
{
    printf("smtrcall is null\n");
    tpreturn(TPFAIL, -1, NULL, 0, 0);
}

memset(smtrcall, 0x00, size);
smtrcall->header.version = _TMADMIN_VERSION;
smtrcall->header.size = size;
smtrcall->header.reserve_int[0] = gid1;
smtrcall->header.reserve_int[1] = gid2;

n = tmadmin(TMADM_SMTRC, smtrcall, TMADM_AFLAG, TMADM_AFLAG);
if(n < 0)
{
    free(smtrcall);
    tpreturn(TPFAIL, -1, NULL, 0, 0);
}
printf("smtrcall->header.num_entry = %d\n", smtrcall->header.num_entry);

printf("smtrcall->header.num_left = %d\n", smtrcall->header.num_left);
n = 0;
for(i=0; i<smtrcall->header.num_entry + smtrcall->header.num_left; i++)
{
    sprintf(buf+n, "SMTRACE[%d] : gid[%x-%x-%x] seqno[%x] clhno[%x]

            status[%s] name[%s] spri[%d] curtime[%ld], svctime[%ld],

            ucputime[%ld], scputime[%ld]\n",
            i, gid1, gid2, ptr->seqno,
            smtrcall->trc[i].seqno,
            smtrcall->trc[i].clhno,
            smtrcall->trc[i].status,
            smtrcall->trc[i].name,
            smtrcall->trc[i].spri,
            smtrcall->trc[i].curtime.tv_sec,
            smtrcall->trc[i].svctime.tv_sec,
            smtrcall->trc[i].ucputime.tv_sec,
            smtrcall->trc[i].scputime.tv_sec);
    n = n + strlen(buf);
}
free(smtrcall);

```

```

        tpreturn(TPSUCCESS, 0, (char *)buf, strlen(buf), 0);
    }

```

#### <svr01.c>

```

#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmadmin.h>
#include "../sdl/demo_sdl.h"

SDLTOUPPER(TPSVCINFO *msg)
{
    int      i, ret, cd;
    struct  smtrace *stdata;
    tmax_smgid_t smgid;
    char *buf;
    long rcvlen;

    buf = (char *)tpalloc("CARRAY", NULL, 0);
    ret = tmgetsmgid(&smgid);
    if(ret < 0)
        tpreturn(TPFAIL, -1, NULL, 0, 0);
    stdata = (struct smtrace *)msg->data;
    stdata->gid1 = smgid.gid1;
    stdata->gid2 = smgid.gid2;
    stdata->seqno = smgid.seqno;

    cd = tpacall("SMTRACE", msg->data, 0, 0);
    ret = tpgetrply(&cd, (char **)&buf, (long *)&rcvlen, 0);
    if(ret < 0)
        tpreturn(TPFAIL, -1, NULL, 0, 0);
    sleep(20);
    tpreturn(TPSUCCESS, 0, (char *)buf, strlen(buf), 0);
}

```

#### < client.c>

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <usrinc/atmi.h>
#include <usrinc/tmadmin.h>
#include "../sdl/demo.s"

main(int argc, char *argv[])
{

```

```

struct smtrace *sndbuf, *rcvbuf;
long rcvlen, sndlen;
int ret;

if (tpstart((TPSTART_T *)NULL) == -1){
    printf("tpstart failed\n");
    exit(1);
}
...
if (tpcall("SDLTOUPPER", (char *)sndbuf, 0, (char **)&rcvbuf, &rcvlen, 0) ==
1)
{
    printf("Can't send request to service SDLTOUPPER =>\n");
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
    exit(1);
}
printf("rcvbuf = %s\n", rcvbuf);
tpfree((char *)sndbuf);
tpfree((char *)rcvbuf);
tpend();
}

```

---

## 参考

tmadminツールについての詳細は、『Tmax 運用ガイド』を参照してください。

---

## 3.2.14. tmget\_smtrclog

ロギング・データを構造体バッファに保存する関数です。入力時にバッファの最大ロギング件数を設定し、出力時に保存されたロギング件数が保存されます。

### ● プロトタイプ

```

#include <usrinc/tmadmin.h>
int tmget_smtrclog(void *handle, tmax_smtrclog_t *buf, int *count)

```

### ● パラメータ

項目	説明
handle	ロギング・サービスで渡されたデータのポインタ(msg->data)です
buf	ロギング・データを取得するためのバッファです

項目	説明
count	ログイン件数が保存されます

ログイン情報の構造体(tmax\_smtrclog\_t)は以下のとおりです。

< usrinc/tmadmin.h>

```

/* SysMaster Trace Log structure */
typedef struct {
    tmax_smgid_t gid;
    int      clhno;
    char     status[TMAX_NAME_SIZE];
    char     name[TMAX_NAME_SIZE];
    int      spri;
    int      reserved;
    struct timeval curtime;
    struct timeval svctime;
    struct timeval ucputime;
    struct timeval scputime;
} tmax_smtrclog_t;

```

メンバー	説明
gid	Sysmaster GIDです
clhno	要求を受けたclh番号です
status[TMAX_NAME_SIZE]	サービスの状態です
name[TMAX_NAME_SIZE]	サービスの名前です
spri	サーバー・プロセスの索引です
reserved	現在は使用していません
curtime	現在の時間です
svctime	サービス実行時間です
ucputime	ユーザーCPU時間です
scputime	システムCPU時間です

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tmget\_smtrclog()が正常に実行されなかった場合、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	入力された引数が正しくない場合です
[TPESYSTEM]	ノードがSysMaster設定になっていない場合や、SysMaster log serviceが設定されていない場合です
[TPELIMIT]	入力したcountが実際にhandleにある件数より小さい場合です

- 例

<SVR.C>

```
#include <stdio.h>
#include <stdlib.h>
#include <usrinc/atmi.h>
#include <usrinc/tmadmin.h>

SMLOGSERVICE(TPSVCINFO *msg)
{
    tmax_smtrclog_t *smtrclog;
    int ret, count=0, i;
    char *buf;

    smtrclog = (tmax_smtrclog_t *)tpalloc("CARRAY", NULL, 1024);
    if(smtrclog == NULL)
    {
        printf("smtrclog tpalloc fail [%s]\n", tpstrerror(tperrno));
    }
    buf = (char *)tpalloc("STRING", NULL, 0);
    if(buf == NULL)
        tpreturn(TPFAIL, -1, NULL, 0, 0);

    memset(buf, 0x00, 1024);
    memset(smtrclog, 0x00, 1024);
    count = tmget_smtrclog_count((char *)msg->data);
    printf("\n#####\n\n");
    printf("tmget_smtrclog_count = %d\n", count);

    count = 100;
    ret = tmget_smtrclog(msg->data, smtrclog, &count);
    printf("count = %d\n", count);
    for(i=0; i<count; i++)
    {
        printf("smtrclog[%d].gid = %d-%d-%d\n", i, smtrclog[i].gid.gidl,
            smtrclog[i].gid.gid2, smtrclog[i].gid.seqno);
        printf("smtrclog[%d].clhno = %d\n", i, smtrclog[i].clhno);
    }
}
```

```

        printf("smtrclog[%d].status = %s\n", i, smtrclog[i].status);
        printf("smtrclog[%d].name = %s\n", i, smtrclog[i].name);
        printf("smtrclog[%d].spri = %d\n", i, smtrclog[i].spri);
        printf("\n");
    }
    printf("#####\n\n");
    strcpy(buf, "success\n");
    tpreturn(TPSUCCESS, 0, (char *)buf, 0, 0);
}

```

### 3.2.15. tmget\_smtrclog\_count

データ数を返す関数で、現在ロギングするデータの件数を返します。

- プロトタイプ

```

#include <usrinc/tmadmin.h>
int tmget_smtrclog_count(void *handle)

```

- パラメータ

項目	説明
handle	ロギング・サービスで渡されたデータのポインター(msg→data)です

- 戻り値

戻り値	説明
現在ロギングされたデータ数	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tmget\_smtrclog\_count()が正常に実行されなかった場合、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	入力された引数が正しくない場合です
[TPESYSTEM]	ノードがSysMaster設定になっていない場合や、SysMaster log serviceが設定されていない場合です

- 例

[「3.2.14. tmget\\_smtrclog」](#)の例を参照してください。

## 3.2.16. tpadvertise

サーバー・プロセスが提供するサービスをサーバーに登録(advertise)する関数で、特定のサービス名を登録できます。特定サービスを登録する場合、TMMが管理するサーバー別サービスの名前表に該当サービス名を登録できます。登録されたサービスが呼び出された場合、funcが指す関数が呼び出されて実行されます。サーバー・プロセスに、該当サーバーが提供する新規サービスを登録できるようにします。

Tmax 5.0 SP1 Fix4(r7000)、Tmax 5.0 SP2 Fix1以前のバージョンでは、環境設定に登録されたサービスのみ処理が可能で、新しいサービスの登録または解除はできませんでした。前述したバージョンからは、新しいサービスも登録または解除することが可能になりました。

以下は、それぞれのケースにおける動作方式の説明です。

- **環境設定ファイルに登録されたサービス**

tpunadvertiseを実行している場合、再びサービスできる状態に変更します。他のサーバーでtpadvertiseを実行すると失敗します。

- **mksvrで登録されたサービス**

tpunadvertiseを実行している場合、再びサービスできる状態に変更します。他のサーバーでtpadvertiseを実行すると失敗します。サーバー・プロセスがすべて終了すると、tpadvertiseを実行したすべてのサービスは自動削除されます。

- **新規登録するサービス**

新規サービスを登録します。tpunadvertiseを実行している場合、再びサービスできる状態に変更します。他のサーバーでtpadvertiseを実行すると失敗します。サーバー・プロセスがすべて終了すると、tpadvertiseを実行したすべてのサービスは自動削除されます。新規登録するサービスは、SVCTIMEOUTとAUTOTRANを設定できないという制約があります。

以下は、関数の使用方法と例についての説明です。

- **プロトタイプ**

```
#include <atmi/usrinc.h>
int tpadvertise(char svcname, void (func)(TPSVCINFO *));
```

- **パラメータ**

パラメータ	説明
svcname	登録するサービス名です

パラメータ	説明
(func)(TPSVCINFO *)	呼び出す関数名です

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tpadvertise()が正常に実行されなかった場合、tpernoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	サービス名がNULLであるか、funcがNULLの場合です
[TPELIMIT]	サービス名が指定の長さを超えた場合です。あるいは、該当ノードのMAXSVC数に達し、新しいサービスを登録できなくなった場合です
[TPEMATCH]	該当サービスがすでに別の関数でadvertiseされた場合です。あるいは、別のサーバーで登録済みのサービスを登録しようとした場合に発生します
[TPEPROTO]	該当関数が不適切な状況で呼び出された場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <usrinc/atmi.h>

SVC2TN_NEWSVC1( TPSVCINFO *msg )
{
    ...
}

SVC2TN_NEWSVC2( TPSVCINFO *msg )
{
    ...
}

SVC2TN_1( TPSVCINFO *msg )
{
    int      ret;
    char     input[MAXLEN];
```



```

void      (*func)(TPSVCINFO*);

memset(input, 0x00, MAXLEN);
strncpy(input, msg->data, msg->len);

func = SVC2TN_NEWSVC1;
ret = tpadvertise(input, func);
if (ret < 0) {
    tpreturn(TPFAIL, 0, (char*)msg->data, msg->len, 0);
}
tpreturn(TPSUCCESS, 0, (char*)msg->data, msg->len, 0);
}

```

---

## 参考

advertise/unadvertise関数についての詳細は、『*Tmax 運用ガイド*』のadvertise / unadvertiseの説明を参照してください。

---

## 3.2.17. tpcancelctx

tpsavectx()で保存されたクライアント情報のうち該当構造体の内容を取り消す関数です。tprelay()を実行しなくても、サービス・ルーチンが終了すると結果が正常に返されます。

tpgetctx()はサービス・ルーチン内でのみ使用できます。

### ● プロトタイプ

```

#include <ucs.h>
int tpcancelctx(CTX_T *ctxp);

```

### ● パラメータ

パラメータ	説明
ctxp	ライブラリー内部に保存されたCTX_T構造体の内容を削除します

以下は、CTX\_T構造体の定義です。

```

typedef struct {
    int    version[4];
    char   data[CTX_USR_SIZE - 16];
} CTX_T;

```

### ● 例

```

RELAY_SVC(TPSVCINFO *msg) {
    .....
    ctxp = (CTX_T *)tpsavectx();
    .....
    ret=tpcancelctx(ctxp);
    if (ret<0) {
        error process routine
    }
    .....
    tpreturn(TPFAIL, sqlca.sqlcode, NULL, 0, TPNOFLAGS);
}

```

## 3.2.18. tpchkclid

IDに該当するクライアントが当該サーバー・プロセスが位置するノードに接続している状態かを確認する関数です。主にRDP方式のサーバー・プログラムを開発するときに、サービス・ルーチンで接続したクライアントIDを保存し、usermain()ルーチンからtpsendtocli()にメッセージを送る場合に使用すると、不要なエラーを事前に防ぐことができます。

### 参考

RDP方式ではサーバー・プロセスが位置するノードに直接接続された状態でない場合は、tpsendtocli()を使用できません。

### ● プロトタイプ

```

#include <tmaxapi.h>
int tpchkclid(int clid)

```

### ● パラメータ

パラメータ	説明
clid	クライアント番号で、 <b>tpgetclid()</b> を使用して取得する値です

### ● 戻り値

戻り値	説明
-2	該当クライアントはローカル・ノードに接続されたクライアントでない場合です
-1	該当クライアントが接続されていない場合です
1	該当クライアントが正常に接続されている場合です

### ● エラー

tpchkclid()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	クライアントがローカル・ノードに接続されていない場合です。あるいは、入力されたクライアント番号の値が正しくない場合です
[TPENOREADY]	クライアントが正常に接続されていない状態です

- 例

```
int _discon(char **buf)
{
    int clid, n;
    clid = tpgetclid();
    n = tpchkclid(clid);
    if (n < 0) {
        printf("Invalid Client\n");
        return -1;
    }
    ...
}
```

- 関連関数

tpgetclid()

## 3.2.19. tpclrfd

UCS方式プロセス内部のfdsetのソケットFDをオフにするのに使用される関数です。UCS方式サーバー・プロセスの外部ソケットをスケジューリングする場合に使用します。

- プロトタイプ

```
#include <ucs.h>
int tpclrfd (int fd)
```

- パラメータ

パラメータ	説明
fd	オフにする内部fdsetのソケットです

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tpclrfd()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <usrinc/ucs.h>
...
#define SERV_ADDR "168.126.185.129"
#define SERV_PORT 1500

int fd_read(int, char *, int);
extern int errno;

int usermain(int argc, char *argv[])
{
    ...
    int listen_fd, n, newfd;
    struct sockaddr_in my_addr, child_addr;
    socklen_t child_len;
    buf = tpalloc("STRING", NULL, 0);
    if (buf == NULL){ error processing }

    memset((void *)&my_addr, NULL, sizeof(my_addr));
    memset((void *)&child_addr, NULL, sizeof(child_addr));
    listen_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (listen_fd == -1){ error processing }
    my_addr.sin_family = AF_INET;
    inaddr = inet_addr(SERV_ADDR);
    my_addr.sin_port = htons((unsigned short)SERV_PORT);
    if (inaddr != -1){
        memcpy((char *)&my_addr.sin_addr, (char *)&inaddr, sizeof(inaddr));
```

```

    }

    ret = bind(listen_fd, (struct sockaddr *)&my_addr, sizeof(my_addr));
    if (ret == -1){ error processing }
    ret = listen(listen_fd, 5);
    if (ret == -1){ error processing }

    tpsetfd(listen_fd);
    ...
    while(1) {
        n = tpschedule(10);
        ...
        if (n == UCS_USER_MSG){
            if (tpissetfd(listen_fd)) {
                child_len = sizeof(child_addr);
                newfd = accept(listen_fd, &child_addr, &child_len);
                if (newfd == -1){ error processing }
                tpsetfd(newfd);
            }

            if (tpissetfd(newfd)){
                /* ソケットからバッファを読み込みます */
                fd_read(newfd, buf, 1024);
                ret = tpcall("SERVICE", (char *)buf, sizeof(buf), (char **)&buf,
                            (long *)&rlen, TPNOFLAGS);
                if (ret == -1){ error processing }
                ...
                ret = tpclrfd(newfd);
                if (ret == -1){ error processing }
                close(newfd);
            }
            ...
        }
        return 1;
    }
}

```

- 関連関数

tpissetfd()

### 3.2.20. tpclrfd\_w

UCS方式プロセス内部のWritable FDSETのソケットFDをオフにするのに使用される関数です。UCS方式サーバー・プロセスの外部ソケットをスケジューリングする場合に使用します。UCS方式サーバー・プロセスのFDSETでパラメータ値として与えられたソケットFDを除去させる関数です。

tpclrfd()関数がReadable FDSETから除去する一方、この関数はWritable FDSETから除去します。UCS方式プロセスの外部ソケット・スケジューリングに使用されます。FDはWritable FDSETから除去するソケットFD値です。

tpissetfd\_w()、tpsetfd\_w()、tpclrfd\_w()関数は、tpschedule()とtpuschedule()のUCSスケジューラーに、TMM、CLHだけでなく、外部ホスト/クライアントと接続して通信するユーザー・ソケットFDでもスケジューリングできます。ユーザーが指定したソケットFDに送るメッセージが準備されると、tpschedule()はUCS\_USER\_MSGを返します。どのソケットFDにメッセージが準備されたのかを認識するためにはtpissetfd\_w()を使用します。

tpissetfd()、tpsetfd()、tpclrfd()、tpissetfd\_w()、tpsetfd\_w()、tpclrfd\_w()などのソケットFD関連マクロ関数は、一般ネットワーク・プログラムで使用するFD\_SET、FD\_CLR、FD\_ISSETと類似しています。

- プロトタイプ

```
#include <ucs.h>
int tpclrfd_w (int fd)
```

- パラメータ

パラメータ	説明
fd	オフにする内部fdsetのソケットです

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tpclrfd()が正常に実行されなかった場合、tpernoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

[「3.2.38. tpissetfd」](#)、[「3.1.91. tpsetfd」](#)、[「3.2.19. tpclrfd」](#)関数の例を参照してください。

- 関連関数

tpissetfd(), tpsetfd(), tpclrfd()

### 3.2.21. tpforward

サーバーからのサービス要求を他のサービス・ルーチンに渡す関数です。自身のサービス処理を終了し、クライアントの要求をsvcサービス・ルーチンに渡します。

tpforward()はサービス・ルーチンで最後に呼び出すものであり、**tpreturn()**のように作動します。tpreturn()と同様に、tpforward()がTmaxシステムで正常に返されるために、tpforward()はTmaxシステムが制御するサービス・ルーチン内で呼び出される必要があります。

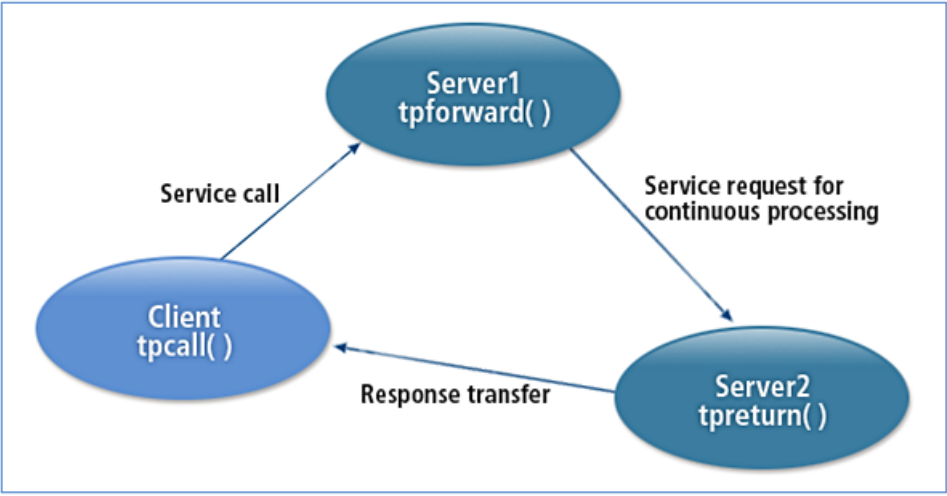
dataが指すデータを使用して、svcと指定されたサービスに要求を渡します。要求を渡すサービス・ルーチンはいかなる応答も受信しません。要求が渡された後、サービス・ルーチンはTmaxシステムに返します。また、サーバーは自由に他の作業を実行できます。tpforward()は要求者からいかなる応答も期待しないため、特にエラーなく、どのサービスにでも渡すことができます。

サービス・ルーチンがトランザクション・モードの場合、そのトランザクションはトランザクション開始者(originator)がtx\_commit()あるいはtx\_rollback()のいずれかを実行してトランザクションを完了することで完了します。tpforward()は、tpreturn()と同様にトランザクションを完了しません。トランザクションがサービス・ルーチン内でtx\_begin()を使用して開始したのであれば、そのトランザクションはtpforward()の呼び出し前に、tx\_commit()あるいはtx\_rollback()ツールのうちの1つを先に完了する必要があります。tpforward()で接続されたすべてのサービスは、すべてがトランザクション・モードであるか、あるいはすべてがトランザクション・モードでない必要があります。最終的に渡されたサーバー・プロセスが、tpreturn()を使用して最初のサービスを要求したクライアントに応答を送ります。tpforward()は、応答を待機している要求者に応答を送信する責任を他のサーバー・プロセスに回すことで、マルチノード間でもサービスが行われます。

tpforward()は、サービス・ルーチンが要求したすべてのサービスに対する応答を受信後に呼び出される必要があります。受信されていない応答に対する記述子は無効化され、転送された要求は送信されません。対話型サービスでは、tpforward()を呼び出すことができません。

以下は、tpforward関数のフロー・チャートです。

[図 3.1] tpforward



● プロトタイプ

```
#include <atmi.h>
void tpforward (char *svc, char *data, long len, long flags)
```

● パラメータ

パラメータ	説明
svc	バッファーを受けるサービス名です
data	<p>NULLでない場合、tpalloc()によって以前に割り当てられたバッファーを指す必要があります。</p> <p>バッファーがサービス・ルーチンに送信されたのと同じバッファーの場合、Tmaxシステムがこのバッファーに対する処理責任をもちます。サービス・ルーチン作成者がこのバッファーを解除しようとした場合、これは失敗と処理されます。しかし、tpforward()で送信されるバッファーが、サービスの呼び出し時に渡されたものと同じバッファーでない場合、tpforward()がそのバッファーを解除します</p>
len	<p>送信されるデータ長です。dataが指すバッファーが、長さを指定する必要がないタイプ(たとえば、STRUCT)の場合、lenは無視され、一般的に0が使用されます。dataがNULLの場合、lenは無視され、データ長が0の要求が送信されます。</p> <p>tpforward()の呼び出し後、サービス・ルーチンの作成者は制御権を再取得できないため、TPSIGRSTRTが暗示的に定義された形式のブロッキング送信が使用されます。tpforward()の実行中にシグナルが発生して実行が中止されても、後に再実行されます。また、ブロッキング状況になっても、タイムアウトが発生するまでは待機して送信します</p>
flags	現行バージョンではサポートしていませんが、TPNOFLAGSに設定します



- 戻り値

サービス・ルーチンは、呼び出し元のTmaxシステムにいかなる値も返しません。サービス・ルーチンはvoidと宣言されます。

- エラー

tpforward()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESVCERR]	有効でないバッファーを使用した場合、有効なtpacall()、tpconnect()の戻り値でcdを返す場合、対話型通信でtpreturn()の代わりにtpforward()を使用した場合、TPEV_DISCONIMMイベントが発生した場合、トランザクション・モードでXA operationが失敗した場合(tx_begin()、tx_rollback()、tx_commit())に発生します
[TPETIME]	サービス・ルーチン作業中あるいは要求を転送中にトランザクション・タイムアウトが発生した場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>

SWITCH(TPSVCINFO *msg)
{
    int switch;
    char *buf;

    buf = (char *)tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }

    strcpy(buf, msg->data);
    data process...

    if (switch>5)
        tpforward("SERVICE1", buf, 0, 0);
    else
        tpforward("SERVICE2", buf, 0, 0);
}
```

- 関連関数

tpalloc(), tpconnect(), tpreturn()

## 3.2.22. tpgetclid

Tmaxシステムに接続されたクライアントの番号を確認できる関数です。クライアント番号は、ドメインシステム内で一意の番号です。複数のマルチノードでドメインシステムが構築されていても、一意の番号をクライアントに付与します。この関数はサーバーでのみ使用でき、サービスを要求した該当クライアントIDを求めて、**tpsendtcli()**でクライアントにメッセージを送るために使用されます。

- プロトタイプ

```
#include <tmaxapi.h>
int tpgetclid(void)
```

- 戻り値

戻り値	説明
0以上の整数値	関数の呼び出しに成功した場合です。クライアントの番号に該当する0以上の整数値を返します
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpgetclid()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	tpgetclid()が不適切な状況で呼び出された場合です。たとえば、クライアント・プログラム内で使用された場合に発生します
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

SERVICE(TPSVCINFO *msg)
{
    int ret, clid;
    char *buf;
    buf = (char *)tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
    strcpy(buf, msg->data);
    data process...

    clid = tpgetclid();
```

```

    if (clid== -1) { error process }

    ret=tpsendtocli(clid, buf, strlen(buf), 0);
    if (ret== -1) { error processing }
    data process....

    tpreturn(TPSUCCESS,0,buf, strlen(buf), 0);
}

```

- 関連関数

tpsendtocli()

### 3.2.23. tpgetctx

現在のクライアント情報を、ユーザーが宣言し割り当てたCTX\_T構造体にコピーする関数です。tpgetctx()を使用した場合、tprelay()でこの情報を使用しないと、該当するサービス・ルーチンが完了してもクライアントは応答を待ち続けます。

tpgetctx()で取得した情報は、tpcancelctx()で取り消すことができないため、必ずtprelay()を使用する必要があります。サービス・ルーチン内でのみ使用することができます。

- プロトタイプ

```

#include <tmaxapi.h>
int tpgetctx (CTX_T *ctxp)

```

- パラメータ

パラメータ	説明
ctxp	tpsavectx()で保存されたクライアント情報をCTX_T構造体にコピーします

- 例

```

RELAY_SVC(TPSVCINFO *msg)
{
    CTX_T *ctxp;
    ctxp=(CTX_T *)malloc(sizeof(CTX_T));
    ....
    ret = tpgetctx(ctxp);
    if (ret<0) {
        error process routine
    }
    .....
}

```

## 3.2.24. tpgetdbsessionid

RMセッション情報を取得する関数です。RMのセッション情報をサービスで処理する場合、毎回処理しなければならない手間を省くために、コールバック関数(tpsetdbsessionid)を使用してRMセッション情報を設定できます。ユーザーは、tpgetdbsessionidを使用して、現在接続されているセッションIDを得ることができます。

- プロトタイプ

```
#include <tmaxapi.h>
char * tpgetdbsessionid(int flags);
```

- パラメータ

パラメータ	説明
flags	現行バージョンではサポートしていませんが、TPNOFLAGSに設定します

- 戻り値

戻り値	説明
現在接続されているセッションID	関数の呼び出しに成功した場合です

- 例

```
#include <usrinc/tmaxapi.h>
...
EXEC SQL include sqlca.h;
EXEC SQL begin declare section;
...
char h_ssid[20];
EXEC SQL end declare section;

int tpsetdbsessionid(char dbsessionid[MAX_DBSESSIONID_SIZE], int flags)
{
    EXEC SQL SELECT TO_CHAR(USERENV('sessionid')) into :h_ssid FROM dual;
    if ( sqlca.sqlcode != 0 ){
        printf( "getting session id fail = %d\n",sqlca.sqlcode );
        return -1;
    }
    printf("RM session id = %s\n", h_ssid);
    memset(dbsessionid, 0x00, MAX_DBSESSIONID_SIZE);
    strcpy(dbsessionid, h_ssid);
    return 0;
}
```

```

FDLINS( TPSVCINFO *msg )
{
    char *ssid;
    int ret;
    int flags = 0;

    printf(" >>> current RM session id = %s\n", h_ssid);
    ssid = tpgetdbsessionid(flags);
    tpreturn( ... );
}

```

### 3.2.25. tpgetmaxsvr

サーバー・プロセスの最大実行数を返す関数です。設定ファイルのSERVERセクションに定義したサーバー・プロセスの最大実行数のMAX項目の値を返します。常に自身のサーバー・プロセスの最大数のみを確認できます。

- プロトタイプ

```

#include <tmaxapi.h>
int tpgetmaxsvr(void)

```

- 戻り値

戻り値	説明
整数値	関数の呼び出しに成功した場合です。サーバー・プロセスの最大実行数を返します
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpgetmaxsvr()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```

#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

```

```

SERVICE(TPSVCINFO *msg)
{
    char *buf;
    buf = msg->data;

    data process...
    printf("maxsvr : %d\n",tpgetmaxsvr());
    data process...
    tpreturn(TPSUCCESS,0,buf, strlen(buf), 0);
}

```

- 関連関数

tpgetminsvr()

### 3.2.26. tpgetmaxuser

サーバー・プロセスが属しているノードの最大同時接続者数を返す関数です。

- プロトタイプ

```

#include <tmaxapi.h>
int tpgetmaxuser(void)

```

- 戻り値

戻り値	説明
整数	関数が正常に実行された場合です。最大同時接続者数を返します
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpgetmaxuser()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログ・ファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```

#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

SERVICE(TPSVCINFO *msg)
{
    char *buf;
    buf = msg->data;
    data process...
    printf("maxusr : %d\n",tpgetmaxusr());
    data process...
    tpreturn(TPSUCCESS,0,buf, strlen(buf), 0);
}

```

### 3.2.27. tpgetminsvr

サーバー・プロセスの最小実行数を返す関数です。設定ファイルのSERVERセクションに定義したサーバー・プロセスの最小実行数のMIN項目の値を返します。常に自身のサーバー・プロセスの最小数のみを確認できます。

- プロトタイプ

```

#include <tmaxapi.h>
int tpgetminsvr(void)

```

- 戻り値

戻り値	説明
整数	関数の呼び出しに成功した場合です。サーバー・プロセスの最小実行数を返します
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpgetminsvr()が実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログ・ファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```

#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

SERVICE(TPSVCINFO *msg)
{
    char *buf;
    buf = msg->data;
    data process...
    printf("minsvr : %d\n",tpgetminsvr());
    data process...
    tpreturn(TPSUCCESS,0,buf, strlen(buf), 0);
}

```

- 関連関数

tpgetmaxsvr()

## 3.2.28. tpgetmynode

サーバーで特定ノードの名前とノード番号を取得する関数です。

- プロトタイプ

```

#include <tmaxapi.h>
char *tpgetmynode(int *nodeno)

```

- パラメータ

パラメータ	説明
nodeno	ノード名は戻り値のchar *に返され、ノード番号はnodenoに設定されます

- 戻り値

戻り値	説明
ノード名	関数の呼び出しに成功した場合です
NULL	関数の呼び出しに失敗した場合です。エラーが発生しても、tperrnoにエラー番号が設定されません

- 例

```

#include <usrinc/tmaxapi.h>
SERVICE(TPSVCINFO *msg)

```



```

{
    int i, clid, n;
    char *nodename;
    ...
    for (i = 0; i < msg->len; i++)
        msg->data[i] = toupper(msg->data[i]);
    ...
    if (nodename == NULL){ error processing }
    nodename = tpgetmynode((int *)&n);
    if (nodename == NULL){
        error processing
    }
    else {
        printf("TOUPPER SERVICE node(%dth node) name = %s\n", n, nodename);
    }
    ...
    tpreturn(TPSUCCESS, 0, (char *)msg->data, 0, TPNOFLAGS);
}

```

### 3.2.29. tpgetmysvgno

サーバーでのみ使用できる関数であり、現在自身が属しているサーバー・グループの番号を返します。COUSIN関係にあるマルチサーバー・グループ環境で特定のサーバー・グループに属するサービスを指定してサービス要求を送信する**tpcallsvg()**または**tpacallsvg()**と一緒に使用されます。また、自身が属しているサーバー・グループに呼び出す場合に使用されます。

- プロトタイプ

```

#include <usrinc/tmaxapi.h>
int tpgetmysvgno(void);

```

- 戻り値

戻り値	説明
グループ番号	関数の呼び出しに成功した場合です。該当関数を呼び出したサービスが属するサーバー・グループの番号を返します

- 例

<サーバー・プログラム>

```

FDLToupper1(TPSVCINFO *msg)
{
    FBUF      *sndbuf, *rcvbuf;

```

```

int ret, i, mysvgno;
char sndata[30], rcvdata[30];
char svc[XATMI_SERVICE_NAME_LENGTH];

sndbuf = (FBUF *)tpalloc("FIELD", NULL, 0});
rcvbuf = (FBUF *)tpalloc("FIELD", NULL, 0))

/* tpcallsvg (mysvgno) */
ret = func_tpcallsvg_myno(sndbuf, rcvbuf, msg->name);
if(ret < 0)
    tpreturn(TPFAIL, 0, (char *)NULL, 0, 0);
tpreturn(TPSUCCESS, 0, (char *)rcvbuf, 0, 0);
}

int func_tpcallsvg_myno(FBUF *sndbuf, FBUF *rcvbuf, char *svc)
{
    int ret;
    char sndata[30], rcvdata[30];
    long rcvlen;
    int svgno;

    strcpy(sndata, "starbj81");
    ret = fbput(sndbuf, INPUT, sndata, 0);
    svgno = tpgetmysvgno();
    ret = tpcallsvg(svgno, "FDLToupper2", (char *)sndbuf, 0, (char **)&rcvbuf,
                    &rcvlen, 0);

    if(ret < 0)
    {
        printf("tpcallsvg[%s] failed! [%d][%s]\n", svc, tperrno, tpstrerror(tperrno));

        return -1;
    }
    fbprint(rcvbuf);
    fbinit(sndbuf, 1024);
    fbinit(rcvbuf, 1024);
    return 0;
}

```

- 関連関数

tpcallsvg(), tpacallsvg()

### 3.2.30. tpgetmysvrid

サーバー・プロセスIDを返す関数です。サーバー・プロセスIDは、設定ファイルにあるSERVERセクションのサーバー・プロセスの実行時に付与する番号です。同じサーバー・プロセスを複数実行しても、別のサーバー・プロセス番号が付与されます。返される値は0からであり、動作するサーバー・プロセスの数に従って1ずつ増加した値が使用されます。tpgetmysvrid()関数は、サービス・ルーチン(サーバー・プロセス)でのみ使用できます。

- プロトタイプ

```
#include <tmaxapi.h>
int tpgetmysvrid(void)
```

- 戻り値

戻り値	説明
整数	関数の呼び出しに成功した場合です(サーバー・プロセスのIDに該当する0以上の整数値を返します)
-1	関数の呼び出しに失敗した場合です(tperrnoにエラーコードが設定されます)

- エラー

tpgetmysvrid()が実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEPROTO]	関数が不適切な状況で呼び出された場合です。たとえば、クライアント・プログラム内で使用された場合に発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

SERVICE(TPSVCINFO *msg)
{
    char *buf;
    buf = msg->data;
    data process...
    printf("mysvr : %d\n",tpgetmysvrid());
    data process...
```

```

    tpreturn(TPSUCCESS,0,buf, strlen(buf), 0);
}

```

### 3.2.31. tpgetnodename

サーバーで指定されたノード名を取得する関数です。

- プロトタイプ

```

#include <tmaxapi.h>
char *tpgetnodename(int nodeno)

```

- パラメータ

パラメータ	説明
nodeno	ノード名を照会するノード番号を渡します

- 戻り値

戻り値	説明
ノード名	関数の呼び出しに成功した場合です。正常処理された場合、ノード名を返します
NULL	関数の呼び出しに失敗した場合です。tperno!にはエラー番号が設定されません

- 例

```

#include <usrinc/tmaxapi.h>

SERVICE(TPSVCINFO *msg)
{
    int i, clid, n;
    char *nodename;
    ...
    for (i = 0; i < msg->len; i++)
        msg->data[i] = toupper(msg->data[i]);

    nodename = (char *)tpgetnodename(n);
    if (nodename == NULL){
        error processing
    }
    else {
        printf("%dth node name(original node name) = %s\n", n, nodename);
    }
    ...
}

```

```

    tpreturn(TPSUCCESS,0,(char *)msg->data, 0,TPNOFLAGS);
}

```

## 3.2.32. tpgetnodeno

サーバーでnodenameの名前をもつノード番号を取得する関数です。

- プロトタイプ

```

#include <tmaxapi.h>
int tpgetnodeno(char *nodename)

```

- パラメータ

パラメータ	説明
nodename	ノード番号を取得するノード名を渡します

- 戻り値

戻り値	説明
ノード番号	関数の呼び出しに成功した場合です。正常処理された場合、ノード番号を返します
-1	関数の呼び出しに失敗した場合です。tperrnoにはエラー番号が設定されません

- 例

```

#include <usrinc/tmaxapi.h>

SERVICE(TPSVCINFO *msg)
{
    int i, clid, n;
    char *nodename;
    ...
    for (i = 0; i < msg->len; i++)
        msg->data[i] = toupper(msg->data[i]);

    n = tpgetnodeno("tmax");
    if (n < 0){
        error processing
    }
    else {
        printf("%s's node no(original node no) = %d\n", "tmax", n);
    }
    ...
}

```

```

    tpreturn(TPSUCCESS,0,(char *)msg->data, 0,TPNOFLAGS);
}

```

### 3.2.33. tpgetorgclh

該当クライアントが現在接続しているCLH番号を照会する関数です。

- プロトタイプ

```

#include <tmaxapi.h>
int tpgetorgclh(int clid)

```

- パラメータ

パラメータ	説明
clid	現在接続しているクライアントIDを設定します

- 戻り値

戻り値	説明
CLH番号	関数の呼び出しに成功した場合です。正常処理された場合、該当クライアントが現在接続しているCLH番号を返します
-1	関数の呼び出しに失敗した場合です。tperrnoにはエラー番号が設定されません

- 例

```

#include <usrinc/tmaxapi.h>

SERVICE(TPSVCINFO *msg)
{
    int i, clid, n;
    ...
    for (i = 0; i < msg->len; i++)
        msg->data[i] = toupper(msg->data[i]);

    n = tpgetorgclh((int)*(msg->cltid.clientdata));
    if (n < 0){
        error processing
    }
    else {
        printf("clh number = %d\n", n);
    }
    ...
}

```

```

    tpreturn(TPSUCCESS,0,(char *)msg->data, 0,TPNOFLAGS);
}

```

### 3.2.34. tpgetorgnode

該当クライアントが接続しているノード番号を返す関数です。

- プロトタイプ

```

#include <tmaxapi.h>
int tpgetorgnode(int clid)

```

- パラメータ

パラメータ	説明
clid	現在接続しているクライアントIDを設定します

- 戻り値

戻り値	説明
ノード番号	関数の呼び出しに成功した場合です。正常処理された場合、ノード番号を返します
-1	関数の呼び出しに失敗した場合です。tperrnoにはエラー番号が設定されません

- 例

```

#include <usrinc/tmaxapi.h>

SERVICE(TPSVCINFO *msg)
{
    int i, clid, n;
    char *nodename;
    ...
    for (i = 0; i < msg->len; i++)
        msg->data[i] = toupper(msg->data[i]);

    n = tpgetorgnode((int)*(msg->cltid.clientdata));
    if (n < 0){
        error processing
    }else {
        printf("original node number = %d\n", n);
    }
    ...
}

```

```

tpreturn(TPSUCCESS,0,(char *)msg->data, 0,TPNOFLAGS);
}

```

### 3.2.35. tpgetpeer\_ipaddr

サーバーで接続している相手方のソケット・アドレスを取得する関数です。Tmaxシステムに接続が完了後、相手方(ノード)のソケット・アドレスを返します。この関数は、クライアントが1つのサービスを直接呼び出した場合にのみ、相手方のIPアドレスを確認できます。

クライアントが呼び出したサービスが他のサービスを呼び出したり、他のサービスにサービスを転送させるなど、間接的に他のサービスを経る場合、ゴミ値が入ることがあります。

#### ● プロトタイプ

```

#include <tmaxapi.h>
int tpgetpeer_ipaddr(struct sockaddr *name, int *namelen)

```

#### ● パラメータ

パラメータ	説明
name	アドレスが保存された構造体です。IPv6プロトコル環境ではstruct sockaddr_in6構造体を使用してアドレス情報を確認します。また、struct sockaddr_storage構造体を使用すれば、IPv4とIPv6のどの環境でも使用可能です
namelen	関数を呼び出す前にnameに渡す構造体のサイズで初期化する必要があります。返却に成功すれば、nameに割り当てられた構造体の実際のサイズが保存されます

#### ● 戻り値

戻り値	説明
ソケットアドレス	関数の呼び出しに成功した場合です。相手方のソケット・アドレスが返されます
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

#### ● エラー

tpgetpeer\_ipaddr()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPINVAL]	引数が有効でない場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です



- 例

```
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>
...

DELETE(TPSVCINFO *msg)
{
    struct sockaddr_in cli;
    char ipAddr[16];
    int cli_len, ret;

    data process...
    memset((char *)&cli, 0, sizeof(cli));
    ret = tpgetpeer_ipaddr((struct sockaddr *)&cli, &cli_len);
    if (ret == -1){
        error processing
    }
    else {
        memcpy(ipAddr, inet_ntoa(cli.sin_addr), 16);
    }
    printf("ip = %s , port = %d\n", ipAddr, ntohs(cli.sin_port));
    data process...

    tpreturn(TPSUCCESS, 0, 0, 0, 0);
}
```

- 例 - IPv6プロトコル環境

```
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>
...

DELETE(TPSVCINFO *msg)
{
    struct sockaddr_storage cli_saddr;
    struct sockaddr_in *cli_sin4;
    struct sockaddr_in6 *cli_sin6;
    char ipaddrbuf[INET6_ADDRSTRLEN];
    const char *ipaddr = NULL;
    int cli_len, ret;
```

```

int portno;

data process...
memset((char *)&cli_saddr, 0, sizeof(cli_saddr));
cli_len = sizeof(cli_saddr);
ret = tpgetpeer_ipaddr((struct sockaddr *)&cli_saddr, &cli_len);
if (ret == -1){
    error processing
}
else {
    if (cli_saddr.ss_family == AF_INET) {
        cli_sin4 = (struct sockaddr_in *)&cli_saddr;
        ipaddr = inet_ntop(AF_INET, &(cli_sin4->sin_addr), ipaddrbuf,
            sizeof(ipaddrbuf));
        portno = ntohs(cli_sin4->sin_port);
    } else if (cli_saddr.ss_family == AF_INET6) {
        cli_sin6 = (struct sockaddr_in *)&cli_saddr;
        ipaddr = inet_ntop(AF_INET6, &(cli_sin6->sin6_addr), ipaddrbuf,
            sizeof(ipaddrbuf));
        portno = ntohs(cli_sin6->sin6_port);
    }
    if (ipaddr == NULL)
        ipaddr = "unknown";
}
printf("ip = %s , port = %d\n", ipaddr, portno);
data process...

tpreturn(TPSUCCESS, 0, 0, 0, 0);
}

```

- 関連関数

tpgetpeername(), tpgetsockname(), tpstart()

### 3.2.36. tpgetsvcname

サービス索引からサービス名を取得する関数です。応答メッセージの破棄時に呼び出されるLossサービスでTPSVCINFOのcltid.clientdata[3]のサービス索引値をパラメータとして使用します。この関数はTmaxサーバーでのみ使用することができ、Tmaxクライアントでは使用できません。返されるバッファは内部静的バッファであるため、バッファの内容を直接変更せずに、別のバッファにコピーして使用します。

- プロトタイプ

```

#include <tmaxapi.h>
char* tpgetsvcname(int svc_index)

```

- パラメータ

パラメータ	説明
svc_index	サービスの索引値です

- 戻り値

戻り値	説明
サービス名	関数の呼び出しに成功した場合です。サービス名が保存されたバッファのアドレス値を返します
NULL	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpgetsvcname()が実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	svc_indexの範囲が正しくない場合です
[TPENOENT]	該当するsvc_indexのsvc名が存在しない場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

SVC01(TPSVCINFO *msg)
{
    int i;

    printf("\nSVC01 service is started!\n");
    printf("INPUT : data=%s\n", msg->data);

    for (i = 0; i < msg->len; i++)
        msg->data[i] = toupper(msg->data[i]);

    printf("OUTPUT: data=%s\n", msg->data);
    sleep(10);
    tpreturn(TPSUCCESS, 0, (char *)msg->data, 0, 0);
}

LOSS_SVC(TPSVCINFO *msg)
{
    long svcindex;
```

```

char *svcname;

printf("\nLOSS_SVC service is started!\n");
printf("INPUT : data = %s\n", msg->data);
printf("TPERROR : %d\n", msg->cltid.clientdata[1]);
printf("TPURCODE : %d\n", msg->cltid.clientdata[2]);

svcindex = msg->cltid.clientdata[3];
printf("SVC INDEX Of Discarded Message : %ld\n", svcindex);

svcname = tpgetsvcname((int)svcindex);
if(NULL == svcname) {
    printf("tpgetsvcname is failed!!\n");
} else {
    printf("SVC Name Of Discarded Message : %s\n", svcname);
}
tpreturn(TPSUCCESS, 0, (char*)msg->data, 0, 0);
}

```

### 3.2.37. tpgetsvrseqno

同じサーバー・プロセス間のサーバー・プロセスのシリアル番号を返す関数です。環境設定ファイルにあるSERVERセクションのMIN、MAX項目と関連して、1つ以上動作するサーバー・プロセスを区分するのに使用されます。返される値は0からであり、動作するサーバー・プロセス数に従って1ずつ増加した値が使用されます。

たとえば、「svr1」という名前のサーバー・プロセスが5つ動作している場合、それぞれ0から4までのシリアル番号が付与されます。3番に該当するサーバー・プロセスが終了した場合、**tmboot -s**や**autorestart!**によって再起動された際に3番をもつことになります。

#### ● プロトタイプ

```

#include <tmaxapi.h>
int tpgetsvrseqno(void)

```

#### ● 戻り値

戻り値	説明
0以上の整数	関数の呼び出しに成功した場合です。サーバー・プロセスのシリアル番号に該当する0以上の整数値を返します
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

#### ● エラー

tpgetsvrseqno()が実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

SERVICE(TPSVCINFO *msg)
{
    char *buf;
    printf("\nINPUT : %s\t processing svr no : %d\n", msg->data, tpgetsvrseqno());

    data process....
    tpreturn(TPSUCCESS, buf, 0, );
}
```

### 3.2.38. tpissetfd

UCSプロセスでソケットFDにデータが到着したかどうかをチェックする関数です。UCS方式サーバー・プロセスの外部ソケットのスケジューリングに使用されます。

- プロトタイプ

```
#include <ucs.h>
int tpissetfd (int fd)
```

- パラメータ

パラメータ	説明
fd	テストするfdset内部のFDを設定します

- 戻り値

戻り値	説明
正数	メッセージが到着した場合です
0	メッセージが到着しなかった場合です

戻り値	説明
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tpissetfd()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <usrinc/ucs.h>
...

#define SERV_ADDR "168.126.185.129"
#define SERV_PORT 1500

int fd_read(int, char *, int);
extern int errno;

int usermain(int argc, char *argv[])
{
    ...
    int listen_fd, n, newfd;
    struct sockaddr_in my_addr, child_addr;
    socklen_t child_len;

    buf = tpalloc("STRING", NULL, 0);
    if (buf == NULL){ error processing }

    memset((void *)&my_addr, NULL, sizeof(my_addr));
    memset((void *)&child_addr, NULL, sizeof(child_addr));

    listen_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (listen_fd == -1){ error processing }

    my_addr.sin_family = AF_INET;
    inaddr = inet_addr(SERV_ADDR);
```

```

my_addr.sin_port = htons((unsigned short)SERV_PORT);
if (inaddr != -1)
    memcpy((char *)&my_addr.sin_addr, (char *)&inaddr, sizeof(inaddr));

ret = bind(listen_fd, (struct sockaddr *)&my_addr, sizeof(my_addr));
if (ret == -1){ error processing }
ret = listen(listen_fd, 5);
if (ret == -1){ error processing }

tpsetfd(listen_fd);
...

while(1) {
    n = tpschedule(10);
    ...
    if (n == UCS_USER_MSG){
        if (tpissetfd(listen_fd)) {
            child_len = sizeof(child_addr);
            newfd = accept(listen_fd, &child_addr, &child_len);
            if (newfd == -1){ error processing }
            tpsetfd(newfd);
        }

        if (tpissetfd(newfd)){
            /*ソケットからバッファを読み込みます*/
            fd_read(newfd, buf, 1024);
            ret = tpcall("SERVICE", (char *)buf, sizeof(buf), (char **)&buf,

                        (long *)&rlen, TPNOFLAGS);
            if (ret == -1){ error processing }
            ...
            tpclrfd(newfd);
            close(newfd);
        }
        ...
    }
}
return 1;
}

```

- 関連関数

tpissetfd(), tpsetfd()

### 3.2.39. tpissetfd\_w

UCS方式サーバー・プロセスのFDSETをチェックして、パラメータ値として与えられたソケットFDに送信するデータがあるかどうかを確認する関数です。tpissetfd()関数が読み取り可能なFDSETをチェックする一方、この関数は書き込み可能なFDSETをチェックします。UCS方式プロセスの外部ソケットのスケジューリングに使用されます。FDは書き込み可能なFDSETでチェックするソケットFD値です。

- プロトタイプ

```
#include <ucs.h>
int tpissetfd_w(int fd)
```

- パラメータ

パラメータ	説明
fd	テストするfdset内部のFDを設定します

- 戻り値

戻り値	説明
正数	メッセージが到着した場合です
0	メッセージが到着しなかった場合です
-1	関数の呼び出しに失敗した場合です。tpernolにエラーコードが設定されます

- エラー

tpissetfd()が実行されなかった場合、tpernolに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	関数の呼び出しに成功した場合です
[TPEOS]	運用システムにエラーが発生した場合です

- 関連関数

tpissetfd(), tpsetfd()

### 3.2.40. tpprechk

RMの状態をチェックするためのユーザー・コールバック関数であり、Tmaxシステムの接続前に呼び出されます。RMの状態をチェックする用途でのみ使用し、それ以外の初期化過程はtpsvrinit()で処理する必要があります。



あります。tpprechk()でブロック状態になった場合、サービスはREADY、サーバーはNOREADY状態になります。tmdownの[-i]オプションを使用してプロセスを終了させることができます。

tpsvrinit()でRM状態を確認中にブロック状態に陥った場合、サービス・スケジューリングになり、ユーザーはサービス・タイムアウトまで待機します。こういった問題を解決するには、tpprechk()を使用する必要があります。

- プロトタイプ

```
#include <tmaxapi.h>
int tpprechk(void)
```

- 戻り値

戻り値	説明
正数	関数の呼び出しに成功した場合です
負数	関数の呼び出しに失敗した場合です。該当サーバーを再起動します

## 3.2.41. tpregcb

サーバーでUCSの非同期型の要求に対する応答を受けるルーチンを設定する関数です。UCS方式プロセスがサーバー・プログラムから応答を受けたときに、それを処理するルーチンを設定します。UCS方式サーバー・プロセスに、tpgetreply()の代わりに使用されます。

- プロトタイプ

```
# include <ucs.h>
int tpregcb (UcsCallback)
```

- パラメータ

パラメータ	説明
UcsCallback	UCSで非同期型要求に対する応答を処理するコールバック関数を指定します

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tpregcb()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/ucs.h>

void reply_receive(UCSMSGINFO *reply);
DUMMY(TPSVCINFO *msg)
{
    data process ....
}

int usermain(int argc, char *argv[])
{
    int ret;
    char *buf

    ret = tpregcb(reply_receive);
    if (ret == -1){ error processing }
    buf=tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
    data process...
    while(1)
    {
        ...
        tpschedule(3);
        cd = tpacall("SERVICE", buf, strlen(buf), TPNOFLAGS);
        if (cd < 0) { error processing }
        ...
    }
}

void reply_receive(UCSMSGINFO *reply)
{
    printf("data....%s\n", reply->data);
}
```

- 関連関数

tpunregcb()

## 3.2.42. tprelay

UCS形式のサーバー・プロセスでのみ使用できる関数です。サービスを要求したクライアントの情報を保存し、他のサービスを要求する方式で、マルチノード間でもサービスが行われます。この形式をとった場合、tprelay()を使用して呼び出されたサービスは、クライアントが自身を直接呼び出したものと認識します。また、サービスの実行結果は、最初にサービスを要求したクライアントに渡されます。

サービスの実行結果を呼び出し元のクライアントに渡すことができるため、UCSプロセスで簡単な構成を通じて、速い応答を誘導できます。通常、結果を取得するまでに2～3回サービスの呼び出しを行わなければならないプログラム・ルーチンの対外機関業務と連動してサービス进行处理する場合に使用するのが望ましいです。

もしtpsavctx()またはtpgetctx()によりクライアント情報を保存した後、tprelay()により他のサービスを要求していない状態でサーバー・プロセスが終了する場合には、自動でサービス呼び出し元にエラー応答が返されます。エラー応答の返却に関連しては、環境設定のSERVERセクションのCTX\_EREPYオプションを参照してください。このような動作は、Tmax v5.0 SP2以降バージョンからサポートされます。

### ● プロトタイプ

```
#include <ucs.h>
int tprelay(char *svc, char *data, long len, long flags, CTX_T *ctxp);
```

### ● パラメータ

パラメータ	説明
svc	Tmax環境ファイルに登録されたサービス名を指定します
data	サービスの呼び出し時に渡されるデータです。NULLでない場合、必ずtpalloc()で割り当てられたバッファを使用します
len	送信するデータ長を指定します。CARRAY、X_OCTET、構造体の配列タイプの場合は必ず設定します
flags	現在サポートしていないパラメータで、TPNOFLAGSを設定します
ctxp	tpgetctx()あるいはtpsavctx()で取得した情報の構造体です

### ● 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

### ● エラー

tprelay()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、ctxpがNULLの場合、あるいは使用したバッファが正しくない場合に発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です

● 例

```

...
#include <stdio.h>
#include <usrinc/ucs.h>
CTX_T *ctx = NULL;

DUMMY(TPSVCINFO *msg)
{
    data process ...
}

usermain(int argc, char *argv[])
{
    int ret, i;
    char *rcvbuf, *sndbuf;
    long sndlen;

    rcvbuf = (char *)tpalloc("CARRAY",NULL, 1024);
    if (rcvbuf == NULL){ error processing }
    i = 0;

    while(1) {
        tpschedule(1);
        if (ctx != NULL)
        {
            i++;
            if ((sndbuf = (char *)tpalloc("CARRAY",NULL, 1024)) == NULL)
            { error processing }
            else
            {
                ...
                ret = tprelay("TPRETURN", sndbuf, sndlen, 0, ctx);
                if (ret==-1) { error processing }
                data process...
                ctx = NULL;
                tpfree(sndbuf);
            }
        }
    }
}

```

```
int RELAY(TPSVCINFO *rqst)
{
    ...
    ctx = tpsavectx();
    tpreturn(TPSUCCESS, 0, rqst->data, rqst->len, 0);
}
```

- 関連関数

tpreturn(), tpforward()

### 3.2.43. tpresumetx

現在停止しているグローバル・トランザクションを再開する関数です。停止しているグローバル・トランザクションを、**tpresumetx()**、**tpsuspendtx()**を使用して再開できます。

- プロトタイプ

```
#include <usrinc/tmaxapi.h>
int tpresumetx(TRANID *tranid, int flags)
```

- パラメータ

パラメータ	説明
tranid	tranidは、TRANID構造体のポインターである必要があり、 <b>tpresumetx()</b> 、 <b>tpsuspendtx()</b> の呼び出し時に使用した構造体のポインターと同じである必要があります
flags	現行バージョンではサポートしていませんが、TPNOFLAGSに設定します

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpresumetx()が実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	tranidがNULLで入力された場合や、flagsがTPNOFLAGSでない場合に発生します

エラーコード	説明
[TPEPROTO]	トランザクションをレジュームできる状況でない場合です
[TPETTRAN]	トランザクションの再開に失敗した場合です

- 関連関数

tpsuspendtx()

## 3.2.44. tpreturn

サーバーのサービスを終了する関数で、サービス・ルーチンの完了を意味します。C言語でのRETURN文と同じ役割をします。tpreturn()が呼び出された場合、サービス・ルーチンはTmaxシステムに返されます。Tmaxシステムに正常に返されるためには、tpreturn()はTmaxシステムが制御するサービス・ルーチン内で呼び出すのが望ましいです。

tpreturn()関数はサービスの応答メッセージを送信します。応答を受信するプログラムが、**tpcall()**、**tpgetrply()**、**tprecv()**で応答を待機している場合、その応答はtpreturn()の呼び出しが成功後、受信者のバッファーを通じて渡されます。

tpreturn()は、対話型サービスが対話型接続を終了できるようにします。サービス・ルーチンは、tpdiscon()を直接呼び出すことができません。正しい結果を保証するには、対話型サービスに接続されたプログラムはtpdiscon()を呼び出さないのが望ましいです。それより、対話型サービスでの完了通知、つまりtpreturn()関数を使用して送信されるTPEV\_SVCSUCCやTPEV\_SVCFAILのようなイベントを待機する必要があります。

サービス・ルーチンがトランザクション・モードであるが、該当サービスを呼び出したクライアントあるいはサービスが明示的にトランザクションを開始しない場合(tx\_beginを使用しない場合)、tpreturnはトランザクションの一部として、TPSUCCESSの場合はコミットあるいはロールバックされます。サービスは同一トランザクション(グローバル・トランザクション)の一部として、複数回呼び出されることもあります。そのため、tx\_beginを使用したトランザクション開始者が、tx\_commitまたはtx\_rollbackのうち1つを呼び出してトランザクションを完了するまでは、完全にコミットあるいはロールバックされません。

tpreturn()関数は、サービス・ルーチンで要求されたサービスからすべての応答を受信後に呼び出されます。そうでない場合、サービス特性に従って、[TPESVCERR]エラーあるいはTPEV\_SVCERRイベントがサービス・ルーチンと通信するプログラムに返されます。受信されていない応答は、Tmaxシステムによって自動的に無視されます。また、このような応答に使用される記述子は無効化されます。

tpreturn()関数は、対話型通信に使用されるサービスで開始されたすべての接続の終了後に呼び出されます。そうでない場合、サービス特性に従って[TPESVCERR]エラーあるいはTPEV\_SVCERRイベントのうち1つがサービス・ルーチンと通信するプログラムに返されます。また、強制的な接続解除イベント(TPEV\_DISCONIMM)が、サービスと接続されたすべての接続従属者に渡されます。

対話型通信で、サービス・ルーチンがtpreturn()の呼び出し時に通信制御権をもっていない場合、2つの結果が発生することができます。

1. サービス・ルーチンがTPFAILのrvalとNULLのdataでtpreturn()を呼び出した場合、対話開始者にTPEV\_SVCFAILイベントが渡されます。
2. これと異なる形式のtpreturn()が呼び出された場合、対話開始者にTPEV\_SVCERRイベントが渡されます。対話型サービスは、サービスで開始していない対話型接続を1つのみもっているため、Tmaxシステムでデータやイベントが送信されるべき記述子を認識しています。したがって、記述子はtpreturn()にパラメータで渡されません。

#### ● プロトタイプ

```
# include <atmi.h>
void tpreturn (int   rval, long   rcode, char   *data, long   len, long   flags)
```

#### ● パラメータ

パラメータ	説明
rval	rvalに設定可能な値は、下の表で説明します。説明に存在しないrval値は、すべてTPFAILと見なされます
rval	<p>以下は、rvalで使用可能な値です。</p> <p>– TPSUCCESS</p> <p>サービスが正常に終了した場合です。データが存在し、tpreturn()の実行中にエラーが発生しなければ、データは送信されます。呼び出し元がトランザクション・モードの場合、このトランザクションの一部をコミットが可能な状態に決定します。トランザクションが最終的に完了するとき、該当トランザクションに属する残りのサービスがすべて正常に完了され、コミットが可能な状態であればコミットし、1つでも失敗した場合はロールバックされます。tpreturn()に対する呼び出しは、必ずしも全体トランザクションを完了することではないことに留意してください。また、呼び出し元がTPSUCCESSで返しても、待機中の応答や対話型接続が存在するか、あるいはサービス内で行った作業がトランザクションをロールバックされるようにした場合、サービス失敗でメッセージが送信されます。応答の受信者が[TPESVCERR]表示、またはTPEV_SVCERRイベントを受信します。サービス・ルーチン内でトランザクションがロールバックされた場合、rvalはTPFAILに設定されることに留意してください。対話型サービスでTPSUCCESSで返された場合、TPEV_SVCSUCCイベントが発生します</p> <p>– TPFAIL</p> <p>サービスがアプリケーションの失敗によって終了しました。応答を受信するプログラムにエラーが返されます。応答を受信する呼び出しが失敗し、受信者は[TPSVCFAIL]値あるいはTPEV_SVCFAILイベントを受信します。この値はデータを送信できません。TPFAILの呼び出し元がトランザクション・モードかつautotransactionの場合、tpreturn()はトランザクションをロールバックします。トランザクションがすでにロールバック状態と決定されていることもあります</p>

パラメータ	説明
	<ul style="list-style-type: none"> <li>– TPEXIT サービスを呼び出した後返す際に、サーバー・プロセスを強制終了しようとする場合に使用されます。tpexit()で終了したプロセスは、TMMIによって再起動されます</li> <li>– TPDOWN TPEXITと似ていますが、TPDOWNで終了したプロセスは、TMMIによって再起動されません</li> <li>– TMSUCCESS TPSUCCESSと同じです</li> <li>– TMFAIL TPFAILと同じです</li> </ul> <p>説明に存在しないrval値はすべてTPFAILにみなされます</p>
rcode	<p>アプリケーション上でユーザーによって定義された戻り値のrcodeは、サービス応答を受信するプログラムに送信されます。このコードはrvalの値と関係なく、応答がクライアントへ無事に送信できる限り(受信する呼び出しが成功するか、[TPSVCFAIL]で返すか、あるいはTPEV_SVCSUCCまたはTPEV_SVCFAILイベントのうち1つを受信する限り)送信されます。</p> <p>rcode値は、受信者にグローバル変数のtpurcodeで渡されます</p>
data	<p>送信される応答データを指します。NULLでない場合、以前にtpalloc()によって割り当てられたバッファを指す必要があります。サービス・ルーチンに渡されたものと同じバッファの場合、Tmaxシステムで処理を行います。</p> <p>したがって、サービス・ルーチンの作成者は、このバッファの削除可否を気にする必要はありません。実際にユーザーがこのバッファを削除しようとした場合は失敗します。しかし、tpreturn()で渡されるバッファがサービスの発生時と同じバッファでない場合、tpreturn()はこのバッファを解除します</p>
len	<p>送信されたデータ長です。</p> <p>dataが指すバッファが、長さを指定する必要がないタイプの場合、lenは無視され、一般的に0が使用されます。dataが指すバッファが、長さを指定する必要がない場合、lenは0になれません。dataがNULLの場合、lenは無視されます。この際、サービスを呼び出したプログラムが応答を期待している場合、データが何もない応答が送信されます。応答が期待されていない場合、tpreturn()は必要に応じてdataを削除し、送信する応答なしで返します</p>
flags	<p>使用されていません。必ず0に設定します。</p>



パラメータ	説明
	<p>サービスが対話型の場合、データが渡されないケースは、次の2つがあります</p> <ul style="list-style-type: none"> <li>– tpreturn()の呼び出す時、対話型接続がすでに終了された場合です。呼び出し元がTPEV_DISCONIMMイベントを受信しました。この場合、tpreturn()はサービス・ルーチンを終了し、トランザクション・モードの場合、現行トランザクションをロールバックします。この場合、呼び出し元のデータは渡せません</li> <li>– 呼び出し元が通信制御権をもっていない場合、上述したように、TPEV_SVCFAILあるいはTPEV_SVCERRのうち1つが対話開始者に送信されます。対話開始者が受信するイベントに関係なく、いかなるデータも渡されません。しかし、対話開始者がTPEV_SVCFAILイベントを受信した場合、リターン・コードは開始者のtpurcode変数で利用できます</li> </ul>

#### ● 戻り値

サービス・ルーチンは、呼び出し元のTmaxシステムにいかなる値も返しません。サービス・ルーチンはtpreturn()を使用して終了することが原則です。サービス・ルーチンがtpreturn()を使用しない場合（たとえば、C言語のRETURN文などを使用して返す場合）、サーバーはサービス要求者にサービス・エラーを返します。対話型通信のために維持されている接続が強制的に終了され、非同期的に待機している応答はすべて無視されます。

サーバーがトランザクション・モードの場合、そのトランザクションはロールバックされます。また、tpreturn()がサービス・ルーチンの外部で使用された場合（たとえば、サービスでないルーチンで使用された場合）、これは何の処理もせずにとただ返すだけです。

#### ● エラー

tpreturn()がサービス・ルーチンを終了させるため、パラメータの処理中にエラーが発生した場合は、呼び出し元のサービス・ルーチンに渡されません。エラーは以下のように渡されます。

区分	説明
同期通信と非同期通信	tpcall()またはtpgetrply()でサービス結果を受信するプログラムに対しては、tperrnoに[TPEV_SVCERR]が渡されます
対話型通信	tpsend()やtprecv()を使用するプログラムに対しては、TPEV_SVCERRイベントを発生させます

#### ● 例

```
#include <stdio.h>
#include <usrinc/atmi.h>

SERVICE1(TPSVCINFO *msg)
{
```

```

char *buf;
long len;

buf=tpalloc("STRING", NULL, 0);
if (buf==NULL) { error processsing }
buf=msg->data;
data process....

ret=tpcall("SERVICE2", buf, sizeof(buf), &buf, &len, TPNOFLAGS);
if (ret==-1) { error processing }
data process....

if (buf != "SUCCESS") {
    printf("svc fail..\n");
    tpreturn (TPFAIL, -1, NULL, 0, 0);
}
else {
    tpreturn(TPSUCCESS, 0, msg->data, msg->len, 0);
}
}

```

- 関連関数

tpalloc(), tpcall(), tpconnect(), tpdison(), tpgetrply(), tprecv(), tpsend()

### 3.2.45. tpsavectx

UCSプロセスで使用され、クライアントの情報を内部的に管理する関数です。tpsavectx()はtprelay()関数と一緒に使用されます。tprelay()は、処理結果を他のサービスに渡す役割をします。一般的なサービス・プログラムでtpforward形式で別のサービスを呼び出したのと同じように動作します。結果的に呼び出されたサービスでは、処理された結果値を該当クライアントに渡します。

tpsavectx()関数は対外機関業務と同様に他のプロトコルが利用され、時間が掛かるためにチャンネルが結合される可能性が多い場合に使用されます。

一般的に使用される形式は以下のとおりです。

```

クライアント → svc1 → svc2(service, tpsavectx) → 対外機関
クライアント ← svc3 ← svc2(usermain, tprelay) ← 対外機関

```

1. クライアントがsvc1にサービスを要求します。
2. svc1はtpforward(...TPNOREPLY)を使用してsvc2を呼び出します。

3. svc2はUCSプロセスに存在するサービスで、サービス・ルーチン内でtpsavectx()を使用してクライアントの情報を保存し、対外機関と通信します。
4. 結果はusermainで受信し、tprelay()を使用してsvc3に渡します。svc3は、svc2でtpforwardを使用して自身を呼び出したものと見なし、最終結果をクライアントに渡します。

svc1でTPNOREPLYを使用してサービスを渡しているため、チャンネルが結合されるのを防ぐことができ、少数の業務プロセスでも多くのクライアントを処理できます。また、1つのUCSプロセスとして送受信プロセスの役割を兼ねることで、比較的単純なシステムを構成できます。これはシステム管理面においても効率的です。

#### ● プロトタイプ

```
#include <ucs.h>
CTX_T * tpsavectx(void)
```

#### ● 戻り値

戻り値	説明
CTX_T	関数の呼び出しに成功した場合です
NULL	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

#### ● エラー

tpsavectx()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEPROTO]	tpsavectx()関数は、必ずサービス・ルーチン内で使用します。サービス・ルーチンでない位置で使用された場合、TPEPROTOエラーが発生します。 <b>tpsvrinit()</b> あるいは <b>tpsvrdone()</b> では使用できません
[TPESYSTEM]	メモリーの割り当てエラーが発生した場合です

#### ● 例

```
...
#include <stdio.h>
#include <usrinc/ucs.h>

CTX_T *ctx = NULL;

usermain(int argc, char *argv[])
{
    int ret, i;
    char *rcvbuf, *sndbuf;
```

```

long sndlen;

rcvbuf = (char *)tpalloc("CARRAY",NULL, 1024);
if (rcvbuf == NULL){ error processing }

i = 0;

while(1) {
    tpschedule(1);
    if (ctx != NULL)
    {
        i++;
        if ((sndbuf = (char *)tpalloc("CARRAY",NULL, 1024)) == NULL)
        { error processing }
        else
        {
            ...
            ret = tprelay("TPRETURN", sndbuf, sndlen, 0, ctx);
            if (ret==-1) { error processing }
            data process...
            ctx = NULL;
            tpfree(sndbuf);
        }
    }
}

int RELAY(TPSVCINFO *rqst)
{
    ...
    ctx = tpsavctx();
    tpreturn(TPSUCCESS, 0, rqst->data, rqst->len, 0);
}

```

- 関連関数

tpreturn(), tpforward(), tprelay()

### 3.2.46. tpschedule

UCS形式のサーバー・プロセスでのみ使用される関数で、UCSサーバー・プロセスでデータの到着を待機します。最大タイムアウト時間の間待機し、その間にデータが到着した場合は即時に返します。

tpschedule()関数は、データの到着時に該当するサービスが自動的に実行された後に返されます。そのため、データが到着後にユーザーが任意でサービスを実行してはいけません。

---

## 注

サービスは常にシステムによって実行されるため、UCS形式のサービス・プログラムでもこの点は注意してください。

---

## ● プロトタイプ

```
#include <ucs.h>
int tpschedule(int timeout)
```

## ● パラメータ

パラメータ	説明
timeout	待機する時間を秒単位で入力します – -1: データが到着したかどうかをチェックのみ行った後、すぐに返します – 0: データが到着するまで無限に待機します

## ● 戻り値

戻り値	説明
正の整数	関数の実行に成功してデータが到着した場合です
-1	タイムアウト時間までにデータが到着していない場合です。あるいは、関数が実行に失敗してエラーが発生した場合です。タイムアウト時間までデータが到着しなかった場合は-1を返し、tperrnoに13番(TPETIME)が設定されます。それ以外の場合、tperrnoにエラーコードが設定されます

## ● エラー

tpschedule()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です
[TPETIME]	タイムアウト時間までにデータが到着していない場合です

## ● 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/ucs.h>
```

```

int usermain(int argc, char *argv[])
{
    ...
    while(1)
    {
        ...
        tpschedule(3);
        ret = tpcall("SERVICE", (char *)buf, strlen(buf), (char **)&buf,
                    (long *)&rlen, TPNOFLAGS);
        if (ret == -1) { error processing}
        ...
    }
}

```

- 関連関数

tpsleep(), tp\_sleep(), tp\_usleep()

## 3.2.47. tpsendtocli

サーバーで使用される関数で、指定されたクライアントに非要求メッセージを送信します。**tpbroadcast()**は、Tmaxシステムに接続されている任意のクライアントに非要求メッセージを送信します。tpsendtocli()は、サーバー・プロセスで該当サーバー・プロセスが提供するサービスを要求したクライアントにのみ非要求メッセージを送るために使用します。

- プロトタイプ

```

#include <tmaxapi.h>
int tpsendtocli (int clid, char *data, long len, long flags)

```

- パラメータ

パラメータ	説明
clid	tpgetclid()で取得したクライアントの一意の番号です
data	tpalloc()によって割り当てられたバッファーです。dataが指すバッファーが、長さを指定する必要がないタイプの場合、lenは無視され、一般的に0が使用されます。dataが指すバッファーが、長さの指定が必要なタイプの場合、lenは0になれません。dataがNULLの場合、lenは無視されます
len	送信するバッファー長です
flags	flagsで使用可能な値は以下のとおりです – TPNOFLAG(0)

パラメータ	説明
	<p>メッセージはクライアントが受信します。しかし、クライアントが受信したメッセージを早急に処理できなければ、要求した結果の受信に時間が掛かることがあります</p> <p>– TPUDP</p> <p>TPUDPフラグは、クライアントとデータを通信する方式がUDPという意味ではありません。呼び出し元がデータを送信する際、送信する内部バッファに渡されるメッセージがいっぱいになって送信できない場合があります。この場合、データは捨てても構わないという意味です。通信のUDPのように、データが途中で紛失することがあります</p> <p>– TPFLOWCONTROL</p> <p>クライアントの状態をチェックし、他のメッセージを要求できるかどうかを確認します。該当クライアントの送信メッセージが多く蓄積している場合、tpsendtcli()は-1を返し、tperrnoをTPEQFULLに設定します。このフラグは、Tmaxシステムの負荷を減らします</p>

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpsendcli()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEBADDESC]	clidが有効でない場合です
[TPEPROTO]	tpsendtcli()が不適切な状況で呼び出された場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です
[TPEQFULL]	送信メッセージがあるため、同じメッセージである場合は再送信する必要があります

- 例

```

#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

SERVICE(TPSVCINFO *msg)
{
    int ret, clid;
    char *buf;

    buf = (char *)tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
    strcpy(buf, msg->data);
    data process...
    clid = tpgetclid();
    if (clid==--1) { error processing }

    ret=tpsendtocli(clid, (char *)buf, 0, 0);
    if (ret==--1) { error processing }
    tpreturn(TPSUCCESS, 0, 0, 0);
}

```

- 関連関数

tpbroadcast()

### 3.2.48. tpsetdbsessionid

RMのセッション情報をサービスで処理する場合、毎回処理を必要とする手間をなくすために、RMセッション情報を設定するコールバック関数です。

- プロトタイプ

```

#include <usrinc/tmaxapi.h>
int tpsetdbsessionid (char dbsessionid[MAX_DBSESSIONID], int flags);

```

- パラメータ

パラメータ	説明
dbsessionid	開発者は、tpsetdbsessionidルーチンを任意で作成し、取得したID値をdbsessionid変数に保存します。Tmaxエンジンでは、RMと新規接続する度に、ユーザーが作成したtpsetdbsessionidルーチンを呼び出して最新値に更新し、内部的に管理します
flags	現行バージョンではサポートしていませんが、TPNOFLAGSに設定します

- 例



```

#include <usrinc/tmaxapi.h>
...
EXEC SQL include sqlca.h;
EXEC SQL begin declare section;
...
char h_ssid[20];
EXEC SQL end declare section;

int tpsetdbsessionid(char dbsessionid[MAX_DBSESSIONID_SIZE], int flags)
{
    EXEC SQL SELECT TO_CHAR(USERENV('sessionid')) into :h_ssid FROM dual;
    if ( sqlca.sqlcode != 0 ){
        printf( "getting session id fail = %d\n",sqlca.sqlcode );
        return -1;
    }
    printf("RM session id = %s\n", h_ssid);

    memset(dbsessionid, 0x00, MAX_DBSESSIONID_SIZE);
    strcpy(dbsessionid, h_ssid);
    return 0;
}

FDLINS( TPSVCINFO *msg )
{
    ...
}

```

### 3.2.49. tpsuspendtx

既存のグローバル・トランザクションを停止する関数です。該当関数を使用した場合、グローバル・トランザクションが実行中の状況で新規トランザクションを開始する時に、現在実行中のグローバル・トランザクションを停止できます。停止されたトランザクションのリソースは、継続してACTIVE状態で存在します。たとえば、トランザクション・タイムアウトが発生した場合、該当トランザクションはロールバック処理されます。停止されたトランザクションは、tpresumetx()、tpsuspendtx()を使用して再開することができます。

- プロトタイプ

```

#include <usrinc/tmaxapi.h>
int tpsuspendtx(TRANID *tranid, int flags)

```

- パラメータ

パラメータ	説明
tranid	開発者が割り当てべき領域で、TRANIDの構造体のポインターである必要があり、現在のトランザクションIDが設定されます
flags	現行バージョンではサポートしていませんが、TPNOFLAGSに設定します

# - 例

```

SVC_TRAN( TPSVCINFO *msg )
{
    str  sndbuf;
    char *rcvbuf;
    char tmp[4096];
    int  cd, ret;
    long rlen;

    /*****
    TSR(Transaction Suspend and Resume) - TPTRANID Structure
    *****/
    TPTRANID *tranid = (TPTRANID *)malloc(sizeof(struct tptranid));
    sndbuf = (str)msg->data;
    rcvbuf=(char *)tpalloc("STRING", NULL, 4096);
    h_empno = sndbuf->empno;
    h_sal   = sndbuf->sal;
    strcpy( h_ename, sndbuf->ename );
    strcpy( h_job   , sndbuf->job   );
    strcpy( h_date  , sndbuf->date  );

    ret = tx_begin();
    cd = tpcall("ORGTRAN1", (char *)msg->data, msg->len, (char **)&rcvbuf,
                (long *)&rlen, 0);
    strcpy(tmp, rcvbuf);

    /*****
    TSR(Transaction Suspend and Resume) - Suspend is Started
    *****/
    Ret = tpsuspendtx(tranid, 0)
    cd = tpcall("NEWTRAN", (char *)msg->data, msg->len, (char **)&rcvbuf,
                (long *)&rlen, 0);
    strcat(tmp, rcvbuf);

    /*****
    TSR(Transaction Suspend and Resume) - Resume is Started
    *****/
    ret = tpresumetx(tranid, 0);
    cd = tpcall("ORGTRAN2", (char *)msg->data, msg->len, (char **)&rcvbuf,
                (long *)&rlen, 0);

```

```

    ret = tx_commit();
    tpreturn( TPSUCCESS, 0, rcvbuf, strlen(rcvbuf), 0 );
}

```

### 3.2.50. tpsvctimeout

サービス・タイムアウトが発生した場合に呼び出されるルーチンです。サービス・タイムアウトが発生した場合、サーバープログラムは自動的にtpsvctimeout()を呼び出します。ユーザーが関数を再定義した場合、再定義した関数を呼び出します。

サービスを実行中にtx\_commit()、tx\_rollback()関数を実行する場合、内部でxa\_commit()、xa\_rollback()段階でサービス・タイムアウトが発生すると、それを区別するためにtperrnoにTPETRANを設定して、tpsvctimeout関数内で参照できるようにします。

- プロトタイプ

```

#include <tmaxapi.h>
void tpsvctimeout(TPSVCINFO *msg)

```

- パラメータ

パラメータ	説明
msg	タイムアウトが発生したサービスの呼び出し時に使用したメッセージです

- 戻り値

tpsvctimeout()は、サービスのタイムアウトが発生した場合、開発者が必要な作業を実行するように作成する関数です。戻り値はなく、エラーも発生しません。

- 例

```

#include <stdio.h>
#include <usrinc/atmi.h>
SERVICE(TPSVCINFO *msg)
{
    ...
    tpreturn(TPSUCCESS, 0, (char *)msg->data, 0, TPNOFLAGS);
}

void tpsvctimeout(TPSVCINFO *msg)
{
    ...
    tpreturn(TPFAIL, TPETIME, (char *)buf, sizeof(buf), TPNOFLAGS);
}

```

## 3.2.51. tpsvrdone

UCS方式サーバー・プロセスを終了する関数です。Tmaxアプリケーション・サーバー・プログラムの分離されたmainは、サービス要求処理をすべて終え、プロセスを終了する前にtpsvrdone()を呼び出します。ルーチンの実行時、そのサーバー・プロセスはシステムの一部ではありますが、サービスはサポートされません。tpsvrdone()ルーチン内でTmax通信が実行されたり、トランザクションが定義されることもあります。

tpsvrdone()が対話型接続を維持している場合は、Tmaxは対話型接続を終了します。非同期性応答を待機している場合は、待機していた非同期性応答を無視します。トランザクション・モードの間はトランザクションを停止し、サーバーはすぐに終了されます。

アプリケーションでtpsvrdone()ルーチンを提供しない場合、Tmaxが提供するデフォルト・ルーチンが代わりに呼び出されます。デフォルトのtpsvrdone()は、トランザクションを処理するサーバー・グループに含まれたサーバーであれば、**tx\_close()**と**userlog()**を呼び出して、サーバーがすぐに終了することを通知します。tpreturn()とtpforward()のうち1つがtpsvrdone()内で呼び出された場合、このようなルーチンは何も作動せず、返すだけです。

- プロトタイプ

```
# include <tmaxapi.h>
int tpsvrdone(void)
```

- 戻り値

tpsvrdone()は、開発者がサーバー・プロセスの終了を実行する前に必要な作業を実行するように作成する関数です。戻り値はなく、エラーも発生しません。

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>

EXEC SQL INCLUDE sqlca.h;

SERVICE(TPSVCINFO *msg)
{
    int ret, cd;
    char *buf;

    EXEC SQL begin declare section;
    ...
    EXEC SQL end declare section;
    EXEC SQL CONNECT : scott IDENTIFIED BY : tiger;
    buf=tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
    data process....
```

```

    cd=tpgetcli();
    if (cd==-1) { error processing }
    ret=tpsendtocli(cd, buf, 0, TPNOFLAGS);
    if (ret==-1) { error processing }
    data process....

    tpsvrdone();
}

void tpsvrdone()
{
    printf(" Sevice end\n");
    EXEC SQL COMMIT WORK RELEASE;
}

```

- 関連関数

tx\_close(), tpsvrinit()

### 3.2.52. tpsvrdown

UCS方式サーバー・プロセスを正常終了させる関数です。UCS方式サーバー・プロセスは、一般的に外部システムとの通信に多く使用されます。代表的な例は、銀行システム間の通信です。何らかの問題が発生し、外部システムがサービス要求を処理できないような状況を避ける方法が必要です。

tpsvrdown()は、UCS方式のサーバー・プロセスを正常終了させるため、障害が発生した外部システムにそれ以上サービスを要求しません。したがって、リソースを節約できるだけでなく、外部システムの負荷も減らすことができます。

- プロトタイプ

```

#include <ucs.h>
int tpsvrdown(void)

```

- 戻り値

tpsvrdown()は、開発者がサーバー・プロセスの終了を実行する前に必要な作業を実行する関数です。そのため、戻り値がなく、エラーも発生しません。

- 例

```

...
#include <usrinc/atmi.h>
#include <usrinc/ucs.h>
int usermain(int argc, char *argv[])

```

```

{
    count=0;
    ...
    while(1)
    {
        ...
        tpschedule(3);
        cd = tpacall("SERVICE", buf, strlen(buf), TPNOREPLY);
        if (cd < 0) { error processing }
        ...
        if (count == 10) tpsvrdown();
    }
}

void reply_receive(UCSMMSGINFO *reply)
{
    printf("data....%s\n", reply->data);
}

```

- 関連関数

tpsvrinit(), tpsvrdone()

### 3.2.53. tpsvrinit

開発者がサービスを実行する前に、Tmaxサーバーの初期化を行う関数です。Tmax応用サーバー・プログラムの分離されたmainの初期化プロセスでtpsvrinit()を呼び出します。このルーチンは、プロセスが実行された後、いかなるサービス要求も処理する前に呼び出されます。そのため、tpsvrinit()ルーチン内にTmax通信が実行されることや、トランザクションが定義されることもあります。

アプリケーションでtpsvrinit()ルーチンを提供しない場合、Tmaxが提供するデフォルト・ルーチンが代わりに呼び出されます。デフォルトのtpsvrinit()は、トランザクションを処理するサーバー・グループに含まれたサーバーである場合、**tx\_open()**と**userlog()**を呼び出し、サーバーが正常に開始されたことを通知します。

アプリケーション別のコマンド・ライン・オプション(CLOPT)はサーバーに渡され、tpsvrinit()で処理されることができます。コマンド・ライン・オプション(CLOPT)についての詳細は、source configファイルのSERVERセクションのうちCLOPT項目を参照してください。オプションは、argcとargvにより渡されます。

**getopt()**がTmaxサーバーのmain()で使用されるため、optarg、optind、opterrがtpsvrinit()内で、オプションのパーシングおよびエラー検出の制御に使用されます。

- プロトタイプ

```

# include <tmaxapi.h>
int tpsvrinit (int  argc, char  **argv)

```

- パラメータ

パラメータ	説明
argc	コマンド・ラインのパラメータ数です
argv	コマンド・ラインのパラメータです

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
負数	関数の呼び出しに失敗した場合です。いかなるサービス要求も受けずにサーバー・プロセスは終了し、エラーは発生しません

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

EXEC SQL INCLUDE sqlca.h;
tpsvrinit(int argc, char **argv)
{
    EXEC SQL begin declare section;
    char user_name[30];
    char user_passwd[30];
    EXEC SQL end declare section;
    EXEC SQL CONNECT scott IDENTIFIED BY tiger;
    return(0);
}

SERVICE(TPSVCINFO *msg)
{
    int ret, cd;
    char *buf;
    buf=tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }

    data process....
    cd=tpgetclid();
    if (cd==-1) { error processing }
    ret=tpsendtocli(cd, buf, 0, TPNOFLAGS);
    if (ret==-1) { error processing }

    data process....
```

```

    printf(" Sevice end\n");
    EXEC SQL COMMIT WORK RELEASE;
    tpreturn(TPSUCCESS, buf, strlen(buf), 0);
}

```

- 関連関数

tx\_open(), tpsvrdone()

### 3.2.54. tpsvrthrdone

マルチスレッドおよびマルチコンテキスト・サーバーでのみ提供される関数です。マルチスレッドおよびマルチコンテキスト・サーバーは、サーバー・プロセスを終了時に、tpsvrdone関数を実行する前にサービス・スレッドを終了します。サービス・スレッドは自身が終了時に、この関数が定義されていれば、自動で呼び出します。開発者はスレッドが終了する前に行う処理ルーチンを作成します。この関数内で、Tmax通信が実行されたり、トランザクションが実行されたりすることができます。このような作業がすべて完了していない状態でそのまま返す場合は、スレッドの終了時に未完了作業をすべて無視します。

関数を使用してあるクライアントが生成済みの他のコンテキストを現在のクライアントに割り当てることができます。ほとんどのATMI関数は、コンテキスト単位で動作します。クライアントは複数のコンテキストを使用できますが、該当する瞬間にはただ1つのコンテキストのみをもちます。たとえば、context1でtpacall()を実行した場合、他のコンテキストを使用したとしても、tpgetrply()を正常に行うためには、tpgetrply()を実行する時点では、context1を現在のコンテキストに設定する必要があります。詳細については、「[3.2.51. tpsvrdone](#)」を参照してください。

- プロトタイプ

```

# include <tmaxapi.h>
int tpsvrthrdone(void)

```

- 戻り値

tpsvrthrdone()は、開発者がサーバー・プロセスの終了を行う前に実行する処理を作成する関数です。戻り値はなく、エラーも発生しません。

- 例

tpsvrthrinit()関数の例を参照してください。

- 関連関数

tpsvrthrinit()



## 3.2.55. tpsvrthrinit

マルチスレッドおよびマルチコンテキスト・サーバーでのみ提供される関数です。Tmaxサーバーは、サーバー・プロセスを開始する際に初期化を行えるtpsvrinit関数を提供します。同様に、マルチスレッドおよびマルチコンテキスト・サーバーでは、tpsvrinit関数が呼び出された後、スレッド・プールで管理されるサービス・スレッドに対しても、スレッドを作成時にそれぞれのスレッドごとに初期化を行える初期化関数を提供します。

スレッド・プールはMINTHR、MAXTHR項目に基づいて動作するので、サーバー・プロセスが最初に起動されるときは、MINTHR数までサービス・スレッドが生成され、tpsvrthrinit()を呼び出します。以降、スレッド・プールにアイドル・サービス・スレッドがなくなると、最大MAXTHRまで必要な数のサービス・スレッドが新たに生成され、この関数を呼び出します。MINTHR項目が0の場合は、プロセス起動初期にサービス・スレッドを作成しないため、tpsvrinit()関数のみ呼び出し、サービス要求を待ちます。

tpsvrinit()関数が呼び出された後、そしてそれぞれのスレッドでサービス要求を処理する前に実行され、tpsvrinit()関数に渡されたものと同じパラメータが渡されます。このパラメータは、環境設定のSERVERセクションのCLOPT項目の設定です。tpsvrinit()とtpsvrthrinit()で渡されるパラメータが同じであることを考慮して作成する必要があります。詳細については「[3.2.53. tpsvrinit](#)」を参照してください。

### ● プロトタイプ

```
# include <tmaxapi.h>
int tpsvrthrinit (int  argc, char  **argv)
```

### ● パラメータ

パラメータ	説明
argc	コマンド・ラインのパラメータ数です
argv	コマンド・ラインのパラメータです

### ● 戻り値

tpsvrthrinit()関数により初期化を行う際、初期化に失敗すると、-1を返します。サーバー・プロセスは、tpsvrthrinit()を呼び出した後、-1が返された場合は、プロセスの起動を取り消して終了します。

戻り値	説明
0	関数の呼び出しに成功した場合です
負数	関数の呼び出しに失敗した場合です。サーバー・プロセスは、プロセスの起動を取り消して終了します

### ● 例

```
#include <stdio.h>
#include <pthread.h>
#include <usrinc/atmi.h>
```

```

void tpsvrthrinit(int argc, char **argv)
{
    param_t *param;
    param = get_threadspecificdata();
    if (pthread_create(&param->tid, NULL, THREAD_ROUTINE, (void *)param) != 0)
    {
        printf("user_create_thread failed\n");
        return -1;
    }
    pthread_mutex_init(&param->mutex, NULL);
    pthread_cond_init(&param->cond, NULL);
    return 0;
}

void tpsvrthrdone()
{
    void *ret;
    param_t *param;

    param = get_threadspecificdata();
    param->state = EXIT;
    pthread_cond_signal(&param->cond);
    pthread_join(param->tid, &ret);

    pthread_mutex_destroy(&param->mutex);
    pthread_cond_destroy(&param->cond);
    printf("user_create_thread destroyed\n");
}

SERVICE(TPSVCINFO *msg)
{
    param_t *param;
    param = get_threadspecificdata();
    ...
    ret = tpgetctxt(&param->ctxtid, TPNOFLAGS);
    ret = pthread_cond_signal(&param->cond);
    ...
}

void *THREAD_ROUTINE(void *arg)
{
    param_t *param;
    param = (param_t *)arg;

    while(1) {
        pthread_mutex_lock(&param->mutex); {

```

```

        pthread_cond_wait(&param->cond, &param->mutex);
        if (param->state == EXIT)
            break;
        if (tpsetctxt(param->ctxtid, TPNOFLAGS) == -1) {
            printf("tpsetctxt(%d) failed, [tperrno:%d]", param->ctxtid,
                    tperrno);
            return NULL;
        }

        tpcall("MTOUPPER", sndbuf, 0, &rcvbuf, &rcvlen, TPNOFLAGS);
        ...

        if (tpsetctxt(TPNULLEXEC, TPNOFLAGS) == -1) {
            printf("tpsetctxt(TPNULLEXEC) failed, [tperrno:%d]", tperrno);

            return NULL;
        }
        ...
    } pthread_mutex_unlock(&param->mutex);
}
return NULL;
}

```

- 関連関数

tpsvrthrdone()

## 3.2.56. tptsleep

TMMからサーバー・プロセス終了のイベントを待機する関数です。tpprechck()コールバック関数で待機する必要がある場合、周期的にtptsleep()を呼び出して、正常なtmddownが実行されるようにします。

- プロトタイプ

```

#include <usrinc/tmaxapi.h>
int tptsleep(struct timeval *timeout)

```

- パラメータ

パラメータ	説明
timeout	終了イベントの待機時のタイムアウト時間を秒単位で指定します。select()システム関数と同様に適用されます

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です

- 例

```
int tpprechk(void)
{
    struct timeval timeout;
    int ret;
    timeout.tv_sec = 5;
    timeout.tv_usec = 0;
    while(1)
    {
        ret = tptsleep(&timeout);
        ...
    }
    return 0;
}
```

## 3.2.57. tpunadvertise

サーバー・プロセスが提供するサービスをサーバーで登録解除(unadvertise)する関数です。特定サービス名を登録(advertise)または登録解除(unadvertise)できます。

特定サービスを登録する場合、TMMが管理するサーバー別サービスの名前表に該当サービス名を登録します。登録解除する場合、サーバー別サービス名を表から削除します。登録の場合、サーバーが提供する新規サービスをサーバー・プロセスが登録します。登録解除の場合、サーバーが提供するサービスを登録解除します。登録解除されたサービスを呼び出した場合、TPENOENTエラーを受信します。

以下の表にそれぞれのケースにおける動作方式を説明します。

- **環境設定ファイルに登録されたサービス**

該当サービスの状態がUNADVIに変更されます。サービスを呼び出す場合、TPENOENTエラーが返されます。

- **mksvrで登録されたサービス**

該当サービスの状態がUNADVIに変更されます。サービスを呼び出す場合、TPENOENTエラーが返されます。該当サーバーのプロセスがすべて終了すると、サービスは自動削除されます。

- **新規登録するサービス**

該当サービスの状態がUNADVに変更されます。サービスを呼び出す場合、TPENOENTエラーが返されます。該当サーバーのプロセスがすべて終了すると、サービスは自動削除されます。

以下は、関数の使用方法についての説明です。

- プロトタイプ

```
#include <atmi/usrinc.h>
int tpunadvertise(char *svcname);
```

- パラメータ

戻り値	説明
svcname	登録解除するサービス名を指定します

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernolにエラーコードが設定されます

- エラー

エラーコード	説明
[TPEINVAL]	サービス名がNULLである場合です
[TPENOENT]	サービスが存在しない場合です
[TPELIMIT]	サービス名が指定された長さを超過して登録された場合です
[TPEPROTO]	関数が不適切な状況で呼び出された場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <usrinc/atmi.h>

SVC2TN_2( TPSVCINFO *msg )
{
    int      ret;
    char     input[MAXLEN];

    memset(input, 0x00, MAXLEN);
    strncpy(input, msg->data, msg->len);
```

```

ret = tpunadvertise(input);
if (ret < 0) {
    tpreturn(TPFAIL, tperrno, (char*)msg->data, msg->len, 0);
}
tpreturn(TPSUCCESS, 0, (char*)msg->data, msg->len, 0);
}

```

## 3.2.58. tpunregcb

UCS方式のサーバー・プロセスから非同期型の要求に対する応答を受けるルーチンを再設定する関数です。サーバー・プログラムから応答を受けた際に実行されるルーチンを再設定(reset)します。

### ● プロトタイプ

```

#include <ucs.h>
int tpunregcb (void)

```

### ● 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

### ● エラー

tpunregcb()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

### ● 例

```

...
#include <usrinc/atmi.h>
#include <usrinc/ucs.h>
void reply_receive(UCSMMSGINFO *reply);

int usermain(int argc, char *argv[])
{
    ...

```

```

ret = tpregcb(reply_receive);
if (ret == -1){ error processing }

ret = tpunregcb();
if (ret == -1){ error processing }
while(1)
{
    ...
    tpschedule(3);
    cd = tpacall("SERVICE", buf, strlen(buf), TPNOFLAGS);
    if (cd < 0) { error processing }
    ...
}
}

void reply_receive(UCSMSGINFO *reply)
{
    printf("first reply receive\n");
    printf("data....%s\n", reply->data);
}

```

- 関連関数

tpregcb()

## 3.2.59. tpschedule

UCSサーバー・プロセスで、データの到着をマイクロ秒単位で入力した時間の間待機する関数です。tpschedule()はUCS形式のサーバー・プロセスでのみ使用可能な関数で、最大タイムアウト時間の間待機し、決められた時間内にデータが到着すれば即時返します。

- プロトタイプ

```

#include <ucs.h>
int tpschedule (int timeout)

```

- パラメータ

パラメータ	説明
timeout	<p>待機する時間をマイクロ秒単位で入力します</p> <ul style="list-style-type: none"> <li>– -1: データが到着したかどうかチェックのみ行い、すぐに終了します</li> <li>– 0: データが到着するまで待機します</li> </ul>

- 戻り値

戻り値	説明
0	タイムアウト時間までにデータが到着していない場合です
正の整数	タイムアウト時間までにデータが到着した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tpschedule()が実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
...
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/ucs.h>

int usermain(int argc, char *argv[])
{
    ...
    while(1)
    {
        ...
        tpschedule(3000000);
        ret = tpcall("SERVICE", (char *)buf, strlen(buf), (char **)&buf,
                    (long *)&rlen, TPNOFLAGS);
        if (ret == -1) { error processing }
        ...
    }
}
```

- 関連関数

tpschedule()



### 3.2.60. tx\_close

サーバーで使用される関数であり、リソース・マネージャーとの接続を終了します。

tx\_close()関数は、呼び出し元と接続しているすべてのリソース・マネージャーとの接続を終了します。リソース・マネージャー別の特定接続終了の呼び出しの代わりに使用されるため、アプリケーションは移植性に問題を引き起こし得るリソース・マネージャー別の特定接続終了の呼び出しを使用しなくても構いません。リソース・マネージャー別に接続終了方式が異なるため、各リソース・マネージャーの終了に必要な情報は、各リソース・マネージャーが準備する必要があります。

tx\_close()関数は内部的に呼び出されるため、ユーザーは使用しなくても構いません。リソース・マネージャー別に接続終了方式が異なるため、tx\_close()関数はトランザクション・マネージャーがリソース・マネージャー別の特定方法に関する情報をリソース・マネージャーに渡します。

tx\_close()関数は、アプリケーション・プロセスがグローバル・トランザクションにそれ以上参加しない場合に呼び出される必要があります。呼び出し元がトランザクション・モードの場合、tx\_close()関数の呼び出しは失敗します。この際、[TX\_PROTOCOL\_ERROR]を返し、いかなるリソース・マネージャーとの接続も終了しません。

#### ● プロトタイプ

```
# include <tx.h>
int tx_close(void)
```

#### ● 戻り値

戻り値	説明
TX_OK	関数の呼び出しに成功した場合です。プロセスに接続されたすべてのリソース・マネージャーとの接続が終了します
負数	関数の呼び出しに失敗した場合です。エラーコードを返します

#### ● エラー

tx\_close()が実行されなかった場合、以下の負数値を返します。

エラーコード	説明
[TX_PROTOCOL_ERROR]	関数が不適切な状況で呼び出された場合です。たとえば、呼び出し元がトランザクション・モードの場合に発生します。いかなるリソース・マネージャーとの接続も終了しません
[TX_ERROR]	トランザクション・マネージャーまたはリソース・マネージャーが一時的にエラーとなった場合です。エラーの原因は製品の特性によって異なります。可能なリソース・マネージャーとの接続がすべて終了します

エラーコード	説明
[TX_FAIL]	致命的なエラーが発生し、トランザクション・マネージャーやリソース・マネージャーがアプリケーションのために作業を実行できなくなった場合です。エラーの原因は製品の特性によって異なります

- 例

```

...
int tpsvrinit(int argc, char *argv[])
{
    int ret;
    ret = tx_close();
    if (ret < 0){ error processing }
    ret = tx_open();
    if (ret < 0){ error processing }
}

int tpsvrdone()
{
    int ret;
    ret = tx_close();
    if (ret < 0) { error processing }
}

```

- 関連関数

tx\_open()

## 3.2.61. tx\_open

関連するリソース・マネージャーと接続する関数です。tx\_close()は、内部的に呼び出されるため、ユーザーは使用しなくても構いません。リソース・マネージャー別に接続方式が異なるため、tx\_open()関数は、トランザクション・マネージャーがリソース・マネージャー別の特定接続情報を呼び出し元と関連するリソース・マネージャーに渡します。

tx\_open()関数は、アプリケーションと関連するすべてのリソース・マネージャーと接続します。これは、移植性の問題を引き起こし得るリソース・マネージャー別の接続関数の代わりに使用されるため、アプリケーションはリソース・マネージャー別の接続関数を使用しなくても構いません。リソース・マネージャー別に接続方式が異なるため、各リソース・マネージャー別の接続情報は、管理者が準備する必要があります。アプリケーションが接続されていないリソース・マネージャーに接続した場合、リソース別の特定エラーが返されます。tx\_open()関数は、プロセスがグローバル・トランザクションに参加前に正常に返される必要があります。

一度tx\_open()が正常に返された場合、再度呼び出されたtx\_open()関数はtx\_close()を呼び出すまでは、何の影響も与えません。リソース・マネージャーと接続された後にtx\_open()が再度呼び出された場合、トラ

ンザクション・マネージャーはいかなるリソース・マネージャーとも再接続せず、正常に返します。**tpopen()**関数は**tx\_open()**関数と同一です。

- プロトタイプ

```
# include <tx.h>
int tx_open(void)
```

- 戻り値

戻り値	説明
TX_OK	関数の呼び出しに成功した場合です。一部のリソース・マネージャーあるいはすべてのリソース・マネージャーと接続されます
負数	関数の呼び出しに失敗した場合です。エラーコードを返します

- エラー

**tx\_open()**が正常に実行されなかった場合、以下の負数値を返します。

エラーコード	説明
[TX_ERROR]	トランザクション・マネージャーあるいはリソース・マネージャーが一時的にエラーとなった場合です。いかなるリソース・マネージャーとも接続されません。エラーの原因は製品の特性によって異なります
[TX_FAIL]	致命的なエラーが発生し、トランザクション・マネージャーやリソース・マネージャーがアプリケーションのために作業を実行できなくなった場合です。エラーの原因は製品の特性によって異なります

- 例

```
int tpsvrinit(int argc, char *argv[])
{
    int ret;
    ret = tx_close();
    if (ret < 0){
        error processing
    }
    ret = tx_open();
    if (ret < 0){
        error processing
    }
}

int tpsvrdoen()
{
    int ret;
```

```

    ret = tx_close();
    if (ret < 0){
        error processing
    }
}

```

- 関連関数

tx\_close()

## 3.2.62. TPADMNTOI

tmadminからadmntoiコマンドによる要求を処理します。関数のプロトタイプのみ存在し、ユーザーがプロトタイプどおり内部処理ロジックを実装する必要があります。実装した後、tpregancb APIを利用して登録すると、tmadminでadmnotiコマンドを使用時に発生したイベントを受信することができます。

- プロトタイプ

```

#include <tmaxapi.h>
typedef int (*TPADMNOTI)(int seqno, int len, char *args);

```

- パラメータ

パラメータ	説明
seqno	tmadminのanコマンド要求の順番です
len	通知メッセージの文字列のサイズです
args	通知メッセージの文字列が保存されたバッファです。このバッファは戻されると、サーバー・ライブラリーによって解除されます

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
	関数の呼び出しに失敗した場合です

- 例

```

...
int admnoti(int regno, int len, char *args);
void foo();
void bar();

```

```

TMAXSVRINFO svrinfo;

int tpsvrinit(int argc, char *argv[])
{
    tmax_my_svrinfo(&svrinfo);
    if (tpregancb(admnoti) < 0)
        error processing;
}

int admnoti(int reqno, int len, char *args)
{
    printf("svg[%s] svr[%s] spri[%d] reqno[%d] message[%s]",
        svrinfo.svgname, svrinfo.svrname, svrinfo.spri, reqno, args);
    if (strncmp(args, "foo", 3) == 0) {
        foo();
    } else if (strncmp(args, "bar", 3) == 0) {
        bar();
    } else {
        return -1;
    }
    return 0;
}

```

- 関連関数

tpretgancb, tpunregancb

## 3.2.63. tpregancb

tadminからadmntoiコマンドによる要求を処理します。

- プロトタイプ

```

# include <tmaxapi.h>
int tpregancb (TPADMNOTI)

```

- パラメータ

パラメータ	説明
TPADMNOTI	コールバック関数です

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpregancb()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	コールバックに指定された関数のアドレスが正しくない場合です

- 例

```

...
int admnoti(int regno, int len, char *args);
void foo();
void bar();
TMAXSVRINFO svrinfo;

int tpsvrinit(int argc, char *argv[])
{
    tmax_my_svrinfo(&svrinfo);
    if (tpregancb(admnoti) < 0)
        error processing;
}

int admnoti(int regno, int len, char *args)
{
    printf("svg[%s] svr[%s] spri[%d] regno[%d] message[%s]",
        svrinfo.svgname, svrinfo.svrname, svrinfo.spri, regno, args);
    if (strncmp(args, "foo", 3) == 0) {
        foo();
    } else if (strncmp(args, "bar", 3) == 0) {
        bar();
    } else {
        return -1;
    }
    return 0;
}

```

- 関連関数

TPADMNOTI, tpunregancb

## 3.2.64. tpunregancb

tadminからadmntoiコマンドによる要求を処理するコールバック関数を解除します。

- プロトタイプ

```
# include <tmaxapi.h>
int tpunregancb ()
```

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tpregancb()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEPROTO]	登録されたコールバック関数がない場合です

- 例

```
...
int admnoti(int regno, int len, char *args);
void foo();
void bar();
TMAXSVRINFO svrinfo;

int tpsvrinit(int argc, char *argv[])
{
    tmax_my_svrinfo(&svrinfo);
    if (tpunregancb() < 0)
        error processing;
}

int admnoti(int regno, int len, char *args)
{
    printf("svg[%s] svr[%s] spri[%d] regno[%d] message[%s]",
        svrinfo.svgname, svrinfo.svrname, svrinfo.spri, regno, args);
    if (strncmp(args, "foo", 3) == 0) {
        foo();
    } else if (strncmp(args, "bar", 3) == 0) {
        bar();
    } else {
```

```

        return -1;
    }
    return 0;
}

```

- 関連関数

TPADMNOTI, tpregancb

## 3.3. クライアント関数

### 3.3.1. gettpurcode

urcodeサービスに設定されているurcodeをクライアントに返す関数です。クライアントが呼び出した`tpreturn()`の2番目のパラメータを出力します。この値は、クライアントとサービス・プログラム間の追加された通信方式に使用されます。たとえば、特定エラーのハンドリング・ルーチンを実行時に、該当エラー番号をクライアントに返すことができます。

- プロトタイプ

```

#include <atmi.h>
long gettpurcode(void)

```

- 戻り値

urcodeを返します。

- 例

```

#include <stdio.h>
#include <usrinc/atmi.h>

void main(int argc, char* argv[])
{
    char *buf;
    long len;
    int ret;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret== -1) { error processing }

    buf=tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
    data process....
}

```



```

ret=tpcall("SERVICE", buf, 0, &buf, &len, TPNOFLAGS);
if (ret==-1) {error processing }
printf("urcode from SERVICE Return: %d", gettpurcode());
data process....

tpend();
}

```

---

#### 参考

gettpurcode関数の詳細については、『Tmax プログラミングガイド(ダイナミックライブラリ)』を参照してください。

---

### 3.3.2. tpchkunsol

関数の呼び出し時にサーバーから非要求メッセージがある場合、該当メッセージを処理するための関数を呼び出す関数です。クライアントは、tpsetunsol()を呼び出した後に非要求メッセージを処理する関数を設定するか、**tpcall()**あるいは**tpacall()**を呼び出した後に非要求メッセージを処理する関数を呼び出す必要があります。

tpchkunsol()関数を呼び出した場合、tpcall()あるいはtpacall()を呼び出す前にも非要求メッセージがあるかどうか確認後、メッセージを取得できます。

#### ● プロトタイプ

```

#include <usrinc/tmaxapi.h>
int tpchkunsol(void)

```

#### ● 戻り値

戻り値	説明
メッセージ数	関数の呼び出しに成功した場合です。サーバーから受信した非要求メッセージの数を返します
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

#### ● エラー

tpchkunsol()が正常に実行されていない場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログ・ファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です
[TPETIME]	タイムアウトが発生した場合です。関数の呼び出し元がトランザクション・モードの場合、トランザクション・タイムアウトが発生し、トランザクションはロールバックされます。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、TPNOTIMEあるいはTPNOBLOCKのどちらも設定されていない状況でブロック・タイムアウトが発生します。トランザクション・タイムアウトが発生した場合、トランザクションがロールバックされるまで新規サービス要求を送信し、まだ受信されていない応答を待機することはすべて[TPETIME]エラーで失敗します
[TPEPROTO]	tpstart()の呼び出し時に、tpsetunsol_flag()を使用してハンドラにより非要求メッセージを受信するTPUNSOL_HNDフラグが設定されている必要があります。TPUNSOL_HNDフラグが設定されていない場合、[TPEPROTO]エラーで失敗します

● 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/ucs.h>
#include <usrinc/tmaxapi.h>

void get_unsol(char *data, long len, long flag)
{
    printf("unsolicited data = %s\n", data);
}

int main(int argc, char *argv[]){
    char *sndbuf;
    char *rcvbuf;
    long rcvlen;
    int RecvCnt = 0;
    TPSTART_T *tpinfo;

    tpinfo = (TPSTART_T *)tpalloc(« TPSTART », NULL, sizeof(TPSTART_T));
    if (tpinfo == NULL) {
        printf("tpalloc failed !<-tpinfo\n");
        exit(1);
    }
    strcpy(tpinfo->username, "starbj81");
    strcpy(tpinfo->cltname, "client");
    if(tpstart((TPSTART_T *)tpinfo) == -1)
    {
```

```

        fprintf(stderr, "tpstart error\n");
        exit(1);
    }
    tpsetunsol_flag(TPUNSOL_HND);
    tp_sleep(5);

    if(tpsetunsol(get_unsol) == TPUNSOLERR)
        printf("tpsetunsol failed..\n");

    RecvCnt = tpchkunsol();
    if(RecvCnt < 0)
        printf("tpchkunsol failed\n");
    else
        printf("Received Unsol Data Count : %d\n", RecvCnt);

    if((sndbuf = (char *)tpalloc("CARRAY", NULL, 1024)) == NULL)
        error processing
    if((rcvbuf = (char *)tpalloc("CARRAY", NULL, 1024)) == NULL)
        error processing
    if((cd = tpcall("LOGIN", sndbuf, 1024, (char **)&rcvbuf,
        (long *)&rcvbuf, 0)) == -1)
        error processing

    printf("After tpacall() received Message from server:%s\n", rcvbuf);
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
}

```

- 関連関数

tpsetunsol(), tpsetunsol\_flag()

### 3.3.3. tpend

クライアントでTmaxシステムとの接続を解除する関数です。クライアントがトランザクション・モードの場合、トランザクションは自動的にロールバックされます。tpend()が正常に返された場合、呼び出し元はいかなるプログラムとも通信できなくなり、トランザクションにも参加できません。また、維持されていた対話型接続は即時終了します。tpend()が1回以上呼び出された場合(Tmaxシステムとの接続がすでに解除された後に再度呼び出された場合)、-1値を返しますが、システムには作動しません。

- プロトタイプ

```

# include <atmi.h>
int tpend (void)

```

- 戻り値

戻り値	説明
-1	関数の呼び出しに失敗した場合です。tpernoにエラーコードが設定されます

- エラー

tpend()が正常に実行されなかった場合、tpernoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEPROTO]	tpend()が不適切な状況で呼び出された場合です。たとえば、呼び出し元がサーバーの場合、あるいは接続が解除された後に再度呼び出された場合に発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <usrinc/atmi.h>

void main(int argc, char *argv[])
{
    char    *sndbuf, *rcvbuf;
    long    rcvlen, sndlen;
    int     ret;

    ret=tpstart((TPSTART_T *)NULL)
    if (ret==-1) {error processing }

    buf = (char *)tpalloc("STRING", NULL, 0)
    if (buf=NULL) { error processing }

    data process ...

    ret=tpcall("SERVICE", buf, 0, &buf, &len, TPNOFLAGS);
    if (ret==-1) { error processing }

    data process ...
    printf(" data: %s\n", buf);
    tpfree((char *)buf);
    tpend();
}
```

- 関連関数

tpstart()

### 3.3.4. tpgethostaddr

クライアントとTmaxシステムの接続可否を確認する場合や、接続されたTmaxシステムのIPアドレスとポート情報を取得する場合に使用する関数です。

- プロトタイプ

```
#include <tuxatmi.h>
int tpgethostaddr(int *ip, int *port, long flags);
```

- パラメータ

パラメータ	説明
ip	該当クライアントのIPアドレスを設定します。IPアドレスはsockaddr_inの構造体にあるs_addrフィールドであるため、dot形式に変えるためにはinet_ntoa()を使用します
port	該当クライアントのポート情報を設定します
flags	flagsに設定可能な値は以下のとおりです <ul style="list-style-type: none"> <li>– GET_SVR_CON Tmaxの接続可否のみを判断します</li> <li>– GET_CUR_IP 接続されたサーバーのIPアドレスとポートの情報も一緒に取得します</li> </ul>

- 戻り値

戻り値	説明
- 1	Tmaxシステムに接続していない場合です
1	プライマリーTmaxシステムに接続している場合です
2	バックアップとして設定されたTmaxシステムに接続している場合です

- 例

```
#include <usrinc/atmi.h>
#include <arpa/inet.h>
#include <usrinc/tmaxapi.h>
```

```

main(int argc, char *argv[])
{
    char    *sndbuf, *rcvbuf;
    long    rcvlen, sndlen;
    int     ret;
    int     i;
    TPSTART_T    *tpinfo;
    struct    sockaddr_in addr;
    char *cli_ip="192.168.1.86";
    int port;

    if ( (ret = tmaxreadenv( "tmax.env","TMAX" )) == -1 ){
        printf( "tmax read env failed\n" );
        exit(1);
    }

    tpinfo = (TPSTART_T *)tpalloc( "TPSTART",NULL,sizeof(TPSTART_T));
    if (tpinfo == -1){
        printf("tpinfo alloc failed\n");
        exit(1);
    }

    ret=tpstart(tpinfo);
    if (ret == -1){
        printf("tpstart failed\n");
        printf("[tperrno(%d) : %s]\n", tperrno,tpstrerror(tperrno));
        exit(1);
    }
    addr1.sin_addr.s_addr = inet_addr(cli_ip);
    port=8888;
    ret=tpgethostaddr(&addr.sin_addr.s_addr, &port, GET_CUR_IP);
    printf("ip = %s ,port = %d\n", inet_ntoa(addr.sin_addr),port);
}

```

### 3.3.5. tpgetunsol

クライアントの要求なく、一方的に渡されたメッセージを処理する関数です。メッセージを送る側で、**tpbroadcast()**あるいは**tpsendtoci()**、**tppost()**を使用して渡します。

tpgetunsol()の呼び出し前に渡された一方的なメッセージは無視されます。tpgetunsol()を使用して非要求メッセージを受け取るには、tpstart()を使用し、Tmaxシステムの接続時にflagsをTPUNSOL\_POLLあるいはTPUNSOL\_HNDと指定する必要があります。tpgetunsol()がプログラムに呼び出された場合、tpstart()のフラグがTPUNSOL\_IGNと設定されていても、内部的にTPUNSOL\_POLLに転換され、サーバーから非要求メッセージを受け取ります。

## ● プロトタイプ

```
#include <tmxapi.h>
int  tpgetunsol (int type, char **data, long *len, long flags)
```

## ● パラメータ

パラメータ	説明
type	サーバーから渡されたメッセージの形式です。UNSOL_TPPOST、UNSOL_TPROADCAST、UNSOL_TPSENDTOCLIなどがあります
data	渡されたメッセージについてのポインターです。クライアントが分からないバッファ・タイプおよびサブタイプの場合があり得ますが、この場合dataを使用できません
len	メッセージの長さの合計です
flags	<p>メッセージのブロッキング処理可否を決定します。</p> <p>flagsで使用可能な値は以下のとおりです</p> <ul style="list-style-type: none"><li>– TPBLOCK</li></ul> <p>tpgetunsol()の呼び出し時にブロッキング状態で一方的なメッセージが来るまで待機します</p> <li>– TPNOCHANGE</li> <p>受信した応答バッファと*dataが指すバッファのタイプが異なる場合、受信者が認識できる限度内で、*dataのバッファ・タイプは受信した応答バッファ・タイプに変更されます。フラグが設定された場合、*dataが指すバッファのタイプは変更できません。受信した応答バッファのタイプおよびサブタイプは、*dataが指すバッファのタイプおよびサブタイプと一致する必要があります</p> <li>– TPNOTIME</li> <p>関数の呼び出し元がブロック・タイムアウトを無視し、応答が受信されるまで無限に待機します。トランザクション・モードでtpgetrply()を実行した場合、トランザクション・タイムアウトが適用されます</p> <li>– TPSIGRSTRT</li> <p>シグナルの割り込みを受け入れる場合に使用します。システム関数の呼び出しが妨害されたとき、システム関数の呼び出しが再実行されます。TPSIGRSTRTフラグが設定されていない状態でシグナルの割り込みが発生した場合、関数は失敗し、tperrnoにTPGOTSIGが設定されます</p> <li>– TPGETANY</li> <p>入力値の記述子(cd)を無視し、これに関係なく受信可能な応答を返します。cdは返された応答に対する呼び出し記述子になります。応答がなければ、一般的にtpgetrply()は応答が到着するまで待機します</p>

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoに状況に該当する値が設定されます

- エラー

tpgetunsol()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEPROTO]	tpgetunsol()が不適切な状況で呼び出された場合です。たとえば、サーバー内で呼び出された場合に発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <string.h>
#include <usrinc/tmaxapi.h>

void main(int argc, char *argv[])
{
    int ret;
    char *buf;
    long len;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf=tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
    data process....
    while(1)
    {
        ret=tp_sleep(2);
        if (ret==-1) { error processing }
        if (ret==0) printf("nothing happened\n");
        else {
            ret=tpgetunsol(UNSOL_TPSENDTOCLI, (char **)&buf, &len, TPNOCHANGE);

            if (ret==-1) { error processing }
```



```

        printf("received data : %s\n", buf);
    }

    data process....
    if (strncmp(buf, "end", 3)==0) break;
}

data process....
tpfree(buf);
tpend();
}

```

- 関連関数

tpbroadcast(), tpsetunsol(), tpstart(), tpend()

### 3.3.6. tpinit

Tuxedoで使用した関数をTmaxシステムにそのまま適用するために使用する関数です。Tuxedoで開発されたプログラムを変更することなく、Tmaxに変換できるようサポートします。関数の機能は**tpstart()**と同一です。

- プロトタイプ

```

#include <tuxatmi.h>
int tpinit (TPINIT *tpinfo)

```

- パラメータ

パラメータ	説明
tpinfo	認証情報およびマルチコンテキストを使用するかどうかについての設定です

- 戻り値

[「3.3.10. tpstart」](#)を参照してください。

- 例

```

#include <stdio.h>
#include <tuxinc/tuxatmi.h>
int main(int argc, char *argv[])
{
    char *buf;
    long len;

```

```

int ret;

ret = tpinit((TPINIT *)NULL);
if (ret == -1){ error processing }
buf = (char *)tpalloc("STRING", NULL, 0);
if (buf == NULL){ error processing }

strcpy(buf, argv[1]);
ret = tpcall("SERVICE", buf, 0, &buf, &len, TPNOFLAGS);
if (ret == -1){ error processing }

printf("data: %s\n", buf);
tpfree(buf);
tpterm();
}

```

- 関連関数

tpstart()

### 3.3.7. tpreset

クライアントで使用され、現在接続されているクライアントをすぐに解除する関数です。クライアント・モジュールにTPESYSTEMエラーが発生する場合は、ほとんどがネットワーク・エラーであるため、Tmaxシステムを再接続します。

tpreset()を使用して接続を解除し、サービスを再要求するか、Tmaxシステムを再度接続します。

- プロトタイプ

```

# include <tmaxapi.h>
int tpreset (void)

```

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

tpreset()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

void main(int argc, char *argv[])
{
    int ret;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1){
        if (tperrno=TPESYSTEM)
        {
            printf ("system error \n");
            ret=tpreset();
            if (ret==-1) { error process }
            tpend();
            exit(1);
        }
        error processing....
    }
    data process..
    tpend();
}
```

### 3.3.8. tpsetunsol

クライアントで使用され、非要求受信メッセージを処理するルーチンを設定する関数です。tpsetunsol()が初めて呼び出される前にTmaxライブラリーによって受信された非要求メッセージは無視されます。NULL関数ポインタのtpsetunsol()の呼び出しも同様に無視されます。

システムが非要求メッセージの通知を受けるのに使用される方法は、アプリケーションに任意で決定されます。これは各クライアント別に変更が可能です。呼び出しで渡された関数ポインタは、与えられたパラメータ定義に適合する必要があります。

- プロトタイプ

```
# include <atmi.h>
Unsolfunc *tpsetunsol (void ( *disp ) ( char *data, long len, long flags )
)
```

## ● パラメータ

パラメータ	説明
data	受信された型付きバッファを指示します。データが伴っていない場合、dataはNULLになれません。dataが、クライアントが認識できないバッファ・タイプおよびサブタイプである場合、データは理解されません。アプリケーションはdataを削除できず、代わりにシステムがこれを削除し、データ領域を無効化して返します
len	データの長さを表します
flags	現在は使用されていません

## ● 戻り値

戻り値	説明
ポインター/NULL	関数の呼び出しに成功した場合です – ポインター: 以前設定された非要求メッセージ処理ルーチンのポインターを返します – NULL: 以前にいかなるメッセージ処理関数も設定されなかったことを表します。正常なリターンです
TPUNSOLERR	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

## ● エラー

tpsetunsol()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEPROTO]	tpsetunsol()が不適切な状況で呼び出された場合です
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

## ● 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include "../sdl/demo.s"
```

```

void get_unsol(char *data, long len, long flags)
{
    printf("get unsolicited data = %s\n", data);
    data process....
}

void main(int argc, char *argv[])
{
    int ret;
    char *buf;
    long len;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    ret=tpsetunsol_flag(TPUNSOL_HND);
    if (ret==-1) { error processing }

    ret=tpsetunsol(get_unsol);
    if (ret==TPUNSOLERR) { error processing }

    buf = (char *)tpalloc("CARRAY", NULL, 20);
    if (buf==NULL) {error processing };

    data process...

    ret=tpcall("SERVICE", buf, 20, (char **)&buf, &len, TPNOFLAGS);
    if (ret==-1) { error processing }

    data process...

    tpfree((char *)buf);
    tpend();
}

```

- 関連関数

tpstart(), tpend(), tpgetunsol()

### 3.3.9. tpsetunsol\_flag

クライアントで使用され、非要求メッセージの受信フラグを変更する関数です。**tpstart()**関数を使用してTmaxシステムと接続する場合、使用したフラグ値を再設定します。

- プロトタイプ

```
# include <atmi.h>
int tpsetunsol_flag (int flag)
```

## ● パラメータ

パラメータ	説明
flag	非要求メッセージを、受信と関連したフラグに設定します。以下のうち1つを設定します – TPUNSOL_IGN : 非要求メッセージを受けません – TPUNSOL_HND, TPUNSOL_POLL : 非要求メッセージを受けます

## ● 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。flagが設定可能な3つのうち1つでない場合で、エラーが発生してもtperrnoには値が設定されません

## ● 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include "../sdl/demo.s"
void get_unsol(char *, long, long);
void main(int argc, char *argv[])
{
    int ret;
    char *buf;
    long len;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    ret=tpsetupsol_flag(TPUNSOL_HND);
    if (ret==-1) { error processing }

    ret=tpsetunsol(get_unsol);
    if (ret==TPUNSOLERR) { error processing }
    buf = (char *)tpalloc("CARRAY", NULL, 20);
    if (buf==NULL) {error processing };
    data process...
    ret=tpcall("SERVICE", buf, 20, &buf, &len, TPNOFLAGS);
    if (ret==-1) { error processing }
```

```

    data process...
    tpfree((char *)buf);
    tpend();
}

void get_unsol(char *, long, long);
{
    printf("get unsolicited data.\n");
    data process....
}

```

- 関連関数

tpstart()

### 3.3.10. tpstart

クライアントをTmaxシステムに接続する関数です。サービス要求やトランザクション処理などに関連するATMI関数を使用する前に、tpstart()を使用してクライアントをTmaxシステムに接続する必要があります。

tpstart()を呼び出す前に他のATMI関数(tpalloc()やtpcall()など)が先に呼び出された場合、内部的にtpstart(NULL)関数が呼び出されます。この動作を望まない場合は、環境変数のTMAX\_ACTIVATE\_AUTO\_TPSTARTをNに指定できます。その場合、他のATMI関数の呼び出し時に、内部的にtpstart(NULL)を呼び出さずにTPEPROTOエラーが発生します。tpstart()が正常に返された場合、クライアントはサービス要求やトランザクション定義などを行うことができます。正常にTmaxシステムに接続された後に再度tpstart()が呼び出された場合、TPEPROTOエラーが発生します。tpstart()を使用してTmaxシステムと接続するには、TmaxシステムがインストールされたサーバーのIPとポート番号を知る必要があります。

以下は、サーバーの情報を認識するための環境変数についての説明です。

変数名	説明
TMAX_HOST_ADDR	クライアントが接続されるノードのIPアドレスです。クライアントがtpstart()を呼び出す場合、内部的にサーバー・システムに接続されるのに使用されます
TMAX_HOST_PORT	<p>クライアントが接続されるノードのポート番号を指定します。クライアントがtpstart()を呼び出す際、内部的にサーバー・システムとの接続のためにTMAX_HOST_ADDRと一緒に使用されます。</p> <p>Tmax環境ファイルにTPORTNOに定義された値である必要があります。クライアントとサーバーが同じノードに存在する場合、TCP/IPソケットを使用せず、ドメイン・ソケットを使用して、より効果的な処理ができます。この場合、Tmax環境ファイルにTPORTNOと定義された値の代わりにPATHDIRを指定しま</p>

変数名	説明
	す。Tmax環境ファイルのDOMAINセクションとNODEセクションのTPORTNO項目を参照します
TMAX_BACKUP_ADDR	TMAX_HOST_ADDRアドレスのノードに障害が発生した場合に備えて、他のTmaxシステム・ノードを指定します。クライアントは、TMAX_HOST_ADDRアドレスのノードで接続を試み、そのノード以外の接続に失敗した場合、TMAX_BACKUP_ADDRアドレスのノードで接続を試みます
TMAX_BACKUP_PORT	TMAX_BACKUP_ADDRアドレスのノードに対するTmaxシステム・ポート番号です
TMAX_CONNECT_TIMEOUT	Tmaxシステム接続時のタイムアウト時間です。micro-seconds resolutionが提供されます(例: 3.5)

## ● プロトタイプ

```
#include <atmi.h>
int tpstart (TPSTART_T *tpinfo )
```

## ● パラメータ

tpinfoは、TPSTART\_Tという構造体のポインターです。TPSTARTというバッファ・タイプを使用し、tpstart()を呼び出す前にtpalloc()によって割り当てられる必要があります。割り当てられたバッファは、tpstart()の呼出し後、tpfree()を使用して除去される必要があります。クライアントはシステム接続時に構造体形式のtpinfoパラメータを使用して必要な情報を渡します。tpinfoパラメータは、クライアント情報、非要求メッセージ処理可否、セキュリティ情報などを含みます。

tpinfoは、NULL値を使用できます。NULLの場合、cltid、dompwd、usrname、usrpwdは長さが0の文字列が与えられます。また、Tmaxセキュリティの特徴を使用せず、flagsは選択されません。

以下は、TPSTART\_T構造体の構成です。

```
struct TPSTART_T{
    char cltid[MAXTIDENT+2]; /*クライアント名 (tpbroadcast())*/
    char dompwd[MAX_PASSWD_LENGTH+2]; /*システム接続セキュリティの暗号*/
    char username[MAXTIDENT+2]; /*ユーザー認証セキュリティのアカウント*/
    char usrpwd[MAX_PASSWD_LENGTH+2]; /*ユーザー認証セキュリティの暗号*/
    int flags; /*非要求メッセージ・タイプとシステム接続方法の決定*/
} ;
```

メンバー	説明
cltid	最大長が30文字まで可能なNULLで終わる文字列で、アプリケーションで定義した名前です。tpbroadcast()で非要求メッセージを送るクライアントの指定時に使用されます



メンバー	説明
dompwd	Tmaxで提供するセキュリティ段階のうち、システム接続制御のために使用されます。Tmax環境ファイルのうち、DOMAINセクションのOWNER項目に設定されたアカウントに対する暗号を登録します
username	Tmaxで提供するセキュリティ段階のうち、ユーザー認証のために使用されます。usernameは、Tmaxシステムのpasswdファイルに登録されたアカウント名です
usrpwd	usernameに該当する暗号です。ユーザー認証セキュリティが設定された場合、クライアントはusernameとusrpwdを登録してからTmaxシステムに接続され、システムで提供するサービスを受けることができます。セキュリティ設定についての内容は、source configファイルのDOMAINセクションにあるSECURITY項目を参照してください
flags	非要求(通知)メッセージ処理およびシステム・アクセスの方法を決定するために使用されます。以下は、非要求メッセージ処理に使用できるフラグです <ul style="list-style-type: none"> <li>TPUNSOL_POLL: 非要求メッセージを受信します</li> <li>TPUNSOL_HND: 非要求メッセージを受信する関数を指定します。詳細については<a href="#">3.3.8. tpsetunsol</a>を参照してください</li> <li>TPUNSOL_IGN: 非要求メッセージを無視します。デフォルト値が定義されていない場合、TPUNSOL_IGNが設定されます</li> </ul>

#### ● 戻り値

戻り値	説明
0または1	関数の呼び出しに成功した場合です。プライマリー・ホストに0を返し、バックアップ・ホストに1を返します
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

#### ● エラー

tpstart()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、tpinfoがNULLの場合やTPSTART_T構造体のポインターでない場合に発生します
[TPEITYPE]	tpinfoがTPSTART_Tの構造体に対するポインターでない場合です
[TPEPROTO]	tpstart()が不適切な状況で呼び出された場合です。たとえば、サーバー・プログラムで使用された場合や、すでに接続されている状況で再度呼び出された場合に発生します

エラーコード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です。あるいは、環境変数の設定が正しくない場合です。たとえば、TMAX_HOST_ADDRやTMAX_HOST_PORTの設定が正しくないために接続に失敗した場合に発生します

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
void main(int argc, char *argv[])
{
    int ret;
    char *buf;
    long len;
    TPSTART_T *tpinfo;

    tpinfo = (TPSTART_T *)tpalloc("TPSTART", NULL, sizeof(TPSTART_T));
    if (tpinfo==NULL) { error processing }

    strcpy(tpinfo->cltname, "cli1");
    strcpy(tpinfo->usrname, "navis");
    strcpy(tpinfo->dompwd, "tmax");
    tpinfo->flags = TPUNSOL_HND;
    ret=tpstart(tpinfo);
    if (ret==-1) { error processing }

    buf = (char *)tpalloc("CARRAY", NULL, 20);
    if (buf==NULL) {error processing };

    ret=tpcall("SERVICE", buf, 20, &buf, &len, TPNOFLAGS);
    if (ret==-1) { error processing }

    data process....
    tpfree((char *) buf);
    tpend();
}
```

- 関連関数

tpend()

### 3.3.11. tpterm

クライアントがTmaxシステムとの接続を解除する関数です。Tuxedoで使用した関数をTmaxシステムにそのまま適用するために使用する関数で、Tuxedoで開発されたプログラムを、変更することなくTmaxに変換できるようサポートします。関数の機能は**tpend()**関数と同一です。

- プロトタイプ

```
#include <tuxatmi.h>
int tpterm(void)
```

- 戻り値

[「3.3.3. tpend」](#)を参照してください。

- 例

```
...
#include <tuxinc/tuxatmi.h>
int main(int argc, char *argv[])
{
    int ret;
    long len;
    char *buf;

    ret = tpinit((TPINIT *)NULL);
    if (ret == -1){ error processing }

    buf=tpalloc("STRING", NULL, 0);
    If (buf==NULL) { error processing }
    data process....

    ret = tpcall("SERVICE", buf, 0, &buf, &len, TPNOFLAGS);
    if (ret == -1){ error processing }

    data process...

    ret = tpterm();
    if (ret ==-1){ error processing }
}
```

- 関連関数

tpend()

### 3.3.12. tptobackup

クライアントがTmaxバックアップ・システムに接続する関数です。**tpstart()**はクライアントがTmaxシステムに接続する関数で、接続するサーバーが異常によって接続できない場合、自動的にバックアップ・システムに接続を試みます。しかし、**tptobackup()**は最初からバックアップ・システムに接続する場合に使用する関数で、ユーザーが任意でバックアップ・システムに接続する場合に使用します。

**tptobackup()**関数は、パラメータでTPSTART\_Tを受けないため、セキュリティが必要なシステムには接続できません。関数で接続されるバックアップ・システムは、セキュリティに関連した項目がconfigに設定された場合は接続できません。また、非要求メッセージを受け取るためには、**tpsetunsol\_flag()**関数を使用し、フラグを変更する必要があります。**tptobackup()**を使用するには、**TMAX\_BACKUP\_ADDR**と**TMAX\_BACKUP\_PORT**をシステム環境ファイルに登録する必要があります。(例: Kornシェルの.profile)

- プロトタイプ

```
#include <tmaxapi.h>
int tptobackup(void)
```

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

- エラー

**tptobackup()**が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEPROTO]	<b>tptobackup()</b> が不適切な状況で呼び出された場合です。たとえば、サーバー・プログラムで使用された場合や、すでに接続されている状況で呼び出された場合に発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です。あるいは、設定された環境変数が正しくない場合です。たとえば、 <b>TMAX_BACKUP_ADDR</b> や <b>TMAX_BACKUP_PORT</b> の設定が正しくないために接続に失敗した場合に発生します

- 例

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>
```

```

void main(int argc, char *argv[])
{
    tpputenv("TMAX_BACKUP_ADDR=xxx.xxx.xxx.xxx");
    tpputenv("TMAX_BACKUP_PORT=xxxx");
    tptobackup();
    data process....
    tpend();
}

```

### 3.3.13. tpgetclid

Tmaxに接続している場合、Tmaxが管理する自身のclidの値を取得する関数です。tpstartを実行しない場合、この関数は意味を持ちません。

- プロトタイプ

```

#include <tmaxapi.h>
int tpgetclid(void)

```

- 戻り値

戻り値	説明
0以上の整数値	関数の呼び出しに成功した場合です。クライアントの番号に該当する0以上の整数値を返します
-1	tpendが呼び出された状態です

- 例

```

#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>

tpstart()...

clid = tpgetclid();
if (clid==-1) { error process }

```

## 3.4. TCP/IPゲートウェイ関数

### 3.4.1. init\_remote\_info

リモート・ノードと接続を確立する前に呼び出される関数です。TCPGWが自身の初期化作業を終了した後に即時呼び出される関数であり、一度だけ呼び出されます。Tmax環境ファイルのCLOPTに[?k ]オプションで共有メモリー・キーを設定した場合、接続情報などを保存するために共有メモリーを作成するロジックを実装することができます。場合によって、内部ロジックを実装しなくても構いません。

- 使用方法

```
int init_remote_info(char *myname, int mynumber, int num_channel, int key)
```

- パラメータ

パラメータ	説明
myname	TCPGWサーバー名です
mynumber	同じTCPGWが同時に複数実行される場合、それぞれのプロセスを識別するためのTCPGWプロセス番号であり、0から開始します
num_channel	TCPGWが接続している最大のチャンネル数です。Tmax環境ファイルの[-n]、[-N]オプションに設定した値の合計です
key	Tmax環境ファイルで[?k ]オプションで設定した共有メモリー・キー値です

### 3.4.2. remote\_connected

リモート・ノードと接続を確立したときに呼び出される関数です。リモート・ノードと接続を確立した後に必要な作業があれば、この関数で実装します。チャンネルと同数の関数が呼び出され、途中チャンネルが解除された後再接続されるときにも呼び出されます。

- 使用方法

```
int remote_connected(int index, int addr, int portno, int type, int fd)
```

- パラメータ

パラメータ	説明
index	TCPGWが[-n]または[-N]オプションで複数のチャンネルを接続している場合、それぞれのチャンネルに対する自身の索引値です
addr	リモート・ノードのアドレスです

パラメータ	説明
portno	接続しているサーバーのポート番号です。TCPGWがクライアント・モードで動作する場合はリモート・ノードのポート番号であり、TCPGWがサーバー・モードで動作する場合は、リスンしているソケットのポート番号です
type	リモート・ノードと接続されたチャンネルのタイプです。IN_CHANNELかOUT_CHANNELかを表します
fd	リモート・ノードと接続されたソケット番号です

### 3.4.3. remote\_connected\_ipv6

IPv6プロトコル環境でリモート・ノードと接続を確立したときに呼び出される関数です。リモート・ノードと接続を確立した後に必要な作業があれば、この関数で実装します。チャンネルと同数の関数が呼び出され、途中チャンネルが解除された後再接続されるときにも呼び出されます。

IPv6環境ではremote\_connectedコールバック関数ではなく、remote\_connected\_ipv6コールバック関数を設定する必要があります。このコールバック関数を設定せずにIPv6環境で使用すると、remote\_connected関数が呼び出されますが、ipaddrの値に任意の値が設定され、slogに警告メッセージが記録されます。

- 使用方法

```
#include <arpa/inet.h>
int remote_connected_ipv6(int index, struct sockaddr *saddr, int portno, int type, int fd)
```

- パラメータ

パラメータ	説明
index	TCPGWが[-n]または[-N]オプションで複数のチャンネルを接続している場合、それぞれのチャンネルに対する自身の索引値です
saddr	リモート・ノードのアドレスおよびポート番号が保存されます。IPv4またはIPv6プロトコル環境に適切なsockaddr型の構造体でキャストして使用します
portno	接続されたサーバーのポート番号です。TCPGWがクライアント・モードで動作する場合はリモート・ノードのポート番号であり、TCPGWがサーバー・モードで動作する場合はリスンしているソケットのポート番号です
type	リモート・ノードと接続されたチャンネルのタイプです。IN_CHANNELかOUT_CHANNELかを表します
fd	リモート・ノードと接続されたソケット番号です

- 例

```

int
remote_connected_ipv6(int index, struct sockaddr *saddr, int portno, int type,
int fd)
{
    char buf[INET6_ADDRSTRLEN];
    char *ipaddr = NULL;
    int cli_portno;

    if (index < 0)
        return -1;

    if (saddr->sa_family == AF_INET6) {
        ipaddr = (char *)inet_ntop(AF_INET6, &(((struct sockaddr_in6
*)saddr)->sin6_addr), buf, sizeof(buf));
        cli_portno = ntohs(((struct sockaddr_in6 *)saddr)->sin6_port);
    } else if (saddr->sa_family == AF_INET) {
        ipaddr = (char *)inet_ntop(AF_INET, &(((struct sockaddr_in
*)saddr)->sin_addr), buf, sizeof(buf));
        cli_portno = ntohs(((struct sockaddr_in *)saddr)->sin_port);
    }
    if (ipaddr == NULL)
        ipaddr = "unknown";

    printf("remote_connected_ipv6, REMOTE[%d] IPADDR[%s] CLIPORT[%d] SVRPORT[%d]
TYPE[%d] fd[%d] connected\n",
        index, ipaddr, cli_portno, portno, type, fd);

    return 1;
}

```

### 3.4.4. remote\_closed

リモート・ノードとの接続を終了したときに呼び出される関数です。リモート・ノードとの接続が切断された後に必要な作業があれば、この関数で実装します。init\_remote\_info関数で共有メモリーを作成するロジックを実装した場合、この関数で解除ロジックを実装します。チャンネルと同数の関数が呼び出されます。

- 使用方法

```
int remote_closed(int index, int type)
```

- パラメータ

パラメータ	説明
index	TCPGWが[-n]または[-N]オプションで複数のチャンネルを接続している場合、それぞれのチャンネルに対する自身の索引値です



パラメータ	説明
type	リモート・ノードと接続されたチャンネルのタイプです。IN_CHANNELかOUT_CHANNELかを表します

### 3.4.5. allow\_connection

リモート・ノードがTCPGWで接続を試みたり、あるいはTCPGWがリモート・ノードに接続を試みて接続に成功した場合、remote\_connected()コールバック関数が呼び出される前に呼び出されます。当該リモートとの接続を許容するかどうかを指定します。

- 使用方法

```
int allow_connection(int addr, int portno, int type, int fd)
```

- パラメータ

パラメータ	説明
addr	リモート・ノードのアドレスです
portno	接続されたサーバーのポート番号です。TCPGWがクライアント・モードで動作する場合はリモート・ノードのポート番号であり、TCPGWがサーバー・モードで動作する場合はリスンしているソケットのポート番号です
type	リモート・ノードと接続されたチャンネルのタイプです。IN_CHANNELかOUT_CHANNELかを表します
fd	リモート・ノードと接続されたソケット番号です

- 戻り値

戻り値	説明
1	当該リモート・ノードとの接続要求を許容します
負数	当該リモート・ノードとの接続を許容しません

- 例

```
int allow_connection(int addr, int portno, int type, int fd)
{
    int i, n, len;
    struct in_addr in;
    char *ipaddr, *endptr;
    char *deny[] = {"10.10.30.1",
                   "121.100.100.*",
                   NULL};
```

```

    if (addr == -1)
        return -1;
    in.s_addr = addr;
    ipaddr = inet_ntoa(in);

    for (i = 0; deny[i] != NULL; i++) {
        if ((endptr = strpbrk(deny[i], "*")) != NULL) {
            if (strncmp(deny[i], ipaddr, (endptr - deny[i])) == 0)
                return -1;
        } else {
            if (strcmp(deny[i], ipaddr) == 0)
                return -1;
        }
    }
    return 1;
}

```

### 3.4.6. allow\_connection\_ipv6

IPv6プロトコル環境でリモート・ノードがTCPGWに接続を試みたり、あるいはTCPGWがリモート・ノードに接続を試みて接続に成功した場合、remote\_connected()コールバック関数が呼び出される前に呼び出されます。当該リモートとの接続を許容するかどうかを指定します。

IPv6環境ではallow\_connectionコールバック関数ではなく、allow\_connection\_ipv6コールバック関数を設定する必要があります。このコールバック関数を設定せずにIPv6環境で使用すると、allow\_connection関数が呼び出されますが、ipaddrの値に任意の値が設定され、slogに警告メッセージが記録されます。

#### ● 使用方法

```

#include <arpa/inet.h>

int allow_connection(struct sockaddr *saddr, int portno, int type, int fd)

```

#### ● パラメータ

パラメータ	説明
saddr	リモート・ノードのアドレスおよびポート番号です。IPv4またはIPv6プロトコル環境に適切なsockaddr型の構造体でキャストして使用します
portno	接続されたサーバーのポート番号です。TCPGWがクライアント・モードで動作する場合はリモート・ノードのポート番号であり、TCPGWがサーバー・モードで動作する場合はリスンしているソケットのポート番号です
type	リモート・ノードと接続されたチャンネルのタイプです。IN_CHANNELかOUT_CHANNELかを表します

パラメータ	説明
fd	リモート・ノードと接続されたソケット番号です

- 戻り値

戻り値	説明
1	当該リモート・ノードとの接続要求を許容します
負数	当該リモート・ノードとの接続を許容しません

- 例

```
int
allow_connection_ipv6(struct sockaddr *saddr, int svr_portno, int type, int fd)
{
    char buf[INET6_ADDRSTRLEN];
    char *ipaddr;
    int cli_portno;

    int i, n, len;
    char *endptr;
    char *allow[] = {"10.10.30.1",
                    "121.100.100.*",
                    NULL};

    if (saddr->sa_family == AF_INET6) {
        ipaddr = (char *)inet_ntop(AF_INET6, &(((struct sockaddr_in6
*)saddr)->sin6_addr), buf, sizeof(buf));
        cli_portno = ntohs(((struct sockaddr_in6 *)saddr)->sin6_port);
    } else if (saddr->sa_family == AF_INET) {
        ipaddr = (char *)inet_ntop(AF_INET, &(((struct sockaddr_in
*)saddr)->sin_addr), buf, sizeof(buf));
        cli_portno = ntohs(((struct sockaddr_in *)saddr)->sin_port);
    }
    if (ipaddr == NULL)
        ipaddr = "unknown";

    for (i = 0; allow[i] != NULL; i++) {
        if ((endptr = strpbrk(allow[i], "*")) != NULL) {
            if (strncmp(allow[i], ipaddr, (endptr - deny[i])) == 0)
                return 1;
        } else {
            if (strcmp(allow[i], ipaddr) == 0)
                return 1;
        }
    }
}
```

```

    return -1;
}

```

### 3.4.7. get\_msg\_length

リモート・ノードから要求や応答が到着し、当該チャンネルでmsg\_header\_t部分だけをrecvした後(msg\_header\_sizeまたはcomm\_header\_sizeで指定した値を読み込む)呼び出す関数で、戻された値の実データを再度読み込みます。

- 使用方法

```

int get_msg_length(msg_header_t *hp)

```

- パラメータ

パラメータ	説明
hp	開発者がcustom.hに定義できるmsg_header_t構造体のポインターです。  一般的に当該チャンネルからデータを読み込む場合、決まったヘッダー・サイズのデータを読み込んだ後、ヘッダーに設定されているデータ・サイズの値を取得して、それを基に次の実データを読み込めるようになっています。hpはTCPGWがリモート・ノードから読み込んだヘッダー・データを渡します

- 戻り値

リモート・ノードから実データのサイズを返します。関数の戻り値としてTCPGWは実データをリモート・ノードから読み込みます。ヘッダーを確認して問題がある場合に-2を戻すと、当該チャンネルが終了します。

### 3.4.8. get\_msg\_info

リモート・ノードから要求や応答が到着してデータを読み込んだ後、Tmaxサービス・プログラムに再度要求や応答を送信する前に当該データ値を加工したり、情報送信のための様々な情報(uid、len、flags、サービス名など)をTCPGWライブラリーとcustom.cとのインターフェースの役割をするinfoで参照または加工する関数です。

- 使用方法

```

int get_msg_info(msg_header_t *hp, char *data, msg_info_t *info)

```

- パラメータ

パラメータ	説明
hp	リモート・ノードから読み込んだメッセージ・ヘッダー・データのポインターです。 get_msg_lengthで使した構造体と同じです
data	リモート・ノードから読み込んだデータです
info	TCPGWライブラリー(libtcpgw.a、libtcpgw.so)とcustom.cとのインターフェースの 役割をする構造体です。ユーザーが受信したデータを基にしてinfo構造体の項目 に各種の情報を関数で設定します

- 戻り値

Tmaxサービスに送信するタイプを定義します。TCPGWは値を基にTmaxでどんな処理を行うかを判断します。たとえば、REMOTE\_REQUEST値は、リモート・ノードから要求が発生したと判断します。REMOTE\_REPLYは、Tmaxサーバーから要求が発生してリモート・ノードから応答が返される場合に戻す値です。REMOTE\_REPLY\_CONTは、応答メッセージが引き続いて返される場合に戻す値であり、REMOTE\_SENTOCLIは非要求メッセージの場合に戻す値です。

-1を戻し、環境設定ファイルのCLOPTセクションに-eオプションを設定すると、当該チャンネルが終了します。

リモート・ノードから応答を受信した場合には、UID値をinfo構造体のuid項目に指定する必要があります。その他の値は状況に合わせて指定してください。

### 3.4.9. get\_channel\_num

Tmaxサービスやクライアントから要求したデータをリモートに送信するときに、ユーザーがチャンネルを選択できるようにする関数です。ユーザーは与えられたデータの特性によって送信するチャンネルを指定できます。ただし、ここで指定するのはリモート・ノードと接続されたソケット番号ではなく、単純なチャンネル番号です。TCPGWはユーザーが指定したチャンネルを使用できない場合は、エラーを返します。

- 使用方法

```
int get_channel_num(char *data)
```

- パラメータ

パラメータ	説明
data	リモート・ノードに送信するデータです

- 戻り値

チャンネル番号を戻します。

## 3.4.10. put\_msg\_info

リモート・ノードにメッセージを送信する場合に呼び出す関数です。同期型通信の場合は、ユーザーが関数でUIDをメッセージに保存する必要があります。UIDはinfo構造体のuid項目の値を使用してもいいし、またはユーザーが任意のUIDを作成して使用した後、infoのuid項目に入力します。

ユーザーはmsg\_header\_t構造体の各項目に適切な値を保存する必要があります。構造体はユーザーが任意に設定できる項目であるため、TCPGWではmsg\_header\_tの構造体項目に値を保存しません。

msg\_header\_t構造体の長さ以上に設定する場合、dataに影響を与えるので、値を注意して使用する必要があります。

- 使用方法

```
int put_msg_info(msg_header_t *hp, char *data, msg_info_t *info)
```

- パラメータ

パラメータ	説明
hp	リモート・ノードに送信するメッセージのヘッダーです
data	リモート・ノードに送信するデータです
info	リモート・ノードに送信するデータの情報です

- 戻り値

ユーザーは実際にリモート・ノードに送信するデータの全体サイズを戻す必要があります。メッセージ・ヘッダーと実データを足したサイズを戻します。

## 3.4.11. put\_msg\_complete

リモート・ノードにメッセージを送信した後呼び出される関数で、データが完全にリモート・ノードに送信されたことを通知するために呼び出されます。

- 使用方法

```
int put_msg_complete(msg_header_t *hp, char *data, msg_info_t *info)
```

- パラメータ

パラメータ	説明
hp	リモート・ノードに送信したメッセージのヘッダーです
data	リモート・ノードに送信したデータです
info	リモート・ノードに送信したデータの情報です

### 3.4.12. get\_service\_name

Tmaxでリモート・ノードに要求を送信するときに、要求を送信するサーバーと結果を受信するサーバーが異なる非ブロック型や非同期型のTCPGWを構成する場合、tpreply()またはtpacall()を実行するサービスの名前をエラーコードによって設定します。

- 使用方法

```
int get_service_name(char *header, int err, char *svc)
```

- パラメータ

パラメータ	説明
header	[-H]または[-h]オプションで設定したTCPGWで保存しているユーザー・ヘッダーのポインターです
err	エラーコードです
svc	tpreply()またはtpacall()を受信するサービスの名前を設定します

### 3.4.13. prepare\_shutdown

TCPGWが終了する直前に呼び出される関数で、一般的にinit\_remote\_info()関数で作成した共有メモリーを解除します。

- 使用方法

```
int prepare_shutdown(int code)
```

- パラメータ

パラメータ	説明
code	終了コードで、現在は使用されていません

### 3.4.14. set\_service\_timeout

サービス・タイムアウトが発生したときに呼び出される関数です。サービス・タイムアウトが発生した後に必要な作業があれば、この関数で実装します。

- 使用方法

```
int set_service_time_out(int uid, char *header)
```

- パラメータ

パラメータ	説明
uid	サービス・タイムアウトが発生したトランザクションのユーザーIDです
header	サービス・タイムアウトが発生したトランザクションのヘッダーです

### 3.4.15. chk\_end\_msg

リモート・ノードからデータを受信するとき、データの最後を表す特定の文字またはビット・ストリームが存在する場合にユーザーが呼び出せる関数です。

- 使用方法

```
int chk_end_msg(int len, char *data)
```

- パラメータ

パラメータ	説明
len	リモート・ノードから読み込んだデータのサイズです
data	リモート・ノードから読み込んだデータです

### 3.4.16. inmsg\_recovery

ノードから要求をtpacall(..., TPBLOCK)で処理した場合、サーバーが起動されていないとエラーが返されますが、そのようなときに呼び出せる関数です。ユーザーは関数内で適切に新しいデータを作成し、データのサイズを返します。

- 使用方法

```
int inmsg_recovery(char *data, msg_info_t *info)
```

- パラメータ

パラメータ	説明
data	リモート・ノードから読み込んだデータです
info	リモート・ノードから読み込んだデータの情報で、意味のある値は以下のとおりです <ul style="list-style-type: none"> <li>– info→svc : tpacall()したサービス名</li> <li>– info→len : tpacall()したデータ長</li> </ul>



パラメータ	説明
	<ul style="list-style-type: none"> <li>info→err : エラーが発生した場合</li> <li>tperrno info→uid : 以前get_msg_info()したときに作成されたUID値</li> </ul>

- 戻り値

ユーザーは新しいデータの長さを返す必要があります。

戻り値	説明
正数	<ul style="list-style-type: none"> <li>info→svcに値が存在する場合 : 当該サービスにtpacall(..., TPNOREPLY)します</li> <li>その他の場合 : リモート・ノードに応答を送ります</li> </ul>
負数	データを捨てます

### 3.4.17. outmsg\_recovery

リモート・ノードに要求を送信した時にエラーが発生した場合に呼び出される関数です。ユーザーは関数内で適切に新しいデータを作成し、希望するサービス名およびUIDなどを設定し、データのサイズを返します。

- 使用方法

```
int inmsg_recovery(char *data, msg_info_t *info)
```

- パラメータ

パラメータ	説明
data	リモート・ノードから読み込んだデータです
info	リモート・ノードから読み込んだデータの情報で、意味のある値は以下のとおりです <ul style="list-style-type: none"> <li>info→svc : tpacall()したサービス名</li> <li>info→len : tpacall()したデータ長</li> <li>info→err : エラーが発生した場合</li> <li>tperrno info→uid : 以前get_msg_info()したとき作成されたUID値</li> </ul>

- 戻り値

戻り値	説明
正数	<ul style="list-style-type: none"> <li>– info→svcに値が存在する場合：当該サービスにtpacall(..., TPNOREPLY)します</li> <li>– info→msgtypeが1000未満の場合：データを呼び出し元に戻します</li> <li>– info→msgtypeが1000以上の場合：データを捨てます</li> </ul>

### 3.4.18. get\_extmsg\_info

get\_msg\_infoと基本的に機能は同じです。ただし、データ・バッファをダブル・ポインター型で渡して、ユーザーがバッファのメモリーを再割り当て(realloc)することができます。この関数はget\_msg\_infoと同時に使用できません。この関数を使用するには、TCPGWをコンパイル時に-D\_TCPGW\_USE\_EXTMSGフラグを設定する必要があります。

#### ● 使用方法

```
int get_extmsg_info(msg_header_t *hp, char **data, int asize, msg_info_t *info)
```

#### ● パラメータ

パラメータ	説明
hp	リモート・ノードから読み込んだメッセージ・ヘッダー・データのポインターです。get_msg_length()で使用した構造体と同じです
data	リモート・ノードから読み込んだデータ・バッファのアドレス値です
asize	リモート・ノードから読み込んだデータ・バッファに割り当てられたメモリーのサイズです
info	TCPGWライブラリー(libtcpgw.a、libtcpgw.so)とcustom.cとのインターフェースの役割をする構造体です。ユーザーが受信したデータを基にしてinfo構造体の項目に各種の情報を設定します

#### ● 戻り値

Tmaxサービスに送信するタイプを定義します。TCPGWは値を基にTmaxでどんな処理を行うかを判断します。たとえば、REMOTE\_REQUEST値はリモート・ノードから要求が発生したと判断します。REMOTE\_REPLYは、Tmaxサービスから要求が発生してリモート・ノードから応答が返される場合に戻す値です。REMOTE\_REPLY\_CONTは、応答メッセージが引き続いて返される場合に戻す値であり、REMOTE\_SENDTOCLは非要求メッセージの場合に戻す値です。リモート・ノードから応答を受信した場合には、UID値をinfo構造体のuid項目に指定する必要があります。その他の値は状況に合わせて指定してください。

### 3.4.19. put\_extmsg\_info

put\_msg\_infoと基本的に機能は同じです。ただし、データ・バッファをダブル・ポインター型で渡して、ユーザーがバッファのメモリを再割り当て(realloc)することができます。この関数はput\_msg\_infoと同時に使用できません。この関数を使用するには、TCPGWをコンパイル時に-D\_TCPGW\_USE\_EXTMSGフラグを設定する必要があります。

- 使用方法

```
int put_extmsg_info(msg_header_t *hp, char **data, int asize, msg_info_t *info)
```

- パラメータ

パラメータ	説明
hp	リモート・ノードに送信するメッセージのヘッダーです
data	リモート・ノードに送信するデータ・バッファのアドレス値です
asize	リモート・ノードに送信するデータ・バッファに割り当てられたメモリ・サイズです
info	リモート・ノードに送信するデータの情報です

- 戻り値

ユーザーは関数で実際にリモート・ノードに送信するデータの全体サイズを戻す必要があります。メッセージ・ヘッダーと実データを足したサイズを戻します。

### 3.4.20. set\_ping\_msg

チャンネル障害を監視するために送信するメッセージやメッセージの周期、タイムアウトなどを設定できるユーザー関数で、[?x]オプションを指定する場合は必ず設定する必要があります。この関数はTCPGWプログラムが起動するときに一度呼び出されます。

- 使用方法

```
int set_ping_msg(msg_header_t *hp, int *interval, int *binterval, int *timeout,  
  
int *mode)
```

- パラメータ

パラメータ	説明
hp	チャンネル障害の監視のために周期的に送信されるメッセージです。ユーザーはこのメッセージを設定する必要があります

パラメータ	説明
interval	<p>チャンネル障害の監視周期です。</p> <p>0を指定する場合、チャンネル障害検知機能は無効になります(単位: 秒)</p>
binterval	<p>バックアップ・チャンネル・モードのときにメイン・チャンネルの状態をチェックする周期です。</p> <p>0を指定する場合、メイン・チャンネルの状態チェック機能は無効になります(単位: 秒)</p>
timeout	<p>チャンネル障害の監視タイムアウトで、時間内に応答がなければ、チャンネルの接続は切断されます(単位: 秒)。</p> <p>実際にチャンネルの接続が終了される時点は、interval値に従ってpingメッセージを送信したとき、reset_ping_msg()が呼び出される前です。0を指定する場合は、pingメッセージだけ送り、pongメッセージは気にしません(半二重通信(Half Duplex)障害の検知)</p>
mode	<p>障害を監視するチャンネルの種類を指定します</p> <ul style="list-style-type: none"> <li>– 0 : OUTチャンネル</li> <li>– 1 : INチャンネル</li> <li>– 2 : OUT/IN両チャンネル</li> </ul>

- 戻り値

エラーが発生したときは負数値を返す必要があり、その場合、監視機能は無効になります。

### 3.4.21. chk\_pong\_msg

リモート・サーバーから受信したメッセージがチャンネル障害検知のための応答メッセージであるかどうかを確認する関数です。

- 使用方法

```
int chk_pong_msg(msg_header_t *hp)
```

- パラメータ

パラメータ	説明
hp	リモート・サーバーから受信したメッセージです

- 戻り値

戻り値	説明
正数	リモート・サーバーから受信したメッセージが正常なチャンネル障害監視の応答メッセージである場合です
0	リモート・サーバーから受信したメッセージがチャンネル障害監視の応答メッセージではなく、一般的なメッセージである場合です。当該メッセージはget_msg_info関数で処理されます
負数	リモート・サーバーから受信したメッセージが異常なチャンネル障害監視の応答メッセージである場合です。当該チャンネルを終了させます

### 3.4.22. set\_extping\_msg

チャンネルの障害監視のために送信するメッセージやメッセージの周期、タイムアウトなどを設定できるユーザー関数で、[?x]オプションを指定する場合は必ず設定する必要があります。この関数はTCPGWプログラムが起動するときに一度呼び出されます。

- 使用方法

```
int set_extping_msg(msg_header_t *hp, char *data, int len, int *interval,
                    int *binterval, int *timeout, int *mode)
```

- パラメータ

パラメータ	説明
hp	チャンネル障害の監視のために周期的に送信するメッセージです。ユーザーはこのメッセージを設定する必要があります
data	チャンネル障害を監視するために周期的に送信するメッセージの後に一緒に送信されるメッセージ本体です。必要な場合に設定します
len	チャンネル障害の監視メッセージ本体の最大長です。[-b]オプションで最大長を指定します。指定しない場合、サイズは0になります
interval	チャンネル障害の監視周期です。  0を指定する場合、チャンネルの障害監視機能は無効になります(単位: 秒)
binterval	バックアップ・チャンネル・モードのときにメイン・チャンネルの状態をチェックする周期です。  0を指定する場合、メイン・チャンネルの状態チェック機能は無効になります(単位: 秒)

パラメータ	説明
timeout	<p>チャンネル障害の監視タイムアウトで、時間内に応答がなければ、チャンネルの接続は切断されます(単位: 秒)。</p> <p>0を指定する場合は、pingメッセージだけ送り、pongメッセージは気にしません(半二重通信(Half Duplex)障害の検知)</p>
mode	<p>障害を監視するチャンネルの種類を指定します</p> <ul style="list-style-type: none"> <li>– 0 : OUTチャンネル</li> <li>– 1 : INチャンネル</li> <li>– 2 : OUT/IN両チャンネル</li> </ul>

- 戻り値

戻り値	説明
0	関数の実行に成功した場合は
負数	関数の実行に失敗した場合は

- 例

```
int set_extping_msg(msg_header_t *hp, char *data, int len, int *interval,
                   int *binterval, int *timeout, int *mode)
{
    msg_body_t *body;

    body = (msg_body_t *)data;
    memset(body->data, 0x00, 52);

    memcpy(body->data, "tmax50", 7);
    body->data[51] = 0;
    printf("set_extping_msg : data = %s\n", body->data);

    hp->len = 7;
    *interval = 10;
    *binterval = 20;
    *timeout = 100;
    *mode = 0; /* OUTBOUND CHANNEL */

    return 1;
}
```

### 3.4.23. chk\_extpong\_msg

リモート・サーバーから受信したメッセージがチャンネル障害検知のための応答メッセージであるかどうかを確認する関数です。

- 使用方法

```
int chk_extpong_msg(msg_header_t *hp, char *data, int len)
```

- パラメータ

パラメータ	説明
hp	チャンネルの障害が検知されたときに、リモート・サーバーから受信したメッセージのヘッダーです
data	チャンネルの障害が検知されたときに、リモート・サーバーから受信したメッセージの本体です
len	受信したメッセージ本体の長さです

- 戻り値

戻り値	説明
正数	リモート・サーバーから受信したメッセージが正常なチャンネル障害監視の応答メッセージである場合です
0	リモート・サーバーから受信したメッセージがチャンネル障害監視の応答メッセージではなく、一般的なメッセージである場合です。当該メッセージはget_msg_info関数で処理されます
負数	リモート・サーバーから受信したメッセージが異常なチャンネル障害監視の応答メッセージである場合です。当該チャンネルを終了させます

- 例

```
int chk_extpong_msg(msg_header_t *hp, char *data, int len)
{
    msg_body_t *body;
    char data2[15];

    body = (msg_body_t *)data;
    printf("chk_extpong_msg : data = %s\n", body->data);

    if (strcmp(body->data, "ping_reply")
        return 1;
    return 0;
}
```

### 3.4.24. reset\_ping\_msg

チャンネル障害検知(TCP/IP ping)メッセージの送信の可否を決定し、送信メッセージを再設定するために周期的に呼び出される関数です。

set\_ping\_msg関数で設定したinterval値に従ってpingメッセージをリモート・サーバーに送信する前に呼び出される関数で、送信メッセージを確認し、必要な場合は修正することができます。また戻り値によってpingメッセージをリモート・サーバーに送信するかどうかを決定します。

- 使用方法

```
int reset_ping_msg(msg_header_t *hp)
```

- パラメータ

パラメータ	説明
hp	チャンネル障害検知のために送信するメッセージのヘッダーです。デフォルト値としてset_ping_msg関数でユーザーが定義したヘッダーが設定されます

- 戻り値

戻り値	説明
正数	pingメッセージをリモート・サーバーに送信します
負数	pingメッセージをリモート・サーバーに送信しません

- 例

```
int reset_ping_msg(msg_header_t *hp)
{
    hp->len = 0;
    hp->msgtype = HEALTH_CHECK;
    return 1;
}
```

### 3.4.25. reset\_extping\_msg

チャンネル障害検知(TCP/IP ping)メッセージの送信の可否を決定し、送信メッセージを再設定するために周期的に呼び出される関数です。

set\_ping\_msg関数で設定したinterval値に従ってpingメッセージをリモート・サーバーに送信する前に呼び出される関数で、送信メッセージを確認し、必要な場合は修正することができます。また戻り値によってpingメッセージをリモート・サーバーに送信するかどうかを決定します。



- 使用方法

```
int reset_extping_msg(msg_header_t *hp, char *data, int len)
```

- パラメータ

パラメータ	説明
hp	チャンネル障害の検知のために送信するメッセージのヘッダーです。デフォルト値としてset_ping_msg関数でユーザーが定義したヘッダーが設定されます
data	再設定するメッセージ本体のポインターです。バッファの最大サイズはlenに設定されたサイズです。実際に送信するバッファのサイズは戻り値に指定します
len	使用可能なメッセージ本体の最大サイズです

- 戻り値

戻り値	説明
正数	pingメッセージをリモート・サーバーに送信します。指定されたサイズのメッセージ本体を送信します
0	pingメッセージをリモート・サーバーに送信します。メッセージ本体は送信しません
負数	pingメッセージをリモート・サーバーに送信しません

- 例

```
int reset_extping_msg(msg_header_t *hp, char *data, int len)
{
    int body_len;
    char *message = "reset_msg";

    body_len = min(len, strlen(message));
    strncpy(data, message, (body_len - 1));
    data[(body_len - 1)] = '\0';
    hp->len = body_len;
    printf("reset_extping_msg : data = %s\n", data);

    return body_len;
}
```

### 3.4.26. set\_error\_msg

TCPGWを介して行うリモート・ノードとのトランザクションの途中でタイムアウトまたはネットワーク切断などのエラーが発生したときに自動で呼び出される関数です。当該関数内でユーザーはユーザー・ヘッダーまたはユーザー・データを修正できます。

- 使用方法

```
int set_error_msg(msg_header_t *hp, int err, char *data, int len)
```

- パラメータ

パラメータ	説明
hp	エラーの発生時に送信されたメッセージのヘッダーで、ユーザーが修正できます
data	エラーの発生時に送信されたデータで、ユーザーが修正できます
len	メッセージ本体の長さです

- 戻り値

戻り値	説明
正数	戻された長さのデータを送信します。ただし、この場合、CLOPT="-l"オプションを設定した場合にのみ適用されます
0	ユーザー・ヘッダー部分まで送信します
負数	当該メッセージをCLHIに送信しません

- エラー

エラーコード	説明
[TPECLOSE]	リモート・ノードにサービスを要求した後、リモート・ノードとの接続が切断された場合に発生します
[TPENOENT]	[-E]オプションを使用する場合、サービス要求ごとにリモート・ノードと接続してデータを送信しますが、同時に呼び出した数が[-n]オプションで指定したチャンネルの数を超えた場合に発生します
[TPENOREADY]	リモート・ノードとの接続が切断され使用できるチャンネルがない場合に発生します
[TPEOS]	TCPGW内部でメモリを確保できない場合に発生します
[TPEPROTO]	tpforwardでTCPGWを呼び出したとき、TCPGWが同期型モードではなく非同期型モードである場合に発生します。tpforwardとtprelayの方式は必ず同期型である必要があります
[TPESVCERR]	put_msg_infoで0または負数を返す場合に発生します

エラーコード	説明
[TPESYSTEM]	TCPGWでコード変換を使用するとき、要求したデータのmapファイルがロードされない場合に発生します。またはコード変換エラーが発生してユーザー関数のput_msg_infoで負数を戻す場合、リモート・ノードにデータを送信したときにエラーが発生した場合です
[TPETIME]	リモート・ノードにサービスを要求し、指定した時間内に応答がない場合に発生します

- 例

```
int set_error_msg(msg_header_t *hp, int err, char *data, int len)
{
    msg_body_t * body;
    body = (msg_body_t *)data;
    strcpy(body->data, "changed hello data");
    /* エラー・メッセージにはデータは含まれないため、
       データまで渡すためには、-Iオプションを使用する必要があります。
       このオプションを使用しない場合、ユーザー・ヘッダー部分までCLHに渡されます。 */
    /* ユーザー・ヘッダーがない場合、hpとdataの値が同じであることがあります。 */
    strcpy(hp->retsvcname, "RECVSVC_CHANGE");
    return len;
}
```

### 3.4.27. get\_msg\_security

get\_msg\_info()またはget\_extmsg\_info()関数が実行された後に呼び出される関数です。ユーザー・データを加工する必要がある場合に使用します。get\_extmsg\_info()関数はこの関数を代替することができます。

- 使用方法

```
int get_msg_security(char **data, int asize, int len)
```

- パラメータ

パラメータ	説明
data	リモート・ノードから読み込んだデータ・バッファのアドレス値です。バッファのサイズが不足する場合は、realloc()によりバッファを拡張することを許容します。realloc()を呼び出してバッファのポインタが変更された場合には、このアドレス値に変更されたポインタ・アドレス値を保存する必要があります
asize	リモート・ノードから読み込んだデータ・バッファに割り当てられたメモリのサイズです
len	メッセージ本体の長さです

- 戻り値

戻り値	説明
正数	加工されたユーザー・データの全体長を返します
0	ユーザー・データの変更事項を反映しません

- 例

```
int get_msg_security(char **data, int asize, int len)
{
    ...
    new_size = len * 2;
    if (new_size > asize) {
        tmpbuf = (char *)realloc(*data, new_size);
        if (tmpbuf == NULL) {
            /* error processing */
        }
        *data = tmpbuf;
    }
    ...
    return new_size;
}
```

### 3.4.28. put\_msg\_security

put\_msg\_info()またはput\_extmsg\_info()関数が実行される前に呼び出される関数です。ユーザー・データを加工する必要がある場合に使用します。put\_extmsg\_info()関数はこの関数を代替することができます。

- 使用方法

```
int put_msg_security(char **data, int asize, int len)
```

- パラメータ

パラメータ	説明
data	リモート・ノードに送信するデータ・バッファのアドレス値です。バッファのサイズが不足する場合は、realloc()によりバッファを拡張することを許容します。realloc()を呼び出してバッファのポインターが変更された場合には、このアドレス値に変更されたポインター・アドレス値を保存する必要があります
asize	リモート・ノードに送信するデータ・バッファに割り当てられたメモリのサイズです
len	メッセージ本体の長さです

- 戻り値

戻り値	説明
正数	加工されたユーザー・データの全体長を返します
0	ユーザー・データの変更事項を反映しません

- 例

```
int put_msg_security(char **data, int asize, int len)
{
    ...
    new_size = len * 2;
    if (new_size > asize) {
        tmpbuf = (char *)realloc(*data, new_size);
        if (tmpbuf == NULL) {
            /* error processing */
        }
        *data = tmpbuf;
    }
    ...
    return new_size;
}
```

## 3.5. TDL関数

TDL関数の詳細については『Tmax プログラミングガイド(ダイナミックライブラリ)』を参照してください。

### 3.5.1. tdlcall

最新バージョンの動的モジュール関数を呼び出す関数です。動的モジュール関数は、必ず**long funcname(void \*args)**形式のプロトタイプをもちます。TDL環境ファイル(tdl.cfg)のVERSIONが1または2に設定された場合に使用できます。動的モジュールは、初めてtdlcall()関数が見つけられる際にロードされます。特にアップデート(tdlupdate)されない場合は再使用され、性能低下を解消します。バージョンの整合性 (Version Consistency)を維持する状況では、整合性を反映したバージョンが使用されます。

- プロトタイプ

```
#include <tdlcall.h>
int tdlcall(char *funcname, void *args, long *urcode, int flags)
```

- パラメータ

パラメータ	説明
funcname	動的モジュールの関数名です(最大サイズ:TDL_FUNCNAME_SIZE - 1)
args	呼び出される動的モジュール関数のパラメータです
urcode	呼び出された動的モジュール関数の戻り値です
flags	TDLTRANに設定可能です。設定時は、必ずurcodeでグローバル・シーケンス番号を渡す必要があります。一方、使用しない場合は、TDL_NOFLAGSに設定します

- 戻り値

戻り値	説明
TDL_OK	正常に終了した場合です
TDL_OPEN_ERROR	dlopen()の使用時にエラーが発生した場合です
TDL_SYM_ERROR	dlsym()の使用時にエラーが発生した場合です
TDL_CLOSE_ERROR	dlclose()の使用時にエラーが発生した場合です
TDL_SYSTEM_ERROR	その他のシステム・コールを使用して、エラーが発生した場合です
TDL_INT_ERROR	ライブラリー内部でエラーが発生した場合です
TDL_ENOFUNC	該当モジュールあるいは関数を見付けることができない場合です
TDL_ENV_ERROR	環境変数の設定にエラーがある場合です
TDL_VER_ERROR	TDL共有メモリーのバージョンと環境ファイル(tdl.cfg)のバージョンが一致しない場合です
TDL_ARG_ERROR	設定された引数が正しくない場合です
TDL_ENOLIB	指定したライブラリーを見付けることができない場合です
TDL_TRAN_ERROR	バージョンの整合性処理中にエラーが発生した場合です
TDL_EINACTIVE	モジュールの使用が一時的に停止された状態です

## 3.5.2. tdlcall2

最新バージョンの動的モジュール関数を呼び出す関数です。ライブラリー名と関数名をキーとして使用します。動的モジュール関数は必ず**long funcname(void \*args)**形式のプロトタイプをもちます。ライブラリー名と関数名をキーとして使用することを除けば、tdlcall()関数と同じ機能を提供します。動的モジュール関数は、必ず**long funcname(void \*args)**形式のプロトタイプをもちます。TDL環境ファイル(tdl.cfg)にVERSION=3と設定されている場合に使用できます。

- プロトタイプ

```
#include <tdlcall.h>
int tdlcall2(char *libname, char *funcname, void *args, long *urcode, int flags)
```

# ● パラメータ

パラメータ	説明
libname	動的モジュールのライブラリー名です(最大サイズ:TDL_FUNCNAME_SIZE - 1)
funcname	動的モジュールの関数名です(最大サイズ:TDL_FUNCNAME_SIZE - 1)
args	呼び出される動的モジュール関数のパラメータです
urcode	呼び出された動的モジュール関数の戻り値です
flags	TDLTRANに設定可能です。設定時は、必ずurcodeでグローバル・シーケンス番号を渡す必要があります。一方、使用しない場合は、TDL_NOFLAGSに設定します

# ● 戻り値

戻り値	説明
TDL_OK	正常に終了した場合です
TDL_OPEN_ERROR	dlopen()の使用時にエラーが発生した場合です
TDL_SYM_ERROR	dlsym()の使用時にエラーが発生した場合です
TDL_CLOSE_ERROR	dlclose()の使用時にエラーが発生した場合です
TDL_SYSTEM_ERROR	その他のシステム・コールを使用して、エラーが発生した場合です
TDL_INT_ERROR	ライブラリー内部でエラーが発生した場合です
TDL_ENOFUNC	該当モジュールあるいは関数を見付けることができない場合です
TDL_ENV_ERROR	環境変数の設定にエラーがある場合です
TDL_VER_ERROR	TDL共有メモリーのバージョンと環境ファイル(tdl.cfg)のバージョンが一致しない場合です
TDL_ARG_ERROR	設定された引数が正しくない場合です
TDL_ENOLIB	指定したライブラリーを見付けることができない場合です
TDL_TRAN_ERROR	バージョンの整合性処理中にエラーが発生した場合です
TDL_EINACTIVE	モジュールの使用が一時的に停止された状態です

### 3.5.3. tdlcall2s

最新バージョンの動的モジュール関数を呼び出す関数です。ライブラリー名と関数名をキーとして使用します。動的モジュール関数は、必ず**long funcname(void \*input, void \*output)**形式のプロトタイプをもちます。ライブラリー名と関数名をキーとして使用することを除けば、tdlcall()関数と同じ機能を提供します。TDL環境ファイル(tdl.cfg)にVERSION=3と設定されている場合に使用できます。

- プロトタイプ

```
#include <tdlcall.h>

int tdlcall2s(char *libname, char *funcname, void *input, void *output,
              long *urcode, int flags)
```

- パラメータ

パラメータ	説明
libname	動的モジュールのライブラリー名です(最大サイズ:TDL_FUNCNAME_SIZE - 1)
funcname	動的モジュールの関数名です(最大サイズ:TDL_FUNCNAME_SIZE - 1)
input	呼び出される動的モジュール関数の入力バッファ・ポインターです
output	呼び出される動的モジュール関数の出力バッファ・ポインターです
urcode	呼び出された動的モジュール関数の戻り値です
flags	TDLTRANに設定可能です。設定時は、必ずurcodeでグローバル・シーケンス番号を渡す必要があります。一方、使用しない場合は、TDL_NOFLAGSに設定します

- 戻り値

戻り値	説明
TDL_OK	正常に終了した場合です
TDL_OPEN_ERROR	dlopen()の使用時にエラーが発生した場合です
TDL_SYM_ERROR	dlsym()の使用時にエラーが発生した場合です
TDL_CLOSE_ERROR	dlclose()の使用時にエラーが発生した場合です
TDL_SYSTEM_ERROR	その他のシステム・コールを使用して、エラーが発生した場合です
TDL_INT_ERROR	ライブラリー内部でエラーが発生した場合です
TDL_ENOFUNC	該当モジュールあるいは関数を見付けることができない場合です
TDL_ENV_ERROR	環境変数の設定にエラーがある場合です
TDL_VER_ERROR	TDL共有メモリーのバージョンと環境ファイル(tdl.cfg)のバージョンが一致しない場合です



戻り値	説明
TDL_ARG_ERROR	設定された引数が正しくない場合です
TDL_ENOLIB	指定したライブラリーを見付けることができない場合です
TDL_TRAN_ERROR	バージョンの整合性処理中にエラーが発生した場合です
TDL_EINACTIVE	モジュールの使用が一時的に停止された状態です

### 3.5.4. tdlcall2v

最新バージョンの動的モジュール関数を呼び出す関数です。ライブラリー名と関数名をキーとして使用します。動的モジュール関数は、必ず**long funcname(int argc, char \*argv[])**形式のプロトタイプをもちます。ライブラリー名と関数名をキーとして使用することを除けば、tdlcall()関数と同じ機能を提供します。TDL環境ファイル(tdl.cfg)にVERSION=3と設定されている場合に使用できます。

- プロトタイプ

```
#include <tdlcall.h>
int tdlcall2v(char *libname, char *funcname, int argc, char *argv[],
              long *urcode, int flags)
```

- パラメータ

パラメータ	説明
libname	動的モジュールのライブラリー名です(最大サイズ:TDL_FUNCNAME_SIZE - 1)
funcname	動的モジュールの関数名です(最大サイズ:TDL_FUNCNAME_SIZE - 1)
argc	呼び出される動的モジュール関数のパラメータ数です
argv	呼び出される動的モジュール関数のパラメータ・ベクトルです
urcode	呼び出された動的モジュール関数の戻り値です
flags	TDLTRANに設定可能です。設定時は、必ずurcodeでグローバル・シーケンス番号を渡す必要があります。一方、使用しない場合は、TDL_NOFLAGSに設定します

- 戻り値

戻り値	説明
TDL_OK	正常に終了した場合です
TDL_OPEN_ERROR	dlopen()の使用時にエラーが発生した場合です
TDL_SYM_ERROR	dlsym()の使用時にエラーが発生した場合です

戻り値	説明
TDL_CLOSE_ERROR	dlclose()の使用時にエラーが発生した場合です
TDL_SYSTEM_ERROR	その他のシステム・コールを使用して、エラーが発生した場合です
TDL_INT_ERROR	ライブラリー内部でエラーが発生した場合です
TDL_ENOFUNC	該当モジュールあるいは関数を見付けることができない場合です
TDL_ENV_ERROR	環境変数の設定にエラーがある場合です
TDL_VER_ERROR	TDL共有メモリーのバージョンと環境ファイル(tdl.cfg)のバージョンが一致しない場合です
TDL_ARG_ERROR	設定された引数が正しくない場合です
TDL_ENOLIB	指定したライブラリーを見付けることができない場合です
TDL_TRAN_ERROR	バージョンの整合性処理中にエラーが発生した場合です
TDL_EINACTIVE	モジュールの使用が一時的に停止された状態です

### 3.5.5. tdlcallva

最新バージョンの動的モジュール関数を呼び出す関数で、関数名をキーに使用します。動的モジュール関数のプロトタイプが固定されていない場合には、パラメータをそのまま渡して関数を呼び出します。ただし、渡されるパラメータはすべて(void \*)ポインター型を保持しなければなりません。動的モジュール関数でも渡されるパラメータはすべて(void \*)ポインター型で作成されている必要があります。

TDL環境ファイル(tdl.cfg)でVERSION=1またはVERSION=2に設定されている場合に使用できます。

#### ● プロトタイプ

```
#include <tdlcall.h>
int tdlcallva(char *funcname, long urcode, int flags, int rettype, void *retval,
               int argc, ...);
```

#### ● パラメータ

パラメータ	説明
funcname	動的モジュールの関数名です(最大サイズ: TDL_FUNCNAME_SIZE - 1)
urcode	他のtdlcall()系の関数と違い、グローバル・シーケンス番号を渡すときにのみ使用します。使用しない場合には0を入力します
flags	TDLTRANで設定可能です。設定時には、必ずurcodeでグローバル・シーケンス番号を渡す必要があります。使用しない場合には、TDL_NOFLAGSで設定します
rettype	呼び出される動的モジュール関数の返却型を定義します。設定可能な値は下の表を参照してください

パラメータ	説明
retval	呼び出される動的モジュール関数の戻り値を保存するバッファのポインタを渡します
argc	呼び出される動的モジュール関数に渡される引数(argument)の数を入力します。現在、最大10個まで可能です
...	呼び出される動的モジュール関数に渡される実際のパラメータを可変引数で入力します。必ず(void *)のポインタ型のパラメータを渡す必要があります

- 返却型

項目	説明
TDL_VA_CHAR	呼び出される動的モジュール関数の返却型がchar型です
TDL_VA_SHORT	呼び出される動的モジュール関数の返却型がshort型です
TDL_VA_INT	呼び出される動的モジュール関数の返却型がint型です
TDL_VA_LONG	呼び出される動的モジュール関数の返却型がlong型です
TDL_VA_FLOAT	呼び出される動的モジュール関数の返却型がfloat型です
TDL_VA_DOUBLE	呼び出される動的モジュール関数の返却型がdouble型です
TDL_VA_PVOID	呼び出される動的モジュール関数の返却型がvoid *型です
TDL_VA_VOID	呼び出される動的モジュール関数の返却型がvoid型です

- 戻り値

戻り値	説明
TDL_OK	正常に終了した場合です
TDL_OPEN_ERROR	dlopen()の使用時にエラーが発生した場合です
TDL_SYM_ERROR	dlsym()の使用時にエラーが発生した場合です
TDL_CLOSE_ERROR	dlclose()の使用時にエラーが発生した場合です
TDL_SYSTEM_ERROR	その他のシステム・コールを使用して、エラーが発生した場合です
TDL_INT_ERROR	ライブラリー内部でエラーが発生した場合です
TDL_ENOFUNC	該当モジュールあるいは関数を見付けることができない場合です
TDL_ENV_ERROR	環境変数の設定にエラーがある場合です
TDL_VER_ERROR	TDL共有メモリのバージョンと環境ファイル(tdl.cfg)のバージョンが一致しない場合です
TDL_ARG_ERROR	設定された引数が正しくない場合です
TDL_ENOLIB	指定したライブラリーを見付けることができない場合です
TDL_TRAN_ERROR	バージョンの整合性処理中にエラーが発生した場合です

戻り値	説明
TDL_EINACTIVE	モジュールの使用が一時的に停止された状態です

- 例

- 動的モジュール関数

```
int myfunc(double *a, double *b, double *c) {
    double sum;
    return (int)(((double)(*a) + (double)(*b) + (double)(*c))/3);
}

char * myfunc2(int *type) {
    char *msg;
    switch (*type) {
        case 1: msg = "foo"; break;
        case 2: msg = "bar"; break;
    }
    return msg;
}
```

- 呼び出しプログラム

```
#include <tdlcall.h>
int main(int argc, char *argv[]) {
    int n;
    long urcode;
    int retval;
    double a, b, c;
    int type;
    char *retmsg;

    urcode = tdlgetseqno();
    a = 2.5;
    b = 3.0;
    c = 3.5;
    if ((n = tdlcallva2("mylib001", "myfunc", urcode, TDLTRAN,
        TDL_VA_INT, &retval, 3, &a, &b, &c)) != TDL_OK) {
        error processing;
    }

    type = 1;
    if ((n = tdlcallva2("mylib001", "myfunc2", urcode, TDLTRAN,
        TDL_VA_PVOID, &retmsg, 1, &type)) != TDL_OK) {
        error processing;
    }
}
```

```
}  
}
```

## 3.5.6. tdlcallva2

最新バージョンの動的モジュール関数を呼び出す関数で、ライブラリー名と関数名をキーに使用します。動的モジュール関数のプロトタイプが固定されていない場合には、パラメータをそのまま渡して関数を呼び出します。ただし、渡されるパラメータはすべて(void \*)ポインター型を保持しなければいけません。動的モジュール関数でも渡されるパラメータはすべて(void \*)ポインター型で作成されている必要があります。

ライブラリー名と関数名をキーに使用することを除いては、tdlcallva()関数と同様の機能を提供します。TDL環境ファイル(tdl.cfg)にVERSION=3と設定されている場合に使用できます。

### ● プロトタイプ

```
#include <tdlcall.h>  
int tdlcallva2(char *libname, char *funcname, long urcode, int flags, int rettype,  
void *retval, int argc, ...);
```

### ● パラメータ

パラメータ	説明
libname	動的モジュールのライブラリー名です(最大サイズ: TDL_FUNCNAME_SIZE - 1)
funcname	動的モジュールの関数名です(最大サイズ: TDL_FUNCNAME_SIZE - 1)
urcode	他のtdlcall()系の関数と違い、グローバル・シーケンス番号を渡すときにのみ使用します。使用しない場合には0を入力します
flags	TDLTRANに設定可能です。設定時には、必ずurcodeでグローバル・シーケンス番号を渡す必要があります。使用しない場合は、TDL_NOFLAGSに設定します
rettype	呼び出される動的モジュール関数の返却型を定義します。設定可能な値は下表の返却型を参照してください
retval	呼び出される動的モジュール関数の戻り値を保存するバッファのポインターを渡します
argc	呼び出される動的モジュール関数に渡される引数(argument)の数を入力します。現在、最大10個まで可能です
...	呼び出される動的モジュール関数に渡される実際のパラメータを可変引数で入力します。必ず(void *)のポインター型のパラメータを渡す必要があります

### ● 返却型

項目	説明
TDL_VA_CHAR	呼び出される動的モジュール関数の返却型がchar型です
TDL_VA_SHORT	呼び出される動的モジュール関数の返却型がshort型です
TDL_VA_INT	呼び出される動的モジュール関数の返却型がint型です
TDL_VA_LONG	呼び出される動的モジュール関数の返却型がlong型です
TDL_VA_FLOAT	呼び出される動的モジュール関数の返却型がfloat型です
TDL_VA_DOUBLE	呼び出される動的モジュール関数の返却型がdouble型です
TDL_VA_PVOID	呼び出される動的モジュール関数の返却型がvoid *型です
TDL_VA_VOID	呼び出される動的モジュール関数の返却型がvoid型です

- 戻り値

戻り値	説明
TDL_OK	正常に終了した場合は
TDL_OPEN_ERROR	dlopen()の使用時にエラーが発生した場合です
TDL_SYM_ERROR	dlsym()の使用時にエラーが発生した場合です
TDL_CLOSE_ERROR	dlclose()の使用時にエラーが発生した場合です
TDL_SYSTEM_ERROR	その他のシステム・コールを使用して、エラーが発生した場合です
TDL_INT_ERROR	ライブラリー内部でエラーが発生した場合です
TDL_ENOFUNC	該当モジュールあるいは関数を見付けることができない場合です
TDL_ENV_ERROR	環境変数の設定にエラーがある場合です
TDL_VER_ERROR	TDL共有メモリーのバージョンと環境ファイル(tdl.cfg)のバージョンが一致しない場合です
TDL_ARG_ERROR	設定された引数が正しくない場合です
TDL_ENOLIB	指定したライブラリーを見付けることができない場合です
TDL_TRAN_ERROR	バージョンの整合性処理中にエラーが発生した場合です
TDL_EINACTIVE	モジュールの使用が一時的に停止された状態です

### 3.5.7. tdlcreate

最新バージョンの動的モジュールで、クラス・ファクトリーを使用してクラス・インスタンスを作成する関数です。ライブラリー名と名前空間、クラス名をキーとして使用し、TDL環境ファイル(tdl.cfg)にVERSION=4と設定されている場合に使用できます。

- プロトタイプ

```
#include <tdlcall.h>
int tdlcreate(char *libname, char *namespace, char *classname, void *args, void
*object, long *urcode, int flags)
```

# ● パラメータ

パラメータ	説明
libname	動的モジュールのライブラリー名です(最大サイズ:TDL_FUNCNAME_SIZE - 1)
namespace	名前空間を指定します(最大サイズ:TDL_FUNCNAME_SIZE - 1)
classname	インスタンスを作成するクラス名です(最大サイズ:TDL_FUNCNAME_SIZE - 1)
args	tdlcreate_cb()コールバック関数でユーザー定義データを渡すパラメータです
object	tdlcreate_cb()コールバック関数で生成されたクラス・インスタンスの参照を保存する変数のポインターです。関数の呼び出しに成功したら、objectパラメータを適切なタイプに変換して使用します
urcode	tdlcreate_cb()コールバック関数で実行された結果の戻り値です
flags	TDLTRANに設定可能です。設定時には、必ずurcodeでグローバル・シーケンス番号を渡す必要があります。使用しない場合は、TDL_NOFLAGSに設定します

# ● 戻り値

戻り値	説明
TDL_OK	正常に終了した場合です
TDL_OPEN_ERROR	dlopen()の使用時にエラーが発生した場合です
TDL_SYM_ERROR	dlsym()の使用時にエラーが発生した場合です
TDL_CLOSE_ERROR	dlclose()の使用時にエラーが発生した場合です
TDL_SYSTEM_ERROR	その他のシステム・コールを使用して、エラーが発生した場合です
TDL_INT_ERROR	ライブラリー内部でエラーが発生した場合です
TDL_ENOFUNC	該当モジュールあるいは関数を見付けることができない場合です
TDL_ENV_ERROR	環境変数の設定にエラーがある場合です
TDL_VER_ERROR	TDL共有メモリーのバージョンと環境ファイル(tdl.cfg)のバージョンが一致しない場合です
TDL_ARG_ERROR	設定された引数が正しくない場合です
TDL_ENOLIB	指定したライブラリーを見付けることができない場合です
TDL_TRAN_ERROR	バージョンの整合性処理中にエラーが発生した場合です

戻り値	説明
TDL_EINACTIVE	モジュールの使用が一時的に停止された状態です

動的モジュールでクラス・ファクトリーを使用するには、`long tdlcreate_cb(char *namespace, char *classname, void *args, void *object)`形式のコールバック関数を実装しなければなりません。コールバック関数は、`tdlcreate()`で渡したパラメータ情報を利用して実際にクラス・インスタンスを作成し、作成されたインスタンスの参照を`object`に渡すように実装します。

この関数で作成したインスタンスの使用が完了すると、`tdldestroy()`関数によりインスタンスを破棄しなければなりません。`tdlcreate()`で作成したインスタンスを`tdldestroy()`で削除するまでは、`tdlupdate`が途中で呼び出され、動的モジュールが新しいバージョンに変更されても、現在作成されて使用されているインスタンスは既存のバージョンで動作します。このような場合、`tdldestroy()`を呼び出した後、再び`tdlcreate()`を呼び出すと、変更された動的モジュールのインスタンスが生成されます。

クラス・ファクトリーを使用するために、動的モジュールは、`tdlcreate_cb()`および`tdldestroy_cb()`コールバック関数を実装する必要があります。

- コールバック関数のプロトタイプ

```
long tdlcreate_cb(char *namespace, char *classname, void *args, void *object)
```

- パラメータ

パラメータ	説明
namespace	<code>tdlcreate()</code> 関数から渡された名前空間です
classname	<code>tdlcreate()</code> 関数から渡された、インスタンス作成のクラス名です
args	ユーザー定義データです
object	コールバック関数で生成されたクラス・インスタンスの参照を保存する変数のポインターです

### 3.5.8. tdldestroy

最新バージョンの動的モジュールで、クラス・ファクトリーを使用して生成されたクラス・インスタンスを破棄する関数です。ライブラリー名と名前空間、クラス名をキーとして使用し、TDL環境ファイル(`tdl.cfg`)に`VERSION=4`と設定されている場合に使用可能です。

- プロトタイプ

```
#include <tdlcall.h>
int tdldestroy(char *libname, char *namespcae, char *classname, void *args, void *object, long *urcode, int flags)
```

- パラメータ



パラメータ	説明
libname	動的モジュールのライブラリー名です(最大サイズ:TDL_FUNCNAME_SIZE - 1)
namespace	名前空間を指定します(最大サイズ:TDL_FUNCNAME_SIZE - 1)
classname	インスタンスを作成するクラス名です(最大サイズ:TDL_FUNCNAME_SIZE - 1)
args	tdldestroy_cb()コールバック関数でユーザー定義データを渡すパラメータです
object	破棄するクラス・インスタンスの参照を渡します。tdldestroy_cb()コールバック関数で当該参照を破棄します。tdlcreate()関数で作成したオブジェクトのみを使用する必要があります
urcode	tdldestroy_cb()コールバック関数で実行された結果の戻り値です
flags	TDLTRANに設定可能です。設定時には、必ずurcodeでグローバル・シーケンス番号を渡す必要があります。使用しない場合は、TDL_NOFLAGSに設定します

- 戻り値

戻り値	説明
TDL_OK	正常に終了した場合です
TDL_OPEN_ERROR	dlopen()の使用時にエラーが発生した場合です
TDL_SYM_ERROR	dlsym()の使用時にエラーが発生した場合です
TDL_CLOSE_ERROR	dlclose()の使用時にエラーが発生した場合です
TDL_SYSTEM_ERROR	その他のシステム・コールを使用して、エラーが発生した場合です
TDL_INT_ERROR	ライブラリー内部でエラーが発生した場合です
TDL_ENOFUNC	該当モジュールあるいは関数を見付けることができない場合です
TDL_ENV_ERROR	環境変数の設定にエラーがある場合です
TDL_VER_ERROR	TDL共有メモリーのバージョンと環境ファイル(tdl.cfg)のバージョンが一致しない場合です
TDL_ARG_ERROR	設定された引数が正しくない場合です
TDL_ENOLIB	指定したライブラリーを見付けることができない場合です
TDL_TRAN_ERROR	バージョンの整合性処理中にエラーが発生した場合です
TDL_EINACTIVE	モジュールの使用が一時的に停止された状態です

動的モジュールでクラス・ファクトリーを使用するには、**long tdldestroy\_cb(char \*namespace, char \*classname, void \*args, void \*object)**形式のコールバック関数を実装しなければなりません。コールバック関数は、tdldestroy()で渡したパラメータ情報を利用してクラス・インスタンスを破棄するように実装します。

tdlcreate()で作成したインスタンスの使用が完了すると、この関数によりインスタンスを破棄しなければなりません。tdlcreate()で作成したインスタンスをtdldestroy()で削除するまでは、tdlupdateが途中で呼び出され、動的モジュールが新しいバージョンに変更されても、現在作成されて使用されているインスタンスは既存のバージョンで動作します。このような場合、tdldestroy()を呼び出した後、再びtdlcreate()を呼び出すと、変更された動的モジュールのインスタンスが生成されます。

- コールバック関数のプロトタイプ

```
long tdldestroy_cb(char *namespace, char *classname, void *args, void *object)
```

- パラメータ

パラメータ	説明
namespace	tdlcreate()関数から渡された名前空間です
classname	tdlcreate()関数から渡された、インスタンス作成のクラス名です
args	ユーザー定義データです
object	コールバック関数で破棄するクラス・インスタンスの参照です

## 3.5.9. tdlend

明示的なバージョン整合性(Explicit Version Consistency)の維持を終了する関数です。

- プロトタイプ

```
#include <tdlcall.h>
int tdlend(void)
```

- 戻り値

戻り値	説明
TDL_OK	関数が正常に実行された場合です
TDL_TRAN_ERROR	関数の呼び出し前にtdlstart()が実行された場合です。詳細については <a href="#">「3.5.1. tdlcall」</a> を参照してください

## 3.5.10. tdlerror

tdlcall()に対するエラーが発生した場合、文字列形式に変換する関数です。直前のtdlcall()関数でエラーが発生した場合、エラーに対する詳細情報を文字列形式で渡します。戻り値で渡されるポインターは内部の静的変数に対するポインターであるため、次のtdlcall()を呼び出す場合に他の内容で設定され、捨てられることがあることに注意します。

したがって、この内容を保存あるいは修正したい場合は、他の変数にコピーをして使用します。

- プロトタイプ

```
#include <tdlcall.h>
char* tdlerror(int retval)
```

- パラメータ

パラメータ	説明
retval	直前のtdlcall()関数の戻り値です

- 戻り値

戻り値	説明
dlopen fail	tdlcall()の戻り値で、TDL_OPEN_ERRORを受信した場合です
dlsym fai	tdlcall()の戻り値で、TDL_SYM_ERRORを受信した場合です
dlclose fail	tdlcall()の戻り値で、TDL_CLOSE_ERRORを受信した場合です
etc system call error	tdlcall()の戻り値で、TDL_SYSTEM_ERRORを受信した場合です
TDL lib internal error	tdlcall()の戻り値で、TDL_INT_ERRORを受信した場合です
funcname not found	tdlcall()の戻り値で、TDL_ENOFUNCを受信した場合です
environment not found	tdlcall()の戻り値で、TDL_ENV_ERRORを受信した場合です
shared version mismatch	tdlcall()の戻り値で、TDL_VER_ERRORを受信した場合です
invalid arguments	tdlcall()の戻り値で、TDL_ARG_ERRORを受信した場合です
library not found	tdlcall()の戻り値で、TDL_ENOLIBを受信した場合です
transaction error	tdlcall()の戻り値で、TDL_TRAN_ERRORを受信した場合です
inactive funcation	tdlcall()の戻り値で、TDL_EINACTIVEを受信した場合です

### 3.5.11. tdlresume

一時的に停止していたバージョン整合性の維持を再開する関数です。

- プロトタイプ

```
#include <tdlcall.h>
int tdlresume(int sd)
```

- プロトタイプ

パラメータ	説明
sd	tdlsuspend()関数から渡された記述子です

- 戻り値

戻り値	説明
TDL_OK	正常に終了した場合です
TDL_TRAN_ERROR	sdが有効な値でない場合です。詳細については「 <a href="#">3.5.1. tdlcall</a> 」を参照してください

## 3.5.12. tdlstart

明示的なバージョン整合性(Explicit Version Consistency)の維持を開始する関数です。

- プロトタイプ

```
#include <tdlcall.h>
int tdlstart(void)
```

- 戻り値

戻り値	説明
TDL_OK	関数の呼び出しに成功した場合です
TDL_TRAN_ERROR	関数の呼び出し前にtdlstart()が実行された場合です。詳細については「 <a href="#">3.5.1. tdlcall</a> 」を参照してください

## 3.5.13. tdlsuspend

一時的にバージョン整合性の維持を停止する関数です。sd(suspend descriptor)を戻り値として返します。最大同時sd数は8です。

- プロトタイプ

```
#include <tdlcall.h>
int tdlsuspend(void)
```

- 戻り値

戻り値	説明
0、または0より大きい値	関数の呼び出しに成功した場合です

戻り値	説明
TDL_TRAN_ERROR	現在のバージョン整合性の維持モードでない場合です。詳細については <a href="#">「3.5.1. tdlcall」</a> を参照してください

### 3.5.14. tdlgetseqno

グローバル・シーケンス番号を取得する関数です。取得した番号を戻り値として返します。

- プロトタイプ

```
#include <tdlcall.h>
unsigned int tdlgetseqno(void)
```

- 戻り値

戻り値	説明
0より大きい値	関数の呼出しに成功した場合です
0	関数の呼出しに失敗した場合です。 <a href="#">「3.5.10. tdlerror」</a> で確認できます

### 3.5.15. tdlclose

当該モジュールの参照カウントを0に初期化するか、モジュールを直接メモリーから解除します。

- プロトタイプ

```
#include <tdlcall.h>
int tdlclose(char *name, int flags)
```

- パラメータ

パラメータ	説明
name	当該モジュールの名前です
flags	0に設定した場合は、当該モジュールの参照カウントのみを0に初期化し、メモリーから解除しません。TDLCLOSE_HARDに設定する場合は、dlcloseして、当該モジュールをメモリーから解除します

- 戻り値

戻り値	説明
TDL_OK	関数の呼び出しに成功した場合です

戻り値	説明
TDL_ENOLIB	該当する名前のモジュールが存在しない場合です

## 3.5.16. tdlload

tdlcall()を使って特定モジュールを初めて呼び出す場合、共有メモリーのハッシュ表(Hashtable)の検索およびライブラリーのメモリーへのロードによって若干の時間がかかります。これによって初回呼び出しがやや遅れてしまうことを防ぐために、ハッシュ表の検索とライブラリーのロードを、tdlcall()を使用する前に実行しておき、ローカル・キャッシュに当該モジュールの情報を保存する関数です。

TDLの環境ファイル(tdl.cfg)の設定が、VERSION=1またはVERSION=2の場合はtdlload()を使用します。また、VERSION=3の場合はtdlload2()を使用し、VERSION=4の場合はtdlload3()を使用します。

### ● プロトタイプ

```
#include <tdlcall.h>
int tdlload(char *funcname, int flags)
```

### ● パラメータ

パラメータ	説明
funcname	ロードを実行する関数の名前です
flags	現在使われていません

### ● 戻り値

戻り値	説明
0	関数を正常に呼び出した場合です
TDL_ENOFUNC	関数の引数としてlibnameやfuncnameがNULLで渡された場合や、ハッシュ表に存在しない値が渡された場合です
TDL_OPEN_ERROR	ハッシュ表に存在するダイナミック・ライブラリーをメモリーにロードできない場合です
TDL_SYSTEM_ERROR	ローカル・キャッシュを作成するためのシステム・リソースの割当に失敗した場合です

## 3.5.17. tdlload2

tdlloadと同様の関数です。詳細内容は「[3.5.16. tdlload](#)」を参照してください。

### ● プロトタイプ

```
#include <tdlcall.h>
int tdlload2(char *libname, char *funcname, int flags)
```

- パラメータ

パラメータ	説明
libname	ロードを実行するライブラリーの名前です
funcname	ロードを実行する関数の名前です
flags	現在使われていません

- 戻り値

tdlloadと同様の戻り値を持ちます。詳細内容は「[3.5.16. tdlload](#)」を参照してください。

## 3.5.18. tdlinit

共有メモリーを初期設定する関数です。

- プロトタイプ

```
#include <tdlcall.h>
int tdlinit(int flags)
```

- パラメータ

パラメータ	説明
flags	現在使われていません

- 戻り値

戻り値	説明
TDL_OK	関数の呼出しに成功した場合です
0より小さい値	関数の呼出しに失敗した場合です。「 <a href="#">3.5.10. tdlerror</a> 」で確認できます

## 3.5.19. tdlldone

共有メモリーを初期化する関数です。

- プロトタイプ

```
#include <tdlcall.h>
int tdlldone(int flags)
```

- パラメータ

パラメータ	説明
flags	現在使われていません

- 戻り値

戻り値	説明
TDL_OK	関数の呼出しに成功した場合です
0より小さい値	関数の呼出しに失敗した場合です。 <a href="#">「3.5.10. tdlerror」</a> で確認できます

## 3.5.20. tdlfind

モジュールの索引を探す関数です。TDL環境ファイル(tdl.cfg)がVERSION=1またはVERSION=2に設定されている場合に使用します。

- プロトタイプ

```
#include <tdlcall.h>
int tdlfind(char *funcname, int flags)
```

- パラメータ

パラメータ	説明
funcname	検索する関数の名前です
flags	現在使われていません

- 戻り値

戻り値	説明
0、または0より大きい値	関数の呼出しに成功した場合です
0より小さい値	関数の呼出しに失敗した場合です。 <a href="#">「3.5.10. tdlerror」</a> で確認できます



## 3.5.21. tdlfind2

モジュールの索引を探す関数です。TDL環境ファイル(tdl.cfg)がVERSION=3に設定されている場合に使用します。

- プロトタイプ

```
#include <tdlcall.h>
int tdlfind2(char *libname, char *funcname, int flags)
```

- パラメータ

パラメータ	説明
libname	検索するライブラリーの名前です
funcname	検索する関数の名前です
flags	現在使われていません

- 戻り値

戻り値	説明
0、または0より大きい値	関数の呼出しに成功した場合です
0より小さい値	関数の呼出しに失敗した場合です。 <a href="#">「3.5.10. tdlerror」</a> で確認できます

## 3.5.22. tdlstat

TDLの統計情報を出力する関数です。

TDLの環境ファイル(tdl.cfg)がVERSION=1またはVERSION=2に設定されている場合に使用します。環境設定をMONITOR=Yに設定する必要があります。

- プロトタイプ

```
#include <tdlcall.h>
int tdlstat(char *funcname, struct timeval svc_time, struct timeval cpu_time)
```

- パラメータ

パラメータ	説明
funcname	統計情報を収集する関数の名前です
svc_time	統計情報のうちサービス・タイムが記録される変数です
cpu_time	統計情報のうちCPUタイムが記録される変数です

- 戻り値

戻り値	説明
TDL_OK	関数の呼出しに成功した場合です
0より小さい値	関数の呼出しに失敗した場合です。 <a href="#">「3.5.10. tderror」</a> で確認できます

## 3.5.23. tdlstat2

TDLの統計情報を出力する関数です。

TDL環境ファイル(tdl.cfg)がVERSION=3に設定されている場合に使用します。環境設定をMONITOR=Yに設定する必要があります。

- プロトタイプ

```
#include <tdlcall.h>
int tdlstat2(char *libname, char *funcname, struct timeval svc_time, struct
timeval cpu_usertime, struct timeval cpu_sysstime)
```

- パラメータ

パラメータ	説明
libname	統計情報を収集するライブラリーの名前です
funcname	統計情報を収集する関数の名前です
svc_time	統計情報のうちサービス・タイムが記録される変数です
cpu_usertime	統計情報のうちCPUのユーザー・タイムが記録される変数です
cpu_sysstime	統計情報のうちCPUのシステム・タイムが記録される変数です

- 戻り値

戻り値	説明
TDL_OK	関数の呼出しに成功した場合です
0より小さい値	関数の呼出しに失敗した場合です。 <a href="#">「3.5.10. tderror」</a> で確認できます

## 3.6. Windows関連関数

### 3.6.1. WinTmaxAcall

Windowsシステム環境でクライアントを使用する関数で、マルチスレッド環境での非同期サービスの送信を要求します。マルチスレッド環境で**tpacall()**と同じ機能をする関数で、新規スレッドを作成し、スレッド内で**tpstart()**→**tpacall()**→**tpgetrply()**を実行します。tpgetrply()を呼び出した後、SendMessage(wHandle、msgType、(UINT)&msg、tperrno)を呼び出します。

- プロトタイプ

```
# include <tmaxapi.h>
int WinTmaxAcall(TPSTART_T *sinfo, HANDLE wHandle, unsigned int msgtype,
                 char *svc, char *sndbuf, int len, int flags)
```

- パラメータ

パラメータ	説明
sinfo	Tmaxシステムにクライアントの情報を渡す必要がある場合に使用する構造体で、tpstart()のパラメータと同じです
wHandle	メッセージを受け取るウィンドウ・ハンドルを指定します
msgtype	到着メッセージを指定します。一般的にWM_USERを開発者の任意で定義して使用します。svc Tmax環境ファイルに登録されたサービス名を指定します
svc	送信を要求するサービスを指定します
sndbuf	サービスの呼び出し時に渡されるデータです。NULLでない場合は、必ずtpalloc()で割り当てられたバッファを使用します
len	送信するデータ長を指定します。CARRAY、X_OCTET、構造体の配列タイプの場合は、必ず設定します
flags	tpacall()のフラグをそのまま使用します。  flagsで使用可能な値は以下のとおりです  – TPBLOCK  フラグなしでtpacall()が使用された場合、svcに呼び出されたサービスがないか、正しくない結果が返されても、正常な結果が返されます。tpgetrply()の呼び出し時にエラーが返されます。TPBLOCKフラグを使用してtpacall()を呼び出した場合、サービス状態の正常可否を確認できます  – TPNOTRAN  トランザクション・モードでsvcがトランザクションをサポートしないサービスの場合、tpacall()がトランザクション・モードで呼び出された場合、フラグは必ずTPNOTRAN

パラメータ	説明
	<p>と設定します。tpacall()の呼び出し元がトランザクション・モード状態でTPNOTRANを設定し、svcサービスを要求した場合、svcサービスはトランザクション・モードから除外されて実行されます。トランザクション・モード内でのtpacall()の呼び出し時、TPNOTRANと設定されていても、トランザクション・タイムアウト(timeout)に影響を受けます。TPNOTRANで呼び出されたサービスが失敗した場合、呼び出し元のトランザクションには影響を与えません</p> <p>– TPNOREPLY</p> <p>tpacall()で送信したサービス要求は、応答を待機せずに即時返します。結果は後にtpacall()で返した記述子を使用してtpgetrply()で結果を受信します。flagsをTPNOREPLYと設定した場合、サービスに対する応答を受けないと設定されます。TPNOREPLYと設定した場合、tpacall()はサービスが正常に呼び出された場合、0を返します。関数の呼び出し元がトランザクション・モードの場合、TPNOTRANフラグと一緒に設定しなければTPNOREPLYフラグを使用できません。TPNOREPLYフラグの場合、サービス状態の正常可否をチェックするためにはTPBLOCKフラグと一緒に設定する必要があります。TPBLOCKフラグを設定していない場合、サービスがNRDYの場合にもエラーを返しません</p> <p>– TPNOBLOCK</p> <p>内部バッファが送信するメッセージでいっぱいになった場合と同じブロッキング状況になれば、サービス要求は失敗するように設定するフラグです。TPNOBLOCKフラグの設定なしでtpacall()の呼び出し時にブロッキング状況が発生した場合、関数の呼び出し元はブロッキング状況が解除されるか、タイムアウト(トランザクション・タイムアウトまたはブロック・タイムアウト)が発生するまで待機します</p> <p>– TPNOTIME</p> <p>関数の呼び出し元がブロック・タイムアウトを無視し、応答が受信されるまで無限に待機するように設定するフラグです。トランザクション・タイムアウト内でtpacall()を呼び出した場合は、トランザクション・タイムアウトが適用されます</p> <p>– TPSIGRSTRT</p> <p>シグナル割り込みを受け入れる場合、使用するフラグでシステム関数の呼び出しが妨害されたとき、関数の呼び出しが再実行されます。TPSIGRSTRTが設定されていない状態でシグナル割り込みが発生した場合、関数は失敗し、tperrnoにTPGOTSIGが設定されます</p>

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です。記述子を返します。記述子は送信されたサービス要求に対する応答を受信するのに使用されます
-1	関数の呼び出しに失敗した場合です。tperrnoにエラーコードが設定されます

## ● エラー

WinTmaxAcall()が正常に実行されなかった場合、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、svcがNULLである場合や、データがtpalloc()で割り当てられていないバッファを指す場合、フラグが有効でない場合に発生します
[TPENOENT]	svcというサービスが存在しないため、サービスを要求できない場合です
[TPEITYPE]	dataのタイプおよびサブタイプが、svcがサポートしていないタイプです。構造体の場合、使用された構造体がSDLFILEファイルに宣言されていません
[TPELIMIT]	処理されていない非同期性サービスの要求数が限界に達したため、呼び出し元のサービス要求が送信されなかった場合です
[TPETRAN]	トランザクション・サービスの呼び出し時に、データベースに問題が発生し、xa_startが失敗した場合です
[TPETIME]	タイムアウトが発生した場合です。関数の呼び出し元がトランザクション・モードの場合、トランザクション・タイムアウトが発生し、トランザクションはロールバックされます。そうでない場合、TPNOTIMEやTPNOBLOCKがすべて設定されていない状態でブロック・タイムアウトが発生します。トランザクション・タイムアウトが発生した場合、トランザクションがロールバックされるまで新規サービスの要求を送信することや、まだ受信していない応答を待機することは、すべて[TPETIME]エラーと処理されます
[TPEBLOCK]	TPNOBLOCKが設定された状態でブロッキング状況が発生した場合です
[TPGOTSIG]	TPSIGRSTRTが設定されていない状態でシグナルを受信した場合です
[TPEPROTO]	トランザクション・モードでのTPNOREPLYサービスの呼び出しや、TPNOTRANフラグを設定していないなどの不適切な状況で発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です
[TPEOS]	運用システムにエラーが発生した場合です

## ● 例

```
...
#include <usrinc/tmaxapi.h>
#define WM_WINTMAX_RECV WM_USER + 1
...
```

```

BEGIN_MESSAGE_MAP(CWinTmaxAcall2TestDlg, CDialog)
...
ON_MESSAGE(WM_WINTMAX_RECV, OnWinTmaxAcall)
...
END_MESSAGE_MAP()

BOOL CWinTmaxAcall2TestDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    ...
    ret = tmaxreadenv("tmax.env", "TMAX");
    if (ret == -1){
        AfxMessageBox("tmaxreadenv fail...");
        return FALSE;
    }
    return TRUE; // return TRUE unless you set the focus to a control
}

void CWinTmaxAcall2TestDlg::OnOK()
{
    ...
    GetDlgItemText(IDC_EDIT1, m_Input);
    lstrcpy((LPTSTR)buf, (LPCTSTR)m_Input.operator const char *());
    ...

    buf = tmalloc("STRING", NULL, 0);
    if (buf == NULL){
        AfxMessageBox("buf alloc fail...");
        return FALSE;
    }

    ret = WinTmaxAcall((TPSTART_T *)NULL, m_hWnd, WM_WINTMAX_RECV,
        "TOUPPER", buf, 0, TPNOFLAGS);
    if (ret == -1){
        error processing
    }
}

LRESULT CWinTmaxAcall2TestDlg::OnWinTmaxAcall(WPARAM wp, LPARAM lp)
{
    char msg[100];
    memset(msg, 0x00, 100);
    TPSVCINFO *get = (TPSVCINFO *)wp;
    if (lp < 0){
        error processing
    }
    ...
}

```

```
SetDlgItemText(IDC_EDIT2, get->data);
return 0;
}
```

● 関連関数

tpacall(), WinTmaxAcall2()

3.6.2. WinTmaxAcall2

Windowsシステム環境でクライアントを使用する関数で、マルチスレッド環境での非同期サービスの送信を要求します。マルチスレッド環境で**tpacall()**と同じ機能をする関数で、新規スレッドを作成し、スレッド内で**tpstart()**→**tpacall()**→**tpgetrply()**を実行します。tpgetrply()を呼び出した後、指定されたロールバック関数に受信されたデータを渡します。

● プロトタイプ

```
# include <tmaxapi.h>
int WinTmaxAcall2(TPSTART_T *sinfo, WinTmaxCallback fn, char *svc,
                  char *sndbuf, int len, int flags)
```

● パラメータ

パラメータ	説明
sinfo	Tmaxシステムにクライアントの情報を渡す必要がある場合に使用する構造体で、tpstart()のパラメータと同一です
fn	サービス要求に対する応答を受けるロールバック関数を指定します
svc	Tmax環境ファイルに登録されたサービス名を指定します
sndbuf	サービスの呼び出し時に渡されるデータです。NULLでない場合、必ずtpalloc()で割り当てられたバッファを使用します
len	送るデータの長さを指定します。CARRAY、X_OCTET、構造体の配列タイプの場合は必ず設定します
flags	tpacall()のフラグをそのまま使します。  flagsで使用可能な値は以下のとおりです  – TPBLOCK  フラグなしでtpacall()が使用されると、svcに呼び出されたサービスがない場合、もしくは正しくない結果が返された場合、正常な結果が返されます。tpgetrply()を呼び出し時にエラーが返されます。TPBLOCKフラグを使用してtpacall()を呼び出した場合、サービス状態が正常かどうかを確認できます

パラメータ	説明
	<p>– TPNOTRAN</p> <p>トランザクション・モードでsvcがトランザクションをサポートしていないサービスの場合、tpacall()がトランザクション・モードで呼び出された場合、フラグは必ずTPNOTRANに設定する必要があります。tpacall()の呼び出し元がトランザクション・モード状態でTPNOTRANを設定してsvcサービスを要求した場合、svcサービスはトランザクション・モードから除外されて実行されます。トランザクション・モード内でtpacall()関数を呼び出した場合、TPNOTRANに設定されていても、トランザクション・タイムアウトに影響を受けます。TPNOTRANで呼び出されたサービスが失敗した場合、呼び出し元のトランザクションには影響を与えません</p> <p>– TPNOREPLY</p> <p>tpacall()で送信したサービス要求は、応答を待たずに即時返します。結果は、後にtpacall()で返した記述子を利用して、tpgetrply()で結果を受信します。</p> <p>flagsをTPNOREPLYに設定した場合、サービスに対する応答を受けないように設定されます。TPNOREPLYに設定した場合、tpacall()はサービスが正常に呼び出された場合に0を返します。関数の呼び出し元がトランザクション・モードである場合、TPNOTRANフラグと一緒に設定しなければ、TPNOREPLYフラグを使用できません。TPNOREPLYフラグの場合、サービス状態が正常かどうかをチェックするためには、TPBLOCKフラグと一緒に設定する必要があります。TPBLOCKフラグを設定していなければ、サービスがNRDYの場合でもエラーを返しません</p> <p>– TPNOBLOCK</p> <p>内部バッファが送信メッセージでいっぱいになったときのようなブロッキング状況になった場合、サービス要求が失敗するように設定するフラグです。TPNOBLOCKフラグが設定されていない状態でtpacall()が呼び出された場合にブロッキング状況が発生すると、ブロッキング状況が解除されるか、タイムアウト(トランザクション・タイムアウト、またはブロック・タイムアウト)が発生するまで関数の呼び出し元は待機します</p> <p>– TPNOTIME</p> <p>関数の呼び出し元がブロック・タイムアウトを無視し、応答が受信されるまで無限に待機するように設定するフラグです。トランザクション・タイムアウト内でtpacall()を実行した場合、トランザクション・タイムアウトが適用されます</p> <p>– TPSIGRSTRT</p> <p>シグナル割り込みを受け入れる場合に使用します。システム関数の呼び出しが妨害されたとき、関数の呼び出しが再実行されます。TPSIGRSTRTが設定されていない状態でシグナル割り込みが発生した場合、関数は失敗し、tpermoにTPGOTSIGが設定されます</p>



- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tpernolにエラー・コードが設定されます

- エラー

WinTmaxAcall2()が正常に実行されなかった場合、tpernolに以下の値のうち1つが設定されます。

エラー・コード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、svcがNULLである場合や、dataがtpalloc()で割り当てられていないバッファを指す場合、フラグが有効でない場合に発生します
[TPENOENT]	svcというサービスが存在しないため、サービスを要求できない場合です
[TPEITYPE]	dataのタイプおよびサブタイプが、svcがサポートしていないタイプです。構造体の場合、使用された構造体がSDLFILEファイルに宣言されていません
[TPELIMIT]	処理されていない非同期性サービス要求数が限界に達したため、呼び出し元のサービス要求が送信されなかった場合です
[TPETRAN]	svcはトランザクションをサポートしていないサービスであり、TPNOTRANが設定されていません
[TPETIME]	タイムアウトが発生した場合です。関数の呼び出し元がトランザクション・モードの場合、トランザクション・タイムアウトが発生したものであり、トランザクションはロールバックされます。そうでない場合、TPNOTIMEやTPNOBLOCKがすべて設定されていない状態でブロック・タイムアウトが発生します。トランザクション・タイムアウトが発生した場合、トランザクションがロールバックされるまで新規サービス要求を送信することや、まだ受信されていない応答を待機することは、すべて[TPETIME]エラーと処理されます
[TPEBLOCK]	TPNOBLOCKが設定された状態でブロッキング状況が発生した場合です
[TPGOTSIG]	TPSIGRSTRTが設定されていない状態でシグナルを受信した場合です
[TPEPROTO]	トランザクション・モードでTPNOREPLYサービスを呼び出す時、TPNOTRANフラグを設定していないなどの不適切な状況で発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
#include <usrinc/tmaxapi.h>
#define WM_WINTMAX_RECV WM_USER + 1
int mycallfn(unsigned int, long);
```

```

...

BEGIN_MESSAGE_MAP(CWinTmaxAcall2TestDlg, CDialog)
...
END_MESSAGE_MAP()

BOOL CWinTmaxAcall2TestDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    ...
    ret = tmaxreadenv("tmax.env", "TMAX");
    if (ret == -1){
        AfxMessageBox("tmaxreadenv fail...");
        return FALSE;
    }
    return TRUE; // return TRUE unless you set the focus to a control
}

void CWinTmaxAcall2TestDlg::OnOK()
{
    ...
    GetDlgItemText(IDC_EDIT1, m_Input);
    lstrcpy((LPTSTR)buf, (LPCTSTR)m_Input.operator const char *());
    ...
    buf = tmalloc("STRING", NULL, 0);
    if (buf == NULL){
        AfxMessageBox("buf alloc fail...");
        return FALSE;
    }
    ret = WinTmaxAcall2((TPSTART_T *)NULL, (WinTmaxCallback)mycallfn,
        "TOUPPER", (char *)buf, 0, TPNOFLAGS);
    if (ret == -1){
        error processing
    }
}

int mycallfn(unsigned int msg, long retval)
{
    TPSVCINFO *svcinfn;
    char infomsg[30];
    memset(infomsg, 0x00, sizeof(infomsg));
    svcinfo = (TPSVCINFO *)msg;
    strncpy(infomsg, svcinfo->data, svcinfo->len);
    if (retval != 0){
        strcpy(infomsg, tpstrerror(retval));
        AfxMessageBox(infomsg);
        return -1;
    }
}

```

```

    } else {
        strncpy(infomsg, svcinfo->data, sizeof(infomsg) - 1);
        AfxMessageBox(infomsg);
        return 1;
    }
}

```

- 関連関数

tpacall(),WinTmaxAcall()

### 3.6.3. WinTmaxEnd

Windowsシステム環境にてクライアントで使用され、Tmaxシステムとの接続を終了する関数です。機能上では**tpend()**と同じです。マルチスレッド環境で使用する場合、スレッドごとに**WinTmaxStart()**を使用して接続を行うため、WinTmaxEnd()もスレッドごとに使用します。

- プロトタイプ

```

#include <WinTmax.h>
int WinTmaxEnd(void)

```

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラー・コードが設定されます

- エラー

WinTmaxEnd()が正常に実行されなかった場合、tperrnoに以下の値のうち1つが設定されます。

エラー・コード	説明
[TPETIME]	臨界領域に入ることができない場合に発生します。システム内部エラーで、システム状態を確認します
[TPEPROTO]	tpend()が不適切な状況で呼び出された場合です。たとえば、呼び出し元がサーバーである場合や、接続が解除された後に再度呼び出された場合に発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログ・ファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です

- 関連関数

tpend(), WinTmaxStart()

### 3.6.4. WinTmaxSend

Windowsシステム環境にてクライアントで使用され、データを送信する関数です。マルチウィンドウ環境で使用され、サービスを要求する関数です。

- プロトタイプ

```
# include <WinTmax.h>
int WinTmaxSend(int recvContext, char *svc, char *data, long len, long flags)
```

- パラメータ

パラメータ	説明
recvContext	Tmaxシステムから応答を受信する場合、受信対象ウィンドウを指定します。WinTmaxSetContext()の戻り値です
svc	サービス名です
data	tpalloc()で割り当てられたバッファに、サービス要求時に渡すデータが保存されます
len	データ長の値で、CARRAYバッファあるいは多重構造体バッファを使用する場合、正確な長さの値を指定する必要があります
flags	WinTmaxSend()で使用できるフラグ値は、下で説明します

WinTmaxSend()は、tpacall()関数と似た方式で動作し、サービスを要求した後、応答が到着するまで待たずに即返します。以降、WinTmaxStart()を呼び出したときに生成されたスレッドで応答を処理し、WinTmaxSetContext()で設定されたウィンドウで応答結果を通知します。したがって、応答を受信するための別途のAPIは提供していません。

以下は、空のスロットを1つ探して(hwd, 0x300)を保存した後、索引を返す例です。WinTmaxSetContext()の戻り値のrcをWinTmaxSend()のrecvContextパラメータとして使用し、WinTmaxSend(rc, svc, data, len, 0)の呼び出し後に応答が返されると、hwd windowsに0 x 300番のメッセージおよび応答結果を送信します。

```
rc = WinTmaxSetContext(hwd, 0x300, -1);
int nSendResult = WinTmaxSend(rc, (LPSTR)(LPCSTR)strService,
(Char*)tpbuf, nLen, TPNOFLAGS);
/*
    以降、内部スレッドで応答メッセージを処理し、該当ウィンドウに次のようなメッセージを
```

送信します。

```
SendMessage(hwd, 0x300, (UINT) &msg, callRet);  
* /
```

メッセージを受信したウィンドウでは、WPARAMに応答データ(msg)を、LPARAMに応答結果(callRet)を受けます。callRetが0の場合は正常な応答であり、-1の場合は、エラーが発生したことを示します。このとき、tperrno値により、エラーの原因を把握できます。

WinTmaxSend()は構造上、トランザクションをサポートせず、Tmaxシステムの環境設定のBLOCKTIME項目またはtpsettimeout()の影響を受けません。ただし、WinTmaxSend()を呼び出すときにTPBLOCKフラグを与えた場合にのみ、そのフラグの動作方式にBLOCKTIMEが適用されます。

WinTmaxSend()で利用できるフラグ値は以下のとおりです。

フラグ値	説明
TPBLOCK	<p>TPBLOCKフラグなしでWinTmaxSend()関数が使用された場合、svcが登録されていなかったり、サービスの実行に失敗したときでも、正常に結果が返されます。エラーは応答を受けたときに確認できます。</p> <p>TPBLOCKフラグを利用する場合、サービスの呼び出し時にその正常可否を確認できます。つまり、WinTmaxSend()はBLOCKTIME以内に要求したサービスが正常に実行できるかを確認して返します。もし要求したサービスが実行できない場合は、それについてのエラーをtperrnoに保存し、-1を返します。</p> <p>BLOCKTIME以内にサービスの実行可否についての確認が不可能な場合には、TPETIMEエラーを返します。この場合、クライアントでは要求したサービスが実行されたかどうかを確認できないため、エラー・ルーチンを作成するときに注意が必要です。もし再要求処理をしなければならない場合は、既存の要求が処理されたかどうかを確認して処理する必要があります</p>
TPNOREPLY	<p>WinTmaxSend()で送信したサービス要求は、応答を待機せずに即時返します。結果は、作業スレッドによって受信して登録されたウィンドウに渡されます。しかし、TPNOREPLYフラグはサービスに対する応答を受けないと設定されます</p>
TPNOTRAN	<p>WinTmaxSend()関数の呼び出し元がトランザクション・モード状態でフラグを設定してsvcサービスを要求した場合、svcサービスはトランザクション・モードから除外されて実行されます。</p> <p>トランザクション・モードでsvcがトランザクションをサポートしていないサービスの場合、WinTmaxSend()関数がトランザクション・モードで呼び出される場合、フラグはTPNOTRANと設定する必要があります。トランザクション・モード内でWinTmaxSend()関数を呼び出した場合、TPNOTRANに設定されていても、トランザクション・タイムアウトに影響を受けます。TPNOTRANで呼び出されたサービスが失敗した場合、呼び出し元のトランザクションは影響を及ぼしません</p>

フラグ値	説明
TPNOBLOCK	TPNOBLOCKフラグが設定した状態で、内部バッファが送信メッセージでいっぱいになったときのようなブロッキング状況になった場合、サービス要求は失敗します。TPNOBLOCKフラグが設定されていない状態でWinTmaxSend()が呼び出された場合にブロッキング状況が発生すると、ブロッキング状況が解除されるか、タイムアウト(トランザクション・タイムアウト、またはブロック・タイムアウト)が発生するまで関数の呼び出し元は待機します
TPNOTIME	関数の呼び出し元がブロック・タイムアウトを無視し、応答が受信されるまで無限に待機するように設定するフラグです。しかし、トランザクション・タイムアウト内でWinTmaxSend()を実行した場合、トランザクション・タイムアウトが適用されます
TPSIGRSTRT	シグナル割り込みを受け入れる場合に使用します。システム関数の呼び出しが妨害されたとき、システム関数の呼び出しが再実行されます。フラグが設定されていない状態でシグナル割り込みが発生した場合、関数は失敗し、tperrnoにTPGOTSIGが設定されます

- 戻り値

戻り値	説明
記述子(descriptor)	関数の呼び出しに成功した場合です。記述子を返します。現在この記述子は使用されていません
-1	関数の呼び出しに失敗した場合です。tperrnoにエラー・コードが設定されます

- エラー

WinTmaxSend()が正常に実行されなかった場合、tperrnoに以下の値のうち1つが設定されます。

エラー・コード	説明
[TPENOENT]	svcというサービスが存在しないため、サービスを要求できない場合です
[TPEITYPE]	dataのタイプおよびサブタイプは、svcがサポートしていないタイプです。構造体の場合、使用された構造体がSDLFILEファイルに宣言されていません
[TPELIMIT]	処理されていない非同期性サービス要求数が限界に達したため、呼び出し元のサービス要求が送信されなかった場合です
[TPETIME]	タイムアウトが発生した場合です。TPNOTIMEとTPNOBLOCKのどちらも設定されていない状況でブロック・タイムアウトが発生します
[TPEBLOCK]	TPNOBLOCKが設定された状態でブロッキング状況が発生した場合です
[TPGOTSIG]	TPSIGRSTRTが設定されていない状態でシグナルを受信した場合です
[TPEPROTO]	トランザクション・モードでTPNOREPLYサービスを呼び出す場合や、TPNOTRANフラグを設定しない場合などの不適切な状況で発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です

エラー・コード	説明
[TPECLOSE]	ネットワーク問題やその他の様々な原因により、Tmaxシステムとの接続が解除された場合です
[TPEOS]	運用システムにエラーが発生した場合です

Windowsに渡されたメッセージのLPARAMが-1の場合、tperrnoに以下の値のうち1つが設定されます。

エラーコード	説明
[TPEBADDESC]	応答メッセージを受信したが、正しい記述子を探せない場合に発生します
[TPEOTYPE]	応答メッセージを受信したが、クライアントとサーバー・プログラムのバッファ・タイプが一致しない場合に発生します
[TPEPROTO]	WinTmaxStart()、WinTmaxEnd()などを不適切な状況で呼び出した場合に発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です
[TPECLOSE]	ネットワーク問題やその他の様々な原因により、Tmaxシステムとの接続が解除された場合です
[TPEOS]	運用システムにエラーが発生した場合です

#### ● 例

```
...
int CTMaxGwView::SendData2(char *data, long nLen)
{
    CString szTemp;
    m_send_length.Format("%d", nLen);
    UpdateData(FALSE);

    char *tpbuf = tpalloc("STRING", NULL, nLen);
    if (tpbuf == NULL) {
        szTemp.Format("tpalloc Error [%s]", tpstrerror(tperrno));
        LogDisplay2(2, (char *) (const char *) szTemp, szTemp.GetLength());

        return -1;
    }

    memcpy(tpbuf, data, nLen);

    CString strService;
    strService.Format("TOUPPER_STRING");
```

```

    int nSendResult=WinTmaxSend(2,(LPSTR)(LPCSTR)strService, (char*)tpbuf,
    nLen, 0);
    tpfree(tpbuf);
    ...
}

```

- 関連関数

WinTmaxStart(),WinTmaxEnd()

### 3.6.5. WinTmaxSetContext

Windowsシステム環境にて、クライアントで使用され、ウィンドウ・ハンドルとメッセージ型を設定する関数です。

マルチウィンドウ環境で使用され、指定されたウィンドウ・ハンドルとメッセージ・タイプをライブラリーの空スロットに保存し、管理するようにします。このように管理される情報は、作業スレッドが応答を受信後、指定されているウィンドウに指定されたメッセージ・タイプでデータを転送します。以下は、データ転送フォーマットです。

```
SendMessage(winhandle, msgType, (UINT) &msg, callRet)
```

応答を受けると、作業スレッドは、SendMessage(winhandle, msgType, (UINT)&msg, callRet)でデータを転送します。この場合、指定されたウィンドウではWPARAMにはmsgが対応し、LPARAMにはcallRetが対応します。msgはTPSVCINFO構造体であり、callRetはサービス処理結果値として、適切なメッセージが受信された場合は0で、正しくないメッセージが受信された場合は-1です。

たとえば、正常なtpreturn(TPSUCCESS, ...)でサービスが処理された場合や、失敗時にtpreturn(TPFAIL, ...)で処理された場合や、非要求メッセージが受信された場合は、適切なメッセージと見なし、callRet値は0になります。しかし、同期型結果値や対話型メッセージが渡される場合、これはマルチウィンドウ環境で使用できないタイプであるため、callRet値は-1になります。callRet値が-1の場合、tperrnoの値を確認してエラーの原因を把握することができます。

- プロトタイプ

```

# include <WinTmax.h>
int WinTmaxSetContext(void *winhandle, unsigned int msgType, int slot)

```

- パラメータ

パラメータ	説明
winhandle	受信データを処理するウィンドウ・ハンドルです
msgType	メッセージ・タイプです



パラメータ	説明
slot	<p>指定されたウィンドウ・ハンドルとメッセージ・タイプを割り当てるスロットを意味します。割り当て可能なスロットの最大数は256で、システムの内部的に0と1は、それぞれデフォルト出力とエラー用に設定されます。データ受信プロセスでエラーが発生したり、出力用ウィンドウが指定されていない場合は、デフォルト・ウィンドウが使用されます。</p> <p>スロットは再定義して使用することができます。非要求メッセージの場合は、ユーザーが指定したウィンドウがないため、0番のデフォルト出力ウィンドウにメッセージが渡されます。</p> <p>以下は、スロットに設定できる値です</p> <ul style="list-style-type: none"> <li>– -1 : システムの内部的に空のスロットを自動的に検索し、指定されたウィンドウ・ハンドルとメッセージ・タイプを割り当てます</li> <li>– 0以上の値 : 与えられた索引のスロットに指定されたウィンドウとメッセージ・タイプを割り当てます</li> </ul>

- 戻り値

戻り値	説明
index	関数の呼び出しに成功した場合です。スロットに対する索引を返し、索引はWinTmaxSend()で最初のパラメータとして使用されます
-1	関数の呼び出しに失敗した場合です。tpernoにエラー・コードが設定されます

- エラー

WinTmaxSetContext()が正常に実行されなかった場合、tpernoに以下の値のうち1つが設定されます。

エラー・コード	説明
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です
[TPEOS]	運用システムにエラーが発生した場合です

- 例

```
...
int CTMaxGwView::Connect()
{
    CString szTemp, FName;
    WinTmaxEnd();
    int Ret = tmaxreadenv(TMAXINI, "TMAX117");
    if(Ret<0)
    {
        szTemp.Format("tmaxreadenv error");
    }
}
```

```

        LogDisplay2(2, (char *) (const char *) szTemp, szTemp.GetLength());
        return FALSE;
    }
    if (WinTmaxStart((TPSTART_T *) NULL) == -1) {
        szTemp.Format("WinTmaxStart    = [%s]", tpstrerror(tperrno));
        LogDisplay2(2, (char *) (const char *) szTemp, szTemp.GetLength());
        return FALSE;
    }

    WinTmaxSetContext(m_hWnd, WM_TMAX_RECV_RDP, 0);
    WinTmaxSetContext(m_hWnd, WM_TMAX_RECV_ERR, 1);
    WinTmaxSetContext(m_hWnd, WM_TMAX_RECV, 2);

    return TRUE;
}

```

- 関連関数

WinTmaxStart(), WinTmaxEnd(), WinTmaxSend()

## 3.6.6. WinTmaxStart

Windowsシステム環境にてクライアントで使用され、マルチウィンドウ環境でTmaxシステムと接続するのに使用される関数です。機能上では**tpstart()**と同じです。マルチスレッドを使用する場合、それぞれのスレッド別にWinTmaxStart()を使用して、別途で接続を行い、サービス呼び出す必要があります。

- プロトタイプ

```

#include <WinTmax.h>
int WinTmaxStart(TPSTART_T *tpinfo)

```

- パラメータ

TPSTART\_Tについては「[3.3.10. tpstart](#)」を参照してください。

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tperrnoにエラー・コードが設定されます

- エラー

WinTmaxStart()が正常に実行されなかった場合、tperrnoに以下の値のうち1つが設定されます。

エラー・コード	説明
[TPEINVAL]	引数が有効でない場合です。たとえば、tpinfoがNULLである場合や、TPSTART_T構造体に対するポインターでない場合に発生します
[TPEITYPE]	tpinfoがTPSTART_T構造体に対するポインターでない場合に発生します
[TPEPROTO]	WinTmaxStart()が不適切な状況で呼び出された場合です。たとえば、サーバー・プログラムで使用された場合や、すでに接続されている状況で再度呼び出された場合に発生します
[TPESYSTEM]	Tmaxシステムにエラーが発生した場合です。詳細情報はログ・ファイルに記録されます
[TPEOS]	運用システムにエラーが発生した場合です。あるいは、環境変数の設定が正しくない場合です。たとえば、TMAX_HOST_ADDRやTMAX_HOST_PORTの設定が正しくないために接続に失敗した場合に発生します

- 関連関数

tpstart(), WinTmaxEnd(), WinTmaxSend(), WinTmaxSetContext()

## 3.7. セキュリティー関数

### 3.7.1. tmax\_init\_crypt

暗号化のための初期化プロセスを実行する関数です。この関数によりクライアントとCASプロセス間の暗号化通信の際に使用されるトークンを作成します。この関数は実際にtpstart()の内部でクライアントとCAS両方で呼び出されます。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_init_crypt(const TMAX_SEC_T *ctoken_from_peer, TMAX_SEC_T
*ctoken_to_peer, TMAX_SEC_T *own_ctoken)
```

- パラメータ

パラメータ	説明
ctoken_from_peer	クライアントはctoken_from_peerを使用しません。CASはクライアントが送信したトークンをctoken_from_peerから受信します
ctoken_to_peer	クライアントからCASに送信するトークンがある場合、ctoken_to_peerに割り当てます。CASはクライアントに再度渡すトークンがある場合、ctoken_to_peerに割り当てます。このctoken_to_peerはクライアントのtmax_accept_crypt関数のctoken_from_peerパラメータから受信します

パラメータ	説明
own_ctoken	クライアントとCASの両方で内部で維持しなければならない暗号化関連トークンをown_ctokenに割り当てます

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です

- 関連関数

tmax\_fini\_crypt(), tmax\_accept\_crypt(), tmax\_wrap(), tmax\_unwrap()

### 3.7.2. tmax\_fini\_crypt

tmax\_init\_crypt関数により作成されたown\_ctokenを整理する関数です。クライアントまたはCASで暗号化プロセスが失敗または正常終了したときに呼び出されます。

- プロトタイプ

```
# include <tmaxapi.h>
int tmax_fini_crypt(TMAX_SEC_T *own_ctoken)
```

- パラメータ

パラメータ	説明
own_atoken	tmax_init_crypt関数により作成されたセキュリティー・トークンです

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です

- 関連関数

tmax\_init\_crypt(), tmax\_accept\_crypt(), tmax\_wrap(), tmax\_unwrap()

### 3.7.3. tmax\_accept\_crypt

tmax\_init\_cryptと同様にtpstart()内部でクライアントとCASの両方で呼び出されます。

クライアントではCASからtmax\_init\_cryptに対する応答がくると、この関数が呼び出されます。CASではクライアントのtmax\_accept\_cryptが成功すると、この関数が呼び出されます。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_accept_crypt(const TMAX_SEC_T *ctoken_from_peer, TMAX_SEC_T
*ctoken_to_peer, TMAX_SEC_T *own_ctoken)
```

- パラメータ

パラメータ	説明
ctoken_from_peer	クライアントはCASのtmax_init_crypt関数のctoken_to_peerトークンをここで受信します。CASはクライアントのtmax_accept_cryptのctoken_to_peerトークンをここで受信します
ctoken_to_peer	クライアントはCASに送信するトークンがある場合、ctoken_to_peerに割り当てます。CASのtmax_accept_cryptはこのパラメータを使用しません
own_ctoken	own_ctokenはtmax_init_crypt関数により作成された暗号化情報が含まれたトークンです。再加工する部分があれば、own_ctokenに割り当てます

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です

- 関連関数

tmax\_init\_crypt(), tmax\_fini\_crypt(), tmax\_wrap(), tmax\_unwrap()

### 3.7.4. tmax\_wrap

クライアントとTmax間のメッセージを暗号化する関数です。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_wrap(const TMAX_SEC_T *own_ctoken, const TMAX_SEC_T *pmsg, TMAX_SEC_T
*cmsg)
```

- パラメータ

パラメータ	説明
own_ctoken	クライアントとCASの両方共、tmax_init_cryptとtmax_accept_cryptにより作成されたown_ctokenです
pmsg	まだ暗号化されていないメッセージです
cmsg	暗号化プロセスが終わったメッセージが割り当てられます

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です

- 関連関数

tmax\_init\_crypt(), tmax\_fini\_crypt(), tmax\_accept\_crypt(), tmax\_unwrap()

## 3.7.5. tmax\_unwrap

クライアントとTmax間の暗号化されたメッセージを復号化する関数です。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_unwrap(const TMAX_SEC_T *own_ctoken, const TMAX_SEC_T *cmsg, TMAX_SEC_T
*pmsg)
```

- パラメータ

パラメータ	説明
own_ctoken	クライアントとCASの両方共、tmax_init_cryptとtmax_accept_cryptにより作成されたown_ctokenです
cmsg	暗号化されたメッセージです
pmsg	復号化プロセスが終わった平文が割り当てられます

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です

戻り値	説明
-1	関数の呼び出しに失敗した場合です

- 関連関数

tmax\_init\_crypt(), tmax\_fini\_crypt(), tmax\_accept\_crypt(), tmax\_unwrap()

### 3.7.6. tmax\_init\_auth

クライアントがtpstart()内部で呼び出す関数です。認証および認可のためのセキュリティー・トークンを作成します。

- プロトタイプ

```
# include <atmi.h>
# include <tmaxapi.h>
int tmax_init_auth(const TPSTART_T *info, TMAX_SEC_T *atoken_to_peer, TMAX_SEC_T
*own_atoken)
```

- パラメータ

パラメータ	説明
info	tpstart()でのTPSTART_T *tpinfo引数をそのまま受信します
atoken_to_peer	クライアントがCASに認証を要求するために渡す情報がある場合、ここに割り当てます
own_atoken	infoを基にクライアントが維持するセキュリティー・トークンを作成します。ここで作成されたセキュリティー情報は認証だけでなく、当該クライアントの認可検査のためにも使用されます

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です

- 関連関数

tpstart(), tmax\_fini\_auth(), tmax\_req\_auth()

### 3.7.7. tmax\_fini\_auth

tmax\_init\_auth関数により作成された当該クライアントのセキュリティ・トークン情報を削除する関数です。

- プロトタイプ

```
# include <atmi.h>
# include <tmaxapi.h>
int tmax_fini_auth(TMAX_SEC_T *own_atoken)
```

- パラメータ

パラメータ	説明
own_atoken	tmax_init_auth関数により作成されたセキュリティ・トークンです

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です

- 関連関数

tmax\_fini\_auth(), tmax\_req\_auth()

### 3.7.8. tmax\_chk\_authen

tpstart()内部でのクライアントのtmax\_init\_auth()の呼び出しに対するCASの認証検査関数です。また、当該クライアントに対する認証・認可・監査のためのセキュリティ・トークン情報を作成します。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_chk_authen(const TMAX_SEC_T *atoken_from_peer, TMAX_SEC_T *own_atoken)
```

- パラメータ

パラメータ	説明
atoken_from_peer	クライアントのtmax_init_auth関数のatoken_to_peerトークンをこのパラメータで受信します
own_atoken	当該クライアントのセキュリティ・トークン情報をこのパラメータに割り当てます

- 戻り値



戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です

- 関連関数

tmax\_init\_auth()

### 3.7.9. tmax\_req\_auth

クライアントのtpcall呼び出し内部で要求するサービスを認可するために呼び出される関数です。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_req_auth(const char *svc_name, TMAX_SEC_T *atoken_to_peer, TMAX_SEC_T
*own_atoken)
```

- パラメータ

パラメータ	説明
svc_name	tpcall()を実行時に呼び出したサービス名です
atoken_to_peer	クライアントがCASに認可の要求のために渡す情報があれば、ここに割り当てます
own_atoken	tmax_init_auth関数により作成されたトークンです。このトークン情報は再びクライアントで維持するので、必要な情報があれば再度割り当てます

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です

- 関連関数

tpcall(), tpacall(), tmax\_init\_auth(), tmax\_fini\_auth()

### 3.7.10. tmax\_chk\_author

CASで認可検査の際に呼び出される関数です。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_chk_author(int desc, const char *svc_name, const TMAX_SEC_T
*atoken_from_peer, TMAX_SEC_T *own_atoken, int *flag)
```

- パラメータ

パラメータ	説明
desc	クライアントからの認可要求である場合は1、それ以外は2に設定されます
svc_name	tpcallで呼び出すサービスの名前です
atoken_from_peer	認証を要求した相手のトークン情報をこのパラメータで受信します
own_atoken	tmax_chk_authen関数で作成された当該クライアントのセキュリティー・トークン情報をこのパラメータで受信します
flag	認可を要求したサービスが自ノードのサービスでない場合、このサービスのあるノードに認可を要求するかどうかを指定するフラグです。0に設定すると要求せず、それ以外は他のノードに要求します

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です

- 関連関数

tmax\_req\_auth()

## 3.7.11. tmax\_crypt\_plugin\_init

Tmax環境設定のCASOPTに設定された値を渡します。CASの起動時に暗号化ライブラリーと関連する初期化プロセスが必要なら、この関数で実装します。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_crypt_plugin_init(int argc, char *argv[])
```

- パラメータ

パラメータ	説明
argc	CASOPTに設定されたオプションの数です
argv	CASOPTに設定されたオプションの値です

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です

- 関連関数

tmax\_crypt\_plugin\_fini()

## 3.7.12. tmax\_crypt\_plugin\_fini

Tmax環境設定のCASOPTに設定された値を渡します。CASの終了時に暗号化ライブラリーと関連する後処理プロセスが必要なら、この関数で実装します。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_crypt_plugin_fini(int argc, char *argv[])
```

- パラメータ

パラメータ	説明
argc	CASOPTに設定されたオプションの数です
argv	CASOPTに設定されたオプションの値です

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です

- 関連関数

tmax\_crypt\_plugin\_init()

### 3.7.13. tmax\_auth\_plugin\_init

Tmax環境設定のCASOPTに設定された値を渡します。CASの終了時に認証・認可・監査ライブラリーと関連する初期化プロセスが必要なら、この関数で実装します。

- プロトタイプ

```
#include <tmaxapi.h>
int tmax_auth_plugin_init(int argc, char *argv[])
```

- パラメータ

パラメータ	説明
argc	CASOPTに設定されたオプションの数です
argv	CASOPTに設定されたオプションの値です

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です

- 関連関数

tmax\_auth\_plugin\_fini()

### 3.7.14. tmax\_tencrypt

Tmaxの暗号化API・復号化APIの注意事項:tencerrno変数はグローバル変数であり、スレッド・セーフではありません。

- プロトタイプ

```
#include <usrinc/tencrypt.h>
int tmax_tencrypt(char *src, int srclen, char **dest, int *destlen, int mode)
```

- パラメータ

パラメータ	説明
src	暗号化または復号化する場合、入力データのアドレス値を指定します
srclen	入力データのサイズを指定します。0を指定すると、内部でstrlen()を呼び出して使用します

パラメータ	説明
dest	暗号化または復号化が実行された後、出力データのバッファ・アドレスが渡されるアドレス値を指定します。内部でバッファをメモリーに割り当ててリターンします。関数の呼び出し後、当該アドレスに対してfree()を呼び出す必要があります
destlen	暗号化または復号化が実行された後の出力データのサイズ
mode	srcを暗号化する場合はTENC_ENCRYPTを、srcを復号化する場合はTENC_DECRYPTを指定します。TENC_ENCRYPTを指定する場合、srcの平文の最大長は255字です

- 戻り値

戻り値	説明
0	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。tencerrnoにエラーコードが設定されます。設定可能な値はtperrnoと同じです

- エラー

tpdeq()が正常に実行されなかった場合、tperrnoに以下の値のいずれかが設定されます。

エラーコード	説明
[TPEPROTO]	暗号化・復号化を実行時に、秘密鍵ファイルをロードできなかった場合です
[TPEINVAL]	暗号化・復号化を実行時に、入力データが正しくないかNULLである場合です
[TPEOS]	メモリーの割り当てに失敗した場合です
[TPELIMIT]	暗号化を実行時に、作成された暗号文が許容可能なサイズを超えた場合です

- 例

```
#include <stdio.h>
#include <string.h>
#include <usrinc/tencrypt.h>

int main(int argc, char **argv)
{
    int n, len;
    char *cipher, *plain;

    if (argc < 2)
        return 1;

    cipher = plain = NULL;
```

```

n = tmax_tencrypt(argv[1], strlen(argv[1]), &plain, &len, TENC_ENCRYPT);
if (n < 0)
    printf("ENC failed. error = %d\n", tencerrno);
else
    printf("ENC [%s] ==> [%s][%d]\n", argv[1], plain, len);

n = tmax_tencrypt(plain, len, &cipher, &len, TENC_DECRYPT);
if (n < 0)
    printf("DEC failed. error = %d\n", tencerrno);
else
    printf("DEC [%s] ==> [%s][%d]\n", plain, cipher, len);

free(cipher);
free(plain);
return 0;
}

```

- ビルド

```
cc test.c -o test -I$TMAXDIR -L $TMAXDIR/lib64 -ltencrypt
```

## 3.8. その他の関数

### 3.8.1. tlog\_close

トランザクション・ログを分析するための関数の1つで、該当ログ・ファイルを閉じます。Tmaxエンジンとは別に使用でき、**<usring/tlog.h>**と**<libtlog.so>**ライブラリーのみあれば使用が可能です。

- プロトタイプ

```

#include <usring/tlog.h>
int tlog_close (tlog_file_t *log);

```

- パラメータ

パラメータ	説明
log	tlog_open()の呼び出し時、該当ログ・ファイルに対する内部情報が設定され、tlog_find()にも使用されていた構造体に対するポインターです

- 戻り値

戻り値	説明
TLOG_OK	正常にログ・ファイルを閉じた場合です
負数	関数の呼び出し中にエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <usrinc/tlog.h>

int main(int argc, char *argv[])
{
    int n;
    tlog_entry_t entry;
    tlog_file_t log;

    printf("\n===== TXLOG =====\n");
    n = tlog_open(NULL, &log, TLOG_LOCAL);
    printf("n = %d\n", n);
    if (n < 0) {
        printf("tlog_open = %d, errno = %d\n", n, errno);
        exit(1);
    }
    ...
    tlog_close(&log);
}
```

- 関連関数

tlog\_close(), tlog\_find(), tlog\_nodeno()

## 3.8.2. tlog\_find

該当トランザクションのログ・ファイルでentryに指定された情報と一致するentryを検索する関数です。結果は、パラメータで渡されたentryに保存されて返されます。Tmaxエンジンとは別に使用でき、<usrinc/tlog.h>と<libtlog.so>ライブラリーのみあれば使用が可能です。

- プロトタイプ

```
#include <usrinc/tlog.h>
int tlog_find (tlog_file_t *log, tlog_entry_t *entry, int flags);
```

- パラメータ

パラメータ	説明
log	tlog_openで作成したtlog情報構造体です
entry	照会するentryを返します
flags	使用可能なフラグについては、下で説明します

tlog\_entry\_tは、該当トランザクションのログ・ファイルに入る内容を表示します。内容は以下のとおりです。

```
typedef struct {
    int decision;
    TXID xid;
    time_t ltime;
    /* following fields have meaning only for TLOG_REMOTE & TLOG_NONTMAX */
    TXID remote_xid;
    char gateway_name[GATEWAY_NAME_SIZE];
} tlog_entry_t;
```

メンバー	説明
decision	該当トランザクションに対してどういう動作が行われたかを記録しています。種類についての定義は、下の内容を参照してください
xid	ログを残したxidです。ゲートウェイで残したトランザクションのログにはremote_xidとgateway_nameの2つのフィールドが追加されます
ltime	ログを残した時間です

以下は、decisionで使用可能な値についての定義です。

```
/* invalid entry */
#define TXDEC_INVALID          -1
/* commit */
#define TXDEC_COMMIT           0
/* rollback */
#define TXDEC_ROLLBACK         1
/* rollback due to svr/cli down or failure during the prepare phase */
#define TXDEC_ABNORMAL_ROLLBACK 2
/* tx initiated from a remote domain */
#define TXDEC_REMOTE           3
/* prepare phase */
#define TXDEC_PREPARE           4
```

以下は、flagsで利用できる6つのマクロのbit-wise ORの定義です。

```
/* get next log entry */
#define TLOG_NEXT              0x0001
/* find an entry logged later than or equal to the ltime field */
```



```

#define TLOG_TIME                0x0002
/* find an entry with matching xid */
#define TLOG_XID                  0x0004
/* find an entry with matching remote_xid */
#define TLOG_REMOTE_XID          0x0008
/* find an entry from the matching gateway_name field */
#define TLOG_GATEWAY              0x0010
/* find an entry with same decision */
#define TLOG_DECISION             0x0020

```

定数	説明
TLOG_NEXT	bitがONの場合、ログ・ファイルの現在の位置から検索します。そうでない場合、ファイルの最初から検索します
TLOG_TIME	bitがONの場合、ログ記録時間がentry→ltime以降のentryのみ検索します
TLOG_XID	entry→xidが一致するentryのみを検索します。XIDの比較をする場合、branch qualifier部分は無視され、グローバル・トランザクションID部分のみ参照します
TLOG_REMOTE_XID	entry→remote_xidが同じentryを検索します
TLOG_GATEWAY	entry→gateway_nameが同じentryを検索します
TLOG_DECISION	entry→decisionが同じentryを検索します

- 戻り値

戻り値	説明
TLOG_OK	関数が正常に実行された場合です
TLOG_INVAL	引数が有効でない場合です
TLOG_NOTFOUND	ファイルの最後まで検索したが、見付からなかった場合です
TLOG_ESYSTEM	Tmaxシステムにエラーが発生した場合です。詳細情報はerrnoを参照してください

- 例

```

#include <stdio.h>
#include <usrinc/tlog.h>

int main(int argc, char *argv[])
{
    int n;
    tlog_entry_t entry;
    tlog_file_t log;

    printf("\nn===== TXLOG =====\n");
}

```

```

n = tlog_open(NULL, &log, TLOG_LOCAL);
printf("n = %d\n", n);
if (n < 0) {
    printf("tlog_open = %d, errno = %d\n", n, errno);
    exit(1);
}
printf("tlog_open success\n");
while(1) {
n = tlog_find(&log, &entry, TLOG_NEXT);
if (n == TLOG_NOTFOUND)
{
    printf("Not found any more\n");
    break;
}

else if (n < 0) {
    printf("tlog_find = %d, errno = %d\n", n,errno);
    tlog_close(&log);
    exit(1);
}
tlog_close(&log);
}

```

- 関連関数

tlog\_open(), tlog\_close(), tlog\_nodeno()

### 3.8.3. tlog\_nodeno

XIDを使用してトランザクションが開始したノード番号を検索する関数です。マルチノード構成の場合、トランザクション・ログはトランザクションが開始されたノードに残るため、XIDから該当ノードを検索する必要があります。tlog\_nodeno()はTmaxエンジンとは別に使用でき、<usring/tlog.h>と<libtlog.so>ライブラリーのみあれば使用が可能です。

- プロトタイプ

```

#include <usring/tlog.h>
int tlog_nodeno (TXID *xid);

```

- パラメータ

パラメータ	説明
xid	ノード番号を検索するXIDです

- 戻り値

戻り値	説明
ノード番号	関数の呼び出しに成功した場合です。トランザクションを開始したノードの番号を返します
負数	関数の呼び出しに失敗した場合です

- 例

```
#include <stdio.h>
#include <usrinc/tlog.h>

int main(int argc, char *argv[])
{
    int n, nodeno;
    tlog_entry_t entry;
    tlog_file_t log;

    printf("\n===== TXLOG =====\n");
    n = tlog_open(NULL, &log, TLOG_LOCAL);
    printf("n = %d\n", n);
    if (n < 0) {
        printf("tlog_open = %d, errno = %d\n", n, errno);
        exit(1);
    }
    printf("tlog_open success\n");
    while(1) {
        n = tlog_find(&log, &entry, TLOG_NEXT);
        ...
    }

    nodeno = tlog_nodeno(&(entry.xid));
    printf("nodeno of txlog = %d\n", nodeno);
    tlog_close(&log);
}
```

- 関連関数

log\_open(), tlog\_close(), tlog\_find()

### 3.8.4. tlog\_open

トランザクション・ログを分析する関数で、該当ログ・ファイルを開きます。Tmaxエンジンとは別に使用でき、**<usrinc/tlog.h>**と**<libtlog.so>**ライブラリーのみあれば使用が可能です。

## ● プロトタイプ

```
#include <usring/tlog.h>
int tlog_open (char *name, tlog_file_t *log, int flags);
```

## ● パラメータ

パラメータ	説明
name	ログ・ファイルの名前です
log	該当ログ・ファイルの内部情報です。tlog_close()やtlog_find()に使用されます
flags	以下は、flagsに設定可能な値についての説明です  – TLOG_LOCAL ローカル・ノードのTXのログを照会します  – TLOG_REMOTE ドメイン・ゲートウェイを介して発生したTXのログを照会します  – TXLOG 当該ノードで2PCの動作が発生した場合に記録を残します  – GWTXLOG 当該ノードで実行中のゲートウェイを介してトランザクションと関連する要求が入ってきた場合、remote_xidとlocal_xid mapping情報を記録します

以下は、各パラメータの設定に従った動作についての説明です。

- (TMAXDIR)/log/tlog/TXLOGファイルを基本に開きます。

```
name = NULL, flags = TLOG_LOCAL
```

- \$(TMAXDIR)/log/tlog/GWTXLOGファイルを開きます。

```
name = NULL, flags = TLOG_REMOTE
```

- name != NULL

```
name = 入力された名前, flags = TLOG_LOCAL
```

## ● 戻り値

戻り値	説明
TLOG_OK	正常にログ・ファイルを開いた場合です
負数	正常にログ・ファイルを開いた際にエラーが発生した場合です

- 例

```
#include <stdio.h>
#include <usrinc/tlog.h>

int main(int argc, char *argv[])
{
    int n;
    tlog_entry_t entry;
    tlog_file_t log;

    printf("\n===== TXLOG =====\n");
    n = tlog_open(NULL, &log, TLOG_LOCAL);
    printf("n = %d\n", n);

    if (n < 0) {
        printf("tlog_open = %d, errno = %d\n", n, errno);
        exit(1);
    }

    printf("tlog_open success\n");
    ...
}
```

- 関連関数

tlog\_close(), tlog\_find(), tlog\_nodeno()

### 3.8.5. Usiginit

ユーザー・シグナル・ハンドリングに必要なマクロ設定に使用される関数です。Windowsシステム環境では使用されません。Usiginit()はTmaxシグナル・ハンドラーを初期化します。

- プロトタイプ

```
# include <usignal.h>
int Usiginit(void)
```

- 戻り値

戻り値	説明
1	関数の呼び出しに成功した場合です
-1	関数の呼び出しに失敗した場合です。エラーが発生した場合もtperrnoには値が設定されません

- 例

```
...
#include <signal.h>
#include <usrinc/usignal.h>

void sig_alrm(int);
SERVICE(TPSVCINFO *msg)
{
    int i, ret;

    data process...

    ret=Usiginit();
    if (ret==-1) { error processing }

    Usignal(SIGALRM, &sig_alrm);
    alarm(1);

    tpreturn(TPSUCCESS,0,(char *)msg->data, 0,0);
}

void sig_alrm(int signo)
{
    if (signo == SIGALRM) printf("SIGALRM recieve\n");
}
```

### 3.8.6. Usignal

ユーザー・シグナル・ハンドリングに必要なマクロ設定に使用される関数です。Windowsシステム環境では使用されません。基本的なTmaxシグナル・ハンドラーは内部実行に必要なSIGALRM、SIGPIPE、SIGTERMをハンドリングします。

signal()を使用してハンドリングする場合は、シグナルによるTmax内部装置が作動しませんが、TmaxライブラリーはUsignal()を提供し、これを防止します。Usignal()はユーザー・シグナル・ハンドラーの記録が保存された表を管理し、シグナルが発生した場合にTmaxシグナル・ハンドラーはこの表を参照します。Tmaxでは、ユーザー・シグナル・ハンドラーがTmaxシグナル・ハンドラーに優先され、ユーザー・シグナル・ハンドリングを終えた後にTmaxシグナル・ハンドラーが作動します。

- プロトタイプ

```
# include <usignal.h>
Sigfunc *Usignal(int sig, Sigfunc *func)
```

- パラメータ

パラメータ	説明
sig	数字あるいは整数値です(例:9あるいはSIGKILL)
func	呼び出す関数値です

- 戻り値

戻り値	説明
ユーザー・シグナル・ハンドラー	以前のユーザー・シグナル・ハンドラーが存在する場合は
SIG_DFL	以前のユーザー・シグナル・ハンドラーが存在しない場合は。エラーが発生した場合も、tperrnoには値が設定されません

- 例

```
...
#include <signal.h>
#include <usrinc/usignal.h>

void sig_almr(int);
SERVICE(TPSVCINFO *msg)
{
    int          i, ret;
    data process...
    ret=Usiginit();
    if (ret==-1) { error processing }
    Usignal(SIGALRM, &sig_almr);
    alarm(1);
    tpreturn(TPSUCCESS,0,(char *)msg->data, 0,0);
}

void sig_almr(int signo)
{
    if (signo == SIGALRM) printf("SIGALRM recieve\n");
}
```

## 3.8.7. Uunixerr

システムを呼び出す途中にエラーが発生した場合、統合されたエラー番号が設定される変数です。

```
int Uunixerr
```

## 3.8.8. Uunix\_err

ATMI APIの呼出しに失敗し、tpernoがTPEOSに設定された場合、システム・エラーの種類をstderrに出力する関数です。

- プロトタイプ

```
# include <Uunix.h>
void Uunix_err (char *msg)
```

- パラメータ

パラメータ	説明
msg	エラーが発生したシステムを呼び出す前に、追加したいメッセージです。一般的にプログラム名を記録します。以下のうち1つが出力されます  – UCLOSE, UCREAT, UEXEC, UFCTNL, UFORK, ULSEEK, UMSGCTL, UMSGGET, UMSGSEND, UMSGRCV, UOPEN, UPLOCK, UREAD, USEMCTL, USEMGET, USEMOP, USHMCTL, USHMGET, USHMAT, USHMDT, USTAT, UWRITE, USBRK, USYSMUL, UWAIT, UKILL, UTIME, UMKDIR, ULINK, UUNLINK, UUNAME, UNLIST

- 例

```
ret=tmaxreadenv("NO THAT FILE", "TMAX");
if (ret<0) {
    Uunix_err("myprog");
    exit(1);
}
```

以下は、上記の例を実行した結果です。

```
myprog: UOPEN
```



## 3.8.9. Ustrerror

システムのエラー・コード(errno)に対する統合エラー・メッセージを返す関数です。

- プロトタイプ

```
# include <Unix.h>
char * Ustrerror(int err);
```

- パラメータ

パラメータ	説明
err	知りたいエラー・メッセージの統合エラー番号です

- 戻り値

関数の呼出しに成功した場合、errnoに対する統合エラー・メッセージのポインター・アドレスが返されます。

以下のリストのうち1つが、chr \*型のポインターとして返されます。

- UCLOSE, UCREAT, UEXEC, UFCTNL, UFORK, ULSEEK, UMSGCTL, UMSGGET, UMSGSEND, UMSGRCV, UOPEN, ULOCK, UREAD, USEMCTL, USEMGET, USEMOP, USHMCTL, USHMGET, USHMAT, USHMDT, USTAT, UWRITE, USBRK, USYSMUL, UWAIT, UKILL, UTIME, UMKDIR, ULINK, UUNLINK, UUNAME, UNLIST

- 例

```
ret=tmaxreadenv("NO THAT FILE", "TMAX");
if (ret<0)
{
    printf("%d->%s\n", Uunixerr, Ustrerror(Uunixerr));
    exit(1);
}
```

以下は、上記の例を実行した結果です。

```
11->UOPEN
```

## 3.8.10. tmaxoserrno

システムの呼び出し中にエラーが発生する場合、統合されたエラー番号が設定される変数です。エラーの発生後、複数のシステム・コールを呼び出すとき、エラー番号が最初のエラー発生後変更され、原因を把握し難くなることがあります。このとき、tmaxoserrno変数を確認すると、tperrnoがTPEOS、TPESYSTEMの発

生時点のシステム・エラー番号を確認できます。WindowsはGetLastError()の値を、UNIXはerrnoの値を保存しています。

# 付録 A. エラー別対応方法

## 2 : TPEBADDESC

説明	cdが有効でない記述子です。関数の使用時に返された記述子が有効ではありません
対応方法	tpcall()を使用して割り当てられたcdを確認します。あるいは、tpstart()を再度呼び出します

## 3 : TPEBLOCK

説明	要求したサービスがブロッキングされた状態です。TPNOBLOCK設定された状態でブロッキングが発生しました。サービスを再度要求するか、ブロッキングが解除された後に再度実行します。tpgetrplyやtpgetunsolがNonBlock Modeで呼び出されたときに発生した場合、呼び出し時にソケットにデータがない状態です。そのため、該当エラーを無視し、一定時間待機した後に再実行するLogicの追加が必要です
対応方法	サービスを再度要求します。あるいは、ブロッキングが解除された後に再度実行します

## 5 : TPELIMIT

説明	システムリソース、あるいはTmaxが提供するリソースが不足しています
対応方法	Tmaxではリソースが充分でない場合、OSが提供するリソースの割り当てを受け、これを利用します

## 6 : TPENOENT

説明	サービス表にサービスが存在しません。あるいは、Tmaxエンジンでサービスを認識できません
対応方法	環境ファイルが修正された場合、gstでサービス表を新規作成し、サーバーアプリケーションのコンパイル時に一緒にコンパイルします

## 7 : TPEOS

説明	OSエラーです
対応方法	OSを始め、ネットワークおよび既存で運用されていた環境が変更されたかどうか、諸般環境をチェックします

## 9 : TPEPROTO

説明	不適切な状況で関数が呼び出されました
対応方法	段階別関数の呼び出しの使用が正しくありません。バッファを設定し、サービスを要求します。あるいは、対話型通信をする場合、sendとrecvなどの段階を遵守します

## 10 : TPESVCERR

説明	サービス実行中にサーバープロセスでエラーが発生しました
対応方法	サーバープロセスを確認します

## 11 : TPESVCFAI

説明	サービス実行中にアプリケーションレベルでエラーが発生しました。サービス要求に対する応答を送信するサービス・ルーチンが、アプリケーション上でエラーが発生しました
対応方法	アプリケーションを確認します

## 12 : TPESYSTEM

説明	Tmaxシステムで異常が発見されました
対応方法	一般的なすべてのエラーを含むため、様々な部分をチェックします。ネットワークに対する障害およびサーバープロセスなど、関連諸般環境をチェックします

## 13 : TPETIME

説明	ブロッキングされています。あるいは、何らかの原因によって指定時間を超過しました
----	---

<b>対応方法</b>	アプリケーションレベルで時間超過の原因を把握し、正常な場合はタイムアウト時間を調整します。アプリケーションでタイムアウト時間を指定することができます。基本的には、環境ファイルのBLOCKTIME値を適用します
-------------	--

## 14 : TPETRAN

<b>説明</b>	トランザクションをサポートしていないサービスです
<b>対応方法</b>	tpcallの呼び出し時にフラグをTPNOTRANと設定します

## 15 : TPGOTSIG

<b>説明</b>	TPSIGRSTRTが設定されていない状態でシグナルが受信されました  COUSIN項目を使用してDDRを行う際、CLHがどのノードにあるサービスを要求すべきか分からない場合にも発生します
-----------	--

## 17 : TPEITYPE

<b>説明</b>	入力されたバッファのタイプが認識できません。COUSIN項目を使用してDDRを行う際、CLHがどのノードにあるサービスを要求すべきか分からない場合にも発生します
<b>対応方法</b>	データ値をチェックします。フィールドキー・バッファを使用する場合は、該当フィールドキーの定義可否を確認します。構造体バッファを使用する場合は、構造体の宣言可否を確認します

## 18 : TPEOTYPE

<b>説明</b>	入力されたバッファのタイプを呼び出し元が認識できません。データのタイプおよびサブタイプと入力されたバッファのタイプが一致しません
<b>対応方法</b>	データ値をチェックします

## 22 : TPEEVENT

<b>説明</b>	対話型モードで発生します。イベントが発生し、データは渡されていません。イベントはrevent値に返されます
<b>対応方法</b>	イベント(revent)値を確認します

## 23 : TPEMATCH

説明	該当条件を満たす除去データを見付けることができません
対応方法	条件が正しいかどうか検討します

## 24 : TPENOREADY

説明	サービスが準備されていません。あるいは、起動はしているがアクティブ状態になっていません。  たとえば、tpcall()の呼び出し時にすでにサービスがNRDY状態の場合に発生します
対応方法	tmadminでst-sを通じてサービス状態を確認し、NRDYに表示された場合は正常に起動していないということです。サーバープロセスを再度確認し、再度起動し、tpstart()を再度呼び出します

## 25 : TPESECURITY

説明	セキュリティ設定の使用時に許可されていないユーザーが接続しました
対応方法	ユーザーの権限を確認します

## 26 : TPEQFULL

説明	要求されたサービスが指定したMaxキューに到達しました
対応方法	Tmax環境ファイルあるいは動的にtmadminツールを使用してMaxキュー値を調節することができます

## 27 : TPEQPURGE

説明	管理者が強制的にキューをページしました。要求されたサービスがキューに待機中、管理者によってページされた場合に発生します
----	---

## 28 : TPECLOSE

説明	Tmaxが起動していません。あるいは、接続することができません
----	---------------------------------

対応方法	Tmaxの正常起動可否を確認後、tpstart()を再度呼び出します
------	------------------------------------

## 29 : TPESVRDOWN

説明	tpcall()を呼び出したサービスのため、サーバーがダウンした状況です
対応方法	サーバープロセスが正常に作動中かどうかを確認します。あるいは、アプリケーション・ロジックの問題として、プログラムが実行中に異常終了することもあるため、アプリケーションを確認します

## 30 : TPEPRESVC

説明	RQセクションに登録して使用したPRESVCに対するエラーです。現在は使用していないため、発生しないエラーです
----	---

## 31 : TPERDOWN

説明	Rolling Downが発生した場合、既存のサーバーキューに蓄積している要求に対するtpgetrplyの呼び出し時にTPERDOWNエラーが設定されます。tpacallの使用時、送信されていたデータがtpgetrplyの受信バッファに再度含まれます。Rolling Downの発生後に呼び出されるtpacallに対しても、TPERDOWNエラーが設定されます
対応方法	リリースノートを参照してください

## 31 : TPERDCLOSE

説明	サーバーキューに滞積した要求がない状況でtpgetrplyが呼び出された場合、TPERDCLOSEエラーがセッティングされます
対応方法	リリースノートを参照してください

## 31 : TPEMAXNO

説明	同時に接続可能なユーザーが限定されていますが、ユーザーが最大数に到達しました
対応方法	tadminでci(client Information)を通じて確認します





# 付録 B. ヘッダー・ファイルの例

## B.1. <atmi.h>

Tmaxシステムを使用してプログラムを開発するには、本ファイルを添付する必要があります。

```
/* ----- usrinc/atmi.h ----- */
/*      */
/*      Copyright (c) 2000 - 2008 Tmax Soft Co., Ltd */
/*      All Rights Reserved      */
/*      */
/* ----- */

#ifndef _TMAX_ATMI_H
#define _TMAX_ATMI_H

#ifdef __EXPORT
#undef __EXPORT
#endif
#ifdef _WIN32
#ifdef _TMAX4GL
#define __EXPORT __stdcall
#else
#define __EXPORT __cdecl
#endif
#else
#define __EXPORT
#endif
#ifndef THLVAR
#define THLVAR
#endif

/* Flags to tpinit() for Tuxedo compatability */
#define TPU_MASK 0x00000007
#define TPU_SIG  0x00000001
#define TPU_DIP  0x00000002
#define TPU_IGN  0x00000004
#define TPSA_FASTPATH 0x00000008
#define TPSA_PROTECTED 0x00000010
```

```

#if defined(_TMAX_MTLIB) || defined(_MCONTEXT)
#define TPSINGLECONTEXT 0
#define TPINVALIDCONTEXT -1
#define TPNULLEXCONTEXT -2
#endif

/* ----- flags from API ----- */
/* Most Significant Two Bytes are reserved for internal use */
#ifndef TPNOFLAGS
#define TPNOFLAGS 0x00000000
#endif
#define TPNOBLOCK 0x00000001
#define TPSIGRSTRT 0x00000002
#define TPNOREPLY 0x00000004
#define TPNOTRAN 0x00000008
#define TPTRAN 0x00000010
#define TPNOTIME 0x00000020
#define TPNOGETANY 0x00000040
#define TPGETANY 0x00000080
#define TPNOCHANGE 0x00000100
#define TPBLOCK 0x00000200
#define TPCONV 0x00000400
#define TPFLOWCONTROL (TPCONV)
#define TPSENDONLY 0x00000800
#define TPRECONLY 0x00001000
#define TPUDP 0x00002000
#define TPRQS 0x00004000
#define TPFUNC 0x00008000 /* API dependent functional flag */
#define TPABSOLUTE (TPFUNC)
#define TPACK (TPFUNC)
#define TPURGENT (TPCONV)

/* --- flags used in tpstart() --- */
#define TPUNSOL_MASK 0x00000007
#define TPUNSOL_HND 0x00000001
#define TPUNSOL_IGN 0x00000002
#define TPUNSOL_POLL 0x00000004
#define TPUNIQUE 0x00000010
#define TPONLYONE 0x00000020
#if defined(_TMAX_MTLIB) || defined(_MCONTEXT)
#define TPMULTICONTEXTS 0x00000040
#endif

/* --- flags used in tpreturn() --- */
#define TPFAIL 0x0001
#define TPSUCCESS 0x0002

```

```

#define TPEXIT          0x0004
#define TPDOWN          0x0008
#define TP_FORWARD     0x0010    /* Internal use only */

/* --- return code used in tpreturn() for Messaging System --- */
#define TPFACK_ACK      -1

/* ----- flags for reply type check ----- */
#define TPREQ           0
#define TPERR           -1

/* ----- for Tuxedo Compatability ----- */
/* Flags to tpscmr() - Valid TP_COMMIT_CONTROL characteristic values */
#define TP_CMT_LOGGED   0x01    /* return after commit decision is logged */
#define TP_CMT_COMPLETE 0x02    /* return after commit has completed */

/* Return values to tpchkauth() */
#define TPNOAUTH        0       /* no authentication */
#define TPSYSAUTH       1       /* system authentication */
#define TPAPPAUTH       2       /* system and application authentication */

#define XATMI_SERVICE_NAME_LENGTH 32    /* where x must be > 15 */

struct clid_t {
    long clientdata[4];
};
typedef struct clid_t CLIENTID;

struct tpsvcinfo {
    char name[XATMI_SERVICE_NAME_LENGTH];
    char *data;
    long len;
    long flags;
    int cd;
    CLIENTID cltid;
};
typedef struct tpsvcinfo TPSVCINFO;

#if defined(_WIN32) || defined(_TMAX_MTLIB) || defined(_MCONTEXT)
#if defined(__cplusplus)
extern "C" {
#endif
#endif

/*
    Internal functions: ONLY BE CALLED FROM AUTOMATICALLY
    GENERATED STUB FILES. DO NOT DIRECTLY CALL THESE FUNCTIONS.
*/

```

```

int *__EXPORT _tmget_tperrno_addr(void);
long *__EXPORT _tmget_tpurcode_addr(void);

#ifdef _TMAX_KERNEL
extern THLVAR int tperrno;
extern THLVAR long tpurcode;
#else
#define tperrno  (_tmget_tperrno_addr())
#define tpurcode (_tmget_tpurcode_addr())
#endif
#if defined(__cplusplus)
}
#endif
#else
extern int tperrno;
extern long tpurcode;
#endif

#define TPEMINNO          1
#define TPEBADDESC        2
#define TPEBLOCK          3
#define TPEINVAL          4
#define TPELIMIT          5
#define TPENOENT          6
#define TPEOS             7
#define TPEPROTO          9
#define TPESVCERR         10
#define TPESVCFAIL        11
#define TPESYSTEM         12
#define TPETIME           13
#define TPETRAN           14
#define TPGOTSIG          15
#define TPEITYPE          17
#define TPEOTYPE          18
#define TPEEVENT          22
#define TPEMATCH          23
#define TPENOREADY        24
#define TPESECURITY       25
#define TPEQFULL          26
#define TPEQPURGE         27
#define TPECLOSE          28
#define TPESVRDOWN        29
#define TPEPRESVC         30
#define TPERDOWN          31
#define TPERDCLOSE        32
#define TPEMAXNO          33

```

```

/* ---- flags used in conv[]: don't use dummy flags ----*/
#define TPEV_DISCONIMM 0x00000001
#define TPEV_SVCERR 0x00000002
#define TPEV_SVCFAIL 0x00000004
#define TPEV_SVCSUCC 0x00000008
#define TPEV_CONVCLOSE 0x00000010 /* don't use this flag */
#define TPEV_SENDOONLY 0x00000020
#define TPCONV_DUMMY1 0x00000800 /* don't use this flag: TPSENDONLY */
#define TPCONV_DUMMY2 0x00001000 /* don't use this flag: TPRECVOONLY */
#define TPCONV_OUT 0x00010000
#define TPCONV_IN 0x00020000

#define X_OCTET "X_OCTET"
#define X_C_TYPE "X_C_TYPE"
#define X_COMMON "X_COMMON"

#define TMTYPEFAIL -1
#define TMTYPESUCC 0

#ifdef MAXTIDENT
#define MAXTIDENT 16 /* max len of identifier */
#endif

#ifdef MAX_PASSWD_LENGTH
#define MAX_PASSWD_LENGTH 16
#endif

#ifdef MAX_MNAME_LENGTH
#define MAX_MNAME_LENGTH 16
#endif

struct tpstart_t {
    char username[MAXTIDENT+2]; /* usr name */
    char cltname[MAXTIDENT+2]; /* application client name */
    char dompwd[MAX_PASSWD_LENGTH+2]; /* domain password */
    char usrpwd[MAX_PASSWD_LENGTH+2]; /* passwd for usrid */
    int flags;
};
typedef struct tpstart_t TPSTART_T;

/* X/Open Typed Buffer related Function */

#ifdef __cplusplus
extern "C" {
#endif

```

```

/* ----- client API ----- */
int __EXPORT tpstart(TPSTART_T *);
int __EXPORT tpend(void);
char *__EXPORT tpalloc(char *type, char *subtype, long size);
char *__EXPORT tprealloc(char *ptr, long size);
long __EXPORT tptypes(char *ptr, char *type, char *subtype);
void __EXPORT tpfree(char *ptr);
int __EXPORT tpcall(char *svc, char *idata, long ilen, char **odata,
                    long *olen, long flags);
int __EXPORT tpacall(char *svc, char *data, long len, long flags);
int __EXPORT tpgetrply(int *cd, char **data, long *len, long flags);
int __EXPORT tpcancel(int cd);
char *__EXPORT tpstrerror(int err_no);

/* ----- conversational API ----- */
int __EXPORT tpconnect(char *svc, char *data, long lenl, long flagsl);
int __EXPORT tpdison(int cd);
int __EXPORT tpsend(int cd, char *data, long lenl, long flagsl, long *revent);
int __EXPORT tprecv(int cd, char **data, long *len, long flagsl, long *revent);

/* ----- server API ----- */
void __EXPORT tpreturn(int rval, long rcode, char *data, long len, long flags);
void __EXPORT tpforward(char *svc, char *data, long len, long flags);
int __EXPORT tpadvertise(char *svcname, void (* func) (TPSVCINFO *));
int __EXPORT tpunadvertise(char *svcname);

/* ----- etc API ----- */
int __EXPORT tpnotify(CLIENTID *id, char *data, long len, long flags);
int __EXPORT gettperrno(void);
long __EXPORT gettpurcode(void);

/* ----- multithread/multicontext API ----- */
int __EXPORT tpsetctxt(int ctxtid, long flags);
int __EXPORT tpgetctxt(int *ctxtid, long flags);

#ifdef __cplusplus
}
#endif

#endif          /* end of _TMAX_ATMI_H */

```

## B.2. <tmaxapi.h>

このヘッダー・ファイルはTmaxシステムによって提供される非標準関数を含んでいます。

Tmaxシステムを使用してプログラムを開発するには、本ファイルを添付する必要があります。

```
/* ----- usrinc/tmaxapi.h ----- */
/*      */
/*      Copyright (c) 2000 - 2008 Tmax Soft Co., Ltd */
/*      All Rights Reserved      */
/*      */
/* ----- */

#ifndef _TMAXAPI_H
#define _TMAXAPI_H

#ifndef _CE_MODULE
#include <sys/types.h>
#endif
#include <usrinc/atmi.h>
#ifdef _WIN32
#ifdef _CE_MODULE
#include <winsock.h>
#else
#include <winsock2.h>
#endif /* _CE_MODULE */
#include <usrinc/svct.h>
#include <usrinc/sdl.h>
#else
#ifndef ORA_PROC
#include <sys/socket.h>
#endif
#endif

/* client logout type */
#define CLIENT_CLOSE_NORMAL      0
#define CLIENT_CLOSE_ABNORMAL    1
#define CLIENT_PRUNED            2

/* RQ Sub-queue type */
#define TMAX_ANY_QUEUE           0
#define TMAX_FAIL_QUEUE          1
#define TMAX_REQ_QUEUE           2
#define TMAX_RPLY_QUEUE          3
#define TMAX_MAX_QUEUE           4
```

```

extern char _rq_sub_queue_name[TMAX_MAX_QUEUE][XATMI_SERVICE_NAME_LENGTH];

/* RQ related macros */
#define RQ_NAME_LENGTH      16

/* unsolicited msg type */
#define UNSOL_TPPOST        1
#define UNSOL_TPBROADCAST   2
#define UNSOL_TPNOTIFY      3
#define UNSOL_TPSENDTOCLI   4
#define UNSOL_ANY           5

/* RM type */
#define ORACLE_TYPE         1
#define SYBASE_TYPE         2
#define INFORMIX_TYPE       3
#define DB2_TYPE            4

#define NONXA_MODE          0
#define XA_MODE              1

/* Check SVCINFO cmds */
#define ISSVC_FORWARDED     0x00000001
#define ISSVC_NOREPLY       0x00000002

/* TPEVCTL ctl_flags */
#define TPEV_SVC            0x00000001
#define TPEV_PROC           0x00000002

/* tpgethostaddr flags */
#define GET_SVR_CON         0x00000000
#define GET_CUR_IP          0x00000001

/* tmax_sq_get/tmax_sq_put flags */
#define TPSQ_PEEK           0x00001000
#define TPSQ_UPDATE         0x00002000
#define TPSQ_SYSKEY         0x00004000
#define TPSQ_KEYGEN         0x00008000

#define SQ_KEYLIST_T        void*
#define SQ_KEYINFO_T        sq_keyinfo_t
#define SQ_SYSKEY_SIZE      16

struct sq_keyinfo_s {
    long keylen;
    long datalen;
    time_t starttime;

```



```

    char *key;
};

typedef struct sq_keyinfo_s sq_keyinfo_t;

struct tpevctl {
    long ctl_flags;
    long post_flags;
    char svc[XATMI_SERVICE_NAME_LENGTH];
    char qname[RQ_NAME_LENGTH];
};

typedef struct tpevctl TPEVCTL;
typedef void __EXPORT Unsolfunc(char *, long, long);
#define TPUNSOLERR      ((Unsolfunc *) -1)

struct tptranid {
    int  info[3];
    int  flags;
};

typedef struct tptranid TPTRANID;

/* Multicast call related structures */
struct svglist {
    int ns_entry; /* number of entries of s_list */
    int nf_entry; /* number of entries of f_list */
    int *s_list; /* list of server group numbers */
    int *f_list; /* list of server group numbers */
};

/* Jun/23/2008 KANAKO TPMCALL_UPDATE <<start>> */
struct svglistx {
    int ns_entry; /* number of entries of s_list */
    int nf_entry; /* number of entries of f_list */
    int nr_entry; /* number of entries of r_list */
    int *s_list; /* list of server group numbers */
    int *f_list; /* list of server group numbers */
    int *r_list; /* list of tpurcode of each server group */
};

/* Jun/23/2008 KANAKO TPMCALL_UPDATE <<end>> */

/* My svrinfo */
#ifndef TMAX_NAME_SIZE
#define TMAX_NAME_SIZE      16
#endif

```

```

#ifndef MAX_DBSESSIONID_SIZE
#define MAX_DBSESSIONID_SIZE    128
#endif

typedef struct {
    int nodeno; /* node index */
    int svgi;   /* server group index; unique in the node */
    int svri;   /* server index; unique in the node */
    int spri;   /* server process index; unique in the node */
    int spr_seqno; /* server process seqno ; unique in the server */
    int min, max; /* min/max server process number */
    int clhi;   /* for RDP only, corresponding CLH id */
    char nodename[TMAX_NAME_SIZE];
    char svgname[TMAX_NAME_SIZE];
    char svrname[TMAX_NAME_SIZE];
    char reserved_char[TMAX_NAME_SIZE];
    /* for more detail use tmaxadmin API */
} TMAXSVRINFO;

#define MSGIDLEN    32
#define CORRIDLEN   32

typedef struct { /* control parameters to queue primitives */
    int flags; /* indicates which of the values are set */
    int deq_time; /* absolute/relative time for dequeuing */
    int priority; /* enqueue priority */
    int diagnostic; /* indicates reason for failure */
    char msgid[MSGIDLEN]; /* id of message before which to queue */
    char corrid[CORRIDLEN]; /* correlation id used to identify message */
    char replyqueue[TMAX_NAME_SIZE]; /* queue name for reply message */
    char failurequeue[TMAX_NAME_SIZE]; /* queue name for failure message */
    CLIENTID cltid; /* client identifier for originating client */
    int urcode; /* application user-return code */
    int appkey; /* application authentication client key */
    int delivery_qos;
    int reply_qos;
    int exp_time;
} TMQCTL;

#ifdef _WIN32
typedef int (__EXPORT *WinTmaxCallback)(WPARAM, LPARAM);
#endif

/* Macro functions */
#define tpalivechk() tmax_chk_conn(0)

/* mode for IP-based ACL */

```

```

#define TMAX_ACL_ALLOW 0
#define TMAX_ACL_DENY 1

/* reserved mask value for IP-based ACL */
#define TMAX_ACL_IPADDR 32

#ifdef __cplusplus
extern "C" {
#endif

/* ----- unsolicited messaging API ----- */
long __EXPORT tpsubscribe(char *eventexpr, char *filter, TPEVCTL *ctl, long flags);
long __EXPORT tpsubscribe2(char *eventexpr, char *svcname, long flags);
int __EXPORT tpunsubscribe(long sd, long flags);
int __EXPORT tppost(char *eventname, char *data, long len, long flags);
int __EXPORT tpbroadcast(char *lnid, char *usrname, char *cltname, char *data,
    long len, long flags);
Unsolfunc *__EXPORT tpsetunsol(Unsolfunc *func);
int __EXPORT tpsetunsol_flag(int flag);
int __EXPORT tpgetunsol(int type, char **data, long *len, long flags);
int __EXPORT tpclearunsol(void);
int __EXPORT tpchkunsol(void);

/* ----- RQS API ----- */
int __EXPORT tpenq(char *qname, char *svc, char *data, long len, long flags);
int __EXPORT tpdeq(char *qname, char *svc, char **data, long *len, long flags);
int __EXPORT tpenq_ctl(char *qname, char *svc, TMQCTL *ctl, char *data,
    long len, long flags);
int __EXPORT tpdeq_ctl(char *qname, char *svc, TMQCTL *ctl, char **data,
    long *len, long flags);
int __EXPORT tpqstat(char *qname, long type);
int __EXPORT tpqsvcstat(char *qname, char *svc, long type);
int __EXPORT tpextsvcname(char *data, char *svc);
int __EXPORT tpextsvcinfo(char *data, char *svc, int *type, int *errcode);
int __EXPORT tpreissue(char *qname, char *filter, long flags);
char *__EXPORT tpsubqname(int type);

/* ----- server API ----- */
int __EXPORT tpgetminsvr(void);
int __EXPORT tpgetmaxsvr(void);
int __EXPORT tpgetmaxuser(void);
int __EXPORT tpgetsvrseqno(void);
int __EXPORT tpgetmysvrid(void);
int __EXPORT tpgetmysvrno(void);
int __EXPORT tpgetmaxuser(void);
int __EXPORT tpsendtocli(int clid, char *data, long len, long flags);
int __EXPORT tpgetclid(void);

```

```

int __EXPORT tpgetpeer_ipaddr(struct sockaddr *name, int *namelen);
int __EXPORT tpchkclid(int clid);
int __EXPORT tmax_clh_maxuser(void);
int __EXPORT tmax_my_svrinfo(TMAXSVRINFO*);
int __EXPORT tmax_cind2clid(int cind);
char *__EXPORT tpgetmynode(int *nodeno);
char *__EXPORT tpgetmysvg(void);
int __EXPORT tpgetmysvgno(void);
int __EXPORT tmax_is_restarted(void);
char *__EXPORT tpgetsvcname(int svci);
int __EXPORT tpsuspendtx(TPTRANID *tranid, long flags);
int __EXPORT tpresumetx(TPTRANID *tranid, long flags);

/* ----- etc API ----- */
int __EXPORT tp_sleep(int sec);
int __EXPORT tptsleep(struct timeval *timeout);
int __EXPORT tp_usleep(int usec);
int __EXPORT tpset_timeout(int sec);
int __EXPORT tpget_timeout(void);
int __EXPORT tmaxreadenv(char *file, char *label);
char *__EXPORT tpgetenv(char* str);
int __EXPORT tpputenv(char* str);
int __EXPORT tpgetsockname(struct sockaddr *name, int *namelen);
int __EXPORT tpgetpeername(struct sockaddr *name, int *namelen);
int __EXPORT tpgetactivesvr(char *nodename, char **outbufp);
int __EXPORT tperrordetail(int i);
int __EXPORT tpreset(void);
int __EXPORT tptobackup(void);
struct svglist *__EXPORT
    tpmcall(char *qname, char *svc, char *data, long len, long flags);
struct svglistx *__EXPORT
    tpmcallx(char *svc, char *data, long len, long flags);
struct svglist *__EXPORT tpgetsvglist(char *svc, long flags);
int __EXPORT tpsvgcall(int svgno, char *qname,
    char *svc, char *data, long len, long flags);
int __EXPORT tpflush(void);
char *__EXPORT tmaxlastsvc(char *idata, char *odata, long flags);
int __EXPORT tpgetorgnode(int clid);
int __EXPORT tpgetorgclh(int clid);
char *__EXPORT tpgetnodename(int nodeno);
int __EXPORT tpgetnodeno(char *nodename);
int __EXPORT tpgetasize(char *data);
int __EXPORT tpgettype(char *data);
char *__EXPORT tpgetsubtype(char *data);
int __EXPORT tpgetcliaddr(int clid, int *ip, int *port, long flags);
int __EXPORT tmax_chk_conn(int timeout);
int __EXPORT tpgethostaddr(int *ip, int *port, long flags);

```

```

char * __EXPORT tpgetdbsessionid(int flags);
int __EXPORT tpcallsvg(int svgno, char *svc, char *idata, long ilen,
    char **odata, long *olen, long flags);
int __EXPORT tpacallsvg(int svgno, char *svc, char *data, long len, long flags);

#if defined(_WIN32)
int __EXPORT WinTmaxAcall(TPSTART_T *sinfo, HANDLE wHandle, unsigned int msgType,

    char *svc, char *sndbuf, int len, int flags);
int __EXPORT WinTmaxAcall2(TPSTART_T *sinfo, WinTmaxCallback fn,
    char *svc, char *sndbuf, int len, int flags);
#endif

#if !defined(_TMAX_KERNEL) && !defined(_TMAX_RCA_H)
/* ----- User supplied routines ----- */
int __EXPORT tpsvrinit(int argc, char *argv[]);
int __EXPORT tpsvrdone(void);
void __EXPORT tpsvctimeout(TPSVCINFO *msg);
int __EXPORT _tmax_event_handler(char *progname, int pid, int tid, char *msg,
    int flags);
int __EXPORT tpsetdbsessionid(char dbsessionid[MAX_DBSSESSIONID_SIZE], int flags);
int __EXPORT tpprechk(void);
#endif

/*
    Internal functions: ONLY BE CALLED FROM AUTOMATICALLY
    GENERATED STUB FILES. DO NOT DIRECTLY CALL THESE FUNCTIONS.
*/
int __EXPORT get_clhfd(void);
int __EXPORT tmax_chk_svcinfo(int cmd);
int __EXPORT _tmax_main(int argc, char *argv[]);
int __EXPORT _tmax_cob_main(int argc, char *argv[]);
#if defined(_WIN32)
int __EXPORT _tmax_regfn(void *initFn, void *doneFn, void *timeoutFn,
    void *userMainFn);
int __EXPORT _tmax_regtab(int svcTabSz, _svc_t *svcTab, int funcTabSz,
    void *funcTab);
int __EXPORT _tmax_regsdl(int _sdl_table_size2, struct _sdl_struct_s *_sdl_table2,

    int _sdl_field_table_size2, struct _sdl_field_s *_sdl_field_table2);
int __EXPORT _tmax_regevthnd(void *evthndFn);
#endif
int __EXPORT _double_encode(char *in, char *out);
int __EXPORT _double_decode(char *in, char *out);
/* --- power builder interface API --- */
int __EXPORT _make_struct_from_pbindata(char *subtype, char *tpidata, int ilen,
    char *indata);

```

```

int __EXPORT _make_field_from_pbindata(char **tpidata, char *indata);
int __EXPORT _make_pbodata_from_struct(char *subtype, char *odata, int olen,
                                       char *tpodata);
int __EXPORT _make_pbodata_from_field(char *form, char *odata, char *tpodata);
int __EXPORT _make_pbfodata_from_field(char *fform, char *fodata, char *tpodata);
int __EXPORT _get_value_from_pbsdata(char *cur, char *vald);
int __EXPORT _get_name_value_from_pbfdata(char *cur, int *n2);
int __EXPORT _get_name_from_form(char *cur);
int __EXPORT _insert_value_to_pbodata(int type, char *out, char *in, int asize,
                                       int asize2);

int __EXPORT tmax_get_db_usrname(char *svgname, char *username, int type);
int __EXPORT tmax_get_db_passwd(char *svgname, char *passwd, int type);
int __EXPORT tmax_get_db_tnsname(char *svgname, char *tnsname, int type);

int __EXPORT tmax_is_xa();

/* 4.0 Sp2 Fix1 added */
int __EXPORT tmax_get_svccnt();
int __EXPORT tmax_get_svclist(char *buf, int bufsize);

/* ----- SessionQ API ----- */
int __EXPORT tmax_sq_put(char *key, long keylen1, char *data, long len1,
                        long flags1);
int __EXPORT tmax_sq_get(char *key, long keylen1, char **data, long *len1,
                        long flags1);
int __EXPORT tmax_sq_count(void);
int __EXPORT tmax_sq_purge(char *key, long keylen1);
int __EXPORT tmax_sq_keygen(char *key, long keylen1);
SQ_KEYLIST_T __EXPORT tmax_sq_getkeylist(char *key, long keylen1);

int __EXPORT tmax_gq_put(char *key, long keylen1, char *data, long len1,
                        long flags1);
int __EXPORT tmax_gq_get(char *key, long keylen1, char **data, long *len1,
                        long flags1);
int __EXPORT tmax_gq_count();
int __EXPORT tmax_gq_purge(char *key, long keylen1);
int __EXPORT tmax_gq_keygen(char *key, long keylen1);
SQ_KEYLIST_T __EXPORT tmax_gq_getkeylist(char *key, long keylen1);

int __EXPORT tmax_get_sessionid(void);
int __EXPORT tmax_keylist_count(SQ_KEYLIST_T keylist);
int __EXPORT tmax_keylist_getakey(SQ_KEYLIST_T keylist, int nth, SQ_KEYINFO_T
*keyinfo);
int __EXPORT tmax_keylist_free(SQ_KEYLIST_T keylist);

int __EXPORT tpgetsprlist(char *svc, int svgno, int *starti, int *endi, long flags1);
int __EXPORT tpspracall(char *svcname, int spri, char *data, long len1, long flags1);

```

```

int __EXPORT tpspracall2(char *svcname, int startspri, int nth, char *data, long
lenl,
                        long flagsl);

/* API for IP-based ACL */
int __EXPORT tmax_add_acl(int nodeno, char *ip, int mask, int mode, int flags);

#if defined (__cplusplus)
}
#endif

#endif          /* end of _TMAXAPI_H */

```





# 索引

## シンボル

#client, 41  
#rcah, 41  
#thread per rcah, 41

## A

allow\_connection(), 409  
allow\_connection\_ipv6(), 410

## C

C Client/Server API  
  tgsterror, 88  
  tmax\_grid\_count, 105  
  tmax\_grid\_create, 96  
  tmax\_grid\_create2, 98  
  tmax\_grid\_dequeue, 114  
  tmax\_grid\_destroy, 100  
  tmax\_grid\_enqueue, 112  
  tmax\_grid\_get, 103  
  tmax\_grid\_get\_child\_with\_index, 116  
  tmax\_grid\_get\_children, 115  
  tmax\_grid\_is\_exist, 104  
  tmax\_grid\_keylist\_free, 117  
  tmax\_grid\_lock, 106  
  tmax\_grid\_set, 101  
  tmax\_grid\_set\_watcher, 108  
  tmax\_grid\_unlock, 107  
  tmax\_grid\_wait\_watcher, 111  
cfl, 15  
chk\_end\_msg(), 416  
chk\_extpong\_msg, 423  
chk\_pong\_msg(), 420

## F

fdlc, 18

## G

get\_channel\_num(), 413  
get\_extmsg\_info(), 418  
get\_msg\_info(), 412  
get\_msg\_length(), 412  
get\_msg\_security(), 427  
get\_service\_name(), 415  
gettperrno, 87  
gettpurcode, 384  
gst, 21

## H

hkcf, 23

## I

idlcオプション  
  -clang, 24  
  -cstub, 24  
  -dbtype, 24  
  -header, 24  
  -inf, 24  
  -sql, 24  
  -sstub, 24  
  -xa, 24  
init\_remote\_info(), 406  
inmsg\_recovery(), 416

## M

mkacl, 26  
mkcli, 25  
mkgrp, 27  
mkpw, 28  
mksvr, 30

## O

outmsg\_recovery(), 417

## P

pid, 41  
prepare\_shutdown(), 415  
put\_extmsg\_info(), 419

put\_msg\_complete(), 414  
put\_msg\_info(), 414  
put\_msg\_security(), 428

## R

racd, 33  
racdr, 36  
rca\_dir, 41  
rca\_mode, 41  
rca\_port, 41  
rcah\_name, 41  
rcah\_no, 41  
rcakill, 39  
rcal\_name, 41  
rcal\_pid, 41  
rcastat, 40  
rcastatオプション  
    [-h], 39, 40  
    [-k], 39, 40  
    [-n], 39, 40  
    [-p], 39, 40  
remote\_closed(), 408  
remote\_connected(), 406  
remote\_connected\_ipv6(), 407  
reset\_extping\_msg, 424  
reset\_ping\_msg(), 424  
Rolling Down [tmdown -R], 76

## S

SDL, 41  
sdlc, 41  
set\_error\_msg(), 426  
set\_extping\_msg(), 421  
set\_ping\_msg(), 419  
set\_service\_timeout(), 415  
shmkey, 41  
shmsize, 41  
svcrpt, 44

## T

tdlcall, 429  
tdlcall2, 430

tdlcall2s, 432  
tdlcall2v, 433  
tdlcallva, 434  
tdlcallva2, 437  
tdlclean, 45  
tdlclose, 445  
tdlcreate, 438  
tdldestroy, 440  
tdldone, 447  
tdlend, 442  
tdlerror, 442  
tdlfind, 448  
tdlfind2, 449  
tdlgetseqno, 445  
tdlinit, 47, 447  
tdlload, 446  
tdlload2, 446  
tdlnm, 48  
tdlresume, 443  
tdlrm, 49  
tdlseqno, 49  
tdlshm, 50  
tdlstart, 444  
tdlstat, 449  
tdlstat2, 450  
tdlsuspend, 444  
tdlsync, 52  
tdltrace, 53  
tdlupdate, 54  
tencrypt, 55  
tgsterror, 88  
tlog\_close, 478  
tlog\_find, 479  
tlog\_nodeno, 482  
tlog\_open, 483  
tmadmin, 57, 284  
tmadminツールで使用できるコマンド, 58  
tmapm, 60  
tmax\_accept\_crypt, 469  
tmax\_auth\_plugin\_init, 476  
tmax\_chk\_authen, 472  
tmax\_chk\_author, 473  
tmax\_chk\_conn, 88

tmax_crypt_plugin_fini, 475	tmax_sq_getkeylist, 122
tmax_crypt_plugin_init, 474	tmax_sq_keygen, 123
tmax_fini_auth, 472	tmax_sq_purge, 124
tmax_fini_crypt, 468	tmax_sq_put, 125
tmax_get_db_passwd, 290	tmax_tencrypt, 476
tmax_get_db_tnsname, 292	tmax_unwrap, 470
tmax_get_db_username, 294	tmax_wrap, 469
tmax_get_sessionid, 118	tmaxlastsvc, 126
tmax_get_svccnt, 295	tmaxlibver, 61
tmax_get_svclist, 296	tmaxoserrno, 489
tmax_gq_count, 91	tmaxreadenv, 127
tmax_gq_get, 92	tmaxtrace, 62
tmax_gq_getkeylist, 92	tmboot, 65
tmax_gq_keygen, 93	tmbootコマンドを実行時のtmconfigパス参照, 70
tmax_gq_purge, 94	tmd, 71
tmax_gq_put, 95	tmdown, 73
tmax_grid_count, 105	tmget_smtrclog, 307
tmax_grid_create, 96	tmget_smtrclog_count, 310
tmax_grid_create2, 98	tmgetsmgid, 301
tmax_grid_dequeue, 114	tmmbfgen, 77
tmax_grid_destroy, 100	tmsnmpd, 78
tmax_grid_enqueue, 112	Tmax SNMP MIB, 80
tmax_grid_get, 103	環境設定, 81
tmax_grid_get_child_with_index, 116	tp_sleep, 128
tmax_grid_get_children, 115	tp_usleep, 129
tmax_grid_is_exist, 104	tpabort, 131
tmax_grid_keylist_free, 117	tpacall, 132
tmax_grid_lock, 106	tpacallsvg, 136
tmax_grid_set, 101	TPADMNTOI, 380
tmax_grid_set_watcher, 108	tpadvertise, 311
tmax_grid_unlock, 107	tpalivechk, 138
tmax_grid_wait_watcher, 111	tpalloc, 140
tmax_init_auth, 471	tpbegin, 142
tmax_init_crypt, 467	tpbroadcast, 144
tmax_is_restarted, 298	tpcall, 146
tmax_is_xa, 298	tpcallsvg, 151
tmax_keylist_count, 118	tpcancel, 153
tmax_keylist_free, 119	tpchkclid, 314
tmax_keylist_getakey, 120	tpchkunsol, 385
tmax_my_svrinfo, 299	tpclrfd, 315
tmax_req_auth, 473	tpclrfd_w, 317
tmax_sq_count, 121	tpcommit, 155
tmax_sq_get, 121	tpconnect, 156

tpdeq, 159  
tpdeq\_ctl, 162  
tpdiscon, 167  
TPEBADDESC, 491  
TPEBLOCK, 491  
TPECLOSE, 494  
TPEEVENT, 493  
TPEITYPE, 493  
TPELIMIT, 491  
TPEMATCH, 494  
TPEMAXNO, 495  
tpend, 387  
TPENOENT, 491  
TPENOREADY, 494  
tpenq, 169  
tpenq\_ctl, 172  
TPEOST, 492  
TPEOTYPE, 493  
TPEPRESVC, 495  
TPEPROTO, 492  
TPEQFULL, 494  
TPEQPURGE, 494  
TPERDCLOSE, 495  
TPERDOWN, 495  
tperr, 81  
tperrordetail, 176  
TPESECURITY, 494  
TPESVCERR, 492  
TPESVCFAI, 492  
TPESVRDOWN, 495  
TPESYSTEM, 492  
TPETIME, 492  
TPETRAN, 493  
tpextsvcinfo, 177  
tpextsvcname, 179  
tpforward, 319  
tpfree, 181  
tpget\_timeout, 182  
tpgetactivesvr, 183  
tpgetcliaddr, 186  
tpgetcliaddr\_ipv6, 187  
tpgetclid, 322  
tpgetclid\_c, 405  
tpgetctxt, 189  
tpgetdbsessionid, 324  
tpgetenv, 192  
tpgethostaddr, 389  
tpgetlev, 193  
tpgetmaxsvr, 325  
tpgetmaxuser, 326  
tpgetminsvr, 327  
tpgetmynode, 328  
tpgetmysvgno, 329  
tpgetmysvrid, 331  
tpgetnodename, 332  
tpgetnodeno, 333  
tpgetorgclh, 334  
tpgetorgnode, 335  
tpgetpeer\_ipaddr, 336  
tpgetpeername, 194  
tpgetrcahseqno, 196  
tpgetrcainfo, 197  
tpgetrply, 197  
tpgetsockname, 201  
tpgetsprlist, 203  
tpgetsvcname, 338  
tpgetsvglist, 205  
tpgetsvrseqno, 340  
tpgetunsol, 390  
TPGOTSIG, 493  
tpgprio, 206  
tpinit, 393  
tpissetfd, 341  
tpissetfd\_w, 344  
tpmcall, 208  
tpmcallx, 210  
tpnotify, 213  
tppost, 215  
tpprechk, 344  
tpputenv, 217  
tpqstat, 218  
tpqsvcstat, 220  
tprealloc, 221  
tprecv, 223  
tpregancb, 381  
tpregcb, 345

tpreissue, 227  
tprelay, 347  
tpremoteconnect, 228  
tpreset, 394  
tpresumetx, 349  
tpreturn, 350  
tpsavectx, 354  
tpschedule, 356  
tpscmt, 229  
tpsend, 230  
tpsendtocli, 358  
tpset\_timeout, 235  
tpsetctx, 246  
tpsetdbsessionid, 360  
tpsetfd, 237  
tpsetsvctimeout, 234  
tpsetunsol, 395  
tpsetunsol\_flag, 397  
tpsleep, 240  
tpspracall, 242  
tpsprio, 244  
tpstart, 399  
tpstrerror, 249  
tpsubqname, 250  
tpsubscribe, 251  
tpsuspendtx, 361  
tpsvctimeout, 363  
tpsvrdone, 364  
tpsvrdown, 365  
tpsvrinit, 366  
tpsvrthrdone, 368  
tpsvrthrinit, 369  
tpterm, 403  
tptobackup, 404  
tptsleep, 371  
tptypes, 253  
tpunadvertise, 372  
tpunregancb, 383  
tpunregcb, 374  
tpunsubscribe, 254  
tpuschedule, 375  
tuxgetenv, 255  
tuxputenv, 256

tuxreadenv, 257  
twagent, 83  
tx\_begin, 259  
tx\_close, 377  
tx\_commit, 261  
tx\_info, 263  
tx\_open, 378  
tx\_rollback, 265  
tx\_set\_commit\_return, 268  
tx\_set\_transaction\_control, 270  
tx\_set\_transaction\_timeout, 273

## U

ulogsync, 275  
uncfl, 84  
untmmbfgen, 85  
UserLog, 278  
userlog, 276  
Usiginit, 485  
Usignal, 486  
Ustrerror, 489  
Uunix\_err, 488  
Uunixerr, 488

## W

WinTmaxAcall, 451  
WinTmaxAcall2, 455  
WinTmaxEnd, 459  
WinTmaxSend, 460  
WinTmaxSetContext, 464  
WinTmaxStart, 466

## X

xwsdlgen, 85

## は

フィールド・キー, 20

