

# Tmax アプリケーション開発ガイド

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 アプリケーション開発ガイド

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 .....	xiii
<b>第1章 Tmaxアプリケーションについて .....</b>	<b>1</b>
1.1. 概要 .....	1
1.2. 構成 .....	1
1.3. 特徴 .....	2
<b>第2章 クライアント・プログラム .....</b>	<b>5</b>
2.1. プログラムの特徴および構成 .....	5
2.2. 開発環境およびツール .....	6
2.3. プログラム・フロー .....	6
2.4. プログラム・コンパイル .....	9
2.5. プロセス起動および終了 .....	11
2.5.1. sdlc .....	13
2.5.2. fdlc .....	14
<b>第3章 サーバー・プログラム .....</b>	<b>15</b>
3.1. プログラムの特徴および構成 .....	15
3.1.1. TCS .....	16
3.1.2. UCS .....	18
3.2. 開発環境およびツール .....	19
3.3. プログラムのフロー .....	19
3.3.1. TCS .....	20
3.3.2. UCS .....	22
3.4. プログラム・コンパイル .....	25
3.4.1. TCS .....	26
3.4.2. UCS .....	30
3.5. プロセス作成および終了 .....	31
<b>第4章 通信タイプ .....</b>	<b>33</b>
4.1. 概要 .....	33
4.2. 同期型通信 .....	34
4.3. 非同期型通信 .....	35
4.4. 会話型通信 .....	36
4.4.1. 会話型通信関連のイベント .....	37
<b>第5章 バッファ・タイプ .....</b>	<b>39</b>
5.1. 概要 .....	39
5.2. バッファのタイプ .....	39
5.3. バッファの管理 .....	41
5.3.1. 構造体バッファ .....	41
5.3.2. フィールド・バッファ .....	42
<b>第6章 トランザクション .....</b>	<b>43</b>

6.1.	概要 .....	43
6.2.	分散トランザクション .....	44
6.2.1.	XAモード .....	47
6.2.2.	非XAモード .....	48
6.3.	トランザクション関連エラー .....	50
6.3.1.	TXエラー .....	50
6.3.2.	XAエラー .....	50
<b>第7章</b>	<b>マルチスレッドとマルチコンテキスト .....</b>	<b>53</b>
7.1.	概要 .....	53
7.2.	クライアント・プログラム .....	53
7.2.1.	プログラムのフロー .....	53
7.2.2.	プログラムの実装 .....	54
7.2.3.	プログラムの例 .....	56
7.3.	サーバー・プログラム .....	63
7.3.1.	概要 .....	64
7.3.2.	プログラムのフロー .....	65
7.3.3.	プログラムの実装 .....	68
7.3.4.	サービス処理プログラムの例 .....	70
7.3.5.	コンテキスト共有プログラムの例 .....	73
<b>第8章</b>	<b>セキュリティ・システム .....</b>	<b>79</b>
8.1.	概要 .....	79
8.2.	1段階セキュリティ(システム接続制御) .....	79
8.3.	2段階セキュリティ(ユーザー認証) .....	80
8.4.	3段階セキュリティ(サービス・アクセス制御) .....	82
<b>第9章</b>	<b>クライアントAPI .....</b>	<b>85</b>
9.1.	概要 .....	85
9.2.	接続および解除 .....	88
9.2.1.	tpstart .....	88
9.2.2.	tpend .....	92
9.3.	同期型通信 .....	93
9.3.1.	tpcall .....	93
9.4.	非同期型通信 .....	98
9.4.1.	tpacall .....	98
9.4.2.	tpgetrply .....	102
9.4.3.	tpcancel .....	105
9.5.	会話型通信 .....	107
9.5.1.	tpconnect .....	107
9.5.2.	tpsend .....	111
9.5.3.	tprecv .....	114
9.5.4.	tpdiscon .....	118
9.6.	非要求メッセージの処理 .....	120
9.6.1.	tpsetunsol .....	121

9.6.2.	tpgetunsol .....	123
9.7.	タイムアウトの変更 .....	125
9.7.1.	tpset_timeout .....	126
9.7.2.	tpsetsvctimeout .....	127
9.8.	バッファの管理 .....	129
9.8.1.	tpalloc .....	129
9.8.2.	tprealloc .....	131
9.8.3.	tpfree .....	133
9.8.4.	tpypes .....	134
9.9.	トランザクションの管理 .....	136
9.9.1.	tx_begin .....	136
9.9.2.	tx_commit .....	138
9.9.3.	tx_info .....	140
9.9.4.	tx_rollback .....	142
9.9.5.	tx_set_transaction_timeout .....	145
9.9.6.	tx_set_transaction_control .....	147
9.9.7.	tx_set_commit_return .....	149
9.10.	RQシステム .....	152
9.10.1.	tpenq .....	152
9.10.2.	tpdeq .....	155
9.10.3.	tpqstat .....	158
9.10.4.	tpextsvcname .....	160
9.11.	イベントを活用するAPI .....	162
9.11.1.	tpsubscribe .....	162
9.11.2.	tpunsubscribe .....	164
9.11.3.	tpost .....	165
9.12.	ブロードキャストとマルチキャスト .....	166
9.12.1.	tpbroadcast .....	166
9.13.	環境プログラム .....	169
9.13.1.	WinTmaxAcall .....	170
9.13.2.	WinTmaxAcall2 .....	174
9.13.3.	WinTmaxStart .....	178
9.13.4.	WinTmaxEnd .....	179
9.13.5.	WinTmaxSetContext .....	180
9.13.6.	WinTmaxSend .....	182
9.14.	マルチ・スレッドとマルチ・コンテキスト .....	186
9.14.1.	tpgetctxt .....	186
9.14.2.	tpsetctxt .....	189
<b>第10章</b>	<b>サーバーAPI .....</b>	<b>193</b>
10.1.	TCS .....	193
10.1.1.	tpreturn .....	196
10.1.2.	tpforward .....	200

10.1.3.	tpsvrinit .....	203
10.1.4.	tpsvrdone .....	205
10.1.5.	tpsvrthrinit .....	206
10.1.6.	tpsvrthrdone .....	209
10.1.7.	tpgetctxt .....	210
10.1.8.	tpsetctxt .....	210
10.1.9.	tpsendtocli .....	210
10.1.10.	tpgetclid .....	212
10.1.11.	tpchkclid .....	213
10.2.	UCS .....	215
10.2.1.	tpschedule .....	217
10.2.2.	tpuschedule .....	218
10.2.3.	tpsetfd .....	220
10.2.4.	tpissetfd .....	222
10.2.5.	tpclrfd .....	225
10.2.6.	tpsavectx .....	227
10.2.7.	tpgetctx .....	229
10.2.8.	tpcancelctx .....	230
10.2.9.	tprelay .....	231
10.2.10.	tpregcb .....	233
10.2.11.	tpunregcb .....	235
<b>第11章</b>	<b>エラー処理 .....</b>	<b>237</b>
11.1.	概要 .....	237
11.2.	APIレベルのエラー処理 .....	237
11.2.1.	tpstrerror .....	237
11.3.	システム・レベルのエラー処理 .....	238
11.3.1.	Unixerr .....	239
11.3.2.	Unix_err .....	239
11.3.3.	Ustrerror .....	240
11.3.4.	tmaxoserrno .....	240
11.4.	デバッグ .....	241
11.4.1.	デバッグCLH .....	241
11.4.2.	デバッグ・ライブラリー .....	242
<b>第12章</b>	<b>例 .....</b>	<b>243</b>
12.1.	通信タイプの例 .....	243
12.1.1.	同期型通信 .....	243
12.1.2.	非同期型通信 .....	246
12.1.3.	会話型通信 .....	250
12.2.	グローバル・トランザクション・プログラムの例 .....	256
12.3.	データベース・プログラム .....	264
12.3.1.	Oracle Insertプログラム .....	265
12.3.2.	Oracle Selectプログラム .....	269



12.3.3.	Informix Insertプログラム	276
12.3.4.	Informix Selectプログラム	285
12.3.5.	DB2プログラム	293
12.4.	データベース連携プログラム	302
12.4.1.	同期型モード(同機種)	302
12.4.2.	同期型モード(異機種)	310
12.4.3.	非同期型モード(同機種)	313
12.4.4.	会話型モード(同機種)	319
12.5.	TIPを利用したプログラム	327
12.5.1.	TIPの構造	327
12.5.2.	TIPの使用	330
12.5.3.	TIPの使用例	333
12.5.4.	システム環境情報照会プログラム	343
12.5.5.	システム統計情報照会プログラム	346
12.5.6.	サーバー・プロセスの起動および終了プログラム	350
12.6.	再帰呼び出し(Local recursive call)	356
<b>付録 A.</b>	<b>Tmax環境設定</b>	<b>361</b>
A.1.	環境ファイル	361
A.1.1.	DOMAINセクション	362
A.1.2.	NODEセクション	362
A.1.3.	SVRGROUPセクション	363
A.1.4.	SERVERセクション	364
A.1.5.	SERVICEセクション	364
A.2.	メイクファイル	365
<b>索引</b>		<b>367</b>



# 図目次

[図 1.1]	Tmaxアプリケーション・プログラム .....	2
[図 2.1]	クライアント・プログラムのフロー .....	7
[図 2.2]	クライアント・プログラムの関数プロセス .....	7
[図 2.3]	クライアント・プログラムのコンパイル・プロセス(構造体バッファの使用) .....	9
[図 2.4]	クライアント・プログラムのコンパイル・プロセス(フィールド・バッファの使用) .....	10
[図 3.1]	Tmaxサーバー・プロセスのタイプ .....	16
[図 3.2]	サーバー・プログラムのフロー .....	20
[図 3.3]	TCS方式のサーバー・プログラムのフロー .....	21
[図 3.4]	UCS方式のサーバー・プログラムのフロー .....	22
[図 3.5]	サーバー・プログラムのコンパイル(構造体バッファ) .....	26
[図 3.6]	サーバー・プログラムのコンパイル(フィールド・バッファ) .....	27
[図 4.1]	同期型通信 .....	34
[図 4.2]	同期型通信モデル .....	34
[図 4.3]	非同期型通信 .....	35
[図 4.4]	非同期型通信モデル .....	35
[図 4.5]	会話型通信 .....	36
[図 4.6]	会話型通信モデル .....	36
[図 5.1]	Tmax通信バッファのタイプ .....	40
[図 5.2]	構造体バッファを利用したアプリケーション・プログラムのコンパイル .....	42
[図 5.3]	フィールド・バッファを利用したアプリケーション・プログラムのコンパイル .....	42
[図 6.1]	2相コミット・プロトコル .....	44
[図 6.2]	X/Open DTPの構造 .....	45
[図 6.3]	分散トランザクションの処理過程 .....	46
[図 6.4]	XAモード .....	47
[図 6.5]	非XAモード .....	49
[図 7.1]	Tmaxクライアントのマルチスレッド・アプリケーション .....	54
[図 7.2]	Tmaxクライアントのマルチコンテキスト・アプリケーション .....	54
[図 7.3]	Tmaxサーバーのマルチスレッド・アプリケーション .....	66
[図 7.4]	Tmaxサーバーのマルチコンテキスト・アプリケーション .....	68
[図 10.1]	tpforward .....	201
[図 10.2]	UCS方式のサービス・フォワーディング .....	216
[図 12.1]	2つのデータベースの接続 .....	257
[図 12.2]	同期型モードのフロー図(同機種のデータベース接続) .....	303
[図 12.3]	同期型モードのフロー図(異機種のデータベース接続) .....	310
[図 12.4]	非同期型モードのフロー図(同機種のデータベース接続) .....	313
[図 12.5]	会話型モードのフロー図(同機種のデータベース接続) .....	319



# このガイドについて

## 対象読者

本書は、Tmax<sup>®</sup>(以下、Tmax)を使用してプログラムを開発するユーザーを対象としています。

Tmaxを利用してアプリケーションを開発するための基本概念およびクライアントとサーバー・プログラムの基本フロー、APIを使用した開発方法などについて説明します。Tmaxにより構築できる多様な開発環境に関する説明と、その環境における開発方法を提示します。また、多様な例を示すことにより、ユーザーのプログラムへの理解を高めます。

## 前提知識

本書は、Tmaxシステムを全般的に理解し、Tmaxシステムが提供する各種機能と特性を習得するための基本的な手引書です。本書を理解するには、以下の内容についての事前知識が必要です。

- ミドルウェアおよびUNIXシステム
- Tmaxの基本概念
- JavaおよびCプログラミング

## 制限事項

本書を読む前にTmaxの基本概念を熟知している必要があります。実務での具体的な使用方法や管理および運用に関する事項については、各製品のガイドを参照してください。

---

### 参考

Tmaxについての基本的な内容は、『Tmax 運用ガイド』および『Tmax スタートガイド』を参照してください。Tmaxで提供するコマンドとC、APIについての説明は、『Tmax リファレンスガイド』を参照してください。

---

## 本書の構成

本書は、計12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: Tmaxアプリケーションについて

Tmaxアプリケーションの概要と構成および特徴について説明します。

- 第2章: クライアント・プログラム

クライアント・プログラムの構成、コンパイル、プロセスについて説明します。

- 第3章: サーバー・プログラム

サーバー・プログラムのフローと特徴および構成について説明します。

- 第4章: 通信タイプ

Tmaxが提供する通信タイプについて説明します。

- 第5章: バッファ・タイプ

Tmaxが提供するバッファ・タイプについて説明します。

- 第6章: トランザクション

トランザクションの概念と定義および処理方法について説明します。

- 第7章: マルチスレッドとマルチコンテキスト

マルチスレッドとマルチコンテキストのプログラム・フローと実装方法について説明します。

- 第8章: セキュリティー・システム

Tmaxが提供する第3段階のセキュリティー・システムについて説明します。

- 第9章: クライアントAPI

Tmaxクライアント・プログラムが提供する関数について説明します。

- 第10章: サーバーAPI

Tmaxサーバー・プログラムが提供する関数について説明します。

- 第11章: エラー処理

TmaxのAPIエラーが発生した場合、状況に応じてエラーを処理する方法について説明します。

- 第12章:例

Tmaxシステムが提供する特殊機能を使用するための例について説明します。

- 付録 A:Tmax環境設定

Tmax環境ファイルの構成要素について説明します。

## 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
<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
<b>参考</b>	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 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 スタートガイド	Tmaxの基本概念と構成に関する基本事項について説明しています
Tmax プログラミングガイド(4GL)	4GL言語を使用してTmaxアプリケーションを開発するユーザー向けに、言語別インターフェースと関連関数について説明しています
Tmax リファレンスガイド	Tmaxアプリケーションの開発に使用するコマンドおよびクライアントとサーバーの接続、通信に使用する関数の使用方法と例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 Tmaxアプリケーションについて

本章では、Tmaxアプリケーションの定義、構成、特長などについて説明します。

## 1.1. 概要

Tmaxアプリケーションとは、オープン環境でTmaxをミドルウェアとして使用して開発したクライアントとサーバー・プログラムです。高性能PCの登場とプログラム手法の発展は、ホスト中心の中央処理方式からクライアントとサーバーが作業を分担する環境への変化をもたらしました。クライアント・サーバー環境の登場によって作業の分担が進み、高性能PCの活用とサーバーの小型化、各種ハードウェアの選択幅の拡大などのように、リソースを効率的に活用できるようになりました。

ところが、クライアント/サーバー環境には以下のような問題があります。

- プログラムを作成するには、開発者に各ハードウェア、運用体制、ネットワーク・プロトコルに関する高度な知識が必要です。
- ユーザー数とデータ量が増加すれば処理速度が著しく低下します。
- 過剰なネットワーク・トラフィックの発生により、処理速度が低下します。
- 分散環境に伴う管理(プロセス管理、通信、セキュリティ、障害対策など)が難しいです。

Tmaxアプリケーションは、クライアント・サーバー・プログラムのメリットを享受しつつも、上記のデメリットを画期的に改善できます。Tmaxをミドルウェアにしてアプリケーションを作成する場合、Tmaxが提供する関数を使用してプログラムを作成できるため、通信プログラムおよびプロセス管理、トランザクション管理など、管理の難点をTmaxが解決します。Tmaxが提供する関数はバッファと通信、記録トランザクション関連の関数であり、サーバー・ライブラリー(libsvr.a)とクライアント・ライブラリー(libcli.a)に区別して提供されます。Tmaxが提供する関数は分散処理国際標準のX/Open DPTモデルを遵守します。

## 1.2. 構成

Tmaxアプリケーションは、大きく**クライアント・プログラム**と**サーバー・プログラム**で構成されます。Tmaxを使用するには環境設定作業が必要です。

- クライアント・プログラム

クライアント・プログラムはサービスを要求し、その要求に対する応答をサーバーから取得し、ユーザーが必要とする形式で結果を転送するプログラムです。Tmaxシステムに接続するには、Tmaxアドレス(Address)、ポート番号などを設定したUNIX環境ファイル(例: .profile)が必要です。クライアント・プログラ

ムはユーザー環境ファイルの環境変数を参照してTmaxシステムに接続し、構造体バッファまたはフィールド・バッファを使用する場合、それぞれに該当するバイナリー・ファイルを参照します。

- サーバー・プログラム

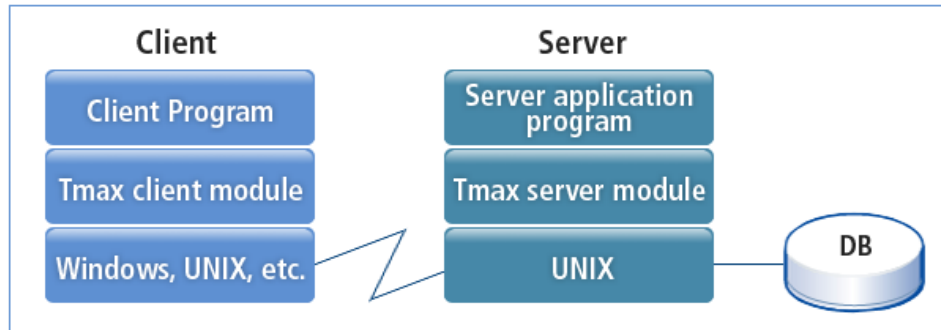
サーバー・プログラムはクライアントから受信した要求を処理して適切な応答を作成するプログラムです。システム・リソースへのアクセスを管理する機能とクライアントからの要求を処理して作成した応答をクライアントに転送する機能をします。

サーバー・プログラムについては、Tmaxシステムがmain()ルーチンを管理するので、開発者はサービス・ルーチンのみを作成すればいいです。サービス・ルーチンはTmaxのユーティリティによって作成される各サービス・テーブルと一緒にコンパイルします。構造体バッファを使用する場合、ユーティリティを使用して取得できるコンバージョン・ルーチンも一緒に使用する必要があります。

Tmaxアプリケーション・サーバーは、**サーバー・プログラム**と**Tmax環境ファイル**を必要とします。Tmax環境ファイルはTmaxが使用するリソースに対する全システムの構成を設定するファイルであり、Tmax管理者が作成します。このファイルはTmaxシステムとサーバー・プログラムを起動するときや、サービス・テーブルを作成するときに参照されます。

以下は、Tmaxアプリケーション・プログラムの構造です。

**[図 1.1] Tmaxアプリケーション・プログラム**



---

**参考**

クライアント・サーバー・プログラムの詳細については、「[第2章 クライアント・プログラム](#)」と「[第3章 サーバー・プログラム](#)」を参照してください。

---

## 1.3. 特徴

Tmaxアプリケーションは、UNIXプログラムに比べ、以下のような特徴があります。

- 分散処理国際標準(X/Open DTP model)を遵守します。



ネットワークを制御するルーチンはTmax関数を使用します。この関数は分散処理国際標準を遵守するので、規格を遵守する他のミドルウェアで開発されたプログラムも別途の修正無しで使用できます。

- クライアントの要求を処理する純粋なサービス・ルーチンのみ作成します。

サーバー・アプリケーション・プログラムを作成するとき、クライアントは基本的なC言語の形式に合わせてプログラム(内部にmain()を含む)を開発する必要があります。また、サーバー・プログラムはクライアントのサービス要求を処理する純粋な業務サービス・ルーチン(内部にmain()が含まれていない)のみを開発します。

- 多様な通信タイプを提供します。

サービス要求には同期型通信、非同期型通信、会話型通信の3種類があります。多様な通信タイプを提供しているため、開発者は通信ネットワークへの深い知識がなくても簡単にプログラミングできます。

以下は、サービス要求の各タイプについての説明です。

通信タイプ	説明
同期型通信	サービスを要求(tpcall)した後、応答が返されるまで待機します
非同期型通信	サービスを要求(tpacall)した後、応答が返されるまで(tpgetrply)、他の業務を実行できます
会話型通信	初期接続を設定(tpconnect)した後、メッセージの送信(tpsend)と受信(tprecv)を繰り返して実行できます

---

#### 参考

通信タイプの詳細については、「[第4章 通信タイプ](#)」を参照してください。

---

- 多様なバッファーを使用できます。

サービスを要求するとき、7種類のバッファーを使用できます。通常、開発者は異種間のデータ通信をするとき、データの整合性を保証することに苦難を感じます。異種ハードウェアと運用体制によっては、データをメモリーに割り当てる方式とデータ・タイプ別に定義された長さが異なる場合があります。そのため、特定のデータ値を送信したとき、受信側では他の値に認識されることもあり得るので、一般的にすべてのハードウェアと運用体制で同じく認識できる文字型に変換してデータを送受信する必要があります。

Tmaxでは、7種類のバッファーをサポートすることで開発者に選択の幅を広げただけでなく、マシン、運用体制、ネットワークへの深い知識がなくても開発できるようにしました。文字型のほか、構造体タイプなどのバッファーを提供しているため、システムの性能向上はもちろん、ネットワークの負荷を減少させます。

---

#### 参考

バッファーの詳細については、「[第5章 バッファー・タイプ](#)」を参照してください。

---

- 開発が簡単です。

- ハードウェア、通信ネットワークへの深い知識がなくても簡単にプログラミングできます。
  - サービス要求を処理するサーバーのアドレスが分からない場合にもサービスを要求することができます。
  - クライアントは名前だけでサービスを要求することができます。
- トランザクション処理の整合性を提供します。

プログラムではトランザクションの範囲のみを設定すれば、Tmaxは2相コミット方式でデータの整合性を提供します。

---

#### 参考

2相コミット(Two-Phase Commit, 2PC)は、2つ以上のデータベースが連動するとき、2段階(準備段階、コミット段階)で処理することをいいます。詳細については、「[第6章 トランザクション](#)」を参照してください。

---

## 第2章 クライアント・プログラム

本章では、クライアント・プログラムのフローと開発環境、特長、そして構成について説明します。

### 2.1. プログラムの特徴および構成

Tmaxは通信ネットワーク、バッファ管理のための関数を提供してアプリケーションの開発を容易にします。Tmaxが提供する関数はライブラリーの形で提供され、アプリケーションと一緒にコンパイルされます。開発者はどのマシンのどのサーバーでどのようなサービスを提供するかに係わらず、プログラムを開発できます。

クライアント・プログラムのコーディングが完了すれば、コンパイルして実行ファイルを作成します。クライアント・プログラムをコンパイルするには、開発者が作成したクライアント・プログラム、Tmaxクライアント・ライブラリー、構造体ファイル(構造体バッファを使用する場合)、そしてフィールド・テーブルが定義されているファイル(フィールド・キー・バッファを使用する場合)が必要です。

以下は、クライアント・プログラムの構成に対する説明です。

- クライアント・プログラム

開発者が作成したクライアント・プログラムです。

- クライアント・ライブラリー

Tmaxが提供するライブラリー(libcli.a / libcli.so)でクライアント・プログラムを開発するときに使用する関数のオブジェクト・コードです。

- 構造体ファイル(.s)

クライアント・プログラムで構造体(STRUCT、X\_C\_TYPE、X\_COMMON)を使用した場合、<ファイル名.s>の構造体ファイルが必要です。構造体ファイルをsdhcコマンドを利用して事前にコンパイルします。コンパイルにより、構造体の各データが標準の通信型に変換されるために必要な情報が含まれているバイナリー・ファイルが作成されます。このファイルはクライアント・プログラムを実行するとき、標準の通信型でデータが送受信されるときに使用します。

- フィールド・キー・ファイル(.f)

フィールド・キー構造を使用した場合、<ファイル名.f>のフィールド定義ファイルが必要です。fdlcコマンドを使用してファイルをコンパイルすると、フィールド・キー・バッファ・ファイルはキー・マッピング(Key Mapping)を利用して<フィールド・キー・バッファ名\_fdl.h>を作成してプログラムを使用します。既存の構造体ファイルとは異なり、ユーザーが必要とするフィールド・キー値のみを操作して送信できるので、リソースの浪費を低減することができます。ただし、キー・マッピングの過程でオーバーヘッドが発生することがあるので注意が必要です。

## 2.2. 開発環境およびツール

以下は、Tmaxクライアント・プログラムの開発環境と開発ツールです。

- 開発環境

UNIX、WindowNT、Windows95/98/2000、Windows 3.1、MS-DOSなどの運用体制

- 開発ツール

ANSI C、VC++、BC++、VB、VB .Net、C#、Delphi、PowerBuilder、Embedded VCなど

---

### 参考

開発ツールを使用したプログラムの詳細については、『Tmax プログラミングガイド(4GL)』を参照してください。

---

## 2.3. プログラム・フロー

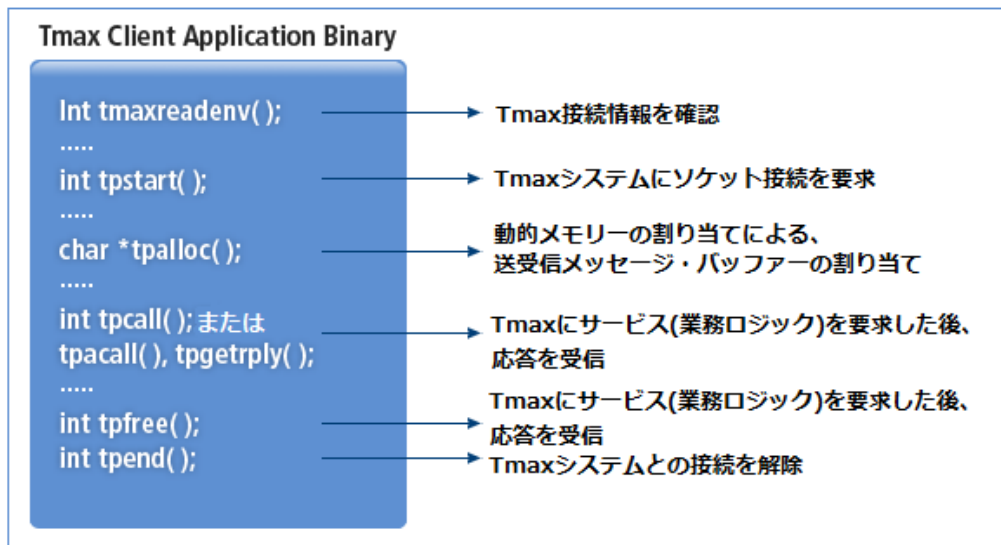
クライアント・プログラムは、ユーザーの要求を受信してサーバーに転送し、サーバーで処理された結果を取得してユーザーに表示する機能をするプログラムです。

以下は、クライアント・プログラムを作成する順序です。

```
main()  
{  
    tpstart バッファの割り当て  
    tpstart バッファの初期化  
  
    Tmaxに接続  
    送受信メッセージ・バッファの割り当て  
    while {  
        送信メッセージ・バッファにユーザーの要求を入力  
        送信メッセージ・バッファをサーバーに送信(サービス要求)  
        サーバーからの応答を送信メッセージ・バッファで受信  
        送信メッセージバッファの内容をユーザーに表示  
    }  
    送受信メッセージ・バッファを削除  
  
    Tmax接続の解除  
}
```

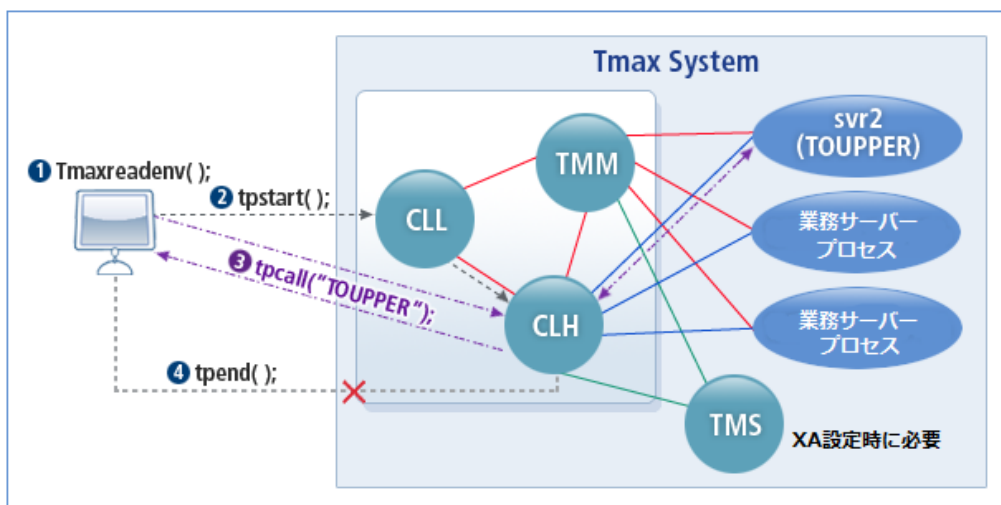
以下は、クライアント・プログラムのフローを示す図です。

【図 2.1】クライアント・プログラムのフロー



以下の図は、クライアント・プログラムにおける各関数のプロセスです。

【図 2.2】クライアント・プログラムの関数プロセス



## 参考

Tmaxがアプリケーション開発のために提供する通信ネットワークとバッファ管理用関数の詳細については、「[第9章 クライアントAPI](#)」を参照するか、『Tmax リファレンスガイド』を参照してください。

以下は、クライアント・プログラムで使われる主要関数についての説明です。

- Tmax通信環境関数

関数	説明
tmaxreadenv	<p>Tmaxクライアント・プログラムを実行するときに基本的に参照することになる一連のTmax環境変数を特定のファイルに記述できます。</p> <p>APIを呼び出すプログラムの作成時に特定ファイルのパス名とファイル名を設定すれば、当該クライアント・プログラムを実行するときに設定されたファイルのファイル・ポインターを受信した後、内容を分析(parsing)します。分析した内容のうち、TMAX_HOST_ADDR、TMAX_HOST_PORTなどのTmax環境変数は、クライアント・プログラムの実行時に確保されたメモリー領域内にインポートした後、関連API関数を呼び出す場合に利用します</p>

- クライアントとサーバー接続および終了関数

関数	説明
tpstart	CLLプロセスでソケット接続を要求し、承認(Accept)された接続をCLH側に転送します。tpstart()を呼び出すとき、TMAX_HOST_ADDR、TMAX_HOST_PORTに設定された値を使用します
tpend	CLLプロセスでソケット接続を終了します

- 通信バッファと解除関数

関数	説明
tpalloc	Tmaxを介してクライアントとサーバー間でデータを送受信する場合、使用するメモリーを動的に割り当てます。データを送受信するときに必要な長さ情報を予測し、そのサイズのみだけ事前に割り当てる必要があります。動的に割り当てられたメモリーは必ず明示的に解除します
tpfree	tpalloc()によって割り当てられた動的メモリーを明示的に解除(返却)します。割り当てられたメモリー・バッファは返す必要があります。メモリーを返さないとgabage(メモリー・リーク)が発生します

- クライアント同期型通信関数

関数	説明
tpcall	CLH側にサービス要求および送信データを転送してクライアントの要求サービスを確認した後、サーバー・アプリケーション・プロセスに渡します。クライアントはサービス要求への応答が返されるまで待機します

- クライアント非同期型通信関数

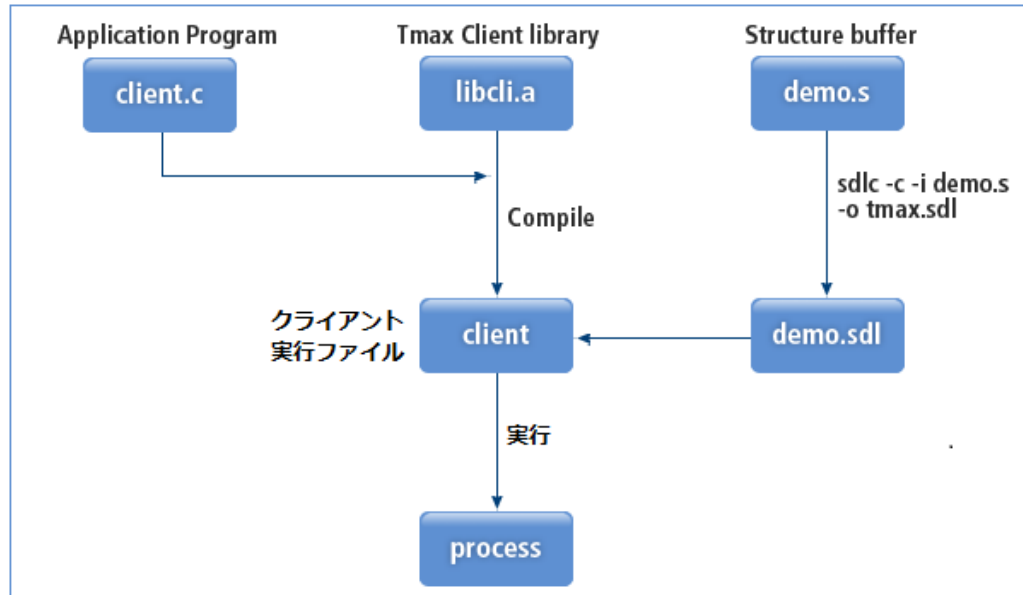
関数	説明
tpacall	<p>CLH側にサービス要求および送信データを転送してクライアントの要求サービスを確認した後、サーバー・アプリケーション・プロセスにサービスを渡します。</p> <p>クライアントはサービス要求に対する応答とは関係なくtpacall()を実行した後、ロジックを実行します</p>
tpgetrply	<p>tpacall()のパラメータ(cd)値を通じてCLH側に応答データを要求し、当該クライアントに送信できる応答データがすでに存在していれば、直ちに送ります。</p> <p>パラメータの中でflags設定値に従って応答データが送られるまで待機するか、あるいは、クライアントに応答受信エラーを直ちに通知します</p>

## 2.4. プログラム・コンパイル

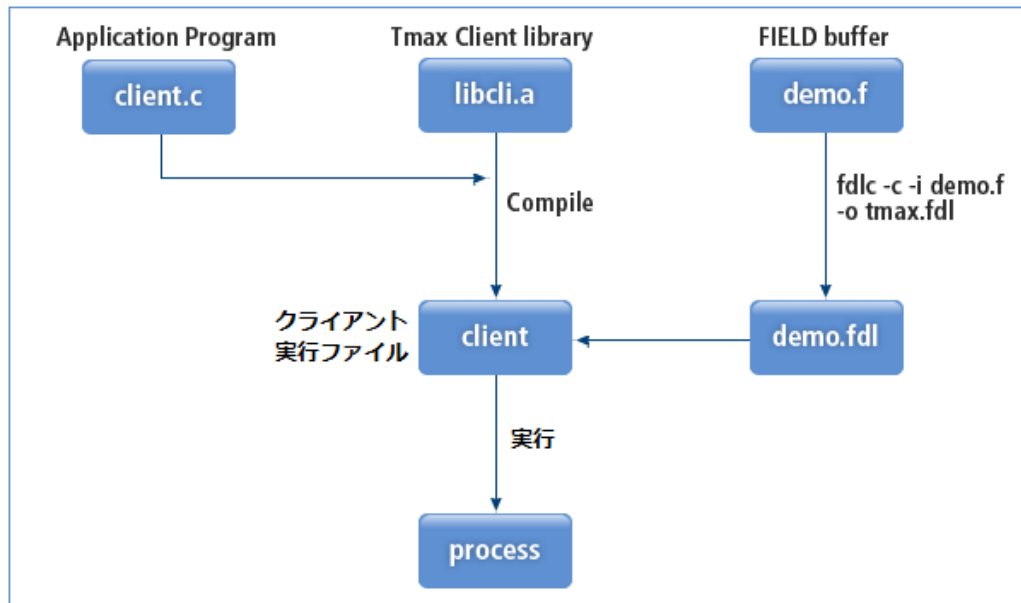
クライアント・プログラムをコンパイルしてオブジェクト・ファイルを作成します。

以下の図は、構造体バッファとフィールド・バッファを使用してクライアント・プログラムをコンパイルするプロセスです。

[図 2.3] クライアント・プログラムのコンパイル・プロセス(構造体バッファの使用)



[図 2.4] クライアント・プログラムのコンパイル・プロセス(フィールド・バッファの使用)



以下は、クライアント・プログラムのコンパイル順序です。

1. UNIX環境ファイルを作成します。

Tmax接続に必要な環境変数を設定します。(profile、.login、.cshrc)

```
TMAX_HOST_ADDR = Tmaxアドレス (=IP Address)
TMAX_HOST_PORT = ポート番号 (default : 8888)
```

2. クライアント・プログラムを作成します。(client.c)

サーバーにサービスを要求し、サーバーに応答するプログラムを開発します。

3. 開発されたクライアント・プログラムをコンパイルしてライブラリーを作成します。(libcli.a/libcli.so)

4. 構造体バッファを使用する場合、構造体ヘッダー・ファイルをコンパイルします。STRUCT、X\_C\_TYPE、X\_COMMONバッファを使用する場合は、ダミー構造体を作成する必要があります。通信時に標準通信型に変換するためにsdlcを使用してコンパイルします。作成されたsdlファイルは、クライアント・プログラムを実行するときに必要です。

フィールド・バッファを使用する場合には、fdlcを使用してフィールド・キー・ファイルをコンパイルします。

以下は、Tmaxクライアント・プログラムのためのメイクファイル(Makefile)の例です。

```
TARGET = <clientname>
APOBJS = $(TARGET).o

TMAXLIBD = $(TMAXDIR)/lib
```



```

#TMAXLIBSはOSに応じて異なります。
#Solaris : TMAXLIBS = -lsocket -lnsl -lcli
#Compac, HP, IBM, Linux : TMAXLIBS= -lcli

TMAXLIBS = -lcli

#CFLAGSはOSに応じて異なります。
#Solaris 32bit, Compaq, Linux: CFLAGS = -O -I$(TMAXDIR)
#Solaris 64bit: CFLAGS = -xarch=v9 -O -I$(TMAXDIR)
#HP 32bit: CFLAGS = -Ae -O -I$(TMAXDIR)
#HP 64bit: CFLAGS = -Ae +DA2.0W +DD64 +DS2.0 -O -I$(TMAXDIR)
#IBM 32bit: CFLAGS = -q32 -brtl -O -I$(TMAXDIR)
#IBM 64bit: CFLAGS = -q64 -brtl -O -I$(TMAXDIR)

CFLAGS = -O -I$(TMAXDIR)

#
.SUFFIXES : .c

.c.o:
    $(CC) $(CFLAGS) -c $<

#
# client compile
#

$(TARGET): $(APOBJS)
    $(CC) $(CFLAGS) -L$(TMAXLIBD) -o $(TARGET) $(APOBJS) $(TMAXLIBS)

#
clean:
    -rm -f *.o core $(TARGET)

```

Tmaxが提供する例題プログラムでは、コンパイルというシェル・スクリプトを使用してメイクファイルを実行しています。コンパイルというシェル・スクリプトを使用するには、コマンドに以下のように入力します。コンパイルでメイクファイルのEXEに拡張子を除いたクライアント・プログラム名を入力します。

```
compile c cli (=>拡張子を除いたクライアント・プログラム名)
```

## 2.5. プロセス起動および終了

Tmaxを起動するには環境変数の設定が必要です。

クライアント・プログラムはTmaxに接続するとき環境変数で必要な情報を取得するために予めUNIXシェル環境ファイル(cシェル: .cshrc、kornシェル: .profile、bash シェル: .bash\_profileなど)にTMAX\_HOST\_ADDR、

TMAX\_HOST\_PORTが設定されていることを確認します。障害に備えてTMAX\_BACKUP\_ADDRとTMAX\_BACKUP\_PORTを設定することができます。また、ネットワーク障害に備えてTMAX\_CONNECT\_TIMEOUTを設定することができます。

UNIX環境ファイルにTmax環境変数が設定されていれば、クライアント・プロセスの作成に必要な準備は完了します。クライアント・プロセスは、他の実行ファイルと同じ方法で実行および終了します。

以下は、各環境変数についての説明です。

環境変数	説明
TMAX_HOST_ADDR	TmaxサーバーのIPを設定する部分であり、非常に重要な役割をします。 クライアントが要求したサービスが最初に接続設定されたサーバーに存在しなければ、内部的に再接続を設定してサービスを受信します。  クライアントがトランザクションを要求した場合、最初に接続が設定されたTmaxがトランザクションのアジャスターとなり、2PCをコントロールします。 したがって、最も多く使われるサービスが存在しているサーバーのアドレスを設定し、ネットワーク負荷の軽減はもちろん応答時間の短縮効果も期待できます
TMAX_HOST_PORT	Tmaxサーバーのポートを設定する部分であり、Tmax環境ファイルにTPORTNOが設定されていないと、デフォルト値の8888が使われます
TMAX_BACKUP_ADDR	クライアントがサービスを要求すると、まずTMAX_HOST_ADDRが指定するサーバーへ接続を試み、そのサーバーに障害が発生している状態であれば、TMAX_BACKUP_ADDRが指定するサーバーに再接続を試みます。TMAX_BACKUP_ADDRが指定されていないとクライアントの要求は再接続を試みず、失敗として終わります
TMAX_BACKUP_PORT	TMAX_BACKUP_ADDRが指定するTmaxサーバーのポート番号です
TMAX_CONNECT_TIMEOUT	ネットワーク障害に備えてTMAX_CONNECT_TIMEOUTを指定します。 クライアントは環境変数に設定した時間の間待機します。  TMAX_CONNECT_TIMEOUTに指定された時間内に設定されないとtpstart()は失敗で終わります。ネットワーク障害の場合、tperronoをTPETIMEに設定します。TMAX_BACKUP_ADDR / TMAX_BACKUP_PORTが設定されるとtpstart()は、バックアップ・ホストを実行するため、他のTMAX_CONNECT_TIMEOUTを試みます
SDLFILE	構造体通信をするクライアント・プログラムは必ず設定します
FDLFILE	フィールド・バッファ通信をするクライアント・プログラムは必ず設定します

## 参考

sdlcとfdlcコマンドの詳細については、『Tmax リファレンスガイド』を参照してください。

## 2.5.1. sdlc

構造体通信をするクライアント・プログラムは、変数の**SDLFILE**が必ず設定されている必要があります。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にコンパイルしたSDLファイルがSDLFILE変数に定義されている必要があります。構造体通信を実行するとき、構造体情報がSDLFILEに存在していれば、Tmaxサーバーと通信することができます。

構造体ファイルは以下の形式でコンパイルします。

```
$ sdlc -c -i 構造体ファイル名 [ -o sdl ファイル名 ] [ -h ヘッダーファイル名 ]
```

オプション	説明
-i構造体ファイル名	コンパイル対象の構造体ファイル名を指定します。ファイルは1つまたは複数を定義することができ、拡張子は関係ありません。アスタリスク(*)がすべてのファイルに対応するワイルドカードとして使われます。  構造体ファイルには、「struct名 { ... };」の形式で構造体が1つ以上定義されている必要があります
[-o sdlファイル名]	作成されるSDLファイル名を指定する変数であり、SDLFILE変数に定義される名前です
[-hヘッダーファイル名]	構造体情報をヘッダー・ファイル形式で作成します。[-h]オプションが省略される場合、<名前_sdl.h>が作成されます

[-o]オプションが省略される場合、デフォルト値で作成されるSDLファイルは以下のとおりです。

- 構造体ファイルが1つの場合、作成されるファイル名は<拡張子を除いた構造体ファイル名.sdl>になります。

以下の場合、作成されるファイル名は<demo.sdl>です。

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

- 構造体ファイルが複数記述された場合、作成されるファイル名は<最初の構造体ファイル名.sdl>になります。

以下の場合、作成されるファイル名は<stra.sdl>です。

```
sdlc -c -i stra.s bana.s orage.s
```

以下は、ディレクトリーに存在する.sファイルがa.s、b.s、c.sの場合、ファイルを作成する例として作成されるファイル名は<a.sdl>になることを示しています。

```
sdlc -c -i *.s
```

## 2.5.2. fdlc

フィールド・バッファ通信をするクライアント・プログラムは、環境変数の**FDLFILE**が必ず設定されている必要があります。クライアント・プログラムで使用するフィールド・バッファ・ファイルをfdlcにコンパイルしたFDLファイルはFDLFILE変数に定義されている必要があります。フィールド・バッファの通信を実行するとき、フィールド情報がFDLFILEに存在していれば、Tmaxサーバーと通信することができます。

フィールド・バッファ・ファイルは以下の形式でコンパイルします。

```
$ fdlc -c -i フィールドバッファファイル名 [ -o fdl ファイル名 ] [ -h ヘッダーファイル名 ]
```

項目	説明
-iフィールドバッファファイル名	コンパイルする対象のフィールド・バッファ・ファイル名を指定します
[-o fdlファイル名]	作成されるFDLファイル名を指定する変数であり、FDLFILE変数に定義される名前です。[-o]オプションが省略される場合、デフォルト値で作成されるFDLファイルは<tmax.fdl>です
[-hヘッダーファイル名]	構造体情報をヘッダー・ファイル形式で作成します。[-h]オプションが省略される場合、<名前_fdl.h>が作成されます

# 第3章 サーバー・プログラム

本章では、サーバー・プログラムのフローと特徴および構成について説明します。

## 3.1. プログラムの特徴および構成

サーバー・プログラムはユーザーの要求を受けて処理し、クライアントにそれに対する応答を返すプログラムです。

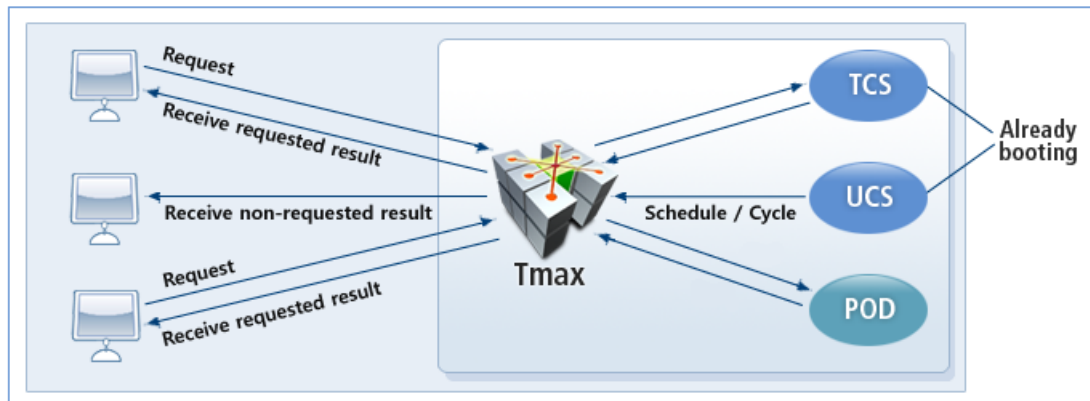
以下は、Tmaxサーバー・プログラムの特徴です。

- 開発者はサービス・ルーチンのみを開発すればいいので開発が簡単であり、期間を短縮することができます。
- サーバーは1つ以上のサービスで構成され、サーバー自身がクライアントになって他のサーバーにサービスを要求することができます。クライアント情報と結果をすべて渡して、クライアントの管理を任せることもできます。
- Tmax関数が通信またはデータ変換を提供するので、開発者はUNIX内部システムとプロトコルを作成する必要はありません。また、ネットワーク・プログラミングについての深い知識がなくてもプログラムを開発することができます。
- XAモードの場合、Tmaxが管理するルーチンでデータベースの接続と解除を管理するので、開発者がデータベースの接続と管理を行う必要はありません。

Tmaxアプリケーション・サーバー・プログラムは、Tmaxが提供するmain()と開発者が作成した業務処理サービス・ルーチンで構成されます。サーバー・プログラムは、アプリケーション・クライアント・プログラムのようにTmax関数を使用して作成されます。この関数は、ライブラリー(libsvr.a、libsvr.so)として提供され、サーバー・プログラムと一緒にコンパイルされます。サーバー・プログラムはTmaxがサポートするプロセス・タイプに応じて構成と開発方法が異なります。

以下は、Tmaxがサポートするサーバー・プロセスのタイプです。

【図 3.1】 Tmaxサーバー・プロセスのタイプ



- TCS(Tmax Control Server)

クライアントの要求に従ってプロセスを処理します。

- UCS(User Control Server)

TCSが受動的な方法であるに対して、UCSは能動的なプロセス処理方法です。クライアントの要求がなくとも、クライアントに継続してデータを送信します。

- RDP(Realtime Data Processor)

特定の用途のためにUCSタイプのプロセスを改善した方法です。

### 3.1.1. TCS

TCS方式のサーバー・プログラムは、以下のように構成されています。

- サーバー・プログラム

開発者が作成したサービス・ルーチンであり、クライアントの要求を処理します。SQL文を使用した場合は、該当するデータベース・ベンダーが提供するツールを使用してプリコンパイルを完了する必要があります。

- Tmaxサーバー・ライブラリー(libsvr.a / libsvr.so)

Tmaxが提供するサーバー・ライブラリーであり、サーバーmain()とtpsvrinit()、tpsvrdone()および各種Tmax関数が含まれています。

- サービス・テーブル

サービス・テーブルは、サーバーごとに提供されるサービス名が記述されているファイルであり、Tmax環境ファイルを参照して作成します。サービス・テーブルは、サービスを実行するとき、実際にサーバー内で該当するサービス・ルーチンを検索するために使われており、システム管理者が提供します。

以下は、サービス・テーブルの作成方法です。

1. バイナリのTmax環境ファイル(例: tmconfig)を参照してgst(generate service table)コマンドを使用します。

```
gst [-f binary configuration file]
```

gstを実行すると、Tmaxディレクトリー下のsvctディレクトリーにTmax環境ファイルのSERVERセクションに登録されているサーバーごとに「サーバー名\_svctab.c」というサービス・テーブルが作成されます。これはSERVICEセクションを参照してサーバー別に提供するサービスが登録されています。

2. システム管理者が作成したシステム全体の構成に関するテキスト・ファイル(sample.m)をcflコマンドを使ってコンパイルした後、バイナリのTmax環境ファイル(tmconfig)が作成されます。

```
$cfl [-i Tmax 環境ファイル]
```

---

## 参考

cflおよびサービス・テーブルの詳細については、『Tmax 運用ガイド』を参照してください。

---

- 構造体バイナリ・テーブル(SDLFILE)

構造体型バッファー(STRUCT、X\_C\_TYPE、X\_COMMON)を使用した場合、このバッファーを定義する<xxxx.s>形式の構造体ファイルが必要です。構造体ファイルには、標準通信タイプ(構造体ファイル名\_sdl.c)と構造体ヘッダー・ファイル(構造体ファイル名\_sdl.h)があります。

構造体ファイルは、**sdlc -c**コマンドを使用してコンパイルします。コンパイルが完了するとバイナリ・テーブル・ファイルが作成されます。作成されたファイルは、サーバー・プログラムを実行するときに構造体型データを標準通信型に変換および逆変換するために使われます。

- フィールド・バッファー・バイナリ・テーブル(FDLFILE)

フィールド・バッファーを使用した場合、<xxxx.f>形式で定義されたフィールド・バッファー・ファイルが必要です。

フィールド・バッファー・ファイルは**fdlc**コマンドを使用してコンパイルします。コンパイルが完了すると、1) フィールド・キーとデータを対応させるバイナリ・テーブル・ファイルと、2) フィールド・キーとフィールド・キーの名前を対応させるヘッダー・ファイル(xxxx\_fdl.h)の、2種類のファイルが作成されます。既存の構造体ファイルと違って、ユーザーが必要とするフィールドの値のみを操作および転送できるので、多様なタイプのデータを使用する際、リソースの浪費を低減することができます。ただし、データ値とフィールド・キー値を一緒に管理するため、多量のフィールドを使用しないと、かえって逆効果を生んでしまうこともあります。

- 構造体-標準バッファーの変換および逆変換プログラム

サーバー・プログラム内で構造体バッファを使用するには、sdlcコマンドにより作成された構造体-標準バッファの変換および逆変換プログラム(xxxx\_sdl.c, xxxx\_sdl.h)をコンパイルして一緒にリンクする必要があります。構造体を使用しない場合は、TMAXDIR/lib/sdl.oをリンクしてください。

## 3.1.2. UCS

UCS方式のサーバー・プログラムはTmaxで管理するmain()ルーチンとサービス・ルーチン、usermain()ルーチンで構成されます。

- main()ルーチン

データベースの接続および解除、コマンド・ラインのオプション処理などを行います。

- サービス・ルーチン

実際にクライアントの要求を受けて業務を処理します。

以下は、サービス・ルーチンを宣言する方法です。

```
<サービス名>(TPSVCINFO *msg)
```

項目	説明
サービス名	NULL終了文字を含めて63字以内にします。アンダーライン(_)で始まる名前はTmaxで内部的に使用するので使用しないでください
msg	サービス・ルーチンはTPSVCINFO構造体をパラメーターとして受けて当該作業を処理します

- usermain()ルーチン

usermain()ルーチンは、サービス要求または制御メッセージのような特別なメッセージがない間、独自の作業を実行します。一定の時点にスケジューリングAPIを通じてTMMおよびCLHからの制御コマンドやサービス実行コマンドを処理します。usermain()が終了すると、該当するサーバー・プロセスも終了するので、一般に無限ループの形で作成されます。クライアントのサービス要求やTMMおよびCLH制御メッセージを処理するために、この無限ループ内には必ず1つ以上のスケジューリングAPIが使われなければなりません。

usermain()はコマンド・ライン・パラメーターを受け取ることができ、この値はTmaxシステム環境ファイルのSERVERセクションのCLOPT項目に設定された値として、tpsvrinit()に渡される値と同じです。usermain()が終了すると、main()ルーチンはtpsvrdone()ルーチンを実行して一緒に終了されます。

```
int usermain(int argc, char *argv[])
{
    .....
    while(1) {
        .....
```



```
        tpschedule(-1);
        .....
    }
    return 0;
}
```

UCS方式のサーバー・プログラムの場合、サービス・ルーチンは不要です。この場合、サーバー・プロセスはデーモン・プロセスと類似した特性を持ちます。

以下は、TOUPPERのサービス・ルーチンを宣言する例です。

```
TOUPPER(TPSVCINFO *msg)
{
    .....
}
```

---

#### 参考

SERVERセクションの詳細については、『Tmax 運用ガイド』を参照してください。

---

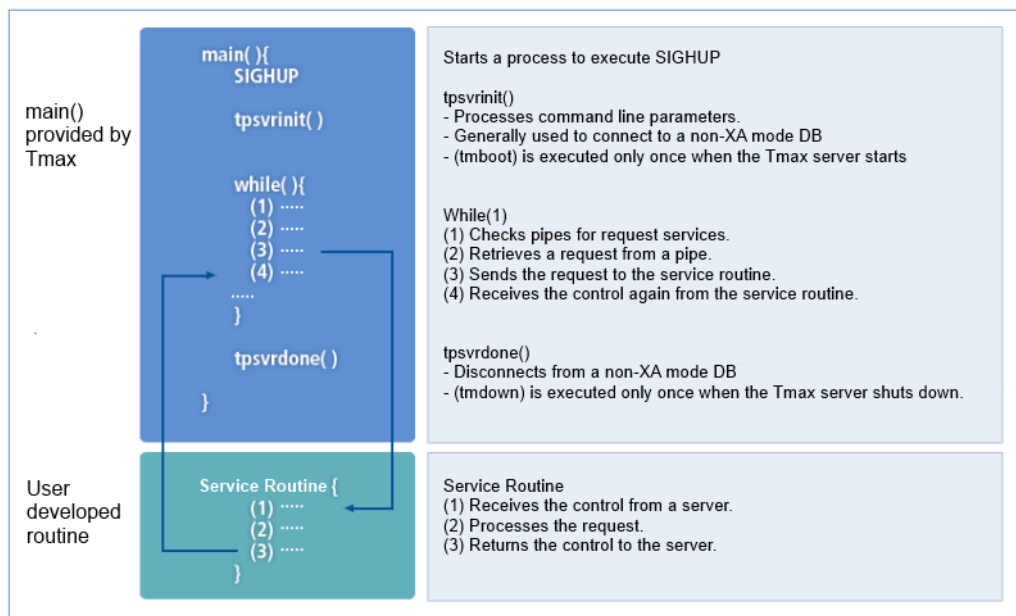
## 3.2. 開発環境およびツール

Tmaxアプリケーション・サーバー・プログラムのOS環境は、ハードウェアを問わずすべてのUNIXマシンに対応しており、UNIX標準エディターのviで開発することができます。

## 3.3. プログラムのフロー

Tmaxサーバー・プログラムは、TCS方式とUCS方式によって異なるフローで実行されます。一般的なサーバー・プログラムのTCS方式は、サービス・ルーチン内ですべての処理が行われます。一方、UCS方式のサーバー・プログラムは、サービス・ルーチンとusermain()を適切に使用し、以前は実現できなかった多様な方式のサービスを提供することができます。サーバー・プログラムには、Tmaxが提供するmain()ルーチンと、開発者が開発するサービス・ルーチンがあります。

【図 3.2】サーバー・プログラムのフロー



#### ● main()ルーチン

main()ルーチンは、コマンド・ラインに入力されたパラメーターの処理、接続されたデータベースの接続および解除処理を行います。また、サービス・ルーチンで使用するバッファを割り当てたり、クライアント要求を処理するサービス・ルーチン呼び出ししたりします。

#### ● サービス・ルーチン

サービス・ルーチンはクライアントの要求を処理します。クライアントの要求は、まずmain()でのサービス要求に関連する情報を持っているTPSVCINFO構造体から受けます。クライアントが要求するバッファの内容はTPSVCINFO→DATAにあります。サービス・ルーチンではTPSVCINFO→DATAからデータを受信して要求を処理した後、その結果をクライアントに返すか、また別の処理をするため、他のサーバーにサービス処理を要求します。

#### 参考

サービス・ルーチンで使用する関数の詳細については、『Tmax リファレンスガイド』および『Tmax プログラミングガイド(UCS)』を参照してください。

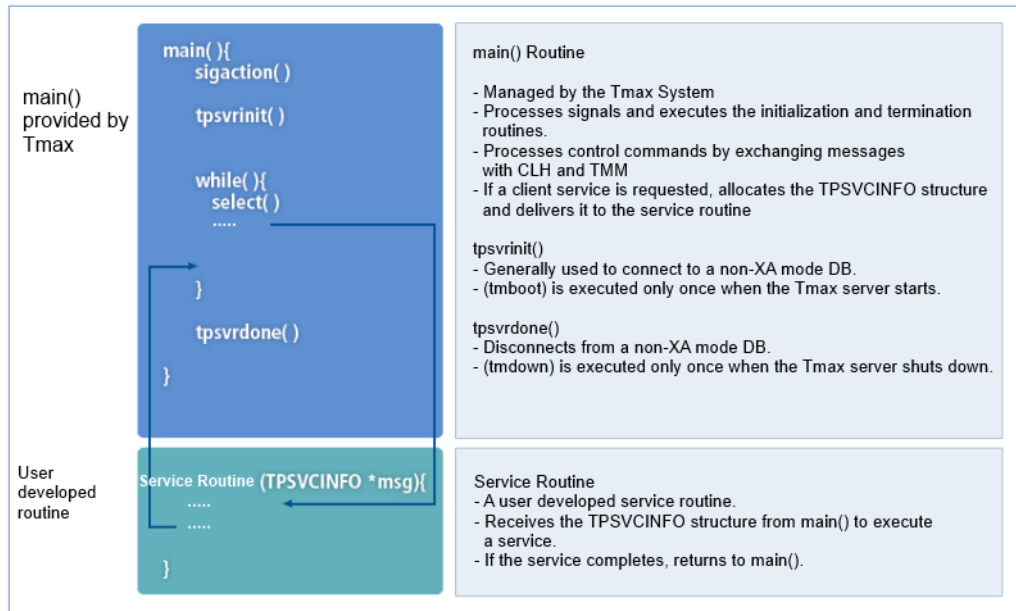
### 3.3.1. TCS

TCSプロセスは、一般的な3階層の概念を持つサーバー・プロセスです。Tmaxシステムは要求した結果を受信し、それをクライアントに返す適切なプロセスにクライアントの要求をスケジューリングします。Tmaxは、TCS方式のサービス・プログラムのためのmain()関数を提供しているため、開発者はサービス・ルーチンの

開発のみを行います。TCS方式のプログラムは、指定されたサービス名を関数にして開発することになります。

TCS方式のmain()ルーチンとサービス・ルーチンは以下のとおりです。

**[図 3.3] TCS方式のサーバー・プログラムのフロー**



#### ● main()ルーチン

Tmaxが管理するmain()ルーチンは、コマンド・ラインに入力されたパラメーターの処理のほか、必要に応じてデータベースの接続と解除処理およびサービス・ルーチンで使用するTPSVCINFO構造体バッファの割り当てや管理を行います。TMMおよびCLHからのメッセージを待機し、サービス要求が受信されたら、データを取得してサービス・ルーチンを呼び出し、実際のサービスが実行されるようにします。

#### ● サービス・ルーチン

サービス・ルーチンはクライアントの要求を処理します。main()ではクライアントから受信したサービス要求を多様な情報と一緒にTPSVCINFO構造体に格納してサービス・ルーチンに渡し、サービス・ルーチンはサービス実行の結果をクライアントに渡します。

以下は、TCS方式のプログラムの例です。

```

SDLTOUPPER( TPSVCINFO *msg )
{
    ...
    struct kstrdata *stdata;
    stdata = (struct kstrdata *)msg->data;
    ...
    for ( i = 0; i < stdata->len; i++)

```

```

        stdata->sdata[i] = toupper(stdata->sdata[i]);
        ...
        tpreturn(TPSUCCESS,0,(char *)stdata, sizeof(struct kstrdata), TPNOFLAGS);
    }

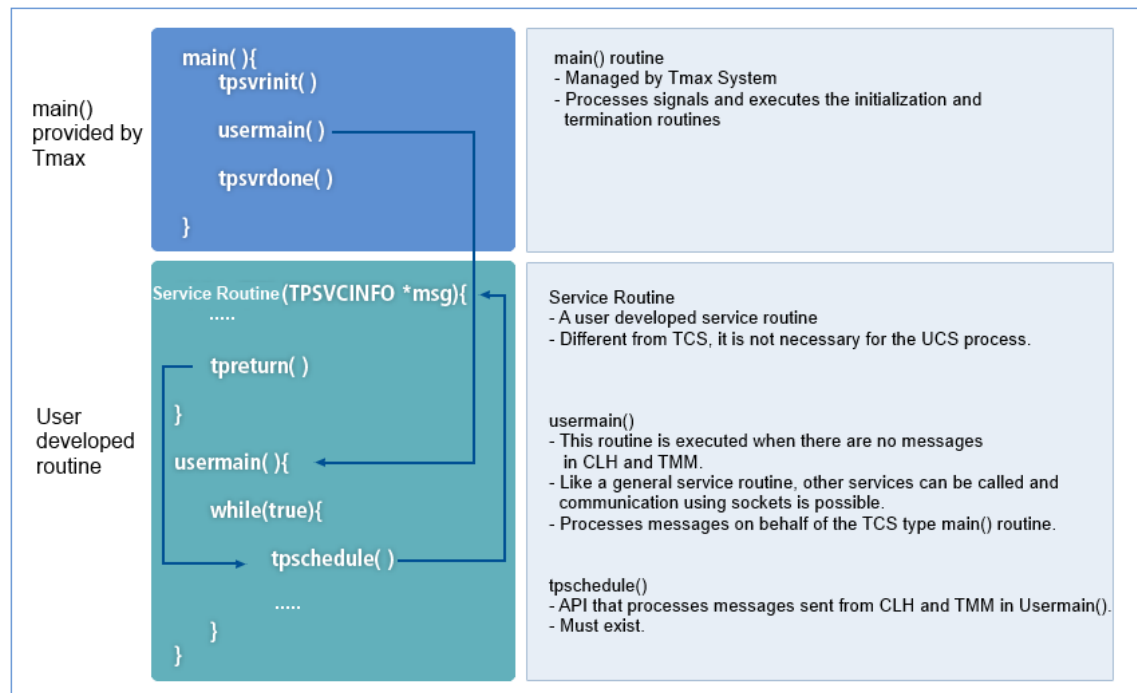
```

### 3.3.2. UCS

典型的なTCS方式のプロセスは、クライアントからの要求があるときだけその要求を処理しますが、UCS方式はクライアントの要求がなくても接続してサービスを提供します。UCS方式は、迅速にサービスを提供することが可能であり、かつTCS方式をサポートしているため、より多くの機能を提供することができます。UCS方式のプログラムは、`usermain()`とサービス・ルーチンで構成されています。

UCS方式の`main()`ルーチンとサービス・ルーチンは以下のとおりです。

**[図 3.4] UCS方式のサーバー・プログラムのフロー**



- **main()ルーチン**

UCS方式の`main()`ルーチンはTCS方式と類似していますが、それに加え、`usermain()`ルーチンも持っています。

- **サービス・ルーチン**

サービス・ルーチンはTCSと同様であり、UCS方式のプロセスはUCSライブラリー(`libsvrucs.a`)とリンクされている必要があります。

- **usermain()ルーチン**

UCS方式のusermain()ルーチンは、他のコマンドのない間、独自で繰り返し作業を実行しており、スケジューリングAPIを使用してCLHまたはTMMから送られるコマンドを処理します。UCS方式のプロセスがtmdownコマンドを使用して正常に終了するためには、ループされる間、**tpschedule()**を使用する必要があります。

以下は、usermain()関数の例です。

```
...
int usermain(int argc, char *argv[])
{
    ...
    while (1) /* 無限ループ */
    {
        clid = tpgetcli();
        ret = tpsendtocli(clid, sndbuf, slen, TPNOFLAGS);
        if (ret < 0)
        {
            error processing routine
        }
        ...
        tpschedule(10); /* UCS方式のプロセスはwhileループにこの関数を追加する必要が
あります。*/
        /* tmdownを呼び出すと、イベントをここに送ります。*/
        ...
    } /* end of while */
}
```

– 基本的にusermain()はループ中の作業を処理します。usermain()は特定のクライアントにデータを送るためにtpsendtocli()を使用し、TCS方式のサービスを処理するためにtpschedule()を使用します。

– tpschedule()はUCS方式のサーバー・プロセスでのみ使用される関数です。

タイムアウト・パラメーターを持つ関数はタイムアウト(秒)の間データを待機し、データが受信されたらそれを直ちに返します。ただし、その時間内にデータが受信されないと、タイムアウトが適用されます(単位: 秒)

パラメーターが「-1」に設定されている場合は、返されたデータの有無をチェックします。チェックの結果、返されたデータがない場合は、次の段階に返します。パラメーターが「0」に設定されていると、クライアントの要求を受信するまで待機します。

tpschedule()関数は、データを受信したら該当するサービスが自動実行された後に返されます。そのため、データを受信した後、任意でサービスを実行してはなりません。サービスは常にシステムによって実行されるので、UCS方式のサービス・プログラムにおいても、この点に注意する必要があります。

– usermain()から非同期型通信(tpacall)でサービスを呼び出す場合、それに対する応答はtpregcb()関数で受信します。

以下は、UCS方式のプロセスを使用するためのTmax環境ファイルの例です。

```
*DOMAIN
res1      SHMKEY = 66999, MAXUSER = 256

*NODE
tmax      TMAXDIR = "/user/tmax",
          APPDIR  = "/user/tmax/appbin",
          PATHDIR = "/user/tmax/path",
          TLOGDIR = "/user/tmax/log/tlog",
          ULOGDIR = "/user/tmax/log/ulog",
          SLOGDIR = "/user/tmax/log/slog"

*SVRGROUP
svg1      NODENAME = tmax

*SERVER
svr1      SVGNAME = svg1

#UCS方式のプロセスは、SVRTYPE = UCSという項目を追加する必要があります。
ucssvr1   SVGNAME = svg1, SVRTYPE = UCS
ucssvr2   SVGNAME = svg1, SVRTYPE = UCS
ucssvr3   SVGNAME = svg1, SVRTYPE = UCS

*SERVICE
TOUPPER   SVRNAME = svr1
TOLOWER   SVRNAME = svr1
DUMMY1    SVRNAME = ucssvr1
DUMMY2    SVRNAME = ucssvr2
```

以下は、UCS方式のプログラムの例です。

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

DUMMY1(TPSVCINFO *msg)
{
    printf("svc start!");
    tpreturn(TPSUCCESS, 0, (char *)msg->data, 0, TPNOFLAGS);
}

int usermain(int argc, char *argv[])
{
    ...
    while (1) {
```

```

        jobs = tpschedule(-1);
        cnt++;
        ...
        ret = tpcall("TOUPPER", sbuf, 0, &rbuf, &rlen, TPNOFLAGS);
        if (ret < 0) {
            error processing routine
        }
    }
}

```

## RDP

RDPとは、変化し続けるデータをクライアントに迅速かつ有効に転送するため、UCS方式のプロセスをカーネル・レベルで改善したサーバー・プロセスです。少量のデータを多数のクライアントに頻繁に送る必要があるとき(1秒当たり10回以上)、プロセス占有率や処理速度の面で優れた性能を発揮します。1つのノードには1つのRDP方式のサーバーのみを使用できます。RDPは形式的にはUCS方式と同様ですが、環境ファイル設定とコンパイル時のライブラリーが異なります。

以下は、RDP方式のプロセスを使用するためのTmax環境ファイルの例です。

```

*DOMAIN
tmax1  SHMKEY = 7090, MINCLH = 2, MAXCLH = 2

*NODE
tmax1  TMAXDIR = "/home/navis/tmax",
        APPDIR = "/home/navis/tmax/appbin",
        PATHDIR = "/home/navis/tmax/path",
        TLOGDIR = "/home/navis/tmax/log/tlog",
        ULOGDIR = "/home/navis/tmax/log/ulog",
        SLOGDIR = "/home/navis/tmax/log/slog",
        REALSVR = "real", rscpc = 2

```

---

### 参考

RDPについての詳しい説明は、『Tmax 運用ガイド』および『Tmax プログラミングガイド(UCS)』を参照してください。

---

## 3.4. プログラム・コンパイル

開発者がサーバー・プログラムの作成を完了すると、Tmaxがそれをコンパイルして実行ファイルを作成します。サーバー・プログラムをコンパイルするには、開発者が作成したサーバー・プログラム、Tmaxサーバー・ライブラリー、サービス・テーブルなどが必要です。構造体バッファを使用した場合は、構造体-標準バッファの変換および逆変換プログラムと構造体バイナリ・テーブルが必要です。フィールド・バッファを使用

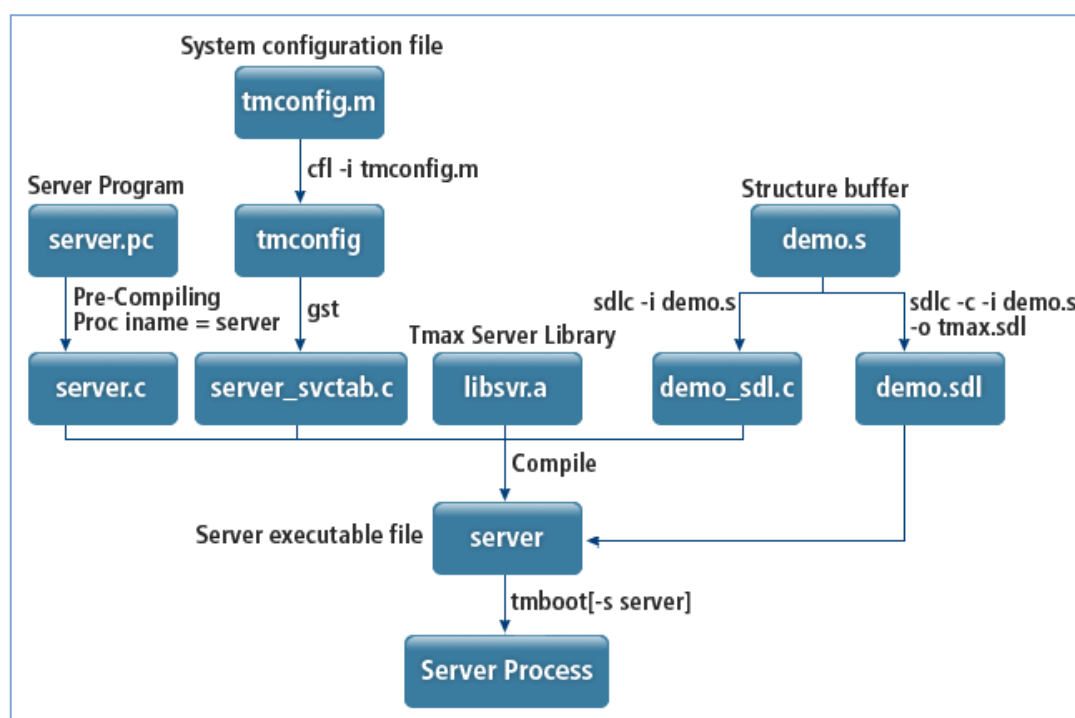
した場合は、フィールド・ヘッダー・ファイルとフィールド・バッファー・バイナリ・テーブルが必要です。サーバー・プログラムのコンパイル方法は、TCS方式とUCS方式に分けて説明します。

### 3.4.1. TCS

一般的なCコンパイラを使用するコンパイル方法です。Tmaxをインストールすると生成されるシェル・プログラムを使用するか、またはmksvrユーティリティを使用して簡単にコンパイルできます。

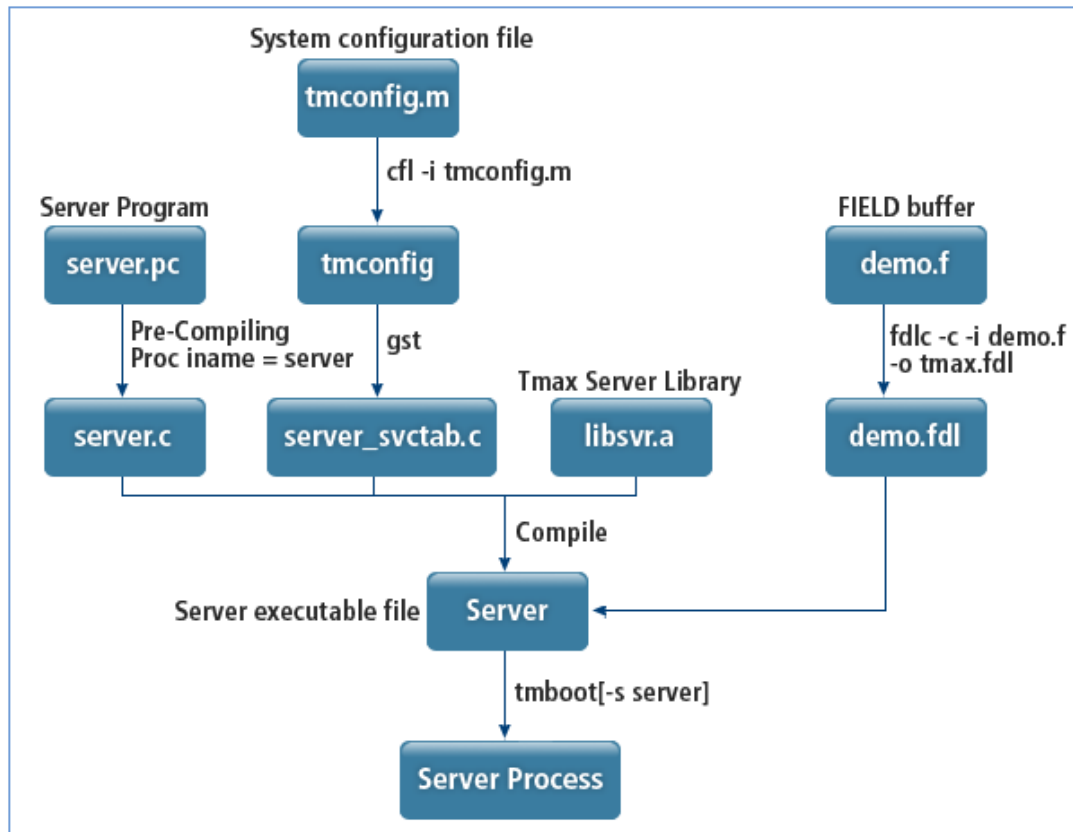
以下の図は、構造体バッファーとフィールド・バッファーを使用するサーバー・プログラムの作成順を示しています。

[図 3.5] サーバー・プログラムのコンパイル(構造体バッファー)





【図 3.6】 サーバー・プログラムのコンパイル(フィールド・バッファ)



以下は、Linuxでの実行例です。開発環境(32ビット/64ビット)とプラットフォームに応じて使われるフラグとライブラリーが若干異なります。

1. サーバー・プログラムをコンパイルしてオブジェクト・ファイルを作成します。

サーバー・プログラムはTmaxが提供するヘッダー・ファイルを参照します。必要に応じて構造体ファイルやフィールド・バッファ・ヘッダー・ファイルも参照します。SQL文を使用した場合、まず該当するデータベース・ベンダーが提供するツールを使用してプリコンパイルを完了する必要があります。

```
$cc -c -I/home/tmax/usrinc app.c → app.o
```

2. 構造体通信の場合、構造体ファイルをコンパイルします。

この段階は、1)sdlcを使用して構造体・標準バッファの変換および逆変換プログラムを作成する段階と、2)作成したプログラムをオブジェクト・ファイルに再コンパイルする段階の2段階で行われます。構造体ファイルを使用しないと、**TMAXDIR/lib/sdl.o**を使用します。

```
$sdlc -i demo.s -> demo_sdl.c
$cc -c -I/home/tmax/usrinc demo_sdl.c → demo_sdl.o
```

3. システム管理者が提供したサービス・テーブルをコンパイルしてオブジェクト・ファイルを作成します。

```
$cc -c app_svctab.c → app_svctab.o
```

4. 上記により作成されたオブジェクト・ファイルとTmaxシステムが提供するサーバー・ライブラリーを一緒にリンクしてサーバー実行プログラムを作成します。

```
$cc -o app app.o demo_sdl.o app_svctab.o libsvr.a libnodb.a → aptest
```

## シェル・プログラムを使用したコンパイル

Tmaxをインストールすると、例題サーバー・プログラム・ディレクトリーには基本的に例題プログラムのほか、4つのメイクファイルとそれを使用するシェル・プログラムがインストールされます。(Makefile.c、Makfile.sdl、Makfile.pc、Makefile.psdl、compile)

以下は、構造体バッファを使用するが、データベースは使用しないTmaxサーバー・プログラムのメイクファイル(Makefile.sdl)の例です。\$COMP\_TARGETには開発したプログラム名が含まれ、シェル・プログラムのコンパイルを使用して渡します。

```
# Server makefile

TARGET = $(COMP_TARGET)
APOBJS = $(TARGET).o
SDLFILE = demo.s

#Not use Db
LIBS = -lsvr -lnodb
#Solarisの場合-lsocket -lnslがインストールされます。
#Oracleの場合-lnodbの代わりに-loras、Informixの場合は-linfs
#Db2の場合は-ldb2, Sybaseの場合は-lsybsが挿入されます。

OBJS = $(APOBJS) $(SDLOBJ) $(SVCTOBJ)

SDLOBJ = ${SDLFILE:.s=_sdl.o}
SDLC = ${SDLFILE:.s=_sdl.c}
SVCTOBJ = $(TARGET)_svctab.o

CFLAGS = -O -I$(TMAXDIR)
#CFLAGは使用環境(32ビット/64ビット)とプラットフォームに応じて異なります。
#Solaris 32bit, Compaq, Linux: CFLAGS = -O -I$(TMAXDIR)
#Solaris 64bit: CFLAGS = -xarch=v9 -O -I$(TMAXDIR)
#HP 32bit: CFLAGS = -Ae -O -I$(TMAXDIR)
#HP 64bit: CFLAGS = -Ae +DA2.0W +DD64 +DS2.0 -O -I$(TMAXDIR)
#IBM 32bit: CFLAGS = -q32 -brtl -O -I$(TMAXDIR)
#IBM 64bit: CFLAGS = -q64 -brtl -O -I$(TMAXDIR)

APPDIR = $(TMAXDIR)/appbin
SVCTDIR = $(TMAXDIR)/svct
LIBDIR = $(TMAXDIR)/lib
#64ビット環境を使用する場合、$(TMADIR)/lib64になります。
```

```

#
.SUFFIXES : .c

.c.o:
    $(CC) $(CFLAGS) -c $<

#
# server compile
#

$(TARGET): $(OBJS)
    $(CC) $(CFLAGS) -L$(LIBDIR) -o $(TARGET) $(OBJS) $(LIBS)
    mv $(TARGET) $(APPDIR)/.
    rm -f $(OBJS)

$(APOBJS): $(TARGET).c
    $(CC) $(CFLAGS) -c $(TARGET).c

$(SVCTOBJ):
    touch $(SVCTDIR)/$(TARGET)_svctab.c
    $(CC) $(CFLAGS) -c $(SVCTDIR)/$(TARGET)_svctab.c

$(SDLOBJ):
$(TMAXDIR)/bin/sdlc -i ../sdl/$(SDLFILE)
    $(CC) $(CFLAGS) -c ../sdl/$(SDLC)

#
clean:
    -rm -f *.o core $(TARGET)

```

各メイクファイルの使用法は以下のとおりです。svr1、svr2、svr3、fdltest、sdltestはすべてTmaxのインストール時に提供されるサンプル・プログラムです。詳細については、該当するメイクファイルを参照してください。

```

$./compile sdl svr1
$./compile c svr2
$./compile c svr3
$./compile pc fdltest
$./compile psdl sdltest

```

## mksvrユーティリティーを使用したコンパイル

mksvrユーティリティーを使用する場合、Tmaxシステム環境ファイルの作成時に、予めサービス名を指定する必要がなく、gstを使用したサービス・テーブルの作成も不要です。一方、構造体を使用する場合は、構造体-標準バッファーの変換および逆変換プログラムが事前に作成されている必要があります。また、データ

ベースを使用する場合は、プリコンパイル済みのファイルを使用するか、あるいはRMファイルを指定しなければなりません。

```
$cfl -i sample.m

$sdlc -i demo.s
$sdlc -c -i demo.s -o tmax.sdl
$mksvr -s SDLToupper,SDLTOlower -o svr1 -f svr1.c -S ../sdl/demo_sdl.c

$mksvr -s TOUPPER,TOLOWER -o svr2 -f svr2.c

$fdlc -c -i demo.s -o tmax.fdl
$mksvr -s FDLTOupper,FDLTOlower -o svr3 -f svr3.c

$proc iname=fdltest include $TMAXDIR
$mksvr -s FDLINS,FDLSEL,FDLDEL,FDLUPT -o fdltest -f fdltest.c

$mksvr -s SDLINS,SDLSEL,SDLDEL,SDLUPT -o sdltest -f sdltest.c
-S ../sdl/demo_sdl.c -r ORACLE
```

---

#### 参考

mksvrの詳細については、『Tmax リファレンスガイド』を参照してください。

---

## 3.4.2. UCS

UCSサーバー・プログラムのコンパイルは、TCSサーバー・プログラムと類似しています。準備事項とコンパイル順は同様であり、サーバー・ライブラリーはlibsvr.a(またはlibsvr.so)ではなく、libsvrucs.a(またはlibsvrucs.so)を使用します。一般的なCコンパイラを使用したコンパイル方法でシェル・プログラムを使用するか、mksvrユーティリティーを利用すると簡単にコンパイルできます。

以下は、Linuxでの実行例です。開発環境(32ビット/64ビット)とプラットフォームに応じて使われるフラグとライブラリーが若干異なります。使用するライブラリー以外はTCS方式と同様なので、詳細については、「[3.4.1. TCS](#)」を参照してください。

1. サーバー・プログラムをコンパイルしてオブジェクト・ファイルを作成します。

サーバー・プログラムはTmaxが提供するヘッダー・ファイルを参照します。必要に応じて構造体ファイルやフィールド・バッファー・ヘッダー・ファイルも参照します。SQL文を使用した場合、まず該当するデータベース・ベンダーが提供するツールを使用してプリコンパイルを完了する必要があります。

```
$cc -c -I/home/tmax/usrinc app.c → app.o
```

2. 構造体通信の場合、構造体ファイルをコンパイルします。

この段階は、1)sdlcを使用して構造体-標準バッファの変換および逆変換プログラムを作成する段階と、2)作成したプログラムをオブジェクト・ファイルに再コンパイルする段階の2段階で行われます。構造体ファイルを使用しない場合は、この過程を実行せずにTMAXDIR/libディレクトリー内の<sdl.o>を使用して一緒にコンパイルします。

```
$sdlc -i demo.s -> demo_sdl.c
$cc -c -I/home/tmax/usrinc demo_sdl.c -> demo_sdl.o
```

3. システム管理者が提供するサービス・テーブルをコンパイルしてオブジェクト・ファイルを作成します。

以下は、データベースを使用しない場合の例です。データベースを使用する場合はデータベースに応じてliboras.so(a)、libinfs.so(a)、libdb2.so(a)、libsybs.so(a)を使用します。シェル・プログラムを使用する場合は、該当するメイクファイルの-lsvrを-lsvrucsに変更します。使用法は同じです。

```
$cc -c app_svctab.c -> app_svctab.o
```

## mksvrユーティリティーを使用したコンパイル

mksvrユーティリティーを使用する場合、Tmaxシステム環境ファイルの作成時に、予めサービス名を指定する必要がなく、gstを使用したサービス・テーブルの作成も不要です。一方、構造体を使用する場合は、構造体-標準バッファの変換および逆変換プログラムが事前に作成されている必要があります。また、データベースを使用する場合は、プリコンパイル済みのファイルを使用するか、あるいはRMファイルを指定しなければなりません。

```
$cfl -i sample.m
$sdlc -i demo.s
$sdlc -c -i demo.s -o tmax.sdl
$mksvr -s UCSSAMPLE -o ucs_svr -f ucs_svr.c -S ../sdl/demo_sdl.c -t UCS
```

---

### 参考

mksvrの詳細については、『Tmax リファレンスガイド』を参照してください。

---

## 3.5. プロセス作成および終了

サーバー・プロセスの作成は管理者の仕事です。サーバー・プロセスの作成時にはTmax関連の環境を参照するので、Tmaxアプリケーション・サーバーは、UNIX実行ファイルのように単独では実行されません。これを踏まえ、**tmboot**コマンドを使用してサーバー・プロセスを作成し、**tmdown**コマンドを使って終了させます。

Tmaxアプリケーション・サーバー・プロセスを作成する前に、システム管理者が作成したTmax環境ファイル(例: sample.m)がコンパイル(cflコマンド利用)されている必要があります。コンパイルされたバイナリ環境ファイル(例: tmconfig)を参照してサーバー・プロセスを作成します。

以下は、プロセスの作成および終了のために使用するコマンドです。

- プロセスの作成

- Tmaxシステム・プロセスとバイナリのTmax環境ファイルに登録されたすべてのサーバー・プロセスを作成します。

```
$tmboot [-f バイナリの構成ファイル]
```

- 特定のサーバー・プロセスのみを作成します。

```
$tmboot [-s サーバー・プログラム名]
```

- プロセスの終了

- Tmaxシステム・プロセスとバイナリのTmax環境ファイルに登録されたすべてのサーバー・プロセスを終了させます。

```
$tmdown [-f バイナリの構成ファイル]
```

- 特定のサーバー・プロセスのみを終了します。

```
$tmdown [-s サーバー・プログラム名]
```

---

## 参考

コマンドの詳細については、『Tmax リファレンスガイド』を参照してください。

---

## 第4章 通信タイプ

本章では、Tmaxがサポートする通信タイプについて説明します。

### 4.1. 概要

Tmaxシステムにおける応用サーバーとクライアント間の通信には、**通信型通信**、**非同期型通信**、**会話型通信**の3タイプがあります。

- 同期/非同期型通信

- 一般的には同期型通信を使用します。クライアントは1回に1つの要求を送って応答を受けます。サーバーは、要求の処理時、1件ずつ処理を完了した後、他の要求を受けます。
- `tpcall()`と`tpacall()`でサービス要求をします。データを送る前に`tpalloc()`で送受信バッファを割り当てる必要があります。サーバーの業務処理サービス・ルーチンは`tpreturn()`または`tpforward()`によって終了されます。
- 同期型通信はクライアントが`tpcall()`を呼び出してサーバーに要求を送り、サーバーからの応答が来るまでブロッキングされて待機します。

非同期型通信はクライアントが`tpacall()`を呼び出してサーバーを要求を送った後、サーバーからの応答が来るまで待機せずに他の業務を処理します。応答を受ける場合、`tpgetrply()`を呼び出してサーバーから応答を受けます。

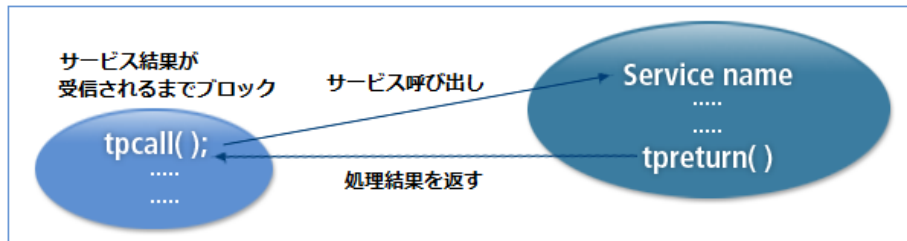
- 会話型通信

- サーバーは同期/非同期型通信とちがって、必ず会話型サービス関数のみを使用してプログラムを開発しなければなりません。会話型モードは通常、接続維持時間が長いため、使用する回数は多くありません。
- `tpconnect()`でサービス要求を初期化します。データを送受信するためには、`tpsend()`と`tprecv()`を利用します。データを送る前に`tpalloc()`で送受信バッファを割り当てる必要があります。
- 会話型通信では`tpforward()`を使用することができません。

## 4.2. 同期型通信

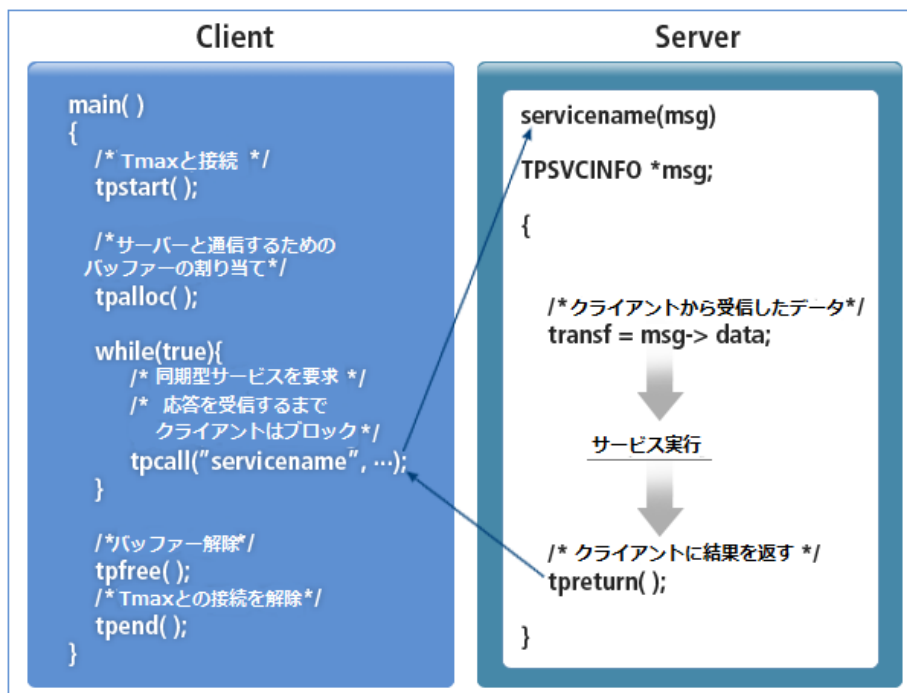
同期型通信はクライアントがサービスを要求し、その応答が来るかタイムアウトになるまで待機します。サービスを要求した後、クライアントはブロッキングされた状態で応答を待ちます。

[図 4.1] 同期型通信



以下の図は同期型通信を使用するクライアント・サーバー・プログラムの動作原理です。

[図 4.2] 同期型通信モデル



### 参考

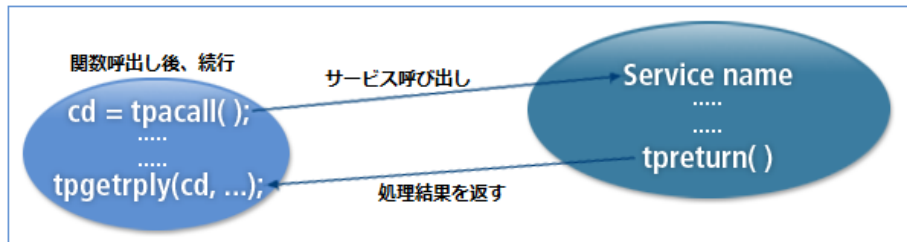
同期型通信関連の関数については、「[9.3. 同期型通信](#)」または『Tmaxリファレンスガイド』を参照してください。



## 4.3. 非同期型通信

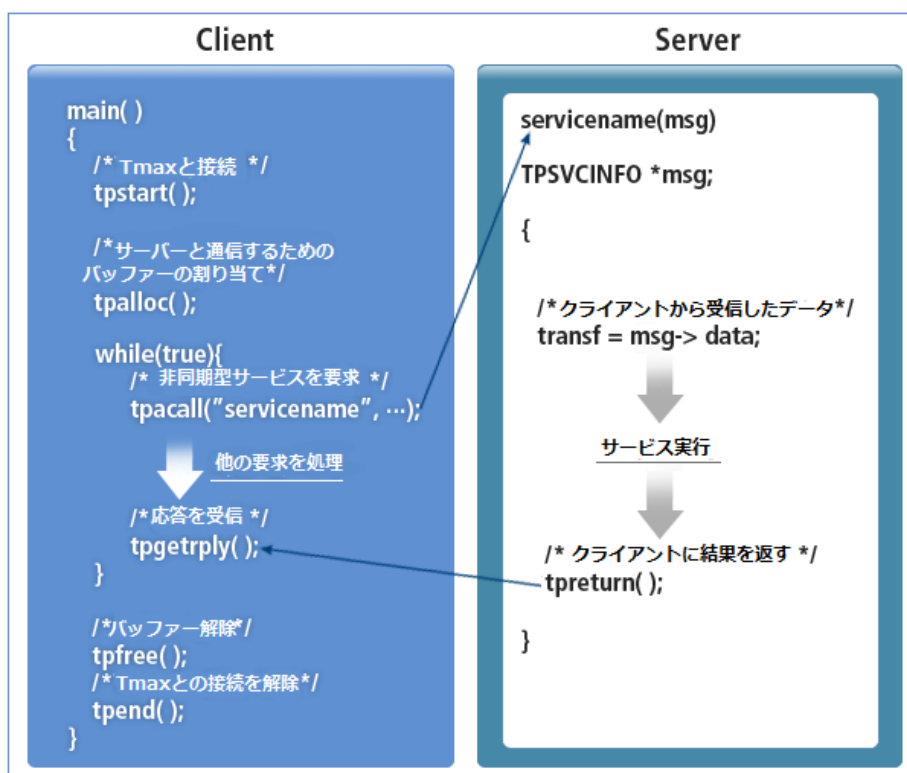
非同期型通信はクライアントがサービスを要求しては応答を待たずに他の業務を続けることができます。また、応答を受けるときに当該要求に対する応答を受信します。サービスを要求するtpacall()は、呼び出すときに即刻返却され、他のサービス要求を処理することができます。しかし、応答を受けるために呼び出すtpgetrply()は応答が来るかタイムアウトになるまでブロッキングされた状態で待機します。

[図 4.3] 非同期型通信



以下の図は非同期型通信を使用するクライアント・サーバー・プログラムの動作原理です。

[図 4.4] 非同期型通信モデル



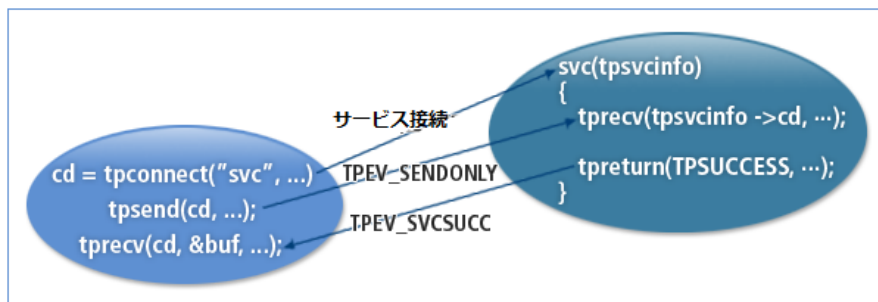
### 参考

非同期型通信関連の関数については、「[9.4. 非同期型通信](#)」または『Tmaxリファレンスガイド』を参照してください。

## 4.4. 会話型通信

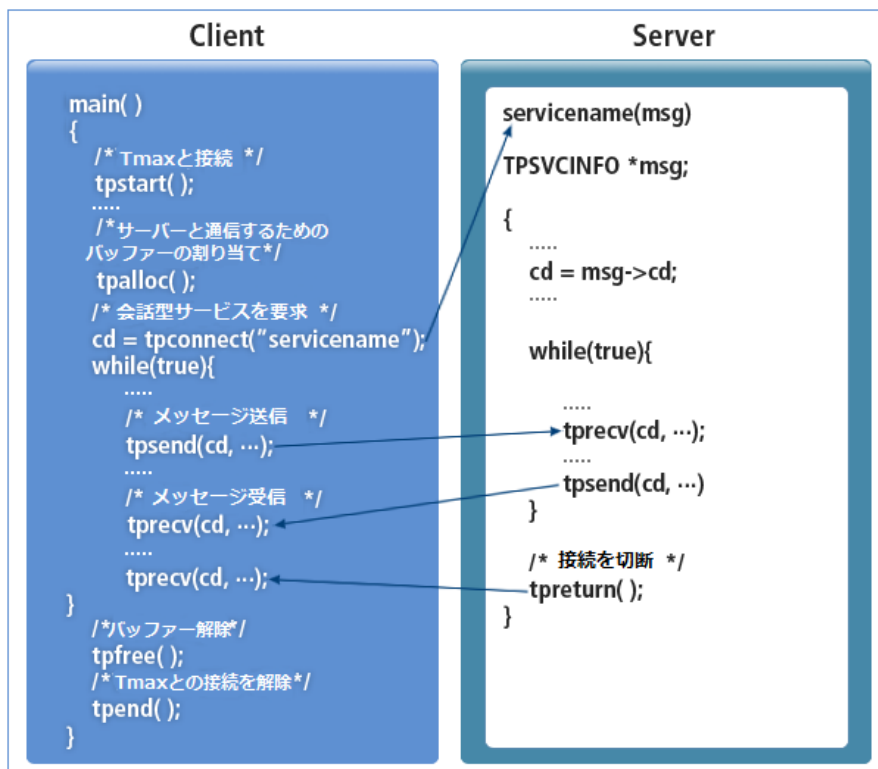
会話型通信はクライアントとサーバー間の半二重(Half Duplex)通信です。最初に会話型モードで接続するときに会話の主導権を設定し、会話の主導権を握る側は送信、そしてその相手は受信のみ可能です。会話の主導権を握っている側はいつでも相手側に主導権を渡すことができるので、メッセージを相互に送受信できます。会話の主導権の制御は送信側のフラグを利用して実行され、受信側はこうした変化をイベント値を使って確認できます。以下は簡単な会話型通信の接続コントロールを渡すことと接続を終了する過程についての説明です。

[図 4.5] 会話型通信



会話型通信はtpconnect()に接続を行った後、tpsend()とtprecv()を呼び出してデータを送受信します。データを送るには、tpconnect()とtpsend()を利用し、フラグの値の設定だけで動く接続コントロールを保持する必要があります。以下の図は会話型通信を使用するクライアント・サーバー・プログラムの動作原理です。

[図 4.6] 会話型通信モデル



- 会話型通信は**tpconnect()**を呼び出すことでスタートします。関数は、当該会話型サービスにデータを送ると同時に、会話の主導権を設定します。tpconnect()が正常に実行されると、それぞれの会話モードを区分するための「接続区切り子(cd)」を返却します。この区切り子を利用してメッセージを送受信できます。
- 会話型通信はサービスのうちコンテキスト(context)によって追加情報が必要な場合に使います。ところが、会話型通信は他の通信に比べて長時間接続を維持しなければなりません。つまり、接続設定(tpconnect())から接続解除(**tpdiscon()**)または**tpreturn()**までは接続が維持されます。こうした長時間の接続維持はネットワークへの負荷を増加させるので、できれば、同期/非同期型通信を使用することを推奨します。
- 一般に、会話型通信はサーバー側が**tpreturn()**を呼び出して完了します。こうなると、相手側はデータと共にTPEV\_SVCSUCCイベントを受信することになります。会話を実行した側がtpdiscon()を呼び出して会話型通信を強制に終了することもできます。ところが、こうした場合、送信中のデータが消滅されることもあるほか、実行中のトランザクションはロールバックされます。
- 会話の主導権は主導権を持っている側のみが相手側に渡すことができます。主導権を渡すときは、**tpsend()**の**TPRECVONLY**フラグを利用します。相手側はデータと共に**TPEV\_SENDONLY**イベントを受信し、主導権を持つことになります。
- コントロールを相手側に渡す場合は、**tpconnect()**または**tpsend()**でflagsパラメータを**TPRECVONLY**に設定して呼び出します。そうすると相手側に**TPEV\_SENDONLY**というイベントが発生してコントロールが自分側に渡されたことが分かります。その後からはtpsend()を呼び出してデータを送ることができます。もちろん、相手側はデータの受信のみ可能です。
- 会話型モードも同期/非同期型通信のようにバッファを使用するので、予め**tpalloc()**を割り当てておきます。つまり、Tmax通信で使用するすべてのバッファはtpalloc()で割り当てられなければなりません。

---

#### 参考

会話型通信関連の関数については、「[9.5. 会話型通信](#)」または『Tmaxリファレンスガイド』を参照してください。

---

### 4.4.1. 会話型通信関連のイベント

会話型通信には以下の5つのイベントが存在します。

イベント	受信関数	説明
TPEV_SENDONLY (0x0020)	tprecv()	接続コントロールが存在する場所を知らせます

イベント	受信関数	説明
TPEV_DISCONIMM (0x0001)	tesend() tprecv() tpreturn()	接続が非規則的に切断したときに受けるイベントです。tpdiscon()が呼び出されるか下位サービスが依然としてオープン状態のまま残っている中で、tpreturn()を呼び出すと発生します
TPEV_SVCERR (0x0002)	tpsend() tprecv()	接続のコントロールを持っていないのにtpreturn()を呼び出すときに発生します。tpreturn()のパラメーターは正しいものの、何らかのエラーが生じたときに発生します
TPEV_SVCFAIL (0x0004)	tpsend() tprecv()	接続のコントロールを持っていない状態でtpreturn()を呼び出すとき、またはtpreturn()でTPFAIL、TPEXITとして設定して呼び出すときに発生します
TPEV_SVCSUCC (0x0008)	tprecv()	正常にサービスを終了するときに発生します。TPSUCCESと設定されてtpreturn()が呼び出されます

#### 参考

関数の詳細については、『Tmaxリファレンスガイド』を参照してください。

# 第5章 バッファ・タイプ

本章ではTmaxがサポートするバッファ・タイプについて説明します。

## 5.1. 概要

多数のサーバーを運用する中、相互異なるハードウェアおよび運用体制間にデータを交換する必要がある場合、プラットフォームごとに異なるメモリー割り当て方式によって問題が発生することがあります。こうした問題を克服するには、すべてのプラットフォームについての詳細な知識と共にデータを転換する複雑な過程が必要です。

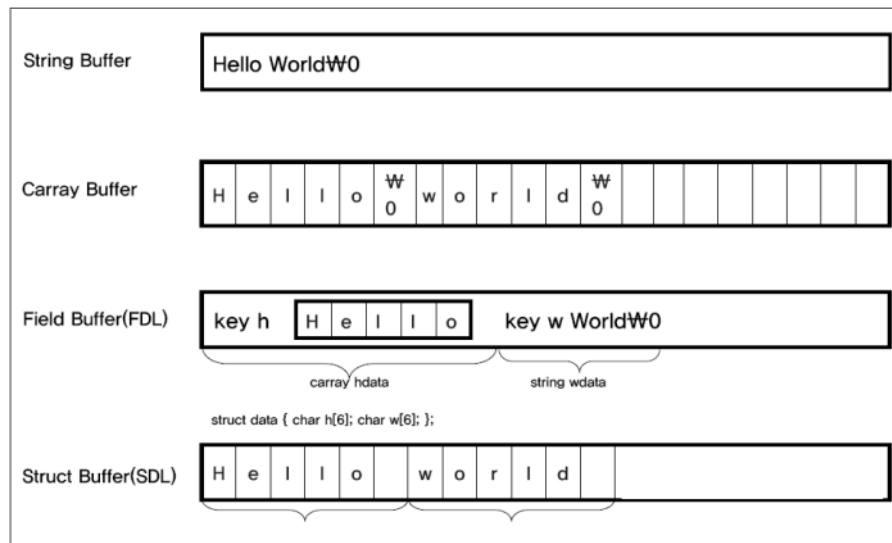
一方、Tmaxは多様なタイプの標準バッファを提供してこうした問題を解決します。また、不要なネットワーク負荷を減らし、開発者の選択の幅を広げ、開発期間を短縮するなどの追加メリットも提供します。Tmaxアプリケーション開発者はTmaxが提供する多様なバッファを使用して以下のような効果を得ることができます。

- アプリケーションは多様なバッファで業務プログラムをより柔軟に開発できるようにします。
- データ変換に必要なプログラムを必要としません。
- ネットワーク負荷を減少させます。初期に設定したデータタイプを送受信できるようにし、文字型データへの変換にかかるネットワーク負荷を減少させます。

## 5.2. バッファのタイプ

Tmaxはネットワーク負荷の減少と開発利便性を高めるために、以下のバッファを提供しています。

**[図 5.1] Tmax通信バッファのタイプ**



- **STRINGバッファ**

NULL値で終わる文字列で、バッファの名前を別途明示する必要がありません。プラットフォームの違いによって発生する問題は発生しません。

- **CARRAY、X\_OCTETバッファ**

長さが指定されているByte列で、通常、バイナリタイプのデータを送るときに使われます。データを交換するときは必ず長さを明示しなければなりません。プラットフォームの違いによる問題は発生しません。

- **STRUCT、X\_C\_TYPEバッファ**

C言語の構造体をデータ通信に使う場合に使用します。構造体のメンバーにはすべての原始タイプを使用でき、宣言された構造体自体もメンバーとして使用できます。また、構造体の配列も使用できます。

- **X\_COMMONバッファ**

メンバータイプがchar、int、longに限られたC構造体です。

- **FIELDバッファ**

フィールド・キーとデータ値を一組として管理するデータバッファで、すべての原始タイプのデータを格納することができます。交換されるデータ型が流動的な場合に使用します。このバッファを使用すれば、多様なデータ・アクセス方法と変換APIを提供します。

## 5.3. バッファの管理

バッファの整合性を保証するために**標準通信型バッファ**を使用します。標準通信型バッファは、通信する前にデータ型とメモリー割り当て方式に基準を定めて使用するものです。送信する前にバッファのデータを標準通信型に変換し、受信する前に逆変換します。Tmaxは異種間に通信をする場合、データの変換/逆変換によって、文字型だけでなく構造体バッファも使用できます。

Tmaxでは、標準通信型への変換のために**sdlc** プログラムが使われます。sdlcプログラムを利用すると、クライアントには標準通信型データへの変換のための情報テーブルが生成され、サーバーには標準通信型への変換/逆変換プログラムが生成されます。こうした過程により、異種間に通信する場合にも文字列やByte列だけでなくプラットフォームごとに異なる原始タイプのデータ型を使用する構造体バッファやフィールド・バッファを何の制約無しに使用できます。

構造体バッファのSTRUCT、X\_C\_TYPE、X\_COMMONは標準通信型に変換されなければならない、コンパイルするときに追加される必要があります。STRING、CARRAY、X\_OCTETバッファは異種間にもstring型通信ができるので、標準バッファ型への変換は要りません。

---

### 参考

バッファ関連関数の詳細については、「[9.8. バッファの管理](#)」または『Tmaxリファレンスガイド』を参照してください。

---

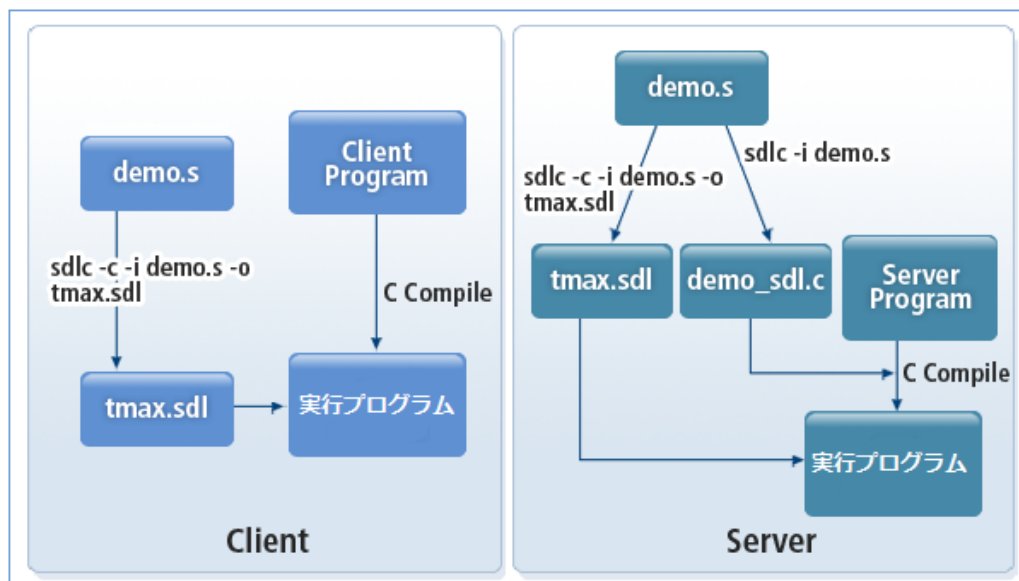
### 5.3.1. 構造体バッファ

開発者は使用する構造体を構造体ファイル(xxx.s)に記録します。そしてこのファイルは変換テーブルと変換プログラムを作成する基盤になります。構造体ファイルにはtypedef、includeのようなコマンドは使用できません。一方、構造体メンバーはすべての原始タイプのデータと構造体を使用できます。

構造体バッファを標準通信型に使用するには、Tmaxが提供する**sdlc**プログラムを使用して標準通信型データへの変換のための情報テーブルと標準通信型の変換/逆変換プログラムを作成する必要があります。変換/逆変換プログラムは開発されたサービス・ルーチンと一緒にコンパイルして使用します。情報テーブルは構造体バッファを使用するとき自動で使われます。

以下の図は構造体バッファ(STRUCT, X\_C\_TYPE, X\_COMMON)を利用してアプリケーションを開発するとき、サーバー・クライアント・プログラムのコンパイル順序です。構造体バッファの名前はdemo.sです。

[図 5.2] 構造体バッファを利用したアプリケーション・プログラムのコンパイル

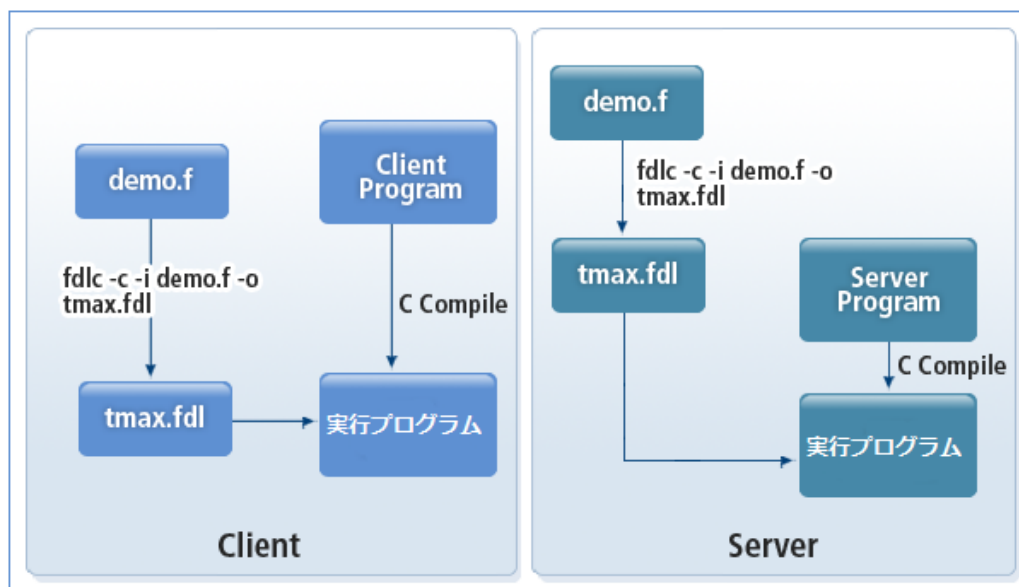


### 5.3.2. フィールド・バッファ

開発者は使用するフィールド・バッファをフィールド・バッファ・ファイル(xxx.f)に記録します。そしてこのファイルは変換テーブルを作成する基盤になります。フィールド・バッファを標準通信型として使用するには、Tmaxが提供する**fdlc**プログラムを使用して情報テーブルを作成する必要があります。また、このテーブルはフィールド・バッファを使用するとき自動で使われます。

以下の図は、フィールド・バッファを使用してアプリケーションを開発する際、サーバー、クライアント・プログラムのコンパイル順です。フィールド・バッファ・ファイル名はdemo.fです。

[図 5.3] フィールド・バッファを利用したアプリケーション・プログラムのコンパイル





# 第6章 トランザクション

本章では、トランザクションの概念と定義、処理方法について説明します。

## 6.1. 概要

トランザクションはリソースを、一つの一貫した状態から他の一貫した状態に変化させる処理の単位です。

Tmaxはローカル・トランザクションとグローバル・トランザクションをサポートします。

- ローカル・トランザクション

ローカル・トランザクション(Local Transaction Processing)は1つのリソース管理者(データベース)が参加するトランザクションです。

- グローバル・トランザクション

グローバル・トランザクションは1つ以上のリソース管理者(データベース)と1つ以上の物理的なサイトが1つの論理的な単位で参加するトランザクションです。Tmaxシステムではすべてのトランザクションを一応グローバル・トランザクションとみなします。

Tmaxのトランザクション処理は、ACID(Atomicity, Consistence, Isolation, Durability)属性を守りつつ作業を処理します。トランザクション処理に関連する標準としてX/Open DTPモデルのTX規格に従います。

ACIDの特性は以下のとおりです。

- 原始性(Atomicity)

業務がすべて実行されるか、1つも実行されない(all or nothing)の二つの場合のみ存在するということをいいます。

- 一貫性(Consistency)

トランザクションの前後でデータの整合性が保たれ、矛盾のない状態が継続するように、トランザクションの実行結果を一つの一貫した状態から他の一貫した状態の共有リソースに更新することをいいます。

- 独立性(Isolation)

共有リソースを変更するとき、トランザクション結果がコミット(commit)になるまでトランザクション外部に現れないということをいいます。

- 耐久性(Durability)

下位システムやメディアのエラーを経験したトランザクション・コミットの結果を変更することをいいます。正常完了したトランザクションの結果は永久に反映されます。

## 6.2. 分散トランザクション

2つ以上の同種または異種のデータベースが関連するグローバル・トランザクションでは、トランザクションの属性を保証するため、2PC(Two Phase Commit)を使用します。2PCとは、2つ以上のデータベースが連動するとき、ACID属性を完全に保証するため、2段階の処理(Prepare段階、Commit段階)をすることです。

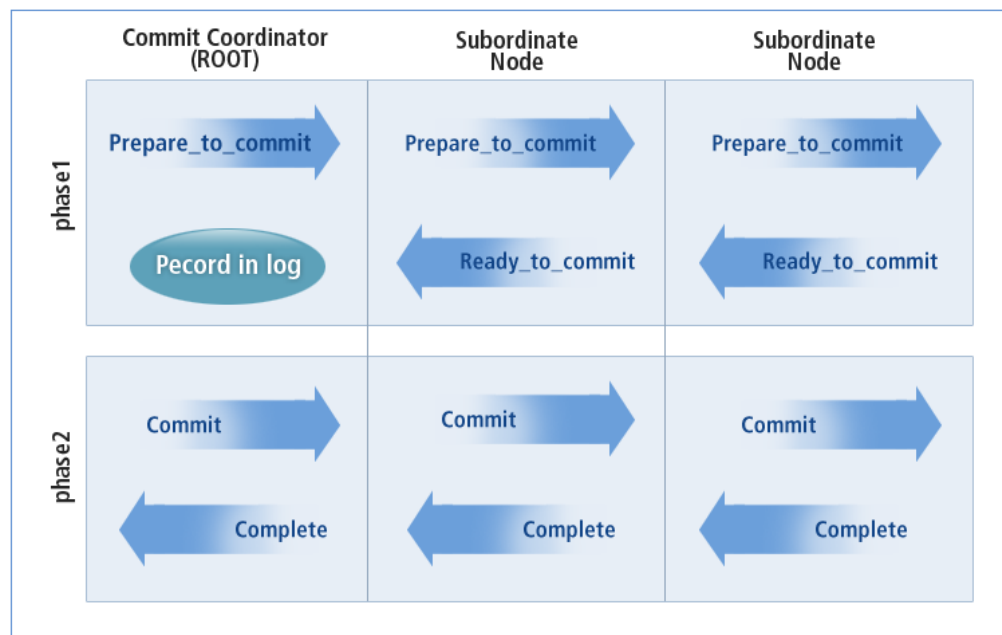
- Prepare段階

トランザクションに関連するすべてのデータベースにトランザクションを処理する準備が終わったかを確認します。すべてのデータベースは、準備が完了すると信号を渡します。

- Commit段階

すべてのデータベースから正常信号を受信するとコミット処理を、1つでも異常信号を受信するとロールバック処理をしてグローバル・トランザクションを完了します。

[図 6.1] 2相コミット・プロトコル



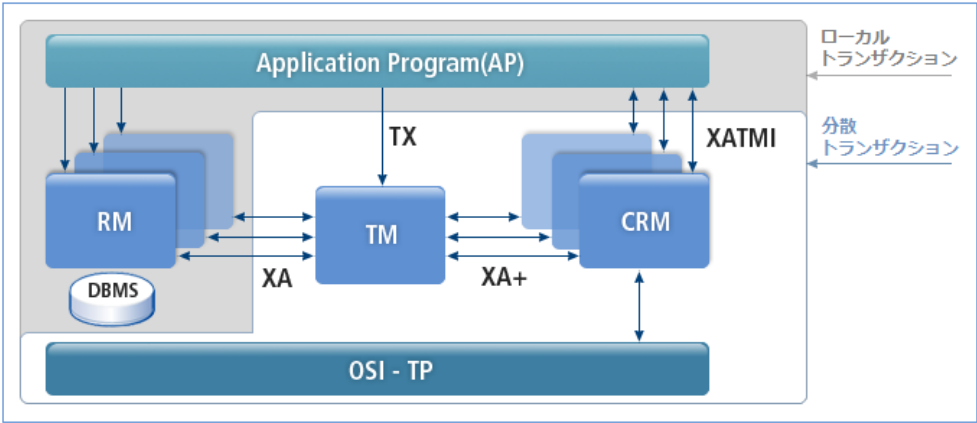
分散トランザクションを処理する場合、2段階のコミットをサポートしてデータ整合性(Integrity)を保証し、簡単な複数の関数(tx\_begin, tx\_commit, tx\_rollbackなど)を提供してグローバル・トランザクションを簡素化します。また、マルチスレッド方式のトランザクション・マネージャを提供して少ないリソースに比べて高い効率性を提供し、動的ロギングによってエラーが発生する場合は迅速に対応するので、リカバリー/ロールバックによる安定性を保証します。すべてのトランザクションは中央で管理するので、トランザクションに対するスケジューリングや管理が簡単です。

# 処理方式

Tmaxの分散トランザクション処理方式は、基本的に国際標準のX/Openモデル(DTP)をベースにします。  
X/OPEN DTPを遵守する標準関数の集合体であるX/OPEN ATMIに従います。

以下は、X/Open DTPの構造です。

[図 6.2] X/Open DTPの構造

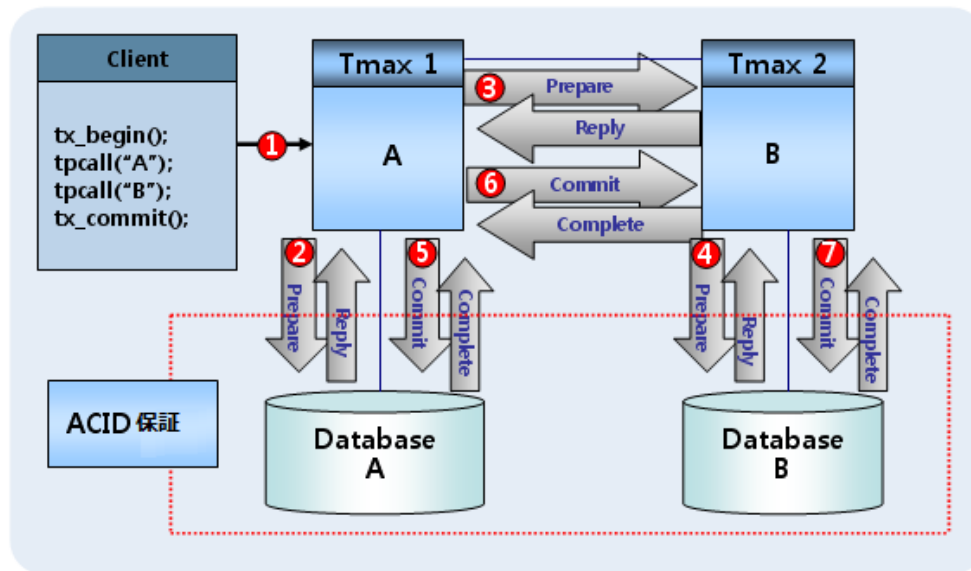


以下は、X/Open DTPの構成要素についての説明です。

区分	説明
AP	AP(Application Program)は、分散トランザクションの境界を提供します
RM	RM(Resource Manager)は、データベースなどのリソースへのアクセスを提供します
TM	TM(Transaction Manager)は、各データベースのIDを作成して実行を管理し、完了および失敗時にリカバリーを行います
CRM	CRM(Communication Resource Manager )は、分散AP間の通信を制御します
OSI-TP	OSI-TP(Open System Interconnection-Transaction Processing)は、互いに異なるTM領域との通信を担当します

X/OPEN ATMIは、X/OPEN DTPを遵守する異種のDBMS間で発生するトランザクションを一括処理します。  
以下は、分散トランザクションの処理過程についての説明です。

[図 6.3] 分散トランザクションの処理過程



トランザクションは、`tx_begin()`を呼び出して`tx_commit()`または`tx_rollback()`が呼び出される部分までです。

`tx_begin()`はトランザクションを開始できるようにし、この関数を呼び出したプロセスはトランザクションの指揮者(coordinator)になります。トランザクション指揮者は、`tx_commit()`または`tx_rollback()`を呼び出してトランザクションを完了する義務を持ちます。`tx_begin()`で開始して`tx_commit()`または`tx_rollback()`で完了する1つのトランザクションに含まれるプロセスをトランザクション参加者(participant)といい、トランザクション参加者は`tpreturn()`で返却される値によってトランザクションの結果に影響を与られます。トランザクション状態で`tx_begin()`によってトランザクションを再開しようとする場合、この試みは失敗し、`tperrno`にTPEPROTOが設定されます。ただし、本来のトランザクションは継続実行されなければなりません。

トランザクション状態にあるクライアントから`tpcall()`または`tpacall()`を呼び出す場合、`flags`パラメーターをTPNOTRANで設定し、サーバーがトランザクション参加から除外されるようにすることができます。TPNOTRANフラグで要求されたサーバーは現在実行中のトランザクション結果に影響を与えることができません。

トランザクションを宣言する方法には、開発者が明示的にトランザクション関連関数(例: `tx_begin`、`tx_commit` など)を直接作成する**Explicitトランザクション方法**と、サーバーからTmax環境ファイルのSVRGROUPセクションでデータベース関連項目を設定し、サービスを内部的にトランザクションとして処理する**Implicitトランザクション方法**があります。

## DBMS関連の確認事項

DBMSと連動する場合、以下の事項を確認する必要があります。

- DBMSベンダーが提供するクライアント・モジュールのインストールを確認します。
- DBMS設定項目のうちデータベースクライアントに対するセッション数の制限などの事項を確認します。

- Tmaxサーバー・アプリケーションの作成に必要な開発ツール・モジュールが必要です。
- XA/非XA使用状態を確認し、適切に構成します。

サーバー・プログラムにおけるXAモードの設定状態はアプリケーション・デザイン段階で決定されなければなりません。XAと非XAは、以下のようにDBMSとTmax間のデータベース接続および分散トランザクション処理方法の違いによって区分されます。

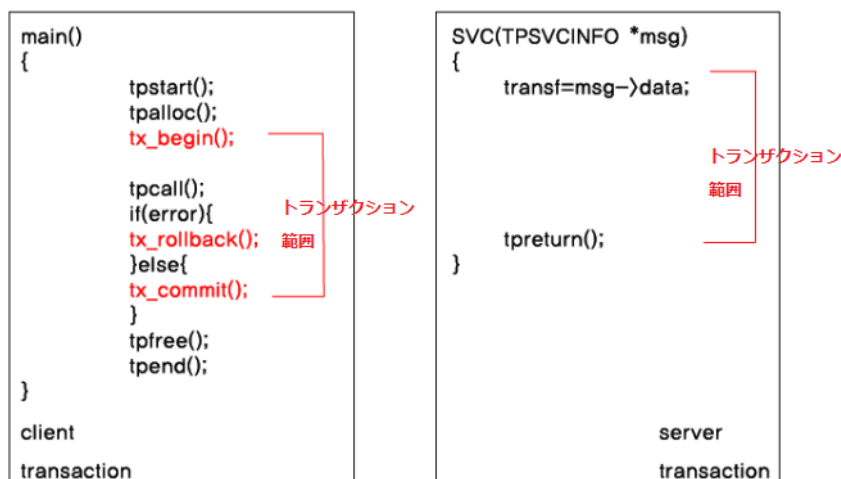
区分	説明
XA	Tmax環境ファイルの設定によってDBMSと接続が行われます。XAトランザクションの終了または取り消しはX/OPEN ATMI関数を使用し、TMSを使って終了または取り消し完了が行われます
非XA	ユーザー(開発者)のESQL文を使ってDBMS接続またはトランザクションの終了および終了の取り消しが処理されます

## 6.2.1. XAモード

XAモードのTmaxシステムは、別途のトランザクション管理者(TMS)を設けてトランザクション全体の管理を担当させています。

XAモードで運用される場合、すべてのトランザクションはグローバル・トランザクションとしてみなされます。データの整合性を確保するには、2PCを使用します。XAモードである場合、リソース管理者との接続と接続解除のため、Tmax環境ファイルに別途の設定が必要です。また、トランザクション管理はTmax APIを使用して処理し、分散環境で1つ以上のデータベースを処理するために必要です。以下は、XAモードのプログラム処理順序です。

[図 6.4] XAモード



tpsvrinit()/tpsvrdone()ロジックのデータベース接続および解除ロジックは不要です。非XAと同様、tmbootされるとき、データベースと接続し、別途にTMSというグローバル・トランザクションの終了と終了取り消しを担当

当するプロセスもデータベースと接続します。グローバル・トランザクションの開始と終了、そして終了の取り消し要求は必ずTmaxのTX APIを呼び出すか、Tmax環境設定によって処理します。XAモードを設定するには、Tmax環境ファイルのうちSVRGROUPセクションでDBMSと連動するサーバー・アプリケーションが含まれるサーバー・グループ名を確認し、それに対するXAを設定します。

以下は、XAモードのSVRGROUPセクションを設定する例です。

```
*SVRGROUP
svg1      NODENAME = tmax,
          DBNAME   = ORACLE,
          OPENINFO  = "ORACLE_XA+Acc=P/scott/tiger+SesTm=600",
          TMSNAME   = svg1_tms

*SERVER
svr1      SVGNAME  = svg1
```

---

#### 参考

サーバー・グループの環境設定については、『Tmax 運用ガイド』を参照してください。

---

## 6.2.2. 非XAモード

非XAモードのTmaxシステムはリソース管理者に直接トランザクション・コマンドを出すもので、分算環境におけるグローバル・トランザクションは使用できません。非XAモードの場合、特に別途設定は必要としないものの、アプリケーションがリソース管理者と別の接続を確立してトランザクションを宣言する必要があります。

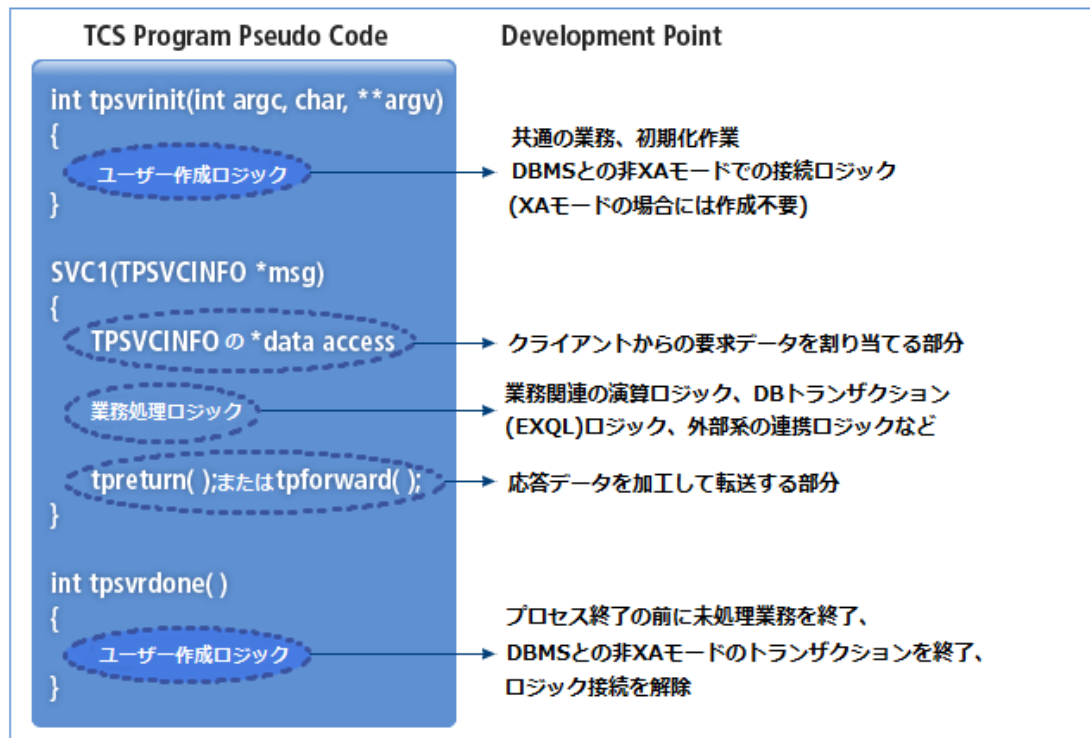
非XAモードはネイティブSQLと正確にRDBMSに接続されたトランザクション制御を持つサーバー・プログラム方式です。非XAモードはTmaxが提供するトランザクションは使用できず、ローカル・トランザクションだけ可能です。非XAモードのサーバー・プログラムは簡単なSelect問合せや実行時間の長いBatchサーバー・プログラムに適合します。非XAモードでもトランザクションの範囲と制御を設定できます。しかし、非XAモードのユーザー・アプリケーションとRDBMS(direct connection)間でのみ可能です。Tmaxとはまったく関係ありません。

非XAモードとXAモードは、以下の2つの点が異なります。

- ユーザー・アプリケーションのSQLを使用してRDBMSと接続するか切断します。
- サーバー・プログラムはトランザクションを要求することができますが、クライアント・プログラムはできません。非XAサーバー・プログラムの場合、データベースと接続するか切断するとき、tpsvrinit()とtpsvrdone()ルーチンを使用することを推奨します。

以下は、非XAモードのプログラム処理順序です。

[図 6.5] 非XAモード



1. tpsvrinit()ロジックに対し、DBMSに接続できるロジックを作成します。
2. tpsvrdone()ロジックに対し、明示的にDBMS接続をロールバックし、リリース・ロジック(release logic)を追加します。
3. tmbootまたはtmboot -S svrnameなどによってTmaxサーバー・アプリケーションを起動すると、DBMSとTmaxサーバー・アプリケーション・プロセス間で接続が作成されます。
4. サービス・ロジックでは、すでに接続されているDBMSで、ESQLによるローカル・トランザクションの転送および終了、終了取り消しの要求を使ってトランザクションを処理できます。

以下は、非XAモードのSVRGROUPセクションを設定する例です。

```
*SVRGROUP
svg1 NODENAME = tmax

*SERVER
svr1 SVGNAME = svg1
```

---

## 参考

トランザクション関連関数の詳細については、「[9.9. トランザクションの管理](#)」または『Tmax リファレンスガイド』を参照してください。

---

## 6.3. トランザクション関連エラー

トランザクション関連で発生できるエラーはTXとXAでのエラーと関連しています。TX関連エラーは<usrinc/tx.h>に定義されており、XA関連エラーは使用するデータベース・ベンダーが提供します。Oracleの場合、<\$ORACLE\_HOME/rdbms/demo/xa.h>を参照してください。

### 6.3.1. TXエラー

以下は、トランザクション・エラーについての説明です。

エラーコード	機能
TX_NOT_SUPPORTED(1)	トランザクションをサポートしないモードの場合に発生するエラーです
TX_OK(0)	正常に処理された場合に発生します
TX_OUTSIDE(-1)	ローカル・トランザクションが実行中の場合に発生します
TX_ROLLBACK(-2)	コミットできない場合に発生するエラーで、ロールバックします
TX_MIXED(-3)	一部はコミットされ、一部はロールバックされる場合です
TX_HAZARD(-4)	正常に完了していない場合に発生します
TX_PROTOCOL_ERROR(-5)	トランザクションが正常に呼び出された場合に発生します
TX_ERROR(-6)	データベース障害が発生した場合です
TX_FAIL(-7)	深刻な障害が発生した場合です
TX_EINVAL(-8)	間違ったパラメータを受信した場合に発生します
TX_COMMITTED(-9)	トランザクションがデータベースに独自実行された場合に発生します
TX_NO_BEGIN(-10)	トランザクションはコミットされたものの、新規トランザクションは開始できない場合に発生します

### 6.3.2. XAエラー

以下は、XAエラーについての説明です。

エラーコード	機能
XA_OK(0)	正常に処理された場合に発生します



エラーコード	機能
XAER_RMERR(-3)	OPENINFOセクションで設定されているリソース・マネージャー・エラーが発生した場合です
XAER_NOTA(-4)	付与されたXIDが有効でない場合に発生します
XAER_INVALID(-5)	間違ったパラメータを受信した場合に発生します
XAER_PROTO(-6)	不適切な段階で呼び出された場合に発生します
XAER_RMFAIL(-7)	データベースとの接続に失敗した場合に発生します
XAER_DUPID(-8)	すでに付与されたXIDを使用した場合に発生します
XAER_OUTSIDE(-9)	トランザクション・モードから離脱した場合に発生します



# 第7章 マルチスレッドとマルチコンテキスト

本章では、マルチスレッドとマルチコンテキストを使用するための設定について説明します。

## 7.1. 概要

Tmaxシステムはマルチスレッドとマルチコンテキスト機能を使用するため、カーネル・レベルのパッケージをサポートしています。スレッドを作成および削除などのプログラムは、開発者がロジックを考慮して作成する必要があります。また、Cで作成されたマルチスレッドとマルチコンテキストのアプリケーションはサポートしていますが、COBOLで作成されたマルチスレッドとマルチコンテキストのアプリケーションはサポートしていません。

---

### 参考

マルチスレッドとマルチコンテキストのクライアント・ライブラリーでは、3.8.15以降のバージョンから使用可能であり、サーバー・ライブラリーではTmax v5.0 SP2以降のバージョンから使用できます。

---

## 7.2. クライアント・プログラム

本節では、マルチスレッドとマルチコンテキストに対するクライアント・プログラムのフローとプログラムの実装方法および例について説明します。

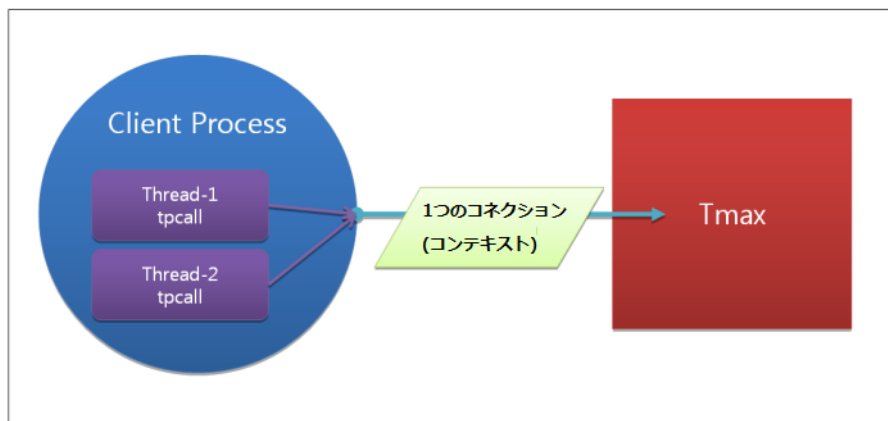
### 7.2.1. プログラムのフロー

#### 7.2.1.1. マルチスレッド

マルチスレッドは1つのプロセス内に1つ以上の実行ユニットを持っているものです。そのため、Tmaxマルチスレッド・アプリケーションでは同じプロセスで同時に複数のサービスを要求することができます。

以下の図は、クライアントのマルチスレッド・アプリケーション・プログラムのフローです。1つのクライアント・プロセスは同時に2つのサービスを呼び出すことができます。

【図 7.1】 Tmaxクライアントのマルチスレッド・アプリケーション

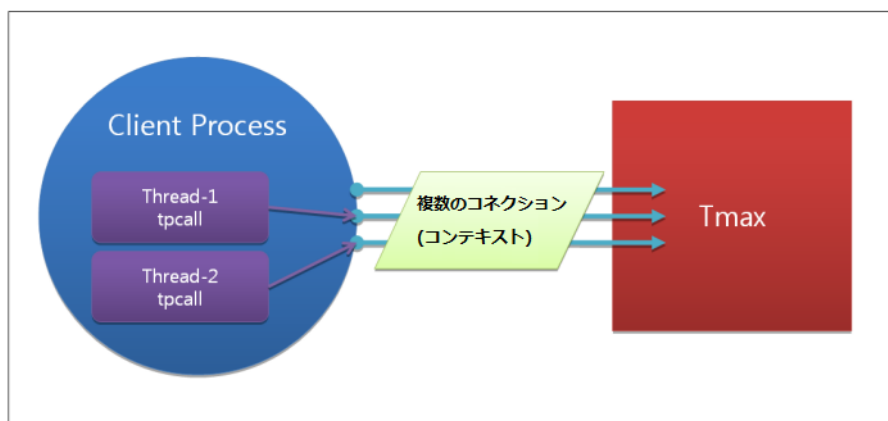


### 7.2.1.2. マルチコンテキスト

マルチコンテキストは1つのクライアントがTmaxシステムと複数の接続を実行して通信できるようにするプログラム手法です。

以下の図は、クライアントのマルチコンテキスト・アプリケーション・プログラムのフローです。1つのクライアントは複数のコンテキストを有しており、各コンテキストはTmaxシステムとそれぞれ1つずつの接続を確立して通信します。したがって、マルチコンテキストを使用する場合、Tmaxシステムは1つのユーザーとして認識します。

【図 7.2】 Tmaxクライアントのマルチコンテキスト・アプリケーション



### 7.2.2. プログラムの実装

マルチスレッドとマルチコンテキストを使用するには、以下の3つのルーチンに従ってクライアント・プログラムを作成する必要があります。

#### 1. 開始文

## 2. 実装文

## 3. 終了文

---

### 参考

クライアント・プログラム実装のために使われる各関数の詳細については、『Tmax リファレンスガイド』を参照してください。

---

## 開始文

マルチスレッドとマルチコンテキストは、以下のような関数で開始できます。

- **tpstart()**

```
int tpstart (TPSTART_T *tpinfo )
```

Tmaxシステムと接続する関数です。最初の引数のTPSTART\_Tのメンバーであるフラグ(flags)を**TPMultiContextS**または**TPSINGLECONTEXT**に設定し、マルチコンテキストまたはシングル・コンテキスト(Singlecontext)としてTmaxシステムと接続することができます。フラグに値を設定しない場合は、デフォルトでTPSINGLECONTEXTが設定され、シングル・コンテキストとして動作します。

また、クライアントは一度シングル・コンテキストに設定されると、継続してシングル・コンテキストとして動作し、マルチコンテキストに設定すると、継続してマルチコンテキストとして使用されます。

---

### 参考

tpstart()についての詳しい内容は、『Tmax リファレンスガイド』の「3.3.10. tpstart」を参照してください。

---

- **tpsetctxt()**

```
int tpsetctxt(int ctxtid, long flags)
```

コンテキストを設定する関数であり、クライアント・プログラムとサーバー・プログラムにおける作成方法が異なります。詳しい説明については、「[9.14.2. tpsetctxt](#)」を参照してください。

## 実装文

マルチスレッドとマルチコンテキストはATMI関数を使用して実装します。ATMI関数を使用する際は、必ず現在のコンテキストなのかを把握して使用しないと正常に使用できません。

- 同期型通信

Tmaxシステムとクライアントが同期型通信をする場合、現在のコンテキストがTPINVALIDCONTEXTでない場合に正常に実行されます。TPINVALIDCONTEXTとは、現在のコンテキストが他のスレッドによってフリー(free)になった場合をいいます。

たとえば、thread1でtpstart()をした後、context1を持ち、thread2ではtpsetctx()でcontext1と一緒に使用する場合、thread1で処理を終えてtpend()をするとcontext1もメモリーから削除されるので、thread2はTPINVALIDCONTEXTを持つことになります。

また、マルチコンテキストではない場合、**tpstart()**をしなくても**tpcall()**などのAPIを呼び出すと自動でTmaxシステムとの接続を実行しますが、マルチコンテキストの場合は必ずtpstart()をした後にtpcall()などのAPIを明示的に使用する必要があります。

- 非同期型通信

マルチコンテキストを使用する場合、同じコンテキストを持つ2つのスレッドがある場合、1つのスレッドで**tpacall()**した後、他のスレッドで**tpgetrply()**を使用して結果を取り込むことができます。このように使用する際は、tpacall()が必ず先に呼び出されなければならない、2つのスレッド間に優先順位が明確な場合で使わないと正しい結果を得ることができないので、注意が必要です。

同期型通信と同様、TPINVALIDCONTEXTではない場合に正常に実行されます。

- トランザクション

1つのスレッドでトランザクションを開始した場合、そのスレッドと同じコンテキストを使用するスレッドはトランザクションを開始した以降は1つのトランザクションになります。このように使用する場合も非同期型通信と同様に優先順位が明確な場合に使用します。また、TPINVALIDCONTEXTでない場合に正常実行されます。

## 終了文

tpend()関数を使用してマルチスレッドとマルチコンテキストを終了します。

```
int tpend()
```

この関数を使用しないと、コンテキストと該当するスレッドに関する情報がメモリーから削除されず、以降問題になる場合があります。したがって、必ずコンテキストを使用した後は、tpend()を実行する必要があります。

### 7.2.3. プログラムの例

クライアント・プログラム、サーバー・プログラム、メイクファイルの例について説明します。

#### クライアント・プログラム

以下は、クライアント・プログラムの例です。

```

/*****
/*          Multi-thread/Multi-context Sample Program          */
/*          */
/* TmaxSoft Co. / QA                                          */
/* remarks: TmaxのTOUPPEサービスが起動されている必要があります。 */
*****/

#include      <pthread.h>
#include      <stdlib.h>
#include      <stdio.h>
#include      <errno.h>
#include      <string.h>
#include      <unistd.h>
#include      <netdb.h>
#include      <sys/types.h>
#include      <usrinc/atmi.h>

#define MAX_CTID_CNT      400          /* 最大コンテキスト数 */
#define MAX_CALL          1
#define NUM_THREADS      2          /* 一度に作成するスレッド数 */
#define NUM_CONTEXTS      40          /* 1つのスレッドで作成するコンテキスト数 */

void *mythread(void *arg);
int svcCall(char* svc, char* arg);

int newContext();
int altContext(int id);
int delContext();

#define CTID_EMPTY      0
#define CTID_OCCUPIED      1
#define THRERR      (void *)-1
#define THRSUC      (void *)1

extern int errno;
extern int _init_wthr_flag;
int      thr_id;
TPSTART_T* tpinfo;

int main()
{
    void      *retVal;
    char      argData[100];
    int      tcnt = 0;
    int      scnt = 0;

    pthread_t      p_thread[NUM_THREADS];

```

```

    memset(argData, 0x00, sizeof(argData));
    strcpy(argData, "...mtmc test...");

    if (tmaxreadenv("tmax.env", "TMAX") == -1)
    {
        printf( "tmax read env failed\n" );
        return FALSE;
    }

    tpinfo = (TPSTART_T *)tpalloc("TPSTART", NULL, 0);
    if (tpinfo == NULL)
    {
        printf("[THR:%d]tpinfo tpalloc fail[%d][%s]\n",pthread_self(),
            tperrno,tpstrerror(tperrno));
    }

#ifdef _MULTI_THREAD_TEST_

    while(scnt<MAX_CALL)
    {
        for ( tcnt=0 ; tcnt<NUM_THREADS ; tcnt++)
        {
            if (pthread_create(&p_thread[tcnt], NULL, mythread,argData))
            {
                fprintf(stderr, "mythread start fail...[%d]\n", errno);
                return FALSE;
            }
        }
        for(tcnt=0 ; tcnt<NUM_THREADS ; tcnt++)
        {
            pthread_join(p_thread[tcnt], &retVal);
        }

        scnt++;
        sleep(1);
    }

#else

    if (pthread_create(&p_thread[tcnt], NULL, mythread, argData))
    {
        fprintf(stderr, "mythread start fail...[%d]\n", errno);
        return FALSE;
    }

    pthread_join(p_thread[tcnt], &retVal);

#endif

```



```

        tpfree((char *)tpinfo);
        return TRUE;
}

/*****
/* Sub Process : myhread */
*****/
void *mythread(void *arg)
{
    int i,j,k;

    printf("[THR:%d] thread start\n",pthread_self());

#ifdef _MULTI_CONTEXT_TEST_
        tpinfo->flags = TPMultiContextS;
        for(i=0;i<NUM_CONTEXTS/2;i++)
        {
            j=newContext();
            k=newContext();

            svcCall("TOUPPER",arg);
            delContext();

            altContext(j);
            svcCall("TOUPPER",arg);
            delContext();

        }
#else
        tpinfo->flags = TPSINGLECONTEXT;
        newContext();
        svcCall("TOUPPER",arg);
        delContext();
#endif

    printf("[THR:%d] thread finish\n",pthread_self());

    return THRSUC;
}

/*****
/* Sub Process : delContext */
*****/

```

```

int delContext()
{
    int i;
    int id;

    i = tpgetctxt(&id, TPNOFLAGS);

    if (i < 0)
    {
        printf("\t[delContext]tpgetctxt fail[%d][%s]\n",
            tperrno, tpstrerror(tperrno));
        return -1;
    }

    tpend();
    printf("\t[THR:%d][CTXT:%d]tpend.\n", pthread_self(), id);
    return 1;
}

/*****
/* Sub Process : newContext */
*****/
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;
}

```

```

/*****/
/* Sub Process : altContext */
/*****/
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;
}

/*****/
/* Sub Process : svcCall */
/*****/
int svcCall(char* svc, char* arg)
{
    int ret;
    long rlen;
    char *sbuf, *rbuf;
    int id;

    ret=tpgetctxt(&id,TPNOFLAGS);

    sbuf = (char *)tpalloc("STRING", NULL, 0);
    if (sbuf == NULL)
    {
        printf("\t[svrCall]tpalloc error[%d][%s]\n",tperrno,
            tpstrerror(tperrno));
        return -1;
    }

    rbuf = (char *)tpalloc("STRING", NULL, 0);
    if (rbuf == NULL)
    {
        printf("\t[svrCall]tpalloc error[%d][%s]\n",tperrno,
            tpstrerror(tperrno));
    }
}

```

```

        return -1;
    }

    strcpy(sbuf, (char *)arg);

    ret = tpcall(svc, (char *)sbuf, strlen(arg), (char **)&rbuf, (long *)&rlen,

                TPNOFLAGS);

    if (ret < 0)
    {
        printf("\t[svcCall]tpcall fail.[%d][%s]\n",tperrno,
                tpstrerror(tperrno));
    }
    else
    {
        printf("\t[THR:%d][CTXT:%d]tpcall success.\n",pthread_self(),id);
    }

    tpfree((char *)sbuf);
    tpfree((char *)rbuf);
}
/*****
/*                                */
/*****/

```

## サーバー・プログラム

以下は、サーバー・プログラムの例です。

```

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

TOUPPER(TPSVCINFO *msg)
{
    int i;

    printf("\tTOUPPER service is started!\n");
    printf("\tINPUT : data=%s\n", msg->data);

    for (i = 0; i < msg->len; i++)
        msg->data[i] = toupper(msg->data[i]);

    printf("\tOUTPUT: data=%s\n", msg->data);
}

```

```
    tpreturn(TPSUCCESS,0,(char *)msg->data, 0,0);  
}
```

## メイクファイル

以下は、メイクファイルの例です。

```
TARGET = $(COMP_TARGET)  
APOBJS = $(TARGET).o  
  
TMAXLIBD = $(TMAXDIR)/lib64  
  
TMAXLIBS = -lcli  
#TMAXLIBS = /home/ancestor/tmax/lib/libclid.a  
  
#multi_thread / multi_contextの場合  
CFLAGS = -q64 -O -I$(TMAXDIR) -D _MULTI_THREAD_TEST_ -D _MULTI_CONXTXT_TEST_  
  
#single_thread / multi_contextの場合  
CFLAGS = -q64 -O -I$(TMAXDIR) -D _MULTI_CONXTXT_TEST_  
  
LDFLAGS = -brtl  
  
#  
.SUFFIXES : .c  
  
.c.o:  
    $(CC) $(CFLAGS) $(LDFLAGS) -c $<  
  
#  
# client compile  
#  
$(TARGET): $(APOBJS)  
    $(CC) $(CFLAGS) $(LDFLAGS) -L$(TMAXLIBD) -o $(TARGET) $(APOBJS) $(TMAXLIBS)  
  
#  
clean:  
    -rm -f *.o core $(TARGET)
```

## 7.3. サーバー・プログラム

本節では、マルチスレッドとマルチコンテキストに対するサーバー・プログラムのフロー、実装、例について説明します。

### 7.3.1. 概要

Tmaxマルチスレッドとマルチコンテキストのサーバー・ライブラリーは、TCS方式のライブラリーは提供していますが、UCS方式およびRDP方式のライブラリーはサポートしていません。

マルチスレッドとマルチコンテキストのサーバー・ライブラリーは、マルチスレッドおよびマルチコンテキスト機能を提供しています。したがって、1つのサーバー・プロセスで複数のスレッドが同時にサービス要求を処理したり、2つ以上のスレッドが1つのコンテキストを共有しながらサービスを処理したりすることができます。

一般的なTmaxサーバー・ライブラリーは、マルチコンテキスト機能を提供していないため、ユーザーが作成したスレッドではTmaxサーバー・ライブラリーが提供するAPIを使用することができません。

マルチスレッドとマルチコンテキストのサーバー・ライブラリーを使用するには、以下の設定が必要です。

- 関連のAPIを使用してサーバー・アプリケーションのコードを作成します。
- サーバー・プログラムをビルドする際、マルチスレッドとマルチコンテキストのサーバー・ライブラリーであるlibsvrmt.soまたはtmaxsvrmt.dllをリンクします。
- Tmax環境設定ファイルの**SERVER**セクションにマルチスレッドとマルチコンテキストのサーバー関連項目を設定します。

該当するサーバーが、マルチスレッドとマルチコンテキストを使用するという情報を設定します。

– 関連項目：SVRTYPE、MINTHR、MAXTHR、STACKSIZE、CPCなど

マルチスレッドを用いたプログラミングは、様々な面で非常に有用な点を持っています。ただし、同時性や性能の面などで慎重にコードを作成しないと、予期しない動作を引き起こす場合があります。したがって、マルチスレッド・プログラミングが持つ問題点について事前に把握しておく必要があります。

以下は、マルチスレッドとマルチコンテキストのサーバー・ライブラリーを使用する際の長短所です。

#### ● 長所

- コードが簡単になり、直感的なプログラミングが可能です。
- サーバー・プロセスの数を減らすことができます。

#### ● 短所

- 同時性の管理やコードの作成などの難易度が高いです。
- エラーが発生した場合、デバッグが難しいです。

- マルチスレッド・プログラムに移植する場合、以前開発されたコードがスレッド・セーフなのかを確認する必要があります。
- RMと連携する場合、マルチスレッドをサポートするのかを確認する必要があります。

---

#### 参考

上記の長短所を考慮し、必要性を十分に検討する必要があります。サーバー・プログラムの実行中に問題が発生する場合はログメッセージを確認します。ログメッセージについての詳しい説明は、『Tmax メッセージリファレンスガイド』を参照してください。

---

## 7.3.2. プログラムのフロー

### 7.3.2.1. マルチスレッド

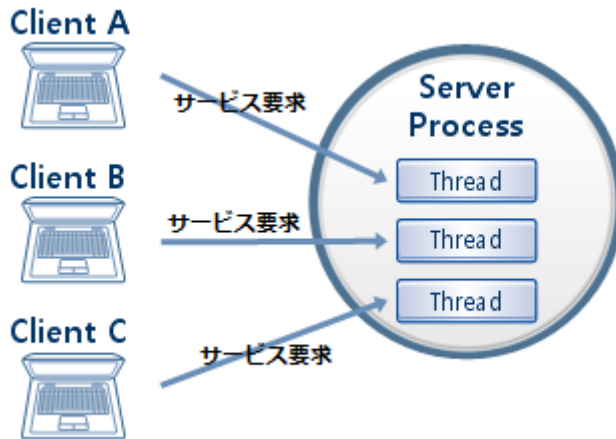
マルチスレッドは、1つのプロセス内に1つ以上のスレッドが存在しており、同時に複数のサービス要求を処理することができます。マルチスレッドとマルチコンテキストのサーバー・ライブラリーでは、スレッドを**サービススレッド**と**ユーザー作成スレッド**に区別しています。

#### サービススレッド

サービススレッドは、マルチスレッドとマルチコンテキストのサーバー・ライブラリーを独自で管理するスレッドです。サーバー・ライブラリーは、複数のサービス要求を同時に処理するため、環境設定の定義に従って一定の数のスレッドを独自で作成し、これをスレッド・プールで管理します。

スレッド・プールは、環境設定のMINTHRとMAXTHR項目の設定に基づいて動作します。サーバー・プロセスが起動する際、最小のスレッド数をデフォルトで作成し、スレッド数がそれ以下になったら、スレッドを追加作成して最小のスレッド数を維持します。サービス要求が増加し、スレッド・プールのアイドル状態のスレッドをすべて使い切った場合は、スレッドを最大数まで追加作成してサービス要求を処理します。

【図 7.3】 Tmaxサーバーのマルチスレッド・アプリケーション



サーバー・プロセスにサービス要求が受信された場合、サービス要求は、それぞれのスレッドで独立的に実行され、その結果をクライアントに返します。

サービス・ルーチンの基本的な動作は一般的なサーバー・プログラムと変わらないため、サービス・ルーチンも一般的なサーバー・ライブラリーのように作成します。ただし、複数のサービス・スレッドが同じサービス・ルーチンを実行できるため、すべてのルーチンはスレッド・セーフで作成される必要があります。

以下は、基本的なサーバー・プログラム作成のフローです。5項以外はサーバー・ライブラリーによって自動的に動作する部分です。

1. サーバー・プロセスが起動しながら`tpsvrinit()`関数を呼び出します。
2. スレッド・プールを構成しつつ、環境設定に従ってサービス・スレッドを作成します。
3. スレッドごとに作成されたら、最初に一度`tpsvrthrinit()`関数を呼び出します。
4. 以降サービス・スレッドはスレッド・プールで待機し、サービス要求が受信されたらアイドル状態のスレッドがアクティブ状態に切り替わり、サービス・ルーチンを実行します。
5. サービス・ルーチンは、[7.2.2節「実装文」](#)に記述された内容に基づいて実行します。
6. `tpreturn()`、`tpforward()`関数の呼出し後、クライアントに処理結果を返した後、スレッド・プールで待機します。
7. サーバー・プロセスが終了すると、作成されたすべてのサービス・スレッドは`tpsvrthrdone()`関数を実行した後、終了します。
8. サーバー・プロセスは`tpsvrdone()`関数を呼び出した後、終了します。



## ユーザー作成スレッド

サービス・ルーチンまたは`tpsvrinit()`、`tpsvrthrinit()`などの初期化ルーチンで、ユーザーが任意のスレッドを作成することができます。ユーザーが作成したスレッドは自身の開始ルーチンを持っているため、サーバー・ライブラリーが管理するスレッド・プールとは無関係であり、基本的にはサービス要求が受信されてもこれらのスレッドでは処理されません。

`tpsvrinit()`、`tpsvrthrinit()`、サービス・ルーチンでユーザーが作成するスレッドは、マルチスレッドとマルチコンテキストのサーバー・ライブラリーとは独立して動作するため、コンテキストを持っていません。これらのスレッドは、開発者の目的に沿って、サービス・ルーチンの代わりになることもあれば、サービス・ルーチンと一緒に動作したり、またサービス・ルーチンとは無関係で動作したりします。必ず[7.3.3節「開発時の注意事項」](#)について熟知してください。

以下は、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する状況における、プログラム作成のフローです。1項と8項は、サーバー・ライブラリーによって自動的に動作する部分です。

1. サービス要求が受信され、サービス・スレッドがサービス・ルーチンの実行を開始します。
2. サービス・ルーチンでは、`tpgetctxt()`関数を呼び出して自身のコンテキストIDを把握します。
3. ユーザー作成スレッドは、サービス要求の前から存在しているか、サービス・ルーチン内で新規作成されます。
4. ユーザー作成スレッドは、サービス・スレッドからコンテキストIDを取得し、`tpsetctxt()`を呼び出してコンテキストを共有します。
5. 両スレッドは、それぞれ自身の指定されたルーチンを実行します。両方のスレッドは両方とも`tpcall()`、`tpacall()`などのATMI APIを呼び出すことができます。
6. サービス・スレッドで`tpreturn()`または`tpforward()`を呼び出す前に、以下のような処理を実行します。
  - すべての同期型通信、非同期型通信、会話型通信などを終了します。
  - トランザクションが実行中であれば、コミットまたはロールバックを行います。
  - ユーザー作成スレッドは、`tpsetctxt()`を呼び出してコンテキストをこれ以上共有しないように設定します。
7. ユーザー作成スレッドは、自身の消滅時期を検討します。
8. サービス・スレッドは`tpreturn()`を実行した後、スレッド・プールに返され、次のサービス要求を待機します。

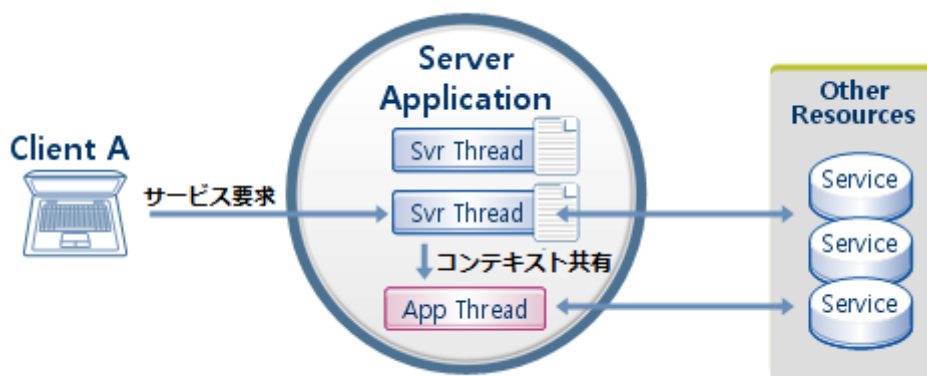
### 7.3.2.2. マルチコンテキスト

サーバー・ライブラリーでのコンテキストは、1つのサービス要求を処理する際に必要な情報です。マルチスレッド環境で、各スレッドごとにサービスを独立して処理するためには、マルチコンテキスト手法が必要です。

スレッド・プールで管理されるすべてのサービス・スレッドは、基本的に自身だけのコンテキストを持っています。また、マルチコンテキスト手法は、複数のスレッドが1つのコンテキストを共有して使用することができます。一方、ユーザー作成スレッドはサービス・スレッドとは違って、自身だけのコンテキストを持っていないため、ユーザー作成スレッドでtpcall()などのTmax APIを使用するためには、tpgetctx()、tpsetctx() APIを呼び出してサービス・スレッドのコンテキストを共有する必要があります。ユーザー作成スレッドが、他のスレッドのコンテキストを共有していない状態でTmax APIを呼び出すと、TPEPROTOエラーコードを返すと同時に呼び出しが失敗処理されます。

マルチスレッドとマルチコンテキストのサーバー・ライブラリーでのコンテキストは、トランザクション、同期型通信、非同期型通信、会話型通信などの情報を管理します。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すると、これらの情報も一緒に共有されます。

[図 7.4] Tmaxサーバーのマルチコンテキスト・アプリケーション



### 7.3.3. プログラムの実装

マルチスレッドとマルチコンテキストのサーバー・プログラムを作成する場合、以下のAPIを利用して開発することができます。開発者は、注意事項を参考にしてサーバー・プログラムを作成してください。

#### 関連API

以下は、サーバー・プログラムを作成する際の関連APIです。

- tpsvrthrinit

```
int tpsvrthrinit(int argc, char *argv[])
```

マルチスレッドと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。マルチスレッドとマルチコンテキスト・サーバーでtpsvrthrinit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、各スレッドの作成時に固有の初期化作業が行えるようにする初期化関数です。tpsvrthrinit()についての詳しい説明は、「[10.1.5. tpsvrthrinit](#)」を参照してください。

- **tpsvrthrdone**

```
int tpsvrthrdone()
```

マルチスレッドと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。マルチスレッドとマルチコンテキスト・サーバーは、サーバー・プロセスが終了すると、tpsvrdone関数を実行する前にサービス・スレッドを終了させます。tpsvrthrdone()についての詳しい説明は、「[10.1.6. tpsvrthrdone](#)」を参照してください。

- **tpsetctxt**

```
int tpsetctxt(int ctxtid, long flags)
```

現在の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・プログラムでの作成方法には違いがあります。tpsetctxt()についての詳しい説明は、「[9.14.2. tpsetctxt](#)」を参照してください。

- **tpgetctxt**

```
int tpgetctxt(int *ctxtid, long flags)
```

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IDを最初のパラメータに返します。tpgetctxt()についての詳しい説明は、「[9.14.1. tpgetctxt](#)」を参照してください。

---

## 参考

これらのAPIのほか、サーバー・ライブラリーが提供するAPIを使用することができます。プログラムを実装するために使われるAPIについての詳しい内容は、『Tmax リファレンスガイド』を参照してください。

---

## 開発時の注意事項

開発者は、サーバー・プログラムを作成する際、以下の内容について注意を払う必要があります。

サービス・スレッドとユーザー作成スレッドの間で**tpsetctxt()**、**tpgetctxt()**などのAPIを使用する場合、以下の内容に注意する必要があります。

- ユーザー作成スレッドの作成と消滅の時期が論理的に妥当である必要があります。サービス・ルーチンでユーザー作成スレッドが作られた後、消滅されない場合、サービス・ルーチンが呼び出されるたびにスレッドが作成され、プログラムに悪影響を与える場合があります。
- ユーザー作成スレッドは基本的にコンテキストを持っていないため、Tmax通信などを実行する場合は、tpsetctxt()関数を使ってコンテキストを共有する必要があります。

- `tpreturn()`または`tpforward()`関数は、サービス・スレッドでのみ呼び出すことができます。ユーザー作成スレッドでは`tpreturn()`を呼び出してはなりません。
- `tpreturn()`または`tpforward()`関数が呼び出される時点で、完了されていない同期型通信、非同期型通信、会話型通信などが存在する場合、サービス呼び出したクライアントに`TPESVCERR`エラーコードを返し、サービスは失敗処理されます。
- `tpreturn()`または`tpforward()`関数が呼び出される時点で、ユーザー作成スレッドが継続してコンテキストを共有してはなりません。開発者は`tpreturn()`を呼び出す前に、ユーザー作成スレッドで`tpsetctxt()`関数を使って`TPNULLCONTEXT`コンテキスト設定を解除するか、または他のコンテキストとして設定するなど、`tpreturn()`を呼び出すサービス・スレッドとのコンテキスト関係を解除する必要があります。そうしなければ、`tpreturn()`はクライアントに`TPESVCERR`エラーコードを返し、サービスは失敗処理されます。
- `tpsetctxt()`関数はサービス・スレッドでは呼び出すことができません。呼び出した場合、`TPEPROTO`エラーコードを返し、失敗処理されます。
- トランザクションのようなコンテキストを共有するスレッドでは、開始スレッドを問わず1つのスレッドでのみコミットやロールバックが可能です。

以下は、前述した内容以外に注意すべき事項です。

- マルチスレッドとマルチコンテキスト・サーバーでサービス・タイムアウトが発生すると、当該サーバー・プロセスは即終了されます。したがって、同時に実行されていた他のサービス要求も一緒に終了することになります。これは、タイムアウトによって特定のスレッドが中断される場合、どのような状態で中断されるかが予想できないからです。たとえば、同期化ソースを先取りした状態であるか、動的メモリーが割り当てられた状態の場合、状況を解決するためのコーディングが複雑になったり、アプリケーションが異常動作したりする可能性があります。
- クライアントで使用する`tpstart()`と`tpend()`関数を使用することができません。
- 自身のサーバー・プロセスでのみ提供するサービスを同期型あるいは非同期型通信で呼び出すことができます。ただし、状況によってはデッドロックが発生する可能性があるため、関連サービスにサービス・タイムアウトを設定するなど、デッドロックが解除できる方法を用意しておく必要があります。

## 7.3.4. サービス処理プログラムの例

クライアント・プログラム、サーバー・プログラム、メイクファイルの例について説明します。

### サーバー・プログラム

以下は、サーバー・プログラムの例です。

```

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

int tpsvrinit(int argc, char **argv)
{
    printf("tpsvrinit()");
    return 1;
}

int tpsvrthrinit(int argc, char **argv)
{
    printf("tpsvrthrinit()");
    return 1;
}

MTOUPPER(TPSVCINFO *msg)
{
    int i;
    printf("MTOUPPER service is started!");
    for (i = 0; i < msg->len; i++)
        msg->data[i] = toupper(msg->data[i]);
    tpreturn(TPSUCCESS,0,msg->data, 0, 0);
}

MTOLOWER(TPSVCINFO *msg)
{
    int i;
    printf("MTOLOWER service is started!");
    for (i = 0; i < msg->len; i++)
        msg->data[i] = tolower(msg->data[i]);
    tpreturn(TPSUCCESS,0,msg->data, 0, 0);
}

int tpsvrthrdone()
{
    printf("tpsvrthrdone()");
    return 1;
}

int tpsvrdone()
{
    printf("tpsvrdone()");
    return 1;
}

```

## メイクファイル

以下は、サーバー・メイクファイルの例です。

```
TARGET = $(COMP_TARGET)
APOBJS = $(TARGET).o
NSDLOBJ = $(TMAXDIR)/lib/sdl.o

LIBS = -lsvrmt -lnodb
OBSJS = $(APOBJS) $(SVCTOBJ)
SVCTOBJ = $(TARGET)_svctab.o

CFLAGS = -I$(TMAXDIR) -D_MCONTEXT

APPDIR = $(TMAXDIR)/appbin
SVCTDIR = $(TMAXDIR)/svct
LIBDIR = $(TMAXDIR)/lib

#
.SUFFIXES : .c

.c.o:
    $(CC) $(CFLAGS) -c $<

#
# server compile
#

$(TARGET): $(OBSJS)
    $(CC) $(CFLAGS) -L$(LIBDIR) -o $(TARGET) $(OBSJS) $(LIBS) $(NSDLOBJ)
    mv $(TARGET) $(APPDIR)/.
    cp $(TARGET).c $(APPDIR)/.
    rm -f $(OBSJS)

$(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)
```

## 環境設定

以下は、環境設定ファイルの例です。

```
*DOMAIN
tmax1  SHMKEY = 77214, MINCLH = 1, MAXCLH = 1,
      TPORTNO = 8888, BLOCKTIME = 30, MAXCACALL = 1024

*NODE
tmax   TMAXDIR = "/home/test/tmax",
      APPDIR = "/home/test/tmax/appbin",
      PATHDIR = "/home/test/tmax/path",
      TLOGDIR = "/home/test/tmax/log/tlog",
      ULOGDIR = "/home/test/tmax/log/ulog",
      SLOGDIR = "/home/test/tmax/log/slog",
      MAXCPC = 200

*SVRGROUP
svg1   NODENAME = tmax

*SERVER
svrmt1 SVGNAME = svg1, SVRTYPE = "STD_MT",
      MIN = 1, MAX = 1,
      CPC = 10, MINTHR = 5, MAXTHR = 10

*SERVICE
MTOUPPER SVRNAME = svrmt1, SVCTIME = 20
MTOLOWER SVRNAME = svrmt1, SVCTIME = 20
```

### 7.3.5. コンテキスト共有プログラムの例

クライアントから「MSERVICE」サービスを呼び出します。サービス・ルーチンではユーザー作成スレッドを作成し、ユーザー・スレッドがサービス・スレッドのコンテキストを共有すると同時にtpcall()を介してサービスを呼び出す方式で動作します。

## サーバー・プログラム

以下は、サーバー・プログラムの例です。

```
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <usrinc/tmaxapi.h>

void * THREAD(void *arg);
```

```

typedef struct {
    int ctxtid;
    TPSVCINFO *svcinfo;
} param_t;

int testcall(char *service, char *msg, long flags)
{
    char *sndbuf, *rcvbuf;
    long sndlen, rcvlen;

    sndlen = strlen(msg);
    if((sndbuf = (char *) tmalloc("STRING", NULL, sndlen)) == NULL) {
        printf("Error allocating send buffer, [tperrno:%d]", tperrno);
        return -1;
    }
    if((rcvbuf = (char *) tmalloc("STRING", NULL, 0)) == NULL) {
        printf("Error allocating recv buffer, [tperrno:%d]", tperrno);
        tpfree(sndbuf);
        return -1;
    }

    strcpy(sndbuf, msg);
    if(tpcall(service, sndbuf, sndlen, (char **)&rcvbuf, &rcvlen, flags) ==
-1)
        printf("tpcall(%s) failed, [tperrno:%d, tpurcode:%d]", service, tperrno,
                tpurcode);
    else
        printf("tpcall(%s) success, [rcvbuf:%s]", service, rcvbuf);

    tpfree(sndbuf);
    tpfree(rcvbuf);
    return 0;
}

MTOUPPER(TPSVCINFO *svcinfo)
{
    int i;

    printf("MTOUPPER service is started! [len:%d, data:%s]\n", svcinfo->len,
            svcinfo->data);

    for (i = 0; i < svcinfo->len; i++)
        svcinfo->data[i] = toupper(svcinfo->data[i]);
}

```



```

        sleep(1);
        printf("MTOUPPER service is finished!\n");
        tpreturn(TPSUCCESS, 0, svcinfo->data, 0, 0);
    }

MTOLOWER(TPSVCINFO *svcinfo)
{
    int i;.

    printf("MTOLOWER service is started! [len:%d, data:%s]\n", svcinfo->len,

           svcinfo->data);
    for (i = 0; i < svcinfo->len; i++)
        svcinfo->data[i] = tolower(svcinfo->data[i]);

    sleep(1);
    printf("MTOLOWER service is finished!\n");
    tpreturn(TPSUCCESS, 0, (char *)svcinfo->data, 0, 0);
}

MSERVICE(TPSVCINFO *svcinfo)
{
    pthread_t tid;
    param_t param;

    printf("MSERVICE service is started!");

    tpgetctx(&param.ctxtid, 0);
    param.svcinfo = svcinfo;
    pthread_create(&tid, NULL, THREAD, &param);

    testcall("MTOLOWER", svcinfo->data, 0);

    pthread_join(tid, NULL);
    printf("MSERVICE service is finished!");
    tpreturn(TPSUCCESS, 0, svcinfo->data, 0L, 0);
}

void *THREAD(void *arg)
{
    param_t *param;
    TPSVCINFO *svcinfo;

    param = (param_t *)arg;
    svcinfo = param->svcinfo;

    if (tpsetctx(param->ctxtid, 0) == -1) {

```

```

        printf("tpsetctxt(%d) failed, [tperrno:%d]", param->ctxtid, tperrno);

        return NULL;
    }

    testcall("MTOUPPER", svcinfo->data, 0);

    if (tpsetctxt(TPNULLEXEC, 0) == -1) {
        printf("tpsetctxt(TPNULLEXEC) failed, [tperrno:%d]", tperrno);
        return NULL;
    }
    return NULL;
}

int tpsvrinit(int argc, char *argv[])
{
    printf("do tpsvrinit()");
    return 1;
}

int tpsvrthrinit(int argc, char *argv[])
{
    printf("do tpsvrthrinit()");
    return 1;
}

int tpsvrthrdone(void)
{
    printf("do tpsvrthrdone()");
    return 1;
}

int tpsvrdone(void)
{
    printf("do tpsvrdone()");
    return 1;
}

```

## メイクファイル

[7.3.4節「メイクファイル」](#)を参照してください。

## 環境設定

以下は、環境設定ファイルの例です。

```
*DOMAIN
tmax1  SHMKEY = 77214, MINCLH = 1, MAXCLH = 1,
      TPORTNO = 8888, BLOCKTIME = 30, MAXCACALL = 1024

*NODE
tmax   TMAXDIR = "/home/test/tmax",
      APPDIR = "/home/test/tmax/appbin",
      PATHDIR = "/home/test/tmax/path",
      TLOGDIR = "/home/test/tmax/log/tlog",
      ULOGDIR = "/home/test/tmax/log/ulog",
      SLOGDIR = "/home/test/tmax/log/slog",
      MAXCPC = 200

*SVRGROUP
svg1   NODENAME = tmax

*SERVER
svrmt2 SVGNAME = svg1, SVRTYPE = "STD_MT",
      MIN = 1, MAX = 1,
      CPC = 10, MINTHR = 5, MAXTHR = 10

*SERVICE
MSERVICE SVRNAME = svrmt2, SVCTIME = 10
MTOUPPER SVRNAME = svrmt2
MTOLOWER SVRNAME = svrmt2
```



# 第8章 セキュリティー・システム

本章では、Tmaxがサポートするセキュリティー・システムについて説明します。

## 8.1. 概要

Tmaxはシステム接続制御、ユーザー認証、サービス・アクセス制御の3段階でセキュリティー機能を提供しています。

段階別のセキュリティー機能は、Tmax環境ファイルのDOMAINセクションの**SECURITY**項目で設定します。**SECURITY**項目の設定値に応じて、以下のようにセキュリティーが設定されます。

設定値	説明
DOMAIN_SEC	システム接続制御
USER_AUTH	ユーザー認証
ACL   MANDATORY	サービス・アクセス制御
NO_SECURITY	セキュリティー機能を適用しない

### 参考

各ファイル(グループ、ユーザー、acl)の作成方法とコマンドについては、『Tmaxリファレンスガイド』を参照してください。

## 8.2. 1段階セキュリティー(システム接続制御)

1段階セキュリティーはシステム接続セキュリティーで、ユーザー・クライアントが最初にTmaxシステムに接続するとき、接続を制限します。Tmaxシステムに対する単一パスワードを設定することで、DOMAINセクションの**OWNER**項目に定義されているアカウントに対するパスワードになります。

Tmaxシステム設定の前にパスワードは**mkpw**ユーティリティで設定します。このセキュリティーが設定されると、クライアントはtpstart()関数を呼び出すとき、TPSTART\_T構造体の**dompwd**項目に当該パスワードを登録します。パスワードが正しい場合のみ、クライアントはTmaxシステム接続に成功します。設定値が「NO\_SECURITY」であれば、dompwd項目に値を設定する必要がなく、NULL値が返されます。

以下は、1段階セキュリティーを使用するTmax環境ファイルの例です。

#システム認証の場合、SECURITY項目に"DOMAIN\_SEC"と設定します。

```
*DOMAIN
res1      SHMKEY = 66999, MAXUSER = 256,
          SECURITY = "DOMAIN_SEC", OWNER = tmax

*NODE
Tmax      TMAXDIR = "/home/tmax",
          APPDIR = "/home/tmax/appbin"

*SVRGROUP
svg1      NODENAME = tmax

*SERVER
upper     SVGNAME = svg1, RESTART = Y, MAXRSTART = 3

*SERVICE
TOUPPER   SVRNAME = upper
TOLOWER   SVRNAME = upper, PRIO = 100
```

以下は、TPSTART\_T構造体のdompwdを使用してパスワードを設定する例です。

```
...
main(int argc, char *argv[])
{
    ...
    TPSTART_T *tpinfo;
    ...
    if ((tpinfo = (TPSTART_T *)tpalloc("TPSTART", NULL, sizeof(TPSTART_T)))== NULL){
        error processing routine
    }
    strcpy(tpinfo->dompwd, "tmax1234");
    if (tpstart(tpinfo) == -1){
        error processing routine
    }
    ...
}
```

## 8.3. 2段階セキュリティ(ユーザー認証)

2段階セキュリティはユーザー認証セキュリティで、Tmaxシステムに認証されたユーザーのみTmaxシステムへの接続を許可します。

クライアントはtpstart()関数を呼び出してTmaxシステムに接続しようとするとき、TPSTART\_T構造体のusernameとusrpwd項目を登録します。usernameはTmaxシステムに認証されたユーザー・アカウントであ

り、usrpwdはパスワードです。ユーザー・アカウントとパスワードは設定前に**mkpw**ユーティリティを使って設定する必要があります。このようなユーザー認証セキュリティが設定された場合、usernameとusrpwdがTmaxシステムに認証されたものと確認されないと、Tmaxシステムに接続できません。

ユーザー認証セキュリティは、システム接続制御セキュリティの次の段階で、システム接続制御セキュリティを含みます。クライアントはdompwd項目も正しく登録する必要があります。dompwdもmkpwを使って設定します。

---

#### 注

パスワード・ファイルは必ずTmaxシステムを設定する前に作成および更新します。

---

以下は、2段階セキュリティを使用するTmax環境ファイルの例です。

```
#ユーザー認証の場合、SECURITY項目に"USER_AUTH"と設定します。
*DOMAIN
res1      SHMKEY = 66999, MAXUSER = 256,
          SECURITY = "USER_AUTH", OWNER = tmax

*NODE
tmax      TMAXDIR = "/home/tmax",
          APPDIR = "/home/tmax/appbin"

*SVRGROUP
svg1      NODENAME = tmax

*SERVER
upper     SVGNAME = svg1, RESTART = Y, AXRSTART = 3

*SERVICE
TOUPPER   SVRNAME = upper
TOLOWER   SVRNAME = upper, PRIO = 100
```

以下は、TPSTART\_T構造体のusername、usrpwdを使用してパスワードを設定する例です。

```
...
main(int argc, char *argv[])
{
    ...
    TPSTART_T *tpinfo;
    if((tpinfo = (TPSTART_T *)tpalloc("TPSTART", NULL, sizeof(TPSTART_T))) == NULL){

        error processing routine
    }
    strcpy(tpinfo->dompwd, "tmax1234");
```

```

strcpy(tpinfo->usrname, "gdhong") ;
strcpy(tpinfo->usrpwd, "hong0000") ;

if (tpstart(tpinfo) == -1){
    error processing routine
}
...
}

```

## 8.4. 3段階セキュリティ(サービス・アクセス制御)

3段階セキュリティはサービス・アクセス制御で、Tmaxシステムに接続したユーザーのうちサービス別にユーザー・アクセスを制限できる機能です。

サービス・アクセス権限制御はユーザー・グループ別に行われます。1つのサービスは、そのサービスに対してアクセス権限が許可されたグループに該当するユーザーのみアクセスできます。したがってこの機能を使用するには、グループファイルを作成しなければならず、当該グループに属するユーザー・ファイルが必要です。そして、サービス別にアクセスできるユーザー・グループを指定する**ACL**というファイルも必要になります。当該機能を使用するには、DOMAINセクションの**SECURITY**項目に**ACL**または**MANDATORY**値を設定します。

3段階セキュリティは、1段階システム接続制御、2段階ユーザー認証までをすべて含んでいます。したがってSECURITY項目がACL(またはMANDATORY)で設定されている場合、TPSTART\_T構造体にdompwd、username、usrpwdフィールドを使用し、1、2段階認証を経てTmaxシステムに正常に接続した場合のみサービス・アクセス権限を使用できるようになります。

以下は、3段階セキュリティを使用するTmax環境ファイルの例です。

```

*DOMAIN
tmax1          SHMKEY = 78350,
               TPORTNO = 8350, SECURITY = ACL, OWNER = starbj0, RACPORT = 3155

*NODE
tmaxh4         TMAXDIR = "/EMC01/starbj81/tmax",
               APPDIR  = "/EMC01/starbj81/tmax/appbin"

tmaxh2         TMAXDIR = "/data1/starbj81/tmax",
               APPDIR  = "/data1/starbj81/tmax/appbin",

*SVRGROUP
svg1           NODENAME = "tmaxh4", COUSIN = "svg2", LOAD = 1
svg2           NODENAME = "tmaxh2", LOAD = 5
svg3           NODENAME = "tmaxh4"

```



```

*SERVER
svr01001      SVGNAME = svg1
svr_ucs1      SVGNAME = svg3, CLOPT = "-u 35", SVRTYPE = UCS, MIN = 1, MAX = 2
svr_ucs2      SVGNAME = svg3, CLOPT = "-u 37", SVRTYPE = UCS, MIN = 1, MAX = 2

*SERVICE
TOUPPER1      SVRNAME = svr01001
TOUPPER2      SVRNAME = svr01001
LOGIN1        SVRNAME = svr_ucs1
LOGIN2        SVRNAME = svr_ucs2

```

### <グループ・ファイル>

```

grp0:x:1001
grp1:x:1
grp2:x:2
grp3:x:3
grp4:x:4
grp5:x:5

```

### <ユーザー・ファイル>

```

starbj0:1002:1
starbj1:1:1
starbj2:2:2
starbj3:3:3
starbj4:4:4
starbj5:5:5

```

### <ACLファイル>

```

TOUPPER1:SERVICE:1
TOUPPER3:SERVICE:5
TOUPPER5:SERVICE:5

```

### <クライアント・プログラム>

```

int main()
{
    ...
    strcpy(tpinfo->usrname, "starbj1");
    strcpy(tpinfo->dcompwd, "starbj0");
    strcpy(tpinfo->usrpwd, "starbj1");
    ...
}

```

```
if(tpcall("TOUPPER1", sndbuf, 0, &rcvbuf, &rcvlen, 0)==-1){  
    error routine..  
}
```

# 第9章 クライアントAPI

本章では、クライアントの開発に使われるAPIについて説明します。

## 9.1. 概要

以下は、クライアントが使用する関数についての説明です。関数の詳細については、『Tmax リファレンスガイド』を参照してください。

- 接続および解除

関数	説明
<a href="#">tpstart</a>	クライアントをTmaxシステムに接続します
<a href="#">tpend</a>	クライアントとTmaxシステムの接続を解除します

- 同期型通信

関数	説明
<a href="#">tpcall</a>	クライアントがサービスを要求して応答が受信される待機する同期型通信関数です

- 非同期型通信

関数	説明
<a href="#">tpacall</a>	クライアントからサービスを要求した後、直ちに返します
<a href="#">tpgetrply</a>	tpacallで送った要求に対する応答バッファのデータを受信します
<a href="#">tpcancel</a>	サーバーとクライアントの応答を取り消します

- 会話型通信

関数	説明
<a href="#">tpconnect</a>	接続を設定して通信を開始します
<a href="#">tpsend</a>	接続コントロールを持つ側がメッセージを送信するときに使用します
<a href="#">tprecv</a>	接続が設定された後、接続コントロールを持たない側でデータを受信したい場合に使用します
<a href="#">tpdiscon</a>	サーバーとクライアント間の会話型通信の接続を終了します

- 非要求メッセージ処理

関数	説明
<a href="#">tpsetunsol</a>	クライアントで使われる関数で、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="#">tpgetunsol</a>	クライアントの要求なしに一方的に渡されたメッセージを処理します

- タイムアウト変更

関数	説明
<a href="#">tpset_timeout</a>	サーバーとクライアントで使われる関数で、ブロッキング・タイムアウト時間を設定します

- バッファ管理

関数	説明
<a href="#">tpalloc</a>	サーバーとクライアントでバッファを割り当てます
<a href="#">tprealloc</a>	サーバーとクライアントでバッファを再割り当てします
<a href="#">tpfree</a>	サーバーとクライアントでバッファに割り当てられたメモリーを解除します
<a href="#">tptypes</a>	サーバーとクライアントで使われる関数で、バッファのタイプと下位タイプに対する情報を提供します

- トランザクション管理

関数	説明
<a href="#">tx_begin</a>	サーバーとクライアントで使用する関数で、トランザクションを開始します
<a href="#">tx_commit</a>	サーバーとクライアントで使用する関数で、トランザクションをコミットします
<a href="#">tx_rollback</a>	サーバーとクライアントで使用する関数で、トランザクションをロールバックします
<a href="#">tx_set_transaction_timeout</a>	サーバーとクライアントでtransaction_timeoutを設定する関数で、transaction_timeoutの特性をタイムアウト値で設定します
<a href="#">tx_set_transaction_control</a>	サーバーとクライアントでtransaction_controlの特性をコントロール値で設定します
<a href="#">tx_set_commit_return</a>	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
<a href="#">tx_info</a>	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します

- RQシステム

関数	説明
<a href="#">tpenq</a>	サーバーとクライアントでRQにデータを保存します
<a href="#">tpdeq</a>	サーバーとクライアントでRQからデータをロードします
<a href="#">tpqstat</a>	サーバーとクライアントで使われる関数で、RQに保存されているデータの統計を要求します
<a href="#">tpextsvcname</a>	サーバーとクライアントでRQからtpdeqにデータを読み込んだ場合、当該データのサービス名を知りたい場合に使用します

- イベントを活用するAPI

関数	説明
<a href="#">tpsubscribe</a>	サーバーとクライアントで使われる関数で、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="#">tpunsubscribe</a>	サーバーとクライアントで使用する関数で、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="#">tppost</a>	サーバーとクライアントで特定イベントを発生させてメッセージを転送します

- ブロード・キャストとマルチ・キャスト

関数	説明
<a href="#">tpbroadcast</a>	クライアントやサーバーが他のクライアントに非要求メッセージを送る場合に使用します

- Windows環境プログラミング

– tmaxmt.dll

関数	説明
<a href="#">WinTmaxAcall</a>	Tmaxに接続してサービスを要求し、結果を受けて再度目的のWindowのプロシージャに渡した後、接続を終了します
<a href="#">WinTmaxAcall2</a>	Tmaxに接続してサービスを要求し、応答は指定したコールバック関数で受け取ります

– WinTmax.dll

関数	説明
<a href="#">WinTmaxStart</a>	メッセージを処理するスレッドを作成し、使用メモリーを初期化します
<a href="#">WinTmaxEnd</a>	メッセージを処理するスレッドを終了し、使用メモリーを解除します
<a href="#">WinTmaxSetContext</a>	Tmaxに到着したデータを送るWindowsとメッセージ番号を指定します

関数	説明
<a href="#">WinTmaxSend</a>	クライアントでサービスを呼び出し、即刻他の業務が処理できるようにします

- マルチ・スレッドとマルチ・コンテキストAPI

関数	説明
<a href="#">tpgetctxt</a>	現在のテキストを返します
<a href="#">tpsetctxt</a>	現在のテキストを設定します

## 9.2. 接続および解除

Tmaxの接続および解除に使われる関数について説明します。

### 9.2.1. 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	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。  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

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、username、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="#">9.6.1. 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()

## 9.2.2. 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。tperrno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、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]	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()

## 9.3. 同期型通信

同期型通信に使われる関数について説明します。

### 9.3.1. 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。**tpacall()**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> <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()

## 9.4. 非同期型通信

非同期型通信に使われる関数について説明します。

### 9.4.1. 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が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



パラメータ	説明
len	<p>送信されるデータ長です</p> <ul style="list-style-type: none"> <li>– 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</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</li> </ul>
flags	<p>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</p> <p>flagsに使用可能な値は以下のとおりです</p> <ul style="list-style-type: none"> <li>– TPBLOCK <p>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</p> </li> <li>– TPNOTRAN <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> </li> <li>– TPNOREPLY <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</p> </li> </ul>

パラメータ	説明
	<p>を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

エラーコード	説明
[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[TPEIME]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()

### 9.4.2. 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ー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ーに受信されるように、バッファー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ーの総サイズより大きい場合、lenが該当バッファー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ー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  – TPGETANY  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

パラメータ	説明
	<p>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> <p>– TPNOCHANGE</p> <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> <p>– TPNOBLOCK</p> <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> <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> <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、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。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
[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

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>
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()

### 9.4.3. 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・トランザクションと関連しています。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 *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()

## 9.5. 会話型通信

会話型通信に使われる関数について説明します。

### 9.5.1. 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	<p>渡すデータ長を指定します</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</li> </ul>

パラメータ	説明
	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
	– TPSIGRSTRT 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

- 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

エラーコード	説明
	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
[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(), tpdicon(), tprecv(), tpsend()

## 9.5.2. 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。tpconnect()あるいはTPSVCINFO媒介変数から返される記述子です
data	tpalloc()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になれます。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	flagsで使用可能な値は以下のとおりです  – 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  – 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  – TPRECVONLY  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

パラメータ	説明
	<p>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> <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>
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> <p>– TPEV_DISCONIMM</p> <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()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</p> <p>– TREV_SVCERR</p> <p>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</p> <p>– TREV_SVCFAIL</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。rvalはTPFAILであり、dataはNULLで実行されたことを通知します</p>

#### ● 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(), tpdiscn(), tprecv(), tpreturn()

### 9.5.3. 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ー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



パラメータ	説明
len	<p>データの長さです。</p> <p>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。</p> <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</p> </li> </ul>

パラメータ	説明
	<p>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> <p>– TPEV_SENDOONLY</p> <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すが、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
[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(), tpdiscon(), tpsend()

## 9.5.4. tpdiscon

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。

tpdiscon()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。

tpdiscon()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。tpdiscon()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。tpdiscon()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  tpdiscon (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、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,TPRECVOONLY);

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()

## 9.6. 非要求メッセージの処理

非要求メッセージの処理関数は、無条件に、あるいは一方的に送るメッセージを処理するための関数です。サーバーから送られる一方的なメッセージ(tpbroadcast())を常にすべてのクライアントが受信できるわけではありません。非要求メッセージを受信するには、まずクライアントがTmaxに接続されていて、Tmax向けに非要求メッセージを受信するとの情報を渡さなければなりません。

tpstart()を呼び出すときに使用するTPSTART\_T構造体のフラグ値をTPUNSOL\_POLLやTPUNSOL\_HNDに設定すると、サーバーから送られる一方的なデータをTmaxが処理します。

## 9.6.1. 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関数ポインターの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ーは、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ーを指示します。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ー・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=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, (char **)&buf, &len, TPNOFLAGS);
    if (ret== -1) { error processing }

    data process...

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



```
    tpend();
}
```

● 関連関数

tpstart(), tpend(), tpgetunsol()

9.6.2. 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 <tmaxapi.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_TPBROADCAST、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ー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	メッセージのブロッキング処理可否を決定します。  flagsで使用可能な値は以下のとおりです  – TPBLOCK  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  – TPNOCHANGE  受信した応答バッファータと*dataが指すバッファータのタイプが異なる場合、受信者が認識できる限度内で、*dataのバッファータタイプは受信した応答バッファータタイプ

パラメータ	説明
	<p>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> <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> <p>– TPSIGRSTRT</p> <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> <p>– TPGETANY</p> <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()

## 9.7. タイムアウトの変更

タイムアウトを変更するときに使われる関数について説明します。

## 9.7.1. 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();
}

```

## 9.7.2. 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);
}
```

# 9.8. バッファの管理

相互異なるハードウェアや運用体制において、メモリー割り当て方式とデータタイプのサイズの違いによって、送信したデータと異なるデータ値を持つことがあります。したがって、一般的なクライアント/サーバー環境で異種間の通信は主に文字型に変換して通信することになります。これはネットワーク負荷を増加させる原因にもなります。

バッファ管理に使われるAPIについて説明します。

## 9.8.1. 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>
subtype	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・

パラメータ	説明
	<p>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()

## 9.8.2. 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ーを返します。

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、返されたポインター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ーに対するポインターです
size	割り当てるバッファーのサイズです

- 戻り値

戻り値	説明
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。適切な型付きバッファー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。tperno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、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が指すバッファーが、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が指すバッファーがtpalloc()で割り当てられていないバッファーです
[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。

---

### 9.8.3. 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にはできません。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 (message2==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。

---

## 9.8.4. 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。tperrnoにエラーコードが設定されます

- エラー

tpypes()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()

## 9.9. トランザクションの管理

トランザクション管理に関連する関数について説明します。

### 9.9.1. 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「[9.9.2. tx\\_commit](#)」と「[9.9.4. 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加しているため、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。参加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, TPRECVOONLY);
    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()

## 9.9.2. 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「[9.9.6. 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()

### 9.9.3. 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()

## 9.9.4. 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「[9.9.6. 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, TPRECVRONLY);
    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()

## 9.9.5. 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()

## 9.9.6. 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()

9.9.7. 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。

設定値	説明
	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()

## 9.10. RQシステム

本節では、RQで使用する関数について説明します。Tmaxシステムは信頼性の高いキュー(Reliable Queue)のためにRQを提供しますが、RQにデータをEnqueue/Dequeueするキュー管理システムとしてENQSVRが動作します(Windows NTまたはWindows 2000環境では使用できません)。

システム障害やエラーによりサービスが不可能な状態でもキューに保存されているデータは整合性が保証されます。主に複数のサーバーがWAN環境で構築されているか業務の集中により処理量が多い場合、RQを利用するとデータの信頼性が保証されつつ、業務を処理できます。

### 9.10.1. 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。

パラメータ	説明
	<p>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tpdeq()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</p>
data	<p>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</p>
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>flagsに使用可能な値は以下のとおりです</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()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、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
[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), TPRS);
    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()

## 9.10.2. 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ー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ーに受信されるようバッファーのサイズを増加させます。

パラメータ	説明
	<p>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。</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>– 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</li> <li>– TPFUNC 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</li> <li>– TPBLOCK 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</li> <li>– TPNOTIME 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</li> <li>– 0(zero) 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</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

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), 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

tpenq(), tpqstat()

### 9.10.3. 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	対象データのタイプです。  flagsで使用可能な値は以下のとおりです  – 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  – 1(TMAX_FAIL_QUEUE) : _fail queueのデータ統計を求める際に使用し ます  – 2(TMAX_REQ_QUEUE) : _request queueのデータ統計を求める際に使用 します  – 3(TMAX_RPLY_QUEUE) : _reply queueのデータ統計を求める際に使用 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

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), TPRQS);
    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), TPRQS);
    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()

## 9.10.4. 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
[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), TPRS);
    if (ret==-1) { error processing }
    data process....

    ret=tpdeq("RQ", "SERVICE", (char **)&buf, (long *)&len, TPRS);
    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()

## 9.11. イベントを活用するAPI

Tmaxはイベントを使って多数のクライアントだけでなくサービス・ルーチンにもメッセージを送ることができます。すべてのクライアントとサービス・ルーチンは開発者が任意でイベントに加入するか取り消すことができ、またイベントを発生させることもできます。そのほか、1つのイベントに重複加入するか、同時に複数のイベントに加入することもできます。

---

### 参考

発生したイベントによるイベント・データの転送とそれに伴う結果はトランザクションの範囲内に入らないので、ご注意ください。

---

### 9.11.1. 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()

## 9.11.2. 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()

## 9.11.3. 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  – 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  – dataがNULLの場合、lenは無視されます
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。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

エラーコード	説明
[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()

## 9.12. ブロードキャストとマルチキャスト

Tmaxは多様な方法でクライアントにデータを渡せます。管理やその他の開発の目的で多数のクライアントにデータを送信する必要がある場合、すべてのクライアントのIDを管理し、それぞれに対してメッセージを送るのも1つの方法ではありますが、システムの同時ユーザーが多い場合は、効率的な方法ではありません。

多数のクライアントをいくつかのグループに区分できる場合、1つのAPIを利用して当該するすべてのクライアントにメッセージを送ることができます。Tmaxはクライアントの接続情報を活用するかイベントを利用する方法としてブロードキャストおよびマルチキャストを実装しています。2つの方式は、どちらも一般的な意味のブロードキャストとマルチキャストが両方とも実行できますが、区分の便宜上、前者をブロードキャストAPI、後者をマルチキャストAPIといいます。

### 9.12.1. 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 *username, char *cltname,
                char *data, long len, long flags)
```

- パラメータ

パラメータ	説明
nodename,	対象クライアントの選択に使用される論理的な名前、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

パラメータ	説明
usrname, cltname	<p>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。</p> <p>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</p>
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	<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NULLの場合もlenは無視されます</li> </ul>
flags	<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>– 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</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</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> </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

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->username, "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()

## 9.13. 環境プログラム

開発されたサービスが一般ユーザーのためのものであれば、クライアント・プログラムは主に個人PCで多く使われるWindows環境をベースに開発されます。そのため、Tmaxは多様な開発ツールのためのインターフェースを提供します。また、開発者の利便性向上のために、多様なタイプのライブラリーを提供します。ライブラリーによっては追加されるAPI、または代替されるAPIが存在し得ます。

---

### 参考

Tmaxがサポートする開発ツールは、PowerBuilder、Delphi、VC++、BC++、VB、VB .Net、C# .Netです。各開発ツールの使用方法については、『Tmax プログラミングガイド(4GL)』を参照してください。

---

以下は、対応ライブラリーについての説明です。

ライブラリー	説明
tmax.lib (dll)	VC++、Delphi、C#で使われます(cdecl)
tmax4gl.lib (dll)	BC++、PowerBuilder、VB、VB.Netで使われます(stdcall)
tmaxmt.lib (dll)	マルチ・スレッド形態をサポートするためのライブラリーです(cdecl)
Wintmax.lib (dll)	マルチ・ウィンドウをサポートするためのライブラリーです(cdecl)
tmaxce.lib (dll)	ウィンドウceのためのライブラリーです(cdecl)

- tmaxmt.dll

Windowsプログラミング環境で使われるMessage-Driven方式のサービス要求のために2つのAPIを追加しました。2つのAPIはほぼ同じであり、指定したWindowsにメッセージを渡すか、指定したコールバック関数に渡すかの違いだけがあるのです。APIを実行するたびにスレッドを1つずつ作成してメッセージを処理します。ヘッダー・ファイルはtmaxpi.hを使用します。

– WinTmaxAcall()

– WinTmaxAcall2()

- WinTmax.dll

プログラミング環境で使われるMessage-Driven方式のサービス要求のためのAPIを追加しました。独立したスレッドを利用してメッセージを処理するので、tpstart()やtpend()の代わりにWinTmaxStart()、WinTmaxEnd()を使用します。

– WinTmaxStart()

- WinTmaxEnd()
- WinTmaxSetContext ()
- WinTmaxSend ()

tmaxmt.dllと類似した方式で動作しますが、Tmaxシステムに自動で接続/解除作業をせず、API1つ当たり1つのスレッドを作成する代わりに1つのスレッドですべてのメッセージを処理します。エラーメッセージと非要求メッセージを処理するために別途Windowを指定しないとメッセージは無視されます。

デバッグまたは管理の目的でMAX\_DEBUG環境変数に指定されたファイルにログを残せます。ヘッダー・ファイルはWinTmax.hを使用します。

### 9.13.1. 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。



パラメータ	説明
	<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 <p>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</p> </li> <li>– TPNOTRAN <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> </li> <li>– TPNOREPLY <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> </li> <li>– TPNOBLOCK <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> </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> </li> </ul>

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、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

エラーコード	説明
[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()

## 9.13.2. 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

パラメータ	説明
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	<p>tpacall()のフラグをそのまま使します。</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 <p>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</p> </li> <li>– TPNOTRAN <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> </li> <li>– TPNOREPLY <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> </li> <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・</p> </li> </ul>

パラメータ	説明
	<p>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、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

WinTmaxAcall2()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、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

エラー・コード	説明
	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 *svcinfo;
    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()

### 9.13.3. 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、サービスを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[「9.2.1. 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()

## 9.13.4. 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。tperno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、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
[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()

## 9.13.5. 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WPARAMIにはmsgが対応し、LPARAMI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
slot	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。  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。  以下は、スロットに設定できる値です － -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 － 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

#### ● 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。tperrno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、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
[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 CMaxGwView::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()

## 9.13.6. 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 = WindTmaxSend(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()で使用できるフラグ値は以下のとおりです。

フラグ値	説明
TPBLOCK	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。

フラグ値	説明
	<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	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
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>
TPNOBLOCK	<p>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</p>
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
[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()

## 9.14. マルチ・スレッドとマルチ・コンテキスト

マルチ・スレッド/マルチ・コンテキスト関連の関数について説明します。マルチ・スレッド/マルチ・コンテキストのサーバー・ライブラリーとクライアント・ライブラリーはマルチ・コンテキスト方式に差があります。

### 9.14.1. 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。

tpgetctx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 tpgetctx(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

tpgetctx()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ー値であったり、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 *svcinfol;
} param_t;

MSERVICE(TPSVCINFO *svcinfol)
{
    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()

## 9.14.2. tpsetctxt

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、tpgetctxt()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、tpgetrply()を正常に行うためには、tpgetrply()を実行する時点に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tpgetrply()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持っており、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。tpsetctx()関数のctxidパラメータは、サービス・スレッドでtpgetctx()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 tpsetctx(int ctxid, long flags)
```

## ● パラメータ

パラメータ	説明
ctxid	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、tpstart()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、ctxtidに0またはTPINVALIDCONTEXTが設定された場合や、flags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、tpstart()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、tpstart()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]	ctxtid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
[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!");
tpgetctx(&param.ctxid, 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 (tpsetctx(param->ctxid, TPNOFLAGS) == -1) {
        printf("tpsetctx(%d) failed, [tperrno:%d]", param->ctxid, tperrno);
        return NULL;
    }

    tpcall("MTOUPPER", sndbuf, 0, &rcvbuf, &rcvlen, TPNOFLAGS);

    if (tpsetctx(TPNULLCONTEXT, TPNOFLAGS) == -1) {
        printf("tpsetctx(TPNULLCONTEXT) failed, [tperrno:%d]", tperrno);
        return NULL;
    }

    return NULL;
}

```

- 関連関数

tpgetctx()

# 第10章 サーバーAPI

本章では、Tmaxサーバーで使用する関数について説明します。ATMIでサーバー・プログラムを作成するときに必要な関数についても紹介します。サーバーをクライアントとして使用することもできるため、クライアントで使われる関数も使用できます。

## 10.1. TCS

TCS方式のサーバー・プログラムで使われる関数について説明します。

- サービス完了関連関数

`tpreturn()`はクライアントに応答を送り、`tpforward()`は必要に応じて他のサーバーへサービス要求を送ってサービスを終了します。

関数	説明
<code>tpreturn</code>	サービス要求に対する応答をクライアントに送り、サービス・ルーチンを終了します
<code>tpforward</code>	別のサービスを処理するために他のサーバーへサービスを依頼します。アプリケーションを解除します

- サーバー初期化及び終了関連関数

アプリケーションと接続するデータベースのオープンとクローズを担当し、コマンドライン・オプション処理などの機能を提供します。このサブルーチンを開発者が作成しない場合はTmaxから基本的に提供されます。

関数	説明
<code>tpsvrinit</code>	サーバーを初期化します
<code>tpsvrdone</code>	サーバーの初期化を解除します

- マルチ・スレッド/マルチ・コンテキスト

関数	説明
<code>tpsetctxt</code>	関数を呼び出すスレッドのコンテキストを一番目のパラメーターのIDに設定します
<code>tpgetctxt</code>	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を一番目のパラメーターに返します

- 非要求関連関数

Tmaxでサーバーは、クライアントが要求していないメッセージをクライアントに一方的に送ることができます。これはTmaxに接続中のクライアントに知らせたいメッセージを送信する場合に使われます。クライアントが非要求メッセージを受信できるためには、Tmaxに接続している必要があり、またフラグ値を設定しなければなりません。

関数	説明
<a href="#">tpsendtocli</a>	クライアントの要求がなくてもサーバーでクライアントが事前登録した要求メッセージを自動送信します
<a href="#">tpgetclid</a>	Tmaxシステムに接続しているクライアントのIDを返します。IDはtpsendtocli()に使用されます
<a href="#">tpchkclid</a>	該当サーバー・プロセスが位置するノードにクライアントIDに該当するクライアントが接続しているかどうかをチェックします

---

## 参考

関数と関連関数の詳細については、『Tmaxリファレンスガイド』を参照してください。

---

## サービス・ルーチン引数

サーバー・プログラムはTmaxが提供するmain()とサービス・ルーチンで構成されています。main()はデータベースの接続及び解除、コマンドラインのオプション処理などの機能をするルーチンで構成され、サービス・ルーチンは実際にクライアントの要求を受けて業務を処理するルーチンで構成されています。

サーバーmain()はクライアントの要求を受けて該当サービス・ルーチンをTPSVCINFO構造体に呼び出して要求を処理させます。TPSVCINFO構造体はサービスを要求したクライアントの情報と処理するデータの情報を格納しています。

TPSVCINFO構造体はatmi.hヘッダー・ファイルに宣言されており、構成要素は以下のとおりです。

```
#define XATMI_SERVICE_NAME_LENGTH 16

struct tpsvcinfo {
    char name[XATMI_SERVICE_NAME_LENGTH]; /* 要求されたサービス名 */
    char *data;                             /* 要求データ */
    long len;                               /* 要求データの長さ */
    long flags;                             /* サービス属性 */
    Int cd;                                 /* 接続記述子 */
};

typedef struct tpsvcinfo TPSVCINFO
```



メンバー	説明
name	クライアントが要求したサービス・ルーチン名です
data	クライアントが要求したデータを受信するために使用されるバッファです。サーバーmain()の中でtpalloc()で事前に割り当てられます
len	要求したデータの長さを示します
flags	トランザクション状態にあるかどうか、または呼び側が応答を必ず要求しているかどうかなどをサービスに知らせます。例えば、フラグがTPTRANであれば、サービスがトランザクション・モードであることを、TPNOTRANであれば、現在のトランザクションに参加できることを知らせます
cd	接続記述子で、どのクライアントに応答するかを設定します。サーバーはクライアントが要求したサービス进行处理するために事前にバッファをmain()で割り当てるため、tpreturn()、tpforward()で通信するときにTPSVCINFOデータの使用を勧めます。TPSVCINFOのデータにアクセスするとき、サーバー・プログラムとクライアント・プログラムのバッファ・タイプが同一でなければなりません

## クライアント・プログラム

以下はクライアント・プログラムの例です。

```
#include <usrinc/atmi.h>
. . .
main()
{
    struct strdata *cltdata;
    if ((cltdata = (struct strdata *)tpalloc("STRUCT", "strdata",
        sizeof(struct strdata)) ) == NULL){
        error processing routine
    }
    . . .
    if ((tpcall ("SEL_SVC", cltdata, 0, (char **)&cltdata, &len,
        TPNOFLAGS))== -1){
        error processing routine
    }
    . . .
}
```

## サーバー・プログラム

以下はサーバー・プログラムの例です。開発者が作成したサービス・ルーチンであり、main()はTmaxから提供されたものです。

```

#include <usrinc/atmi.h>
. . .
SEL_SVC(TPSVCINFO *msg)
/* クライアントとクライアントの要求内容が入っている構造体*/
{
    struct strdata *svcddata;
    /* Bufferタイプと一致するようにデータ型を変換 */
    svcddata = (struct strdata *)msg->data;
    . . .
    svcddata->ip = sip;

    strcpy(msg->data, svcddata);
    tpreturn(TPSUCCESS, 0, msg->data, sizeof(struct strdata), TPNOFLAGS);
};

```

## 10.1.1. 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>

パラメータ	説明
	<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> <p>– TPEXIT</p> <p>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</p> <p>– TPDOWN</p> <p>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</p> <p>– TMSUCCESS</p> <p>TPSUCCESSと同じです</p> <p>– TMFAIL</p> <p>TPFAILと同じです</p> <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	送信されたデータ長です。

パラメータ	説明
	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
flags	<p>使用されていません。必ず0に設定します。</p> <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[TPESVCERR]が渡されます

区分	説明
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(), tpdiscon(), tpgetrply(), tprecv(), tpsend()

## 10.1.2. 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。

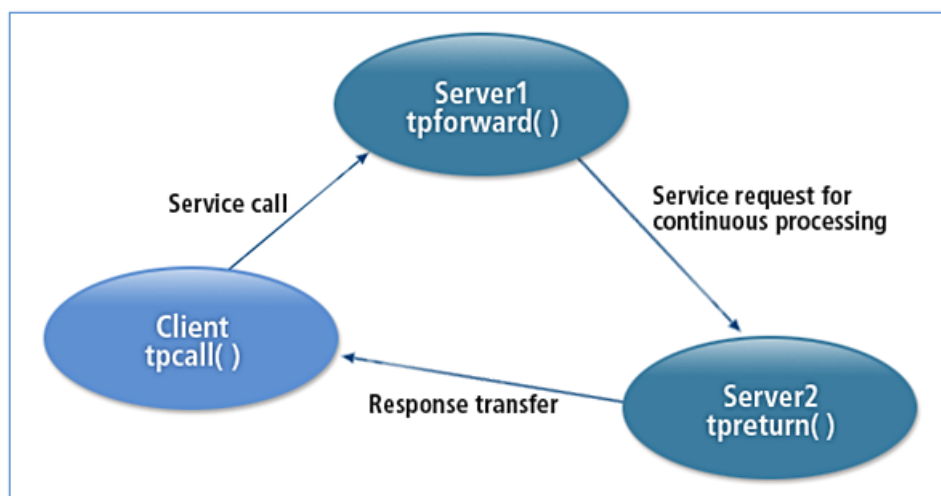
dataが指すデータを使用して、svcと指定されたサービスに要求を渡します。要求を渡すサービス・ルーチンはいかなる応答も受信しません。要求が渡された後、サービス・ルーチンはTmaxシステムに返します。また、サーバーは自由に他の作業を実行できます。tpforward()は要求者からいかなる応答も期待しないため、特にエラーなく、どのサービスにでも渡すことができます。

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 10.1] 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

パラメータ	説明
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を指す必要があります。</p> <p>バッファがサービス・ルーチンに送信されたのと同じバッファの場合、Tmaxシステムがこのバッファ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でない場合、tpforward()がそのバッファを解除します</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	<p>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</p>

- 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]	<p>有効でないバッファを使用した場合、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</p>
[TPETIME]	<p>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</p>

- 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()

### 10.1.3. 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()

## 10.1.4. 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=tpgetclid();
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()

## 10.1.5. 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時は、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「[10.1.3. 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()

## 10.1.6. 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、「[10.1.4. 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()

## 10.1.7. 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に返す関数です。クライアントとサーバー・プログラムでの作成方法は若干異なります。詳しい内容は、「[9.14.1. tpgetctxt](#)」を参照してください。

## 10.1.8. tpsetctxt

現在のコンテキストを設定する関数です。クライアントとサーバー・プログラムでの作成方法は若干異なります。詳しい内容は、「[9.14.2. tpsetctxt](#)」を参照してください。

## 10.1.9. 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	送信するバッファ長です
flags	flagsで使用可能な値は以下のとおりです  – TPNOFLAG(0)  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  – TPUDP



パラメータ	説明
	<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せん

#### ● 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 = tpgetcliid();
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()

## 10.1.10. tpgetcliid

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、**tpsendtocli()**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 tpgetcliid(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

tpgetcliid()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()

## 10.1.11. 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、tpgetclid()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()

## 10.2. UCS

UCS方式のサーバー・プログラムで使われる関数について説明します。

- スケジューリング関連API

UCS方式はユーザーが開発したusermain()ルーチンがmain()ルーチンのように使われるので、スケジューリングAPIによりTMMとCLHからの多様なメッセージを処理しなければなりません。

関数	機能
<a href="#">tpschedule</a>	指定時間(秒)の間にCLHまたはユーザー設定FDにメッセージが到着したかどうかを確認します
<a href="#">tpuschedule</a>	指定時間(マイクロ秒)の間にCLHまたはユーザー設定FDにメッセージが到着したかどうかを確認します

- ソケットFD関連マクロ

ソケット関連マクロは、先述したtpschedule()などのUCSスケジューラーにユーザーのソケットFDと一緒に使用するためのもので、一般ネットワーク・プログラムで使用するFD\_SET、FD\_CLR、FD\_ISSETに似ています。

関数	機能
<a href="#">tpsetfd</a>	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します
<a href="#">tpissetfd</a>	該当FDにメッセージが到着したかどうかを確認します
<a href="#">tpclrfd</a>	該当FDをUCSスケジューラーから解除します

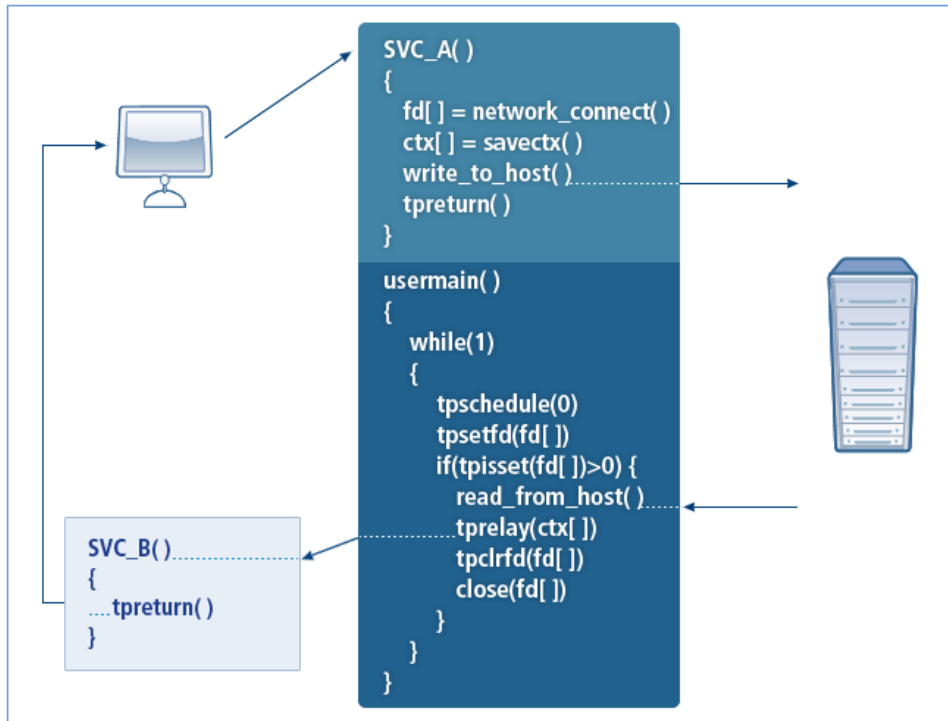
- サービス・フォワーディング

システムでない一般ホスト・システムと連動して運用する場合、一般的にサーバー・プログラムでソケットを開けてデータを送受信する過程をクライアントの代わりに実行します。しかし、外部ホスト・システムとの接続が不安定な場合、またはサービス時間が長い場合に該当サーバー・プログラムはブロック状態に置かれ、追加的なサービスの実行ができません。

このような場合を解決するためにサービス・ルーチンはクライアントの要求を受けてクライアントの情報を保存し、外部ホストに要求する部分までを担当します。応答はusermain()ルーチンを利用して保存された

クライアント情報とともに他のサービス・ルーチンへ渡されます。そのサービス・ルーチンがクライアントに結果値を送るようにすれば、サーバー・プログラムはブロックされずに全過程を実行することができます。

[図 10.2] UCS方式のサービス・フォワーディング



関数	機能
<a href="#">tpsavectx</a>	サーバー・ライブラリー内にクライアント情報を保存します
<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="#">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="#">tprelay</a>	tpgetctx()またはtpsavectx()から取得したクライアント情報/トランザクション情報を渡すとともに他のサービス・ルーチンにクライアント管理を任せます(TCS方式サーバー・プログラムで使われるtpforward() に似ています)

- usermain()ルーチン内における非同期型通信

usermain()ルーチン内でサービス時間の長い非同期型通信を使用する場合、tpgetreply()でブロックが発生してスケジューリングが遅くなる場合があります。サービス結果をtpgetreply()で受けずにコールバック関数を指定すると、スケジューリングに支障なく結果値の処理ができます。しかし、ループ時間の短いusermain()内で毎回非同期型通信が発生する場合、最大非同期型サービス数を超過してしまい、エラーが発生するので、注意を要します。

関数	機能
<a href="#">tpregcb</a>	非同期型サービス要求を処理するコールバック関数を設定します
<a href="#">tpunregcb</a>	設定されたコールバック関数を解除します

---

## 参考

関数と関連関数の詳細については、『Tmaxリファレンスガイド』を参照してください。

---

### 10.2.1. 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、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
[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

tpsleeep(), tp\_sleep(), tp\_usleep()

## 10.2.2. tpuschedule

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。tpuschedule()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 tpuschedule (int timeout)
```

- パラメータ



パラメータ	説明
timeout	待機する時間をマイクロ秒単位で入力します – -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 – 0: データが到着するまで待機します

- 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

tpuschedule()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)
    {
        ...
        tpuschedule(3000000);
        ret = tpcall("SERVICE", (char *)buf, strlen(buf), (char **)&buf,
                    (long *)&rlen, TPNOFLAGS);
        if (ret == -1) { error processing }
        ...
    }
}
```

- 関連関数

tpschedule()

### 10.2.3. 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()

## 10.2.4. 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
-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()

# 10.2.5. 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 = tmalloc("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()

## 10.2.6. 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 = tpsavectx();
    tpreturn(TPSUCCESS, 0, rqst->data, rqst->len, 0);
}

```

- 関連関数

tpreturn(), tpforward(), tprelay()

## 10.2.7. 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
    }
    .....
}
```

## 10.2.8. 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);
}
```

## 10.2.9. 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連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

パラメータ	説明
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tpsavectx()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。tperrno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、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、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()

## 10.2.10. 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(UCSMSGINF *reply);
DUMMY(TPSVCINF *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(UCSMMSGINFO *reply)
{
    printf("data....%s\n", reply->data);
}

```

- 関連関数

tpunregcb()

## 10.2.11. 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(UCSMSGINFO *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()

# 第11章 エラー処理

本章では、エラー処理方法とデバッグ方法について説明します。

## 11.1. 概要

Tmax APIは、エラーが発生した場合、状況に基づいた適切なエラー番号を設定します。エラーメッセージを参照すると、エラーの原因究明に役立ちます。また、API内部のシステム呼び出しレベルのエラー情報が知りたい場合、tuxinc/Unix.hに定義されているエラーメッセージとAPIを参照してください。

また、アプリケーションレベルではなくTmaxシステム運用上で発生する問題のために、運用上の各種情報をコンソールに表示するデバッグ用CLHを提供しているので、参照してください。

---

### 参考

Tmaxの運用中に発生するエラーメッセージは、『Tmax メッセージリファレンスガイド』を参照してください。

---

## 11.2. APIレベルのエラー処理

Tmax APIが失敗した場合の戻り値はAPIごとに異なり、グローバル変数tperrnoにはエラー状況に対するエラー番号が設定されます。

### 11.2.1. 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->dcompwd, "tuxedo");
    if (tpstart(tpinfo) == -1){
        printf("tpstart fail , err = %s\n", tpsterror(tperrno));
        exit(1);
    }

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

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

## 11.3. システム・レベルのエラー処理

Tmax APIは、多くのシステムの呼び出しを使用します。OSやプラットフォーム上に問題があって特定システムの呼び出しでエラーが発生したときにそのエラーを確認する場合や、他の異種プラットフォームにポータリングするとき、エラーメッセージを統合して管理したい場合に、以下で紹介するAPIを使用すると役に立ちます。ヘッダー・ファイルのディレクトリーはTMAXDIR/tuxinc/Uunix.hです。

```
TMAXDIR/tuxinc/Uunix.h
```

## 11.3.1. 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
```

## 11.3.2. Uunix\_err

ATMI APIの呼出しに失敗し、tperrno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
```

### 11.3.3. 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
```

### 11.3.4. 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。

```
# include <atmi.h>
int tmaxoserrno;
```

## 11.4. デバッグ

### 11.4.1. デバッグCLH

TMAXDIR/binディレクトリーにclh.dbgという名前のファイルが存在します。

ファイル名をCLHに変更して使用すると、CLHで行われるすべてのメッセージ転送の内容を確認できます。必ずオリジナルのCLHをバックアップしておいてください。

```
/home/navis/tmax/bin> tmboot

TMBOOT for node(aix51) is starting:
Welcome to Tmax demo system: it will expire 2002/9/15
Today: 2002/7/16
    TMBOOT: TMM is starting: Tue Jul 16 22:39:13 2002
    TMBOOT: CLL is starting: Tue Jul 16 22:39:13 2002
    TMBOOT: CLH is starting: Tue Jul 16 22:39:13 2002
COM: waiting for TMM reply
LIB: read 96 bytes
(I) CLH Current Tmax Configuration:
    Number of client handler(MINCLH) = 1
    Supported maximum user per node = 3944
    Supported maximum user per handler = 3944
LIB: read 96 bytes
CLH: bootpid = 31202
    TMBOOT: SVR(sub) is starting: Tue Jul 16 22:39:13 2002
    TMBOOT: SVR(svr2) is starting: Tue Jul 16 22:39:13 2002
CLH: request_from_server: clh = 0, ind = 0, fd = 8
CLH: msg from server: msgtype = 101, svcname = , len = 0
CLH: register_from_server, spri = 32, svri = 0, maxtms = 32
CLH: reply_to_server: clh = 0, ind = 32, fd =8
CLH: msg to server: msgtype = 1101, svcname = , len = 0
CLH: request_from_server: clh = 0, ind = 0, fd = 9
CLH: msg from server: msgtype = 135, svcname = , len = 0
.....
```

## 11.4.2. デバッグ・ライブラリー

TMAXDIR/libディレクトリーにlibsvrd.a / libsvrd.soというライブラリーが存在しています。libsvr.a / libsvr.soの代わりにこれらのライブラリーを使用すると、コンソール画面に様々なデータ値が出力されるため、サーバー・ライブラリーで起きていることや、エラーが発生する時点などの把握が容易です。libsvrd.aの場合は再コンパイルが必要ですが、libsvrd.soは名前のみ変更します。

```
/oracle/navis/tmax385/lib> tmboot

TMBOOT for node(tmaxc1) is starting:
Welcome to Tmax demo system: it will expire 2002/9/30
.....
GETOPT1: -b 255859
GETOPT1: -s svr2
GETOPT1: -d -1
SVR: delay = -1, _use_lock = 1
COM: waiting for TMM reply
LIB: read 96 bytes
register_to_tmm success
init_shm(78990, 139364) success
init_svctab success
SVR: my info--1 32 0 0 -3
init_clh success
LIB: read 96 bytes
register_to_clh success
init_txinfo success
check_node success
_tmax_init = 1
GETOPT1: -b 255859
GETOPT1: -s fdltest
GETOPT1: -d -1
SVR: delay = -1, _use_lock = 1
COM: waiting for TMM reply
LIB: read 96 bytes
register_to_tmm success
.....
```



# 第12章 例

本章では、Tmaxシステムが提供する固有機能を使用するための例について説明します。

## 12.1. 通信タイプの例

本節では、Tmaxで使用する3種類の通信タイプである同期型、非同期型、会話型アプリケーションの簡単な例を通して、全体的な流れについて説明します。

### 12.1.1. 同期型通信

クライアントはSTRINGバッファに文字列をコピーしてサービスを呼び出し、サーバーのサービス・ルーチンはこの文字列を受信し、大文字列に変えて返すプログラムです。

#### プログラム構成

- 共通プログラム

プログラム・ファイル	説明
sample.m	Tmax環境設定ファイルです

- クライアント・プログラム

プログラム・ファイル	説明
sync_cli.c	クライアント・プログラムです

- サーバー・プログラム

プログラム・ファイル	説明
syncsvc.c	大文字に変えるサービス・プログラムです
Makefile	Tmaxが提供するMakefileを修正する必要があります

#### プログラムの特徴

- クライアント・プログラム

機能	説明
Tmax接続	基本接続(クライアント情報なし)
バッファ・タイプ	STRING
通信タイプ	tpcall()を利用した同期型通信

- サーバー・プログラム

機能	説明
サービス	TOUPPERSTR
データベース接続	なし

## Tmax環境ファイル

以下は、Tmax環境ファイルの例です。

<sample.m>

```
*DOMAIN
resrc          SHMKEY = 77990, MAXUSER = 256

*NODE
tmax           TMAXDIR = "/home/tmax",
               APPDIR  = "/home/tmax/appbin",
               PATHDIR = "/home/tmax/path",
               TLOGDIR = "/home/tmax/log/tlog",
               ULOGDIR = "/home/tmax/log/slog",
               SLOGDIR = "/home/tmax/log/ulog"

*SVRGROUP
svg1           NODENAME = tmax

*SERVER
syncsvc       SVGNAME = svg1,
               MIN = 1, MAX = 5,
               CLOPT = " -e $(SVR).err -o $(SVR).out "

*SERVICE
TOUPPERSTR     SVRNAME = syncsvc
```

## クライアント・プログラム

以下は、クライアント・プログラムの例です。

<sync\_cli.c>

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

main(int argc, char *argv[])
{
    char *sendbuf, *recvbuf;
    long rlen;

    if (argc != 2)
    {
        fprintf(stderr, "Usage: $ %s string \n", argv[0]);
        exit(1);
    }

    if (tpstart((TPSTART_T*)NULL) == -1)
    {
        fprintf(stderr, "Tpstart failed\n");
        exit(1);
    }

    if ((sendbuf = tpalloc("STRING", NULL, 0)) == NULL) {
        fprintf(stderr, "Error allocation send buffer\n");
        tpend();
        exit(1);
    }

    if ((recvbuf = tpalloc("STRING", NULL, 0)) == NULL) {
        fprintf(stderr, "Error allocation recv buffer\n");
        tpend();
        exit(1);
    }

    strcpy(sendbuf, argv[1]);

    if (tpcall("TOUPPERSTR", sendbuf, 0, &sendbuf, &rlen, TPNOFLAGS) == -1)
    {
        fprintf(stderr, "Can't send request to service TOUPPER->%s!\n",
            tpstrerror(tperrno));
        tpfree(sendbuf);
        tpfree(recvbuf);
        tpend();
        exit(1);
    }

    printf("Sent value:%s\n", sendbuf);
}
```

```
printf("Returned value:%s\n ",recvbuf );
tpfree(sendbuf);
tpfree(recvbuf);
tpend( );
```

## サーバー・プログラム

以下は、サーバー・プログラムの例です。

<syncsvc.c>

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

TOUPPERSTR(TPSVCINFO *msg)
{
    int i;

    for (i = 0; i < msg->len ; i++)
        msg->data[i] = toupper(msg->data[i]);
    msg->data[i] = '\0';

    tpreturn(TPSUCCESS, 0, msg->data, 0, TPNOFLAGS);
}
```

### 12.1.2. 非同期型通信

クライアントはSTRUCTバッファのメンバーに文字列をコピーしてサービスを呼び出し、サーバーのサービス・ルーチンはこの文字列を受信して小文字列あるいは大文字列に変えて返すプログラムです。クライアントは非同期型通信でTOUPPERサービスを要求し、再度同期型でTOLOWERサービスを呼び出して結果を受信した後、先に要求したTOUPPERサービスの実行結果を受信します。

## プログラム構成

### ● 共通プログラム

プログラム・ファイル	説明
demo.s	構造体バッファを定義します
sample.m	Tmax環境設定ファイルです

### ● クライアント・プログラム

プログラム・ファイル	説明
async_cli.c	クライアント・プログラムです

- サーバー・プログラム

プログラム・ファイル	説明
asynsvc.c	大文字/小文字に変えるサービス・プログラムです
Makefile	Tmaxが提供するMakefileを修正する必要があります

## プログラムの特徴

- クライアント・プログラム

機能	説明
Tmax接続	基本接続
バッファ・タイプ	STRUCT
通信タイプ	同期型及び非同期型

- サーバー・プログラム

機能	説明
サービス	TOUPPER、TOLOWER
データベース接続	なし
通信タイプ	同期型及び非同期型

## 構造体バッファ

以下は、非同期型通信で使用する構造体バッファです。

<demo.s>

```
struct strdata {
    int flag;
    char sdata[20];
};
```

## Tmax環境ファイル

以下は、Tmax環境ファイルの例です。

<sample.m>

```

*DOMAIN
resrc          SHMKEY = 77990, MAXUSER = 256

*NODE
tmax           TMAXDIR = "/home/tmax",
               APPDIR = "/home/tmax/appbin",
               PATHDIR = "/home/tmax/path"

*SVRGROUP
svg1           NODENAME = tmax

*SERVER
asynccsvc      SVGNAME = svg1, MIN = 1, MAX = 5

*SERVICE
TOUPPER        SVRNAME = asynccsvc
TOLOWER        SVRNAME = asynccsvc

```

## クライアント・プログラム

以下は、クライアント・プログラムの例です。

<async\_cli.c>

```

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

main(int argc, char *argv[ ])
{
    struct strdata *sendbuf, *sendbuf1;
    long dlen, clen;
    int cd;

    if (argc != 3) {
        fprintf(stderr, "Usage: $ %s string STRING\n", argv[0], argv[1]);
        exit(1) ;
    }

    if (tpstart((TPSTART_T *)NULL) == -1) {
        fprintf(stderr, "TPSTART_T failed\n");
        exit(1) ;
    }

    sendbuf = (struct strdata *)tpalloc("STRUCT", "strdata", 0);

```

```

    if ( sendbuf == NULL) {
        fprintf(stderr, "Error allocation send buffer\n");
        tpend () ;
        exit(1) ;
    }

    sendbuf1 = (struct strdata *)tpalloc("STRUCT", "strdata", 0);
    if (sendbuf1 == NULL) {
        fprintf(stderr, "Error allocation send1 buffer\n");
        tpend();
        exit(1) ;
    }

    strcpy(sendbuf->sdata, argv[1]);
    strcpy(sendbuf1->sdata, argv[2]);

    if ((cd = tpacall("TOUPPER", (char *)sendbuf, 0, TPNOFLAGS)) == -1)
    {
        fprintf(stderr, "Toupper error -> %s", tpstrerror(tperrno));
        tpfree((char *)sendbuf);
        tpend();
        exit(1) ;
    }
    if (tpcall("TOLOWER", (char *)sendbuf1, 0, (char **)&sendbuf1, &dlen,
               TPSIGRSTRT) == -1) {
        fprintf(stderr, "Tolower error -> %s", tpstrerror(tperrno));
        tpfree((char *)sendbuf);
        tpend();
        exit(1) ;
    }
    if (tpgetrply(&cd, (char **)&sendbuf, &crlen, TPSIGRSTRT) == -1) {
        fprintf(stderr, "Toupper getrply error -> %s", tpstrerror(tperrno));

        tpfree((char *)sendbuf);
        tpend();
        exit(1) ;
    }
    printf("Return value %s\n %s\n", sendbuf -> sdata, sendbuf1 -> sdata);
    tpfree((char *)sendbuf);
    tpfree((char *)sendbuf1);
    tpend() ;
}

```

## サーバー・プログラム

以下は、サーバー・プログラムの例です。

<asynsvc.c>

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

TOUPPER(TPSVCINFO *msg)
{
    int i = 0;
    struct strdata *stdata;

    stdata = (struct strdata *)msg -> data;

    while (stdata->sdata[ i ] != '\0') {
        stdata->sdata[ i ] = toupper(stdata->sdata[ i ]);
        i++;
    }

    tpreturn(TPSUCCESS, 0, (char *)stdata, 0, TPNOFLAGS);
}

TOLOWER(TPSVCINFO *msg)
{
    int i = 0;
    struct strdata *stdata;

    stdata = (struct strdata *)msg -> data;

    while ( stdata->sdata[ i ] != '\0') {
        stdata->sdata[ i ] = tolower(stdata->sdata[ i ]);
        i++;
    }

    tpreturn(TPSUCCESS, 0, (char *)stdata, 0, TPNOFLAGS);
}
```

### 12.1.3. 会話型通信

クライアントはユーザーからの入力を受け、STRINGバッファーを通じて固有番号を送ります。サーバーのサービス・ルーチンは、データベースに格納されたテーブルから該当固有番号より大きい番号をもつ顧客情報を、構造体を通じて返します。

クライアントは、会話型モードを設定して固有番号を送り、会話主導権をサーバーに渡します。サーバーは、条件を満たすデータベースの全データをカーソルにて読み取り、クライアントに送ります。クライアントは、TPEVSVCSUCCを通じて、正常に全データを読み取ったことを確認できます。



## プログラム構成

- 共通プログラム

プログラム・ファイル	説明
demo.s	構造体を定義したファイルです
sample.m	Tmax環境設定ファイルです
mktable.sql	テーブルを作成するスクリプトです
sel.sql	テーブル及びデータを出力するスクリプトです

- クライアント・プログラム

プログラム・ファイル	説明
conv_cli.c	クライアント・プログラムです

- サーバー・プログラム

プログラム・ファイル	説明
consvsvc.pc	サーバー・プログラムです
Makefile	Tmaxが提供するMakefileを修正する必要があります

## プログラムの特徴

- クライアント・プログラム

機能	説明
Tmax接続	基本接続
バッファ・タイプ	送信時はSTRING、受信時はSTRUCT
通信タイプ	会話型通信STRUCT

- サーバー・プログラム

機能	説明
サービス	MULTI
データベース接続	Oracle使用

# 構造体バッファ

以下は、会話型通信で使用する構造体バッファです。

<demo.s>

```
struct sel_o {
    char seqno[10];
    char corpno[10];
    char compdate[8];
    int totmon;
    float guarat;
    float guamon;
} ;
```

# Tmax環境ファイル

以下は、Tmax環境ファイルの例です。

<sample.m>

```
* DOMAIN
resrc          SHMKEY = 77990, MAXUSER = 256

*NODE
tmax           TMAXDIR = "/home/tmax",
               APPDIR  = "/home/tmax/appbin",
               PATHDIR = "/home/tmax/path"

* SVRGROUP
svg1           NODENAME = tmax,
               DBNAME  = ORACLE,
               OPENINFO = "ORACLE_XA+Acc=P/scott/tiger+SesTm=60",
               TMSNAME = svg1_tms

*SERVER
convsvc       SVGNAME = svg1, CONV = Y

*SERVICE
MULTI         SVRNAME = convsvc
```

以下は、環境設定に追加された項目についての説明です。

項目	説明
DBNAME	使用するデータベース名を定義します
OPENINFO	Oracleデータベースとの連動のための接続情報を設定します

項目	説明
TMSNAME	グローバル・トランザクション処理を行うプロセス名を設定します
CONV	会話型モードのサーバーを指定します

## データベース・スクリプト

以下は、Oracleテーブルを作成するスクリプトの例です。

<mktable.sql>

```
sqlplus scott/tiger << EOF
create table multi_sel
(
    seqno      VARCHAR(10),
    corpno     VARCHAR(10),
    compdate   VARCHAR(8),
    totmon     NUMERIC(38),
    guarat     FLOAT,
    guamon     FLOAT
);
create unique index idx_tdb on multi_sel(seqno);
EOF
```

以下は、Oracleテーブル及びデータを出力するスクリプトの例です。

<sel.sql >

```
sqlplus scott/tiger << EOF
Desc multi_sel;
select * from multi_sel;
EOF
```

## クライアント・プログラム

以下は、クライアント・プログラムの例です。

<conv\_cli.c>

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

main(int argc, char *argv[])
{
    struct sel_o *rcvbuf;
```

```

char *sndbuf;
long sndlen, rcvlen, revent;
int cd;

if (argc != 2) {
    printf("Usage: client string\n");
    exit(1);
}

/* tpstart()と一緒にTmaxに接続. */
if (tpstart((TPSTART_T *) NULL) == -1) {
    printf("tpstart failed\n");
    exit(1);
}

if ((sndbuf = tpalloc("STRING", NULL, 12)) == NULL) {
    printf("tpalloc failed:sndbuf\n");
    tpend();
    exit(1);
}

if ((rcvbuf = (struct sel_o *)tpalloc("STRUCT", "sel_o", 0)) == NULL) {
    printf("tpalloc failed:rcvbuf\n");
    tpfree(sndbuf);
    tpend();
    exit(1);
}

strcpy(sndbuf, argv[1]);

if ((cd = tpconnect ("MULTI", sndbuf, 0, TPRECVOONLY)) == -1){
    printf("tpconnect failed:CONVER service, tperrno=%d\n", tperrno);
    tpfree(sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
    exit(1);
}

/* 会話型通信接続、会話主導権はサーバー側に渡す */
printf("tpconnect SUCESS \"MULTI\" service\n");
while ( 1 ) {    /* 多重データの受信. */
    printf("tprecv strat\n");
    if( tprecv(cd, (char **)&rcvbuf, &rcvlen, TPNOTIME, &revent) < 0 )
    {
        /* サーバーでtpreturn()で終わった場合 */
        if (revent == TPEV_SVCSUCC){
            printf("all is completed\n");
            break;

```

```

    }
    printf("tprecv failed, tperrno=%s, revent=%x\n",
           tpstrerror(tperrno), revent );
    tpfree(sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
    exit(1);
}
printf("seqno = %s\t\t corpno =%s\n", rcvbuf->seqno, rcvbuf->corpno);
printf("compdate = %s\t\t totmon =%d\n", rcvbuf->compdate, rcvbuf->totmon);

printf("guarat = %f\t\t guamon =%f\n\n", rcvbuf->guarat, rcvbuf->guamon)
;
}

tpfree(sndbuf);
tpfree((char *)rcvbuf);
tpend();
printf("FINISH\n");
}

```

## サーバー・プログラム

以下は、サーバー・プログラムの例です。

<consvsc.pc>

```

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

EXEC SQL begin declare section; /* Oracleグローバル変数の宣言 */
    char seq[10];
    struct sel_o *sndbuf;
EXEC SQL end declare section;
EXEC SQL include sqlca;

MULTI(TPSVCINFO *msg)
{
    int i, cd;
    long sndlen, revent;

    memset(seq, 0, 10);
    strcpy(seq, msg->data);

    if ((sndbuf = (struct sel_o *) tmalloc ("STRUCT", "sel_o", 0)) == NULL) {

```

```

        printf("tpalloc failed:\n");
        tpreturn (TPFAIL, -1, NULL, 0, TPNOFLAGS);
    }

    /* 多量のデータのためにカーソル宣言 */
    EXEC SQL declare democursor cursor for
    select *
    from corp
    where seqno > :seq;

    EXEC SQL open democursor;
    EXEC SQL whenever not found goto end_of_fetch;
    if (sqlca.sqlcode != 0){
        printf("oracle sqlerror=%s", sqlca.sqlerrm.sqlerrmc);
        tpreturn (TPFAIL, sqlca.sqlcode, NULL, 0, TPNOFLAGS);
    }

    /* Oracleエラーがない間データ転送 */
    while ( sqlca.sqlcode == 0 ){
        EXEC SQL fetch democursor into :sndbuf;

        if (tpsend (msg->cd, (char *)sndbuf, 0, TPNOTIME, &revent) == -1){
            printf("tpsend failed, tperrno=%d, revent=%x\n", tperrno,
                revent );
            tpfree ((char *)sndbuf);
            tpreturn (TPFAIL, -1, NULL, 0, TPNOFLAGS);
        }
    }

    tpreturn (TPFAIL, sqlca.sqlcode, NULL, 0, TPNOFLAGS);

end_of_fetch:
exec sql close democursor;
printf("tpreturn before");
tpreturn (TPSUCCESS, 0, NULL, 0, TPNOFLAGS);
}

```

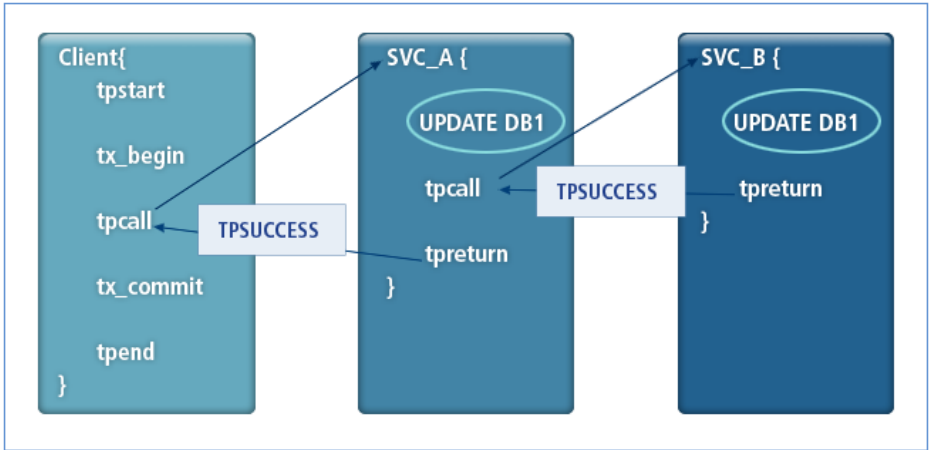
## 12.2. グローバル・トランザクション・プログラムの例

グローバル・トランザクション(Global Transaction Processing)とは、1つ以上のリソース管理者(データベース)と1つ以上の物理的なサイトが、1つの論理的な単位で参与するトランザクションです。Tmaxシステムでは、すべてのトランザクションをグローバル・トランザクションと見なし、データの整合性のために2PC(2 Phase Commit)を使用します。

クライアントは、ユーザーからの入力を受け、構造体バッファを通じて固有番号とデータを送ります。サーバーは該当固有番号のデータを更新し、このデータで他のデータベースを使用するサービスを呼び出し、

テーブルに挿入します。クライアントは、このすべての過程を1つのトランザクションとして指定し、エラーが発生した場合、2個のデータベースを同時にロールバックできるようにします。

[図 12.1] 2つのデータベースの接続



プログラム構成

● 共通プログラム

プログラム・ファイル	説明
demo.s	構造体バッファの設定ファイルです
sample.m	Tmax環境設定ファイルです
mktable.sql	データベース・テーブルを作成するSQLスクリプトです

● クライアント・プログラム

プログラム・ファイル	説明
client.c	クライアント・プログラムです

● サーバー・プログラム

プログラム・ファイル	説明
update.pc	データベースに更新するサーバー・プログラムです
insert.pc	データベースに挿入するサーバー・プログラムです
Makefile	Tmaxが提供するMakefileを修正する必要があります

## プログラムの特徴

- クライアント部分

機能	説明
Tmax接続	基本接続
バッファ・タイプ	STRUCT
通信タイプ	tpcall()による同期通信
トランザクション処理	クライアントでトランザクション範囲を指定

- サーバー部分

機能	説明
サーバー・プログラム	互いに異なるデータベースを使用する2個のサーバー・プログラム
サービス	UPDATE、INSERT
データベース接続	2種類のOracleデータベース

## 構造体バッファ

以下は、グローバル・トランザクションで使用する構造体バッファです。

<demo.s>

```
struct input {
    int account_id;
    int branch_id;
    char phone[15];
    char address[61];
};
```

## Tmax環境ファイル

以下は、Tmax環境ファイルの例です。

<sample.m>



```

*DOMAIN
res          SHMKEY=88000, MINCLH=1, MAXCLH=5, TPORTNO=8880, BLOCKTIME=60

*NODE
tmax1        TMAXDIR = "/user/ tmax ",
              APPDIR  = "/user/ tmax /appbin",
              PATHDIR = "/user/ tmax /path",
              TLOGDIR = "/user/ tmax /log/tlog",
              ULOGDIR="/user/ tmax /log/ulog",
              SLOGDIR="/user/ tmax /log/slog"

tmax2        TMAXDIR = "/user/ tmax ",
              APPDIR  = "/user/ tmax /appbin",
              PATHDIR = "/user/ tmax /path",
              TLOGDIR = "/user/ tmax /log/tlog",
              ULOGDIR="/user/ tmax /log/ulog",
              SLOGDIR="/user/ tmax /log/slog"

*SVRGROUP
svg1         NODENAME = tmax1, DBNAME = ORACLE,
              OPENINFO = "ORACLE_XA+Acc=P/scott/tiger+SesTm=60",
              TMSNAME = svg1_tms

svg2         NODENAME = tmax2, DBNAME = ORACLE,
              OPENINFO = "ORACLE_XA+Acc=P/scott/tiger+SesTm=60",
              TMSNAME = svg2_tms

*SERVER
update       SVGNAME=svg1
insert       SVGNAME=svg2

*SERVICE
UPDATE       SVRNAME=update
INSERT       SVRNAME=insert

```

## データベース・スクリプト

以下は、Oracleテーブルを作成するスクリプトの例です。

<mktable.sql>

```

sqlplus scott/tiger << EOF
drop table ACCOUNT;

create table ACCOUNT (

```

```

ACCOUNT_ID  integer,
BRANCH_ID   integer not null,
SSN         char(13) not null,
BALANCE     number,
ACCT_TYPE   char(1),
LAST_NAME   char(21),
FIRST_NAME  char(21),
MID_INIT    char(1),
PHONE       char(15),
ADDRESS     char(61),
CONSTRAINT ACCOUNT_PK PRIMARY KEY(ACCOUNT_ID)
);
quit
EOF

```

## クライアント・プログラム

以下は、クライアント・プログラムの例です。

<client.c>

```

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

#define TEMP_PHONE "6283-2114"
#define TEMP_ADDRESS "Korea"

int main(int argc, char *argv[])
{
    struct input *sndbuf;
    char *rcvbuf;
    int acnt_id, n, timeout;
    long len;

    if (argc != 2) {
        fprintf(stderr, "Usage:%s account_id \n", argv[0]);
        exit(1);
    }

    acnt_id = atoi(argv[1]);
    timeout = 5;

    n = tmaxreadenv("tmax.env", "tmax");
    if (n < 0) {

```

```

        fprintf(stderr, "tmaxreadenv fail! tperrno = %d\n", tperrno);
        exit(1);
    }

    n = tpstart((TPSTART_T *)NULL);
    if (n < 0) {
        fprintf(stderr, "tpstart fail! tperrno = %s\n", tperrno);
        exit(1);
    }
    sndbuf = (struct input *)tpalloc("STRUCT", "input", sizeof(struct input));

    if (sndbuf == NULL) {
        fprintf(stderr, "tpalloc fail: sndbuf tperrno = %d\n", tperrno);
        tpend();
        exit(1);
    }
    rcvbuf = (char *)tpalloc("STRING", NULL, 0);
    if (rcvbuf == NULL) {
        fprintf(stderr, "tpalloc fail: rcvbuf tperrno = %d\n", tperrno);
        tpend();
        exit(1);
    }
    sndbuf->account_id = acct_id;
    sndbuf->branch_id = acct_id;
    strcpy(sndbuf->phone, TEMP_PHONE);
    strcpy(sndbuf->address, TEMP_ADDRESS);

    tx_set_transaction_timeout(timeout);
    n = tx_begin();
    if (n < 0)
        fprintf(stderr, "tx begin fail! tperrno = %d\n", tperrno);

    n = tpcall("UPDATE", (char *)sndbuf, sizeof(struct input), (char **)&rcvbuf,

               (long *)&len, TPNOFLAGS);
    if (n < 0) {
        fprintf(stderr, "tpcall fail! tperrno = %d\n", tperrno);
        tpend();
        exit(1);
    }

    n = tx_commit();
    if (n < 0) {
        fprintf(stderr, "tx commit fail! tx error = %d \n", n);
        tx_rollback();
        tpend();
        exit(1);
    }

```

```

    }
    printf("rtn msg = %s\n", rcvbuf);
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
}

```

## サーバー・プログラム

以下は、データベースに更新するサーバー・プログラムの例です。

<update.pc>

```

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

#define OKMSG "YOU COMPLETE THE TRANSACTION"

EXEC SQL include sqlca.h;
EXEC SQL BEGIN DECLARE SECTION;
int account_id;
    int      branch_id;
    char     ssn[15];
    char     phone[15];
    char     address[61];
EXEC SQL END DECLARE SECTION;

UPDATE(TPSVCINFO *msg)
{
    struct input *rcvbuf;
    int      ret;
    long     acnt_id, rcvlen;
    char     *send;

    rcvbuf = (struct input *) (msg->data);
    send = (char *) tpalloc("STRING", NULL, 0);
    if (send == NULL) {
        fprintf(stderr, "tpalloc fail errno = %s\n", strerror(tperrno));
        tpreturn(TPFAIL, 0, (char *) NULL, 0, 0);
    }
    account_id = rcvbuf->account_id;
    branch_id = rcvbuf->branch_id;

```

```

strcpy(phone, rcvbuf->phone);
strcpy(address, rcvbuf->address);
strcpy(ssn, "1234567");

EXEC SQL UPDATE ACCOUNT
        SET BRANCH_ID = :branch_id,
          PHONE = :phone,
          ADDRESS = :address,
          SSN = :ssn
        WHERE ACCOUNT_ID = :account_id;
if (sqlca.sqlcode != 0 && sqlca.sqlcode != 1403 ) {
    fprintf(stderr, "update failed sqlcode = %d\n", sqlca.sqlcode);
    tpreturn(TPFAIL, -1, (char *)NULL, 0, 0);
}
rcvbuf->account_id++;
ret = tpcall("INSERT", (char *)rcvbuf, 0, (char **)&send, (long *)&rcvlen,
            TPNOFLAGS);
if (ret < 0) {
    fprintf(stderr, "tpcall fail tperrno = %d\n", tperrno);
    tpreturn(TPFAIL, -1, (char *)NULL, 0, 0);
}
strcpy(send, OKMSG);
tpreturn(TPSUCCESS, 1, (char *)send, strlen(send), TPNOFLAGS);
}

```

以下は、データベースに挿入するサーバー・プログラムの例です。

<insert.pc>

```

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

#define OKMSG "YOU COMPLETE THE TRANSACTION"

EXEC SQL include sqlca.h;
EXEC SQL BEGIN DECLARE SECTION;
    int      account_id;
    int      branch_id;
    char     ssn[15];
    char     phone[15];
    char     address[61];
EXEC SQL END DECLARE SECTION;

INSERT(msg)

```

```

TPSVCINFO  *msg;
{
    struct input *rcvbuf;
    int         ret;
    long        acnt_id;
    char        *send;

    rcvbuf = (struct input *) (msg->data);
    send = (char *) tmalloc("STRING", NULL, 0);
    if (send == NULL) {
        fprintf(stderr, "tpalloc fail errno = %s\n", tpstrerror(tperrno));
        tpreturn(TPFAIL, 0, (char *) NULL, 0, TPNOFLAGS);
    }

    account_id = rcvbuf->account_id;
    branch_id = rcvbuf->branch_id;
    strcpy(phone, rcvbuf->phone);
    strcpy(address, rcvbuf->address);
    strcpy(ssn, "1234567");

    /* Declare && Open Cursor for Fetch */
    EXEC SQL INSERT INTO ACCOUNT (
    ACCOUNT_ID,
    BRANCH_ID,
    SSN,
    PHONE,
    ADDRESS )
    VALUES (
    :account_id, :branch_id, :ssn, :phone, :address);

    if (sqlca.sqlcode != 0 && sqlca.sqlcode != 1403 )
    {
        printf("insert failed sqlcode = %d\n", sqlca.sqlcode);
        tpreturn(TPFAIL, -1, (char *) NULL, 0, TPNOFLAGS);
    }
    strcpy(send, OKMSG);
    tpreturn(TPSUCCESS, 1, (char *) send, strlen(send), TPNOFLAGS);
}

```

## 12.3. データベース・プログラム

代表的なデータベースであるOracleとInformixを使用した例について説明します。

## 12.3.1. Oracle Insertプログラム

クライアントはユーザーからの入力を受け、構造体バッファに入れてサービスを呼び出します。サーバーはこれを受け取り、該当テーブルに追加します。クライアントはトランザクションを指定し、エラーが発生した場合はロールバックできるようにします。

### プログラム構成

- 共通プログラム

プログラム・ファイル	説明
demo.s	構造体バッファの設定ファイルです
sample.m	Tmax環境設定ファイルです
mktable.sql	データベース・テーブルを作成するSQLスクリプトです
sel.sql	データベーステーブルの内容を出力するSQLスクリプトです

- クライアント・プログラム

プログラム・ファイル	説明
oins_cli.c	クライアント・プログラムです

- サーバー・プログラム

プログラム・ファイル	説明
oinssvc.pc	サービス・プログラムのOracleソースです
Makefile	Tmaxが提供するMakefileを修正する必要があります

### プログラムの特徴

- クライアント・プログラム

機能	説明
Tmax接続	基本接続
バッファ・タイプ	STRUCT
通信タイプ	tpcall()による同期通信
トランザクション処理	クライアントでトランザクション範囲を指定

- サーバー・プログラム

区分	説明
サービス	ORAINS
データベース接続	Oracleデータベース

## 構造体バッファ

以下は、構造体バッファの例です。

<demo.s>

```
struct ktran {
    int no;
    char name[20];
};
```

## Tmax環境ファイル

以下は、Tmax環境ファイルの例です。

<sample.m>

```
*DOMAIN
resrc          SHMKEY = 77990, MAXUSER = 256

*NODE
tmax           TMAXDIR = /home/tmax,
               APPDIR  = /home/tmax/appbin,
               PATHDIR = /home/tmax/path

*SVRGROUP
svg1           NODENAME = tmax,
               DBNAME  = ORACLE,
               OPENINFO = "Oracle_XA+Acc=P/scott/tiger+SesTm=60",
               TMSNAME = svg1_tms

*SERVER
oinssvc        SVGNAME = svg1, MIN = 1, MAX = 5

*SERVICE
ORAINS         SVRNAME = oinssvc
```

以下は、環境設定に追加された項目についての説明です。

項目	説明
DBNAME	使用するデータベースを定義します



項目	説明
OPENINFO	Oracleデータベース接続情報を設定する項目です。Oracleデータベースの場合、CLOSEINFOは指定しなくても結構です。tpsvrinfo()で呼び出します
TMSNAME	トランザクション処理を行うプロセス名を指定する項目です。OPENINFOの指定による自動トランザクションを処理します。svg1に属する該当サービスを自動トランザクション状態で処理します

## データベース・スクリプト

以下は、Oracleテーブルを作成するスクリプトの例です。

<mktable.sql>

```
sqlplus scott/tiger << EOF
    create table testdb1 (
        no number(7),
        name char(30)
    ) ;
EOF
```

以下は、Oracleテーブル及びデータを出力するスクリプトの例です。

<sel.sql>

```
sqlpus scott/tiger << EOF
desc testdb1;
select * from testdb1;
select count (*) from testdb1;
EOF
```

## クライアント・プログラム

以下は、クライアント・プログラムの例です。

<oins\_cli.c>

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

main(int argc, char *argv[])
{
    struct ktran *sndbuf, *rcvbuf;
    long sndlen, rcvlen;
    int cd;
```

```

    if (argc != 3) {
        printf("Usage: client no name\n");
        exit(1);
    }

    printf("tpstart-start \n");
    if(tpstart ((TPSTART_T *) NULL) == -1) {
        printf("Tpstart failed\n");
        exit(1);
    }
    printf("tpstart-ok \n");

    if((sndbuf=(struct ktran *) tmalloc("STRUCT","ktran",0))==NULL) {
        printf("tpalloc failed:sndbuf, tperrno=%d\n", tperrno);
        tpend();
        exit(1) ;
    }

    if((rcvbuf = (struct ktran *) tmalloc("STRUCT", "ktran", 0))== NULL) {
        printf("tpalloc failed:rcvbuf, tperrno=%d\n", tperrno);
        tpfree((char *)sndbuf);
        tpend();
        exit(1);
    }

    sndbuf->no = atoi(argv[1]);
    strcpy(sndbuf->name, argv[2]);
    printf("tpcall-start \n");
    tx_begin();

    if(tpcall("ORAINS",(char *)sndbuf,0,(char **)&rcvbuf,&rcvlen,TPNOFLAGS)==-1){

        printf("tpcall failed:ORA service, tperrno=%d", tperrno);
        printf("sql code=%d\n", tpurcode);
        tx_rollback();
        tpfree ((char *)sndbuf);
        tpfree ((char *)rcvbuf);
        tpend();
        exit(1);
    }

    printf("tpcall-success \n");
    tx_commit();

    tpfree ((char *)sndbuf);
    tpfree ((char *)rcvbuf);

```

```
        tpend();  
    }
```

## サーバー・プログラム

以下は、サーバー・プログラムの例です。

<oinssvc.pc>

```
#include <stdio.h>  
#include <usrinc/atmi.h>  
#include "../sdl/demo.s"  
  
EXEC SQL begin declare section;  
char name[20];  
int no;  
  
EXEC SQL end declare section;  
EXEC SQL include sqlca;  
  
ORAINS(TPSVCINFO *msg)  
{  
    struct ktran *stdata;  
    stdata = (struct ktran *)msg->data;  
    strcpy(name, stdata->name);  
    no = stdata->no;  
    printf("Ora service started\n");  
  
    /* データベースに挿入 */  
    EXEC SQL insert into testdb1(no, name) values(:no, :name);  
  
    if (sqlca.sqlcode != 0){  
        printf("oracle sqlerror=%s", sqlca.sqlerrm.sqlerrmc);  
        tpreturn (TPFAIL, sqlca.sqlcode, NULL, 0, TPNOFLAGS);  
    }  
  
    tpreturn (TPSUCCESS, sqlca.sqlcode, stdata, 0, TPNOFLAGS);  
}
```

### 12.3.2. Oracle Selectプログラム

クライアントはユーザーからの入力を受け、構造体バッファーに入れてサービスを呼び出します。サーバーはこれに該当する全データを受け取り、構造体配列を使用して結果を返します。クライアントはトランザクションを指定し、エラーが発生した場合はロールバックできるようにします。

## プログラム構成

- 共通プログラム

構成	説明
demo.s	構造体バッファ設定ファイルです
sample.m	Tmaxシステム環境ファイルです
mktable.sql	データベース・テーブルを作成するSQLスクリプトです
sel.sql	データベーステーブルの内容を出力するSQLスクリプトです

- クライアント・プログラム

構成	説明
oins_cli.c	クライアント・プログラムです
cdata.c	クライアント使用の関数モジュールです

- サーバー・プログラム

構成	説明
oselsvc.pc	サービス・プログラムのOracleソースです
Makefile	Tmaxが提供するMakefileを修正する必要があります

## プログラムの特徴

- クライアント・プログラム

機能	説明
Tmax接続	基本接続
サービス	ORASEL
バッファ・タイプ	STRUCT
通信タイプ	tpcall()による同期通信
トランザクション処理	クライアントでトランザクション範囲を指定

- サーバー・プログラム

機能	説明
サービス	ORASEL

機能	説明
データベース接続	Oracleデータベース
バッファ使用	必要に応じてバッファサイズを再調整

## 構造体バッファ

以下は、構造体バッファの例です。

<demo.s>

```
struct stru_his{
    long    ACCOUNT_ID ;
    long    TELLER_ID ;
    long    BRANCH_ID ;
    long    AMOUNT ;
} ;
```

## Tmax環境ファイル

以下は、Tmax環境ファイルの例です。

<sample.m>

```
*DOMAIN
resrc          SHMKEY = 77990, MAXUSER = 256

*NODE
tmax           TMAXDIR = "/home/tmax",
               APPDIR  = "/home/tmax/appbin",
               PATHDIR = "/home/tmax/path"

*SVRGROUP
svg1           NODENAME= tmax,
               DBNAME  = ORACLE,
               OPENINFO = "Oracle_XA+Acc=P/scott/tiger+SesTm=600",
               TMSNAME = svg1_tms

*SERVER
oselsvc        SVGNAME = svg1, MIN = 1, MAX = 5

*SERVICE
ORASEL         SVRNAME = oselsvc
```

以下は、環境設定に追加された項目についての説明です。

項目	説明
DBNAME	使用するデータベース名を設定します
OPENINFO	Oracleのデータベース接続情報を設定する項目です。OracleではCLOSEINFOは設定しなくても結構です。使用できるオプションは、以下で説明します
TMSNAME	トランザクション処理を管理するプロセス名を指定します
AUTOTRAN	該当サービスを処理する際、自動的にトランザクション状態で処理します

以下は、OPENINFO項目で使用できるオプションで、LogDirとDbgFlを使用できます。

オプション	説明
LogDir	LogDirを指定した場合、指定された位置にXAに関連したログを残すことができます  LogDirを指定しなかった場合、\$ORACLE_HOME/rdbms/logまたは現在のディレクトリに<xa_NULL日付.trc>ファイルが作成されます
DbgFl	何回目の段階でデバッグ・フラグを設定するかを決定します。基本的な段階の0x01、またはOCI段階の0x04などを使用します

以下は、OPENINFO項目のオプションでLogDirとDbgFlを使用する例です。

```
OPENINFO="Oracle_XA+Acc=P/アカウント/パスワード +SesTm=60+LogDir=/tmp+DbgFl=0x01"
```

#### 注

開発時にデバッグモードを解除して使用すると、後にディスクがフル状態になることを避けることができます。

## データベース・スクリプト

以下は、Oracleテーブルを作成するスクリプトの例です。

<mktable.sql>

```
sqlplus scott/tiger << EOF
    create table sel_his(
        account_id number(6),
        teller_id number(6),
        branch_id number(6),
        amount number(6)
    );
create unique index idx_tdbl on sel_his(account_id);
EOF
```

以下は、Oracleテーブル及びデータを出力するスクリプトの例です。

<sel.sql>

```
sqlplus scott/tiger << EOF
desc sel_his;
select * from sel_his;
EOF
```

## クライアント・プログラム

以下は、クライアント・プログラムの例です。

<oins\_cli.c>

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tx.h>
#include "../sdl/demo.s"
#define NARRAY 10
#define NOTFOUND 1403

main(int argc, char *argv[])
{
    struct stru_his *transf;
    int i, j;
    long urcode, nrecv, narray = NARRAY;
    long account_id, teller_id, branch_id, amount;

    if (argc != 2) {
        fprintf(stderr, "Usage: %s ACCOUNT_ID !\n", argv[0]);
        exit(0);
    }

    if (tpstart((TPSTART_T *) NULL) == -1) { /* Tmaxに接続 */
        fprintf(stderr, "TPSTART_T(tpinfo) failed -> %s!\n",
            tpstrerror(tperrno)) ;
        exit(1) ;
    }

    /* c structure構造でバッファ作成 */
    transf = (struct stru_his *) tpalloc("STRUCT", "stru_his", 0);
    if (transf == (struct stru_his *) NULL) {
        fprintf(stderr, "Tpalloc failed->%s!\n",
            tpstrerror(tperrno)) ;
        tpend();
        exit(1);
    }
}
```

```

memset(transf, 0x00, sizeof(struct stru_his));

account_id = atoi(argv[1]);
transf->ACCOUNT_ID = account_id;

/* トランザクション・タイムアウトの設定 */
tx_set_transaction_timeout(30);

/* グローバル・トランザクションの開始 */
if (tx_begin() < 0) {
    fprintf(stderr, "tx_begin() failed ->%s!\n",
        tpstrerror(tperrno));
    tpfree((char*)transf);
    tpend();
    exit(0) ;
}

if (tpcall("ORASEL", (char *)transf, 0, (char **)&transf, &nrecv,
    TPNOFLAGS)== -1){
    /* 同期通信で"ORASEL"サービス要求 */
    fprintf(stderr, "Tpcall(SELECT...)error->%s ! ",
        tpstrerror(tperrno)) ;
    tpfree((char *)transf);
    /* 失敗時、トランザクションをキャンセル */
    tx_rollback();
    tpend();
    exit(0) ;
}

/* 成功時、トランザクションをコミット */
if (tx_commit() == -1) {
    fprintf(stderr, "tx_commit() failed ->%s!\n",
        tpstrerror(tperrno)) ;
    tpfree((char *)transf);
    tpend();
    exit(0) ;
}

/* 受け取ったデータは構造体の配列です。*/
for (j =0 ; j < tpurcode ; j++) {
    /* Oracleで問い合わせたデータ結果を印刷 */
    if (j == 0)
        printf("%-12s%-10s%-10s%-10s\n",
            "ACCOUNT_ID", "TELLER_ID", "BRANCH_ID", "AMOUNT");
    account_id=transf[j].ACCOUNT_ID;
    teller_id=transf[j].TELLER_ID;
    branch_id=(*(transf+j)).BRANCH_ID;
    amount=transf[j].AMOUNT;
    printf("%-12d %-10d %-10d %-10d\n", account_id, teller_id, branch_id, amount);
}

```



```

    }
    /* 問い合わせたデータがないか、最後の場合 */
    if (urcode == NOTFOUND) {
        printf("No records selected!\n");
        tpfree((char *)transf);
        tpend();
        return 0;
    }

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

```

## サーバー・プログラム

以下は、サーバー・プログラムの例です。

<oselsvc.pc>

```

#include <stdio.h>
#include <usrinc/atmi.h>
#include "../sdl/demo.s"
#define NARRAY 10
#define TOOMANY 2112
#define NOTFOUND 1403
EXEC SQL include sqlca.h;

EXEC SQL begin declare section;
    long key, rowno= NARRAY;
    long account_id[NARRAY],teller_id[NARRAY], branch_id[NARRAY],
        amount[NARRAY] ;
EXEC SQL end declare section;

ORASEL(TPSVCINFO *msg)
{
    struct stru_his *transf;
    int i , lastno;
    transf=(struct stru_his *) msg->data;

    /* msgバッファの内容をプログラム変数に渡す*/
    key = transf->ACCOUNT_ID;

    /* transfバッファのサイズを再調整 */
    if(((transf=(struct stru_his *) tprealloc((char*)transf,
        sizeof(struct stru_his) * NARRAY ))==(struct stru_his*)NULL)){
        fprintf(stderr, "tprealloc error ->%s\n",

```

```

        tpstrerror(tperrno));
        tpreturn(TPFAIL, tperrno, NULL, 0, TPNOFLAGS);
    }
EXEC SQL select account_id, teller_id, branch_id, amount
        into :account_id, :teller_id, :branch_id, :amount from sel_his

        where account_id > :key
/* account_idがクライアントで送ったkey値より大きいものを */
order by account_id;          /* グローバル変数に入れる */

/* sqlエラーチェック(問い合わせたものがない場合、あるいは多過ぎる場合は除外) */
if (sqlca.sqlcode!=0 && sqlca.sqlcode!=NOTFOUND && sqlca.sqlcode!=TOOMANY)
{
    fprintf(stderr, "SQL ERROR ->NO(%d):%s\n", sqlca.sqlcode,
        sqlca.sqlerrm.sqlerrmc) ;
    tpreturn(TPFAIL, sqlca.sqlcode, NULL, 0, TPNOFLAGS);
}

/* アクセスした数をlastnoに入れる */
lastno = sqlca.sqlerrd[2];

/* 問い合わせたデータが多過ぎる */
if (sqlca.sqlcode == TOOMANY)
    lastno =rowno;

/* No records */
if (lastno == 0)
    transf->ACCOUNT_ID = 0;

/* Oracleで問い合わせたデータを転送するバッファに入れる */
for ( i = 0 ; i < lastno; i++) {
    transf[i].ACCOUNT_ID = account_id[i];
    transf[i].TELLER_ID = teller_id[i];
    transf[i].BRANCH_ID = branch_id[i];
    transf[i].AMOUNT = amount[i];
}
tpreturn(TPSUCCESS, lastno, transf, i * sizeof(struct stru_his),
TPNOFLAGS );
}

```

### 12.3.3. Informix Insertプログラム

クライアントはユーザーからの入力を受け、構造体バッファに入れてサービスを呼び出します。サーバーはこれを受け取り、該当テーブルに追加します。クライアントはトランザクションを指定し、エラーが発生した場合はロールバックします。

Informixアプリケーションをコンパイルする前に、以下の事項を確認してください。

## 1. UNIX環境を確認します。(profile、.login、.cshrc)

以下の値を設定します。

```
INFORMIXDIR=/home/informix
INFORMIXSERVER=tmax
ONCONFIG=onconfig
PATH=$INFORMIXDIR/bin: ...
LD_LIBRARY_PATH=/home/informix/lib:/home/informix/lib/esql:
...
```

## 2. Makefileを確認します。

以下のオペレーションと設定を確認します。

```
# Server esql makefile

TARGET = <target filename>
APOBJS = $(TARGET).o
SDLFILE = info.s

LIBS = -lsvr -linfs
# solarisの場合、-lnsl -lsocketを追加

OBJS = $(APOBJS) $(SDLOBJ) $(SVCTOBJ)
SDLOBJ = ${SDLFILE:.s=_sdl.o}
SDLC = ${SDLFILE:.s=_sdl.c}
SVCTOBJ = $(TARGET)_svctab.o

CFLAGS = -O -I$(INFORMIXDIR)/incl/esql -I$(TMAXDIR)
# Solaris 32bit, Compaq, Linux: CFLAGS = -O -I$(INFORMIXDIR)/incl/esql
#                                     -I$(TMAXDIR)
# Solaris 64bit: CFLAGS = -xarch=v9 -O -I$(INFORMIXDIR)/incl/esql -I$(TMAXDIR)
# HP 32bit: CFLAGS = -Ae -O -I$(INFORMIXDIR)/incl/esql -I$(TMAXDIR)
# HP 64bit: CFLAGS = -Ae +DA2.0W +DD64 +DS2.0 -O -I$(INFORMIXDIR)/incl/esql
#                                     -I$(TMAXDIR)
# IBM 32bit: CFLAGS = -q32 -brtl -O -I$(INFORMIXDIR)/incl/esql -I$(TMAXDIR)
# IBM 64bit: CFLAGS = -q64 -brtl -O -I$(INFORMIXDIR)/incl/esql -I$(TMAXDIR)

INFLIBD = $(INFORMIXDIR)/lib/esql
INFLIBDD = $(INFORMIXDIR)/lib
INFLIBS = -lifsql -lifasf -lifgen -lifos -lifgls -lm -ldl -lcrypt
#                                     $(INFORMIXDIR)/lib/esql/
checkapi.o -lifglx -lifxa

APPDIR = $(TMAXDIR)/appbin
```

```

SVCTDIR = $(TMAXDIR)/svct
TMAXLIBDIR = $(TMAXDIR)/lib

#
.SUFFIXES : .ec .s .c .o

.ec.c :
    esql -e $*.ec

#
# server compile
#
all: $(TARGET)

$(TARGET):$(OBJS)
    $(CC) $(CFLAGS) -L$(TMAXLIBDIR) -L$(INFLIBD) -L$(INFLIBDD) -o $(TARGET)

    $(OBJS) $(LIBS) $(INFLIBS)
    mv $(TARGET) $(APPDIR)/.
    rm -f $(OBJS)

$(APOBJS): $(TARGET).ec
    esql -e -I$(TMAXDIR)/usrinc $(TARGET).ec
    $(CC) $(CFLAGS) -c $(TARGET).c

$(SVCTOBJ):
    touch $(SVCTDIR)/$(TARGET)_svctab.c
    $(CC) $(CFLAGS) -c $(SVCTDIR)/$(TARGET)_svctab.c

$(SDLOBJ):
    $(TMAXDIR)/bin/sdlc -i ../sdl/$(SDLFILE)
    $(CC) $(CFLAGS) -c ../sdl/$(SDLC)

#
clean:
    -rm -f *.o core $(TARGET) $(TARGET).lis

```

#### <TMS Makefile>

```

#
TARGET = info_tms

INFOLIBDIR = ${INFORMIXDIR}/lib
INFOELIBDIR = ${INFORMIXDIR}/esql
INFOLIBD = ${INFORMIXDIR}/lib/esql
INFOLIBS = -lifsq1 -lifasf -lifgen -lifos -lifgls -lm -ldl -lcrypt
           /opt/informix/lib/esql/checkapi.o

```

```

-lifglx -lifxa
# solarisはライブラリーに-lnsl -lsocket -laio -lelfを追加

CFLAGS =-O -I$(INFORMIXDIR)/incl/esql -I$(TMAXDIR)
# Solaris 32bit, Compaq, Linux: CFLAGS = -O -I$(INFORMIXDIR)/incl/esql
-I$(TMAXDIR)
# Solaris 64bit: CFLAGS = -xarch=v9 -O -I$(INFORMIXDIR)/incl/esql -I$(TMAXDIR)
# HP 32bit: CFLAGS = -Ae -O -I$(INFORMIXDIR)/incl/esql -I$(TMAXDIR)
# HP 64bit: CFLAGS = -Ae +DA2.0W +DD64 +DS2.0 -O -I$(INFORMIXDIR)/incl/esql
-I$(TMAXDIR)
# IBM 32bit: CFLAGS = -q32 -brtl -O -I$(INFORMIXDIR)/incl/esql -I$(TMAXDIR)
# IBM 64bit: CFLAGS = -q64 -brtl -O -I$(INFORMIXDIR)/incl/esql -I$(TMAXDIR)

TMAXLIBDIR = $(TMAXDIR)/lib
TMAXLIBS = -ltms -linfs
# CC = /opt/SUNWspro/bin/cc : solaris only

$(TARGET): $(APOBJ)
    $(CC) $(CFLAGS) -o $(TARGET) -L$(TMAXLIBDIR) -L$(INFOLIBD)
-L$(INFOLIBDIR) -L
$(INFOELIBDIR) $(INFOLIBS) $(TMAXLIBS)
    mv $(TARGET) $(TMAXDIR)/appbin

#
clean:
    -rm -f *.o core $(TARGET)

```

## プログラム構成

- 共通プログラム

プログラム・ファイル	説明
demo.s	構造体バッファの設定ファイルです
sample.m	Tmaxシステム環境ファイルです
mkdb.sql	データベースを作成するSQLスクリプトです。XAモードはデータベースがロギングモードで作成される際にサポートします
mktable.sql	データベース・テーブルを作成するSQLスクリプトです
sel.sql	データベーステーブルの内容を出力するSQLスクリプトです
info.s	SDLFILEです

- クライアント・プログラム

プログラム・ファイル	説明
client.c	クライアント・プログラムです

- サーバー・プログラム

構成	説明
tdbsvr.ec	サーバー・プログラムです
Makefile	Tmaxが提供するMakefileを修正します

## プログラムの特徴

- クライアント・プログラム

機能	説明
Tmax接続	基本接続
バッファ・タイプ	STRUCT
通信タイプ	tpcall()による同期通信
トランザクション処理	クライアントでトランザクション範囲を指定

- サーバー・プログラム

機能	説明
サービス	INSERT
データベース接続	Informixデータベース

## 構造体バッファ

以下は、構造体バッファの例です。

<demo.s>

```
struct info {
    char seq[8];
    char data01[128];
    char data02[128];
    char data03[128];
};
```

## Tmax環境ファイル

以下は、Tmax環境ファイルの例です。

<sample.m>

```
*DOMAIN
resrc          SHMKEY = 77990, MAXUSER = 256

*NODE
tmax           TMAXDIR = "/home/tmax", A
              APPDIR = "/home/tmax/appbin",
              PATHDIR = "/home/tmax/path"

*SVRGROUP
svg1           NODENAME = tmax,
              DBNAME = INFORMIX,
              OPENINFO = "stores7",
              CLOSEINFO = "",
              TMSNAME = info_tms

*SERVER
tdbsvr         SVGNAME = svg1, MIN = 1, MAX = 5

*SERVICE
INSERT         SVRNAME = tdbsvr
```

以下は、環境設定に追加された項目についての説明です。

項目	説明
DBNAME	使用するデータベース名を設定します
OPENINFO, CLOSEINFO	Informixデータベース接続及び解除のための情報を、tpsvrinfo()、tpsvrdone()で呼び出します
TMSNAME	トランザクション処理を行うプロセス名を指定する項目です。OPENINFOの指定による自動トランザクションを処理します。svg1に属する該当サービスを自動トランザクション状態で処理します

## データベース・スクリプト

以下は、Informixテーブルを作成するスクリプトの例です。

<mktable.sql>

```
dbaccess << EOF
create database stores7 with buffered log;
grant connect to public;

database stores7;
drop table testdb1;
```

```
create table testdb1 (
    seq          VARCHAR(8) ,
    data01       VARCHAR(120) ,
    data02       VARCHAR(120) ,
    data03       VARCHAR(120)
) lock mode row;

create unique index idx_tdb1 on testdb1(seq);
EOF
```

以下は、Informixテーブル及びデータを出力するスクリプトの例です。

<sel.sql>

```
dbaccess << EOF
database stores7;
select * from testdb1;
EOF
```

## クライアント・プログラム

以下は、クライアント・プログラムの例です。

<client.c>

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tx.h>
#include "info.s"
main(int argc, char **argv)
{
    struct info *transf;
    char data[256];
    long nrecv;

    /* Tmaxに接続 */
    if ((tpstart((TPSTART_T *)NULL) == -1) {
        fprintf(stderr, "tpstart(TPINFO...) failed ->%s!\n",
            tpstrerror(tperrno)) ;
        exit(1) ;
    }

    /* アプリケーション・プログラムで使用するバッファ・メモリの割り当て */
    if ((transf=(struct info *)tpalloc ("STRUCT","info",0))==(struct info *)NULL){

        fprintf(stderr, "tpalloc(struct info, ...) failed ->%s!\n",
            tpstrerror(tperrno)) ;
        tpend();
    }
}
```



```

        exit(1) ;
    }

    /* 転送するデータフィールドの設定 */
    strcpy(transf->seq, "000001");
    strcpy(transf->data01, "Hello");
    strcpy(transf->data02, "World");
    strcpy(transf->data03, "1234");

    /* トランザクション・タイムアウトの設定 * /
    tx_set_transaction_timeout (30);

    /* トランザクションの開始を通知 */
    if (tx_begin() < 0) {
        fprintf(stderr, "tx_begin() failed ->%s!\n",
            tpstrerror(tperrno)) ;
        tpfree ((char *)transf);
        tpend( ); exit(0);
    }

    /* サービスの呼び出し */
    if (tpcall("INSERT", (char*) transf, 0, (char **)&transf, &nrecv, TPNOFLAGS) == -1) {

        fprintf(stderr, "tpcall(struct info, ...)
        failed ->%s!\n", tpstrerror(tperrno)) ;
        tx_rollback ();
        tpfree ((char *)transf),
        tpend();
        exit(0);
    }

    /* トランザクションが成功 */
    if (tx_commit () < 0) {
        fprintf(stderr, "tx_commit() failed ->%s!\n", tpstrerror(tperrno)) ;
        tpfree ((char *)transf);
        tpend( );
        exit(0);
    }

    tpfree ((char *)transf );
    /* Tmax接続を終了 */
    tpend( ) ;
}

```

## サーバー・プログラム

以下は、サーバー・プログラムの例です。

< tdbsvr.ec>

```
#include <stdio.h>
#include <ctype.h>
#include <usrinc/atmi.h>
#include "info.s"

EXEC SQL include sqlca.h;

/* サービス名 */
INSERT(TPSVCINFO *msg)
{
    /* プログラム上のバッファ・タイプの宣言 */
    struct info *INFO;

    /* SQL文内のバッファ・タイプの宣言 */
    EXEC SQL begin declare section;
    varchar seq[8],buf01[128],buf02[128],buf03[128];
    EXEC SQL end declare section;

    /* メッセージ・バッファから構造体型でデータを受け取る */
    INFO = (struct info *)msg -> data;

    /* 構造体形式で受け取ったデータをデータベース・バッファにコピー */
    strcpy(seq, INFO->seq);
    strcpy(buf01, INFO->data01);
    strcpy(buf02, INFO->data02);
    strcpy(buf03, INFO->data03);

    /* SQL文を挿入 */
    EXEC SQL insert into testdb1 (seq,data01,data02,data03)
    values(:seq, :buf01, :buf02, :buf03);

    /* エラーが発生した場合 */
    if ( sqlca.sqlcode != 0) {
        /* 挿入失敗を通知 */
        printf("SQL error => %d !" ,sqlca.sqlcode);
        tpreturn (TPFAIL, sqlca.sqlcode, NULL, 0, TPNOFLAGS);
    }

    /* 挿入が正常に終了した場合 */
}
```

```

        tpreturn (TPSUCCESS, 0, NULL, 0, TPNOFLAGS);
    }

```

## 12.3.4. Informix Selectプログラム

クライアントはユーザーからの入力を受け、構造体バッファに入れてサービスを呼び出します。サーバーはこれに該当する全データを受け取り、構造体配列を使用して結果を返します。クライアントはトランザクションを指定し、エラーが発生した場合はロールバックできるようにします。.

### プログラム構成

- 共通プログラム

プログラム・ファイル	説明
acct.s	SDLFILEです
sample.m	Tmax環境ファイルです
mkdb.sql	データベースを作成するSQLスクリプトです
mktables.sql	データベース・テーブルを作成するSQLスクリプトです
sel.sql	データベース・テーブルの内容を出力するSQLスクリプトです

- クライアント・プログラム

構成	説明
client.c	クライアント・プログラムです
cdate.c	client.cで使用された関数モジュールです

- サーバー・プログラム

構成	説明
sel_acct.ec	サービス・プログラムのInformixソースです
Makefile	Tmaxが提供するMakefileを修正します

### プログラムの特徴

- クライアント・プログラム

区分	説明
Tmax接続	基本接続
バッファ・タイプ	STRUCT
通信タイプ	tpcall()による同期通信
トランザクション処理	クライアントでトランザクション範囲を指定

- サーバー・プログラム

区分	説明
サービス	SEL_ACCT
データベース接続	Informixデータベース
バッファ使用	必要に応じてバッファサイズを再調整

## 構造体バッファ

以下は、構造体バッファの例です。

<acct.s>

```
struct stru_acct {
    int    ACCOUNT_ID;
    char   PHONE[20];
    char   ADDRESS[80];
};
```

## Tmax環境ファイル

以下は、Tmax環境ファイルの例です。

<sample.m>

```
*DOMAIN
resrc          SHMKEY = 77990, MAXUSER = 256

*NODE
tmax           TMAXDIR = "/home/tmax",
               APPDIR  = "/home/tmax/appbin",
               PATHDIR = "/home/tmax/path"

*SVRGROUP
svg1           NODENAME = tmax,
               DBNAME  = INFORMIX,
               OPENINFO = "stores7",
               CLOSEINFO = "",
```

```

TMSNAME = info_tms

*SERVER
SEL_ACCT      SVGNAME = svg1, MIN = 1, MAX = 5

*SERVICE
SEL_ACCT      SVRNAME = sel_acct

```

以下は、環境設定に追加された項目についての説明です。

項目	説明
DBNAME	使用するデータベースを設定します
OPENINFO, CLOSEINFO	Informixデータベースへの接続及び解除のための情報を、tpsvrinit()、tpsvrdone()で呼び出します
TMSNAME	トランザクション処理を行うプロセス名を指定する項目です。OPENINFOの指定による自動トランザクションを処理します。svg1に属する該当サービスを自動トランザクション状態で処理します

## データベース・スクリプト

以下は、Informixテーブルを作成するスクリプトの例です。

<mkdb.sql>

```

dbaccess << EOF
create database stores7 with buffered log;
grant connect to public;

database stores7;
drop table ACCOUNT;

create table ACCOUNT (
    account_id INTEGER,
    phone VARCHAR(20),
    address VARCHAR(80)
) lock mode row;

create unique index idx_tdb1 on ACCOUNT(account_id);
EOF

```

以下は、Informixテーブル及びデータを出力するスクリプトの例です。

<sel.sql>

```

dbaccess << EOF
database stores7;

```

```
select * from ACCOUNT;
EOF
```

## クライアント・プログラム

以下は、クライアント・プログラムの例です。

<client.c>

```
#include <stdio.h>
#include <time.h>
#include <usrinc/atmi.h>
#include <usrinc/tx.h>
#include "acct.s"
#define NOTFOUND 1403

void htime(char *, int *);

main(int argc, char *argv[ ])
{
    struct stru_acct *transf;
    float tps;
    int i, j, loop, cnt_data = 0, sec1, sec2;
    long urcode, nrecv, narray;
    char ts[30], te[30], phone[20], address[80];
    int account_id, key;

    if(argc != 2) {
        fprintf(stderr, "Usage: %s LOOP (NARRAY = 30) !\n", argv[0]);
        exit(0) ;
    }

    /* ユーザーが実行したい回数分ループを行う */
    loop = atoi(argv[1]);

    /* Tmaxに接続 */
    if (tpstart((TPSTART_T *)NULL) == -1) {
        fprintf(stderr, "tpstart(tpinfo) failed ->%s!\n",
            tpstrerror(tperrno));
        exit(1) ;
    }

    /* sec1 = 開始時間 */
    htime(ts, &sec1); key=0;

    /* メッセージ・バッファの割り当て */
    for( i = 0; i < loop; i++) {
```

```

    if ((transf=(struct stru_acct *)tpalloc("STRUCT","stru_acct",0))
        ==(struct stru_acct *)NULL) {
        fprintf(stderr,"Tpalloc(STRUCT.. )failed->%s!\n" ,
            tpstrerror(tperrno)) ;
        tpend(); exit(1);
    }
    transf -> ACCOUNT_ID = key;

    /* time-out value= 30 */
    tx_set_transaction_timeout(30) ;

    if (tx_begin() < 0) { /* トランザクション開始 */
        fprintf(stderr, "tx_begin() failed ->%s!\n", tpstrerror(tperrno)) ;

        tpfree((char*)transf);
        tpend();
        exit(0);
    }

    /* select service呼び出し */
    if (tpcall("SEL_ACCT", (char *)transf, 0, (char **)&transf, &nrecv,
        TPNOFLAGS)== -1) {
        /* service error : message buffer free & トランザクションのキャンセル & 接
続終了 */
        fprintf(stderr,"Tpcall(SELECT...)error->%s! " ,
            tpstrerror(tperrno)) ;
        tpfree ((char *)transf);
        tx_rollback ( ) ;
        tpend( ) ;
        exit ( 1 ) ;
    }
    urcode = tpurcode;

    /*正常にサービス完了 : トランザクション結果で実際にリソースを変化させる*/
    if (tx_commit() < 0 ) {
        fprintf(stderr, "tx_commit() failed ->%s!\n", tpstrerror(tperrno))
;

        tpfree((char *)transf);
        tpend();
        exit(0);
    }

    /*データを問い合わせた場合*/
    if ( urcode != NOTFOUND) {
        narray =urcode;
        /* 問い合わせたデータの最後のレコード */
        key=transf[narray-1].ACCOUNT_ID;

```

```

/* 問い合わせた個数分の結果をユーザーに出力 */
for ( j = 0 ; j < narray ; j++ ) {
    if ( j == 0 )
        printf("%-10s%-14s%s\n", "ACCOUNT_ID", "PHONE", "ADDRESS" ) ;
        account_id = transf[j].ACCOUNT_ID;
        strcpy(phone, transf[j].PHONE);
        strcpy(address, transf[j].ADDRESS);
        printf("%-10d %-14s %s\n", account_id, phone, address);
    }/* for2 end */

/* 全結果数の増加 */
cnt_data += j;

/* message buffer free */
tpfree ((char *)transf);
if(urcode == NOTFOUND) {
    printf("No records selected!\n");
    break ;
}
}/* for1 end */

/* message buffer free */
tpfree ((char *)transf);
/* Tmax接続の終了 */
tpend ();

/* sec2 = 終了時間 */
htime(te,&sec2);

/* データ1つあたりの処理時間を計算 */
printf("TOT.COUNT = %d\n", cnt_data);
printf("Start time = %s\n", ts);
printf("End time = %s\n", te);
if ((sec2-sec1) != 0)
    tps = (float) cnt_data / (sec2 - sec1);
else
    tps = cnt_data;
printf("Interval = %d secs ==> %10.2f T/S\n", sec2-sec1,tps);
}

htime(char *cdate, int *sec)
{
    long time(), timef, pt;
    char ct[20], *ap;
    struct tm *localtime(), *tmp;

    pt = time(&timef);

```



```

        *sec = pt;
        tmp = localtime(&timef);
        ap = asctime(tmp);

        sscanf(ap, "%*s%*s%*s%*s", ct);
        sprintf( cdate, "%02d. %02d. %02d %s", tmp->tm_year, ++tmp->tm_mon,
                tmp->tm_mday, ct);
    }

```

## サーバー・プログラム

以下は、サーバー・プログラムの例です。

<sel\_acct.pc>

```

#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tx.h>
#include "acct.s"
#define NFETCH 5
#define NOTFOUND 100

EXEC SQL include sqlca.h;
EXEC SQL begin declare section;
long account_id, key;
varchar phone[20], address[80];
EXEC SQL end declare section;

SEL_ACCT(TPSVCINFO *msg)
{
    int i , j , nfetch;
    int return_code;
    struct stru_acct *ACCT_V;

    /*クライアントのデータを受け取る */
    ACCT_V = (struct stru_acct *) msg->data;

    /* 問い合わせたいaccount idをkeyに移す */
    key = ACCT_V->ACCOUNT_ID;

    /* クライアント・バッファサイズのリ割り当て */
    if ((ACCT_V = (struct stru_acct *) tprealloc((char *) ACCT_V,
        sizeof(struct stru_acct)*NFETCH )) == (struct stru_acct *) NULL) {
        fprintf(stderr, "tprealloc error = %s\n", tpstrerror(tperrno));
        tpreturn (TPFAIL, tperrno, NULL, 0, TPNOFLAGS);
    }
}

```

```

/*バッファの初期化*/
ACCT_V->ACCOUNT_ID = 0;
strcpy(ACCT_V->PHONE," " );
strcpy(ACCT_V->ADDRESS," " );

/* ACCOUNTテーブルからphone、 addressフィールドを問い合わせ */
EXEC SQL declare CUR_1 cursor for
    select account_id,phone,address
    into :account_id, :pfone, :address
    from ACCOUNT
    where account_id > :key; /* フィールド・キーよりaccount idが大きいとき */

/* cursor open */
EXEC SQL open CUR_1;
/* cursor open error */
if (sqlca.sqlcode != 0) {
    printf("open cursor error !\n");
    tpreturn(TPFAIL, sqlca.sqlcode, NULL, 0, TPNOFLAGS);
}

nfetch=0 ;
return_code = NOTFOUND;

/* cursor open success */
while (sqlca.sqlcode == 0) {
    /* カーソルからデータを1つずつフェッチ */
    EXEC SQL fetch CUR_1;

    /* fetch error */
    if (sqlca.sqlcode != 0) {
        if (sqlca.sqlcode == NOTFOUND)
            break ;
        break ;
    }

    ACCT_V[nfetch].ACCOUNT_ID = account_id;
    strcpy(ACCT_V[nfetch].PHONE, phone );
    strcpy(ACCT_V[nfetch].ADDRESS, address );

    /* 問い合わせたデータ数の増加 */
    nfetch++;
    return_code = nfetch

    /* NFETCHの数の分だけ問い合わせた場合、while文から抜ける */
    if (nfetch > NFETCH) {

```

```

        nfetch = NFETCH;
        return_code = nfetch;
        break ;
    }
}
/* cursor close */
EXEC SQL close CUR_1;

/* 問い合わせ結果とそのデータをクライアントに返す */
tpreturn ( TPSUCCESS, return_code, (char *)ACCT_V,
          sizeof(struct stru_acct)*nfetch, TPNOFLAGS);
}

```

## 12.3.5. DB2プログラム

クライアントはユーザーの入力を受け、STRINGバッファにEMPNOを入れてサービスを呼び出し、サーバーはこれを受けて該当するテーブルに追加します。クライアントはトランザクションを指定し、エラーが発生した場合にロールバックできるようにします。

### プログラム構成

- 共通プログラム

プログラム・ファイル	説明
sample.m	Tmax環境設定ファイルです
create.ers	データベース・テーブルを作成するSQLスクリプトです

- クライアント・プログラム

プログラム・ファイル	説明
clidb2tx.c	クライアント・プログラムです

- サーバー・プログラム

プログラム・ファイル	説明
svr_db2.sqc	サービス・プログラムのOracleソースです
Makefile	TMSおよびサーバー・プログラムをコンパイルするメイクファイルです

### プログラムの特徴

- クライアント・プログラム

機能	説明
Tmax接続	基本接続
バッファ・タイプ	STRING
通信タイプ	tpcall()による同期型通信
トランザクション処理	クライアントでトランザクション範囲を指定

● サーバー・プログラム

区分	説明
サービス	XASERVICE2
データベース接続	DB2データベース

## DB2連携の前の確認事項

以下は、DB2連携時の考慮事項です。

1. DB2クライアント・エンジンのビット(32あるいは64)を確認します。
2. DB2クライアントのバージョンが8.0以下なのか、9.0以上なのかを確認します。
3. Tmax環境設定ファイルのSVRGROUPセクションのXAOPTION項目を1、2項の結果に基づいて設定します。

– DB2クライアントのバージョンが8.0以下の場合

- DB2クライアント・エンジンが32ビットの場合

```
XAOPTION = "DYNAMIC"
```

- DB2クライアント・エンジンが64ビットであり、Linux/UNIX系の場合

```
XAOPTION = "DYNAMIC XASWITCH32"
```

- DB2クライアント・エンジンが64ビットであり、Windows系の場合

```
XAOPTION = "DYNAMIC"
```

– DB2クライアントのバージョンが9.0以上の場合 (DB2クライアント・エンジンのビットおよびLinux/UNIX/Windowsを問わない)

```
XAOPTION = 未設定
```

4. TMSまたはSVRのコンパイル時、1、2、3項の結果に基づいて適切なライブラリーをリンクします。

- DB2クライアントのバージョンが8.0以下の場合

- DB2クライアント・エンジンが32ビットの場合

```
-ldb2s or $(TMAXDIR)/lib/libdb2s.a
```

- DB2クライアント・エンジンが64ビットであり、Linux/UNIX系の場合

```
-ldb2_64s or $(TMAXDIR)/lib64/libdb2_64s.a
```

- DB2クライアント・エンジンが64ビットであり、Windows系の場合

```
-ldb2s or $(TMAXDIR)/lib/libdb2s.a
```

- DB2クライアントのバージョンが9.0以上の場合 (DB2クライアント・エンジンのビットおよびLinux/UNIX/Windowsを問わない)

```
-ldb2s_static or $(TMAXDIR)/lib/libdb2s_static.a
```

---

#### 参考

バージョン9.0以上のDB2クライアントの動的登録を使用する必要がある場合は、上記の8.0の場合と同様に設定することができますが、使用を推奨しません。

---

## Tmax環境ファイル

以下は、Tmax環境ファイルの例です。

<sample.m>

```
*DOMAIN
tmax1      SHMKEY = 71990, MINCLH = 1, MAXCLH = 3,
           TPORTNO = 7789, BLOCKTIME = 30,
           MAXCPC = 150

*NODE
phk        TMAXDIR = "/home/tmaxha/tmax",
           APPDIR  = "/home/tmaxha/tmax/appbin",
           PATHDIR = "/home/tmaxha/tmax/path",
           TLOGDIR = "/home/tmaxha/tmax/log/tlog",
           ULOGDIR = "/home/tmaxha/tmax/log/ulog",
           SLOGDIR = "/home/tmaxha/tmax/log/slog"

*SVRGROUP
xa_svg_db2  NODENAME = "phk",
           DBNAME  = IBMDB2,
```

```
#          XAOPTION = "DYNAMIC XASWITCH32",
          XAOPTION = "DYNAMIC",
          OPENINFO = "db=test,uid=tmaxha,pwd=ha0115",
          TMSNAME = tms_db2,
          RESTART=N

*SERVER
svr_db2          SVGNAME = xa_svg_db2

*SERVICE
XASERVICE2      SVRNAME = svr_db2
```

以下は、環境設定の追加項目についての説明です。

項目	説明
DBNAME	使用するデータベースを定義します
OPENINFO	DB2データベースの接続情報を設定します
TMSNAME	トランザクション処理を管理するプロセス名を指定します。OPENINFO指定による自動トランザクションを処理します

## データベース・スクリプト

以下は、DB2テーブル作成の例です。

```
# 'db2start' を実行
$ db2start

# 'TPTEST' というデータベースを作成
$ db2 "CREATE DATABASE TPTEST"

# TPTESTデータベースに接続
$ db2 "CONNECT TO TPTEST"

# EMPテーブルの作成
$ db2 -vf create.ers -t

<create.ers>
CREATE TABLE EMP (
    EMPNO          DECIMAL(8) NOT NULL,
    ENAME          VARCHAR(16),
    JOB            VARCHAR(16),
    SAL            DECIMAL(8),
    HIREDATE       DECIMAL(8),
    XID            CHAR(32)
);
```

```
# EMPテーブルが作成されたことを確認
$ db2 "LIST TABLES"
```

以下は、DB2テーブルおよびデータ出力の例です。

```
$ db2 "DESCRIBE TABLE EMP"

Column          Type      Type      Length  Scale Nulls
name            schema   name                               -----
-----
EMPNO           SYSIBM   DECIMAL          8      0 No
ENAME           SYSIBM   VARCHAR         16      0 Yes
JOB             SYSIBM   VARCHAR         16      0 Yes
SAL             SYSIBM   DECIMAL          8      0 Yes
HIREDATE        SYSIBM   DECIMAL          8      0 Yes
XID             SYSIBM   CHARACTER        32      0 Yes

6 record(s) selected.
```

## クライアント・プログラム

以下は、クライアント・プログラムの例です。

<clidb2tx.c>

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <usrinc/atmi.h>

int main(int argc, char *argv[])
{
    char      *sndbuf, *rcvbuf;
    long      rcvlen;
    int       ret;

    if ( (ret = tmaxreadenv( "tmax.env", "TMAX" )) == -1 ){
        printf( "tmax read env failed. [%s]\n", tpstrerror(tperrno));
        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("sndbuf alloc failed! [%s]\n", tpstrerror(tperrno));
    tpend();
    exit(1);
}

if ((rcvbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL) {
    printf("rcvbuf alloc failed! [%s]\n", tpstrerror(tperrno));
    tpfree(sndbuf);
    tpend();
    exit(1);
}

strcpy(sndbuf, argv[1]);

ret = tx_begin();
if (ret < 0) {
    printf("tx_begin is failed. [%s]\n", tpstrerror(tperrno));
    resource_free(sndbuf, rcvbuf);
    exit(1);
} else
    printf("tx_begin success.\n");

if (tpcall("XASERVICE2", sndbuf, strlen(sndbuf), &rcvbuf, &rcvlen, 0) == -1){

    printf("Can't send request to service XASERVICE2. [%s]\n",
tpstrerror(tperrno));
    ret = tx_rollback();
    if (ret < 0)
        printf("tx_rollback is failed. [%s]\n", tpstrerror(tperrno));

    resource_free(&sndbuf, &rcvbuf);
    exit(1);
} else
    printf("XASERVCE2 success.\n");

ret = tx_commit();
if (ret < 0) {
    printf("tx_commit is failed. [%s]\n", tpstrerror(tperrno));
    ret = tx_rollback();
    if (ret < 0)
        printf("tx_rollback is failed. [%s]\n", tpstrerror(tperrno));
} else
    printf("tx_commit success.\n");
resource_free(sndbuf, rcvbuf);

```



```

        return 0;
    }

resource_free(char* sndbuf, char *rcvbuf)
{
    if (rcvbuf != NULL)
        tpfree((char*)rcvbuf);

    if (sndbuf != NULL)
        tpfree((char*)sndbuf);

    tpend();
}

```

## サーバー・プログラム

以下は、サーバー・プログラムの例です。

<svr\_db2.sqc>

```

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

EXEC SQL INCLUDE SQLCA;

EXEC SQL begin declare section;
    sqlint32 h_empno;
    sqlint32 h_count;
EXEC SQL end declare section;

XASERVICE2(TPSVCINFO *msg)
{
    char          *res_msg;
    int   h_count=0;

    h_empno = atoi(msg->data);
    printf("h_empno = %d \n", h_empno);

    EXEC SQL
        INSERT into EMP (EMPNO) VALUES (:h_empno);
    if (sqlca.sqlcode != 0) {
        printf("insertion is failed : sqlcode[%d]%d\n", sqlca.sqlcode,
sqlca.sqlstate);
        tpreturn(TPFAIL, -1, 0, 0, 0);
    } else

    EXEC SQL SELECT COUNT(*)

```

```

        INTO      :h_count
        FROM      emp
        WHERE     empno = :h_empno;

        if (sqlca.sqlcode != 0) {
            printf("insertion is failed : sqlcode[%d]%d\n", sqlca.sqlcode,
sqlca.sqlstate);
            tpreturn(TPFAIL, -1, 0, 0, 0);
        } else
            printf("insertion is success. selcnt(%d) \n", h_count);

        tpreturn(TPSUCCESS, 0, 0, 0, 0);
    }

```

## サーバー・メイクファイル

以下は、サーバー・メイクファイルの例です。

```

# Server makefile for DB2
# Linux 64bit

DB2LIBDIR = $(DB2_HOME)/lib
DB2LIBS    = -ldb2

DB          = TEST
DB2USER     = tmaxha
DB2PASS     = ha0115

TARGET     = $(COMP_TARGET)
APOBJS     = $(TARGET).o
APOBJS2    = utilemb.o
NSDLOBJ    = $(TMAXDIR)/lib64/sdl.o

#OBS        = $(APOBJS) $(APOBJS2) $(SVCTOBJ)
OBS        = $(APOBJS) $(SVCTOBJ)
SVCTOBJ    = $(TARGET)_svctab.o

#CFLAGS     = -m64 -O -I$(TMAXDIR) -I$(DB2_HOME)/include
CFLAGS     = -O -I$(TMAXDIR) -I$(DB2_HOME)/include
LDFLAGS    =

TMAXAPPDIR = $(TMAXDIR)/appbin
TMAXSVCTDIR = $(TMAXDIR)/svct
TMAXLIBDIR = $(TMAXDIR)/lib64

TMAXLIBS    = -lsvr -ldb2s          # dynmic XAOPTION=DYNAMIC (DB2 Client
v8.0(below) 32bit or DB2 Client 64bit & Windows)

```

```

#TMAXLIBS      = -lsvr -ldb2_64s      # dynmic XAOPTION=DYNAMIC (DB2 Client
v8.0(below) 64bit & Linux/Unix)
#TMAXLIBS      = -lsvr -ldb2s_static      #static XAOPTION=none (DB2 Client
v9.0(above))
#
.SUFFIXES : .c

.c.o:
    $(CC) $(CFLAGS) $(LDFLAGS) -c $<

#
# server compile
#
all: $(TARGET)

$(TARGET): $(OBSJS)
    $(CC) $(CFLAGS) $(LDFLAGS) -L$(TMAXLIBDIR) -o $(TARGET) -L$(DB2LIBDIR)
$(DB2LIBS) $(OBSJS) $(TMAXLIBS) $(NSDLOBJ)
    mv $(TARGET) $(TMAXAPPDIR)
    rm -f $(OBSJS)

$(APOBJS): $(TARGET).sqc
    db2 connect to $(DB) user $(DB2USER) using $(DB2PASS)
    db2 prep $(TARGET).sqc bindfile
    db2 bind $(TARGET).bnd
    db2 connect reset
    db2 terminate
    $(CC) $(CFLAGS) $(LDFLAGS) -c $(TARGET).c

$(SVCTOBJ):
    cp -f $(TMAXSVCTDIR)/$(TARGET)_svctab.c .
    touch ./$(TARGET)_svctab.c
    $(CC) $(CFLAGS) -c ./$(TARGET)_svctab.c

#
clean:
    :-rm -f *.o core $(TMAXAPPDIR)/$(TARGET) $(TARGET).bnd

```

## TMSメイクファイル

以下は、TMSメイクファイルの例です。

```

# TMS Makefile for DB2
# Linux 64bit

TARGET    = tms_db2
APOBJ     = dummy.o

```

```

APPPDIR = $(TMAXDIR)/appbin
TMAXLIBD= $(TMAXDIR)/lib64
TMAXLIBS = -lsvr -ldb2s          # dynmic XAOPTION=DYNAMIC (DB2 Client
v8.0(below) 32bit or DB2 Client 64bit & Windows)
#TMAXLIBS = -lsvr -ldb2_64s      # dynmic XAOPTION=DYNAMIC (DB2 Client
v8.0(below) 64bit & Linux/Unix)
#TMAXLIBS = -lsvr -ldb2s_static   #static XAOPTION=none (DB2 Client
v9.0(above))

DB2PATH = $(DB2_HOME)
DB2LIBDIR= $(DB2PATH)/lib
DB2LIB = -ldb2

CFLAGS =
LDFLAGS =
SYSLIBS =

all: $(TARGET)

$(TARGET): $(APOBJ)
    $(CC) $(CFLAGS) $(LDFLAGS) -o $(TARGET) -L$(TMAXLIBD) $(TMAXLIBS) $(APOBJ)
    -L$(DB2LIBDIR) $(DB2LIB) $(SYSLIBS)
    mv $(TARGET) $(APPPDIR)/.

$(APOBJ):
    $(CC) $(CFLAGS) -c dummy.c
#
clean:
    -rm -f *.o core $(APPPDIR)/$(TARGET)

```

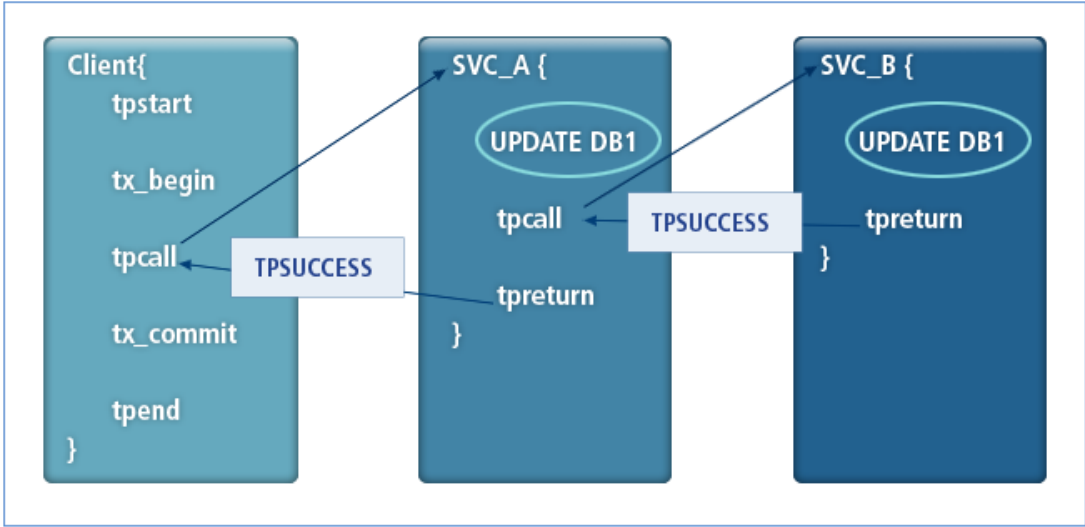
## 12.4. データベース連携プログラム

アプリケーションの開発時に応用して使用できる例を通して、実際のプログラミングの手法を説明します。データベースとの連携は、同一機種のデータベースと異機種データベースに分かれます。

### 12.4.1. 同期型モード(同機種)

以下は、同期型モードで同機種のデータベースに接続する場合のプログラムのフローを表した図です。

[図 12.2] 同期型モードのフロー図(同機種のデータベース接続)



プログラム構成

● 共通プログラム

プログラム・ファイル	説明
demo.s	SDLFILEです
sample.m	Tmax環境ファイルです
tmax.env	環境ファイルです
mktable.sql	データベース・テーブルを作成するSQLスクリプトです

● クライアント・プログラム

プログラム・ファイル	説明
client.c	クライアント・プログラムです

● サーバー・プログラム

プログラム・ファイル	説明
update.pc, insert.pc	サーバー・プログラムです
Makefile	Tmaxが提供するMakefileを修正する必要があります

プログラムの特徴

● クライアント・プログラム

区分	説明
Tmax接続	NULLパラメータで接続
バッファ・タイプ	STRUCT
サブタイプ	input構造体ファイルをsdlcでコンパイルし、SDLファイルの作成が必要
トランザクション可否	クライアントでトランザクションを指定

- サーバー・プログラム

区分	説明
サービス数	UPDATEサービスでINSERTサービスを要求
データベース接続	Oracleデータベースを使用し、Tmax環境ファイルのSVRGROUPセクションにデータベース情報を指定

## プログラム環境

区分	説明
システム	SunOS 5.7 32bit
データベース	Oracle 8.0.5

## 構造体バッファ

以下は、構造体バッファの例です。

<demo.s>

```
struct input {
    int     account_id;
    int     branch_id;
    char    phone[15];
    char    address[61];
};
```

## Tmax環境ファイル

以下は、Tmax環境ファイルの例です。

<sample.m>

```
*DOMAIN
res    SHMKEY=88000, MINCLH=1, MAXCLH=5, TPORTNO=8880, BLOCKTIME=60
```

```

*NODE
tmax          TMAXDIR = "/user/ tmax ",
              APPDIR  = "/user/ tmax /appbin",
              PATHDIR = "/user/ tmax /path",
              TLOGDIR = "/user/ tmax /log/tlog",
              ULOGDIR="/user/ tmax /log/ulog",
              SLOGDIR="/user/ tmax /log/slog"

*SVRGROUP
svg1          NODENAME = tmax, DBNAME = ORACLE,
              OPENINFO = "ORACLE_XA+Acc=P/bmt/bmt+SesTm=60",
              TMSNAME  = svg1_tms

*SERVER
update        SVGNAME=svg1
insert        SVGNAME=svg1

*SERVICE
UPDATE        SVRNAME=update
INSERT        SVRNAME=insert

```

以下は、環境ファイルの例です。

<tmax.env>

```

[tmax]
TMAX_HOST_ADDR=192.168.1.39
TMAX_HOST_PORT=8880
SDLFILE=/user/tmax/sample/sdl/tmax.sdl
TMAX_CONNECT_TIMEOUT=5

```

## データベース・スクリプト

以下は、データベース・テーブルを作成するSQLスクリプトの例です。

<mktable.sql>

```

$ORACLE_HOME/bin/sqlplus bmt/bmt <<!
drop table ACCOUNT;
create table ACCOUNT (
    ACCOUNT_ID integer,
    BRANCH_ID integer not null,
    SSN        char(13) not null,
    BALANCE    number,
    ACCT_TYPE  char(1),
    LAST_NAME  char(21),
    FIRST_NAME char(21),
    MID_INIT   char(1),

```

```

        PHONE      char(15),
        ADDRESS    char(61),
        CONSTRAINT ACCOUNT_PK PRIMARY KEY(ACCOUNT_ID)
);
quit
!
```

## クライアント・プログラム

以下は、クライアント・プログラムの例です。

<client.c>

```

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

#define TEMP_PHONE "6283-2114"
#define TEMP_ADDRESS "Korea"

int main(int argc, char *argv[])
{
    struct input *sndbuf;
    char *rcvbuf;
    int acnt_id, n, timeout;
    long len;

    if (argc != 2) {
        fprintf(stderr, "Usage:%s account_id \n", argv[0]);
        exit(1);
    }

    acnt_id = atoi(argv[1]);
    timeout = 5;

    n = tmaxreadenv("tmax.env", "tmax");
    if (n < 0) {
        fprintf(stderr, "tmaxreadenv fail! tperrno = %d\n", tperrno);
        exit(1);
    }

    n = tpstart((TPSTART_T *)NULL);
    if (n < 0) {
        fprintf(stderr, "tpstart fail! tperrno = %s\n", tperrno);
        exit(1);
    }
}
```



```

sndbuf = (struct input *)tpalloc("STRUCT", "input", `
    sizeof(struct input));
if (sndbuf == NULL) {
    fprintf(stderr, "tpalloc fail: sndbuf tperrno = %d\n", tperrno);
    tpend();
    exit(1);
}

rcvbuf = (char *)tpalloc("STRING", NULL, 0);
if (rcvbuf == NULL) {
    fprintf(stderr, "tpalloc fail: rcvbuf tperrno = %d\n", tperrno);
    tpend();
    exit(1);
}

sndbuf->account_id = acct_id;
sndbuf->branch_id = acct_id;
strcpy(sndbuf ->phone, TEMP_PHONE);
strcpy(sndbuf ->address, TEMP_ADDRESS);

tx_set_transaction_timeout(timeout);
n = tx_begin();
if (n < 0)
    fprintf(stderr, "tx begin fail! tperrno = %d\n", tperrno);

n = tpcall("UPDATE", (char *)sndbuf, sizeof(struct input),
    (char **)&rcvbuf, (long *)&len, TPNOFLAGS);
if (n < 0) {
    fprintf(stderr, "tpcall fail! tperrno = %d\n", tperrno);
    tpend();
    exit(1);
}

n = tx_commit();
if (n < 0) {
    fprintf(stderr, "tx commit fail! tx error = %d \n", n);
    tx_rollback();
    tpend();
    exit(1);
}

printf("rtn msg = %s\n", rcvbuf);
tpfree((char *)sndbuf);
tpfree((char *)rcvbuf);
tpend();
}

```

## サーバー・プログラム

以下は、データベースに更新するサーバー・プログラムの例です。

<update.pc>

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

#define OKMSG "YOU COMPLETE THE TRANSACTION"

EXEC SQL include sqlca.h;
EXEC SQL BEGIN DECLARE SECTION;
    int      account_id;
    int      branch_id;
    char      ssn[15];
    char      phone[15];
    char      address[61];
EXEC SQL END DECLARE SECTION;

UPDATE(TPSVCINFO *msg)
{
    struct input *rcvbuf;
    int      ret;
    long      acnt_id, rcvlen;
    char      *send;

    rcvbuf = (struct input *) (msg->data);
    send = (char *) tmalloc("STRING", NULL, 0);
    if (send == NULL) {
        fprintf(stderr, "tpalloc fail errno = %s\n", strerror(tperrno));
        tpreturn(TPFAIL, 0, (char *) NULL, 0, 0);
    }
    account_id = rcvbuf->account_id;
    branch_id = rcvbuf->branch_id;
}
```

以下は、データベースに挿入するサーバー・プログラムの例です。

<insert.pc>

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

```

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

#define OKMSG "YOU COMPLETE THE TRANSACTION"

EXEC SQL include sqlca.h;
EXEC SQL BEGIN DECLARE SECTION;
    int      account_id;
    int      branch_id;
    char      ssn[15];
    char      phone[15];
    char      address[61];
EXEC SQL END DECLARE SECTION;

INSERT(msg)
TPSVCINFO *msg;
{
    struct input *rcvbuf;
    int      ret;
    long      acnt_id;
    char      *send;

    rcvbuf = (struct input *) (msg->data);
    send = (char *) tmalloc("STRING", NULL, 0);
    if (send == NULL) {
        fprintf(stderr, "tpalloc fail errno = %s\n", tpstrerror(tperrno));
        tpreturn(TPFAIL, 0, (char *) NULL, 0, TPNOFLAGS);
    }

    account_id = rcvbuf->account_id;
    branch_id = rcvbuf->branch_id;
    strcpy(phone, rcvbuf->phone);
    strcpy(address, rcvbuf->address);
    strcpy(ssn, "1234567");

    /* Declare && Open Cursor for Fetch */
    EXEC SQL INSERT INTO ACCOUNT (
        ACCOUNT_ID,
        BRANCH_ID,
        SSN,
        PHONE,
        ADDRESS )
    VALUES (:account_id, :branch_id, :ssn, :phone, :address);

    if (sqlca.sqlcode != 0 && sqlca.sqlcode != 1403 )
    {
        printf("insert failed sqlcode = %d\n", sqlca.sqlcode);
    }
}

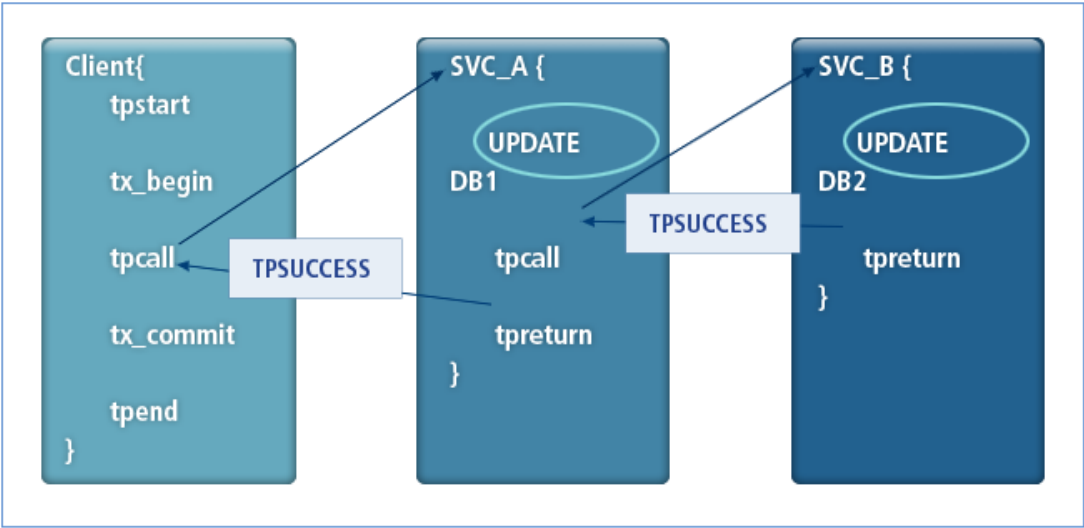
```

```
        tpreturn(TPFAIL, -1, (char *)NULL, 0, TPNOFLAGS);
    }
    strcpy(send, OKMSG);
    tpreturn(TPSUCCESS, 1, (char *)send, strlen(send), TPNOFLAGS);
}
```

12.4.2. 同期型モード(異機種)

以下は、同期型モードで異機種のデータベースに接続する場合のプログラムのフローを表した図です。

[図 12.3] 同期型モードのフロー図(異機種のデータベース接続)



プログラム構成

- 共通プログラム

プログラム・ファイル	説明
demo.s	SDLFILEです
sample.m	Tmax環境ファイルです
tmax.env	環境ファイルです
mktable.sql	データベース・テーブルを作成するSQLスクリプトです

- クライアント・プログラム

構成	説明
client.c	クライアント・プログラムです

- サーバー・プログラム

構成	説明
update.pc, insert.pc	サーバー・プログラムです
Makefile	Tmaxが提供するメイクファイルを変更する必要があります

#### 参考

クライアントとサーバー・プログラムは、「[12.4.1. 同期型モード\(同機種\)](#)」の場合と同じです。マルチノード環境設定についての詳細内容は、『Tmax 運用ガイド』を参照してください。

## プログラムの特徴

- クライアント・プログラム

区分	説明
Tmax接続	NULLパラメータで接続
バッファ・タイプ	STRUCT
サブタイプ	input構造体ファイルをsdlcでコンパイルし、SDLファイルの作成が必要
トランザクション可否	クライアントで明示的にトランザクション範囲を指定

- サーバー・プログラム

区分	説明
サービス数	UPDATEサービスでINSERTサービスを要求
データベース接続	Oracleデータベースを使用し、システム構成ファイルのSVRGROUPセクションにデータベース情報を指定

## プログラム環境

区分	説明
システム	SunOS 5.7 32bit、SunOS 5.8 32bit
データベース	Oracle 8.0.5

## Tmax環境ファイル

以下は、Tmax環境ファイルの例です。

<sample.m>

```
*DOMAIN
res          SHMKEY=88000, MINCLH=1, MAXCLH=5, TPORTNO=8880, BLOCKTIME=60

*NODE
tmax1        TMAXDIR="/user/ tmax ",
              APPDIR="/user/ tmax /appbin",
              PATHDIR = "/user/ tmax /path",
              TLOGDIR = "/user/ tmax /log/tlog",
              ULOGDIR="/user/ tmax /log/ulog",
              SLOGDIR="/user/ tmax /log/slog"

tmax2        TMAXDIR="/user/ tmax ",
              APPDIR="/user/ tmax /appbin",
              PATHDIR = "/user/ tmax /path",
              TLOGDIR = "/user/ tmax /log/tlog",
              ULOGDIR="/user/ tmax /log/ulog",
              SLOGDIR="/user/ tmax /log/slog"

*SVRGROUP
svg1         NODENAME = tmax1, DBNAME = ORACLE,
              OPENINFO = "ORACLE_XA+Acc=P/bmt/bmt+SesTm=60",
              TMSNAME = svg1_tms

svg2         NODENAME = tmax2, DBNAME = ORACLE,
              OPENINFO = "ORACLE_XA+Acc=P/bmt/bmt+SesTm=60",
              TMSNAME = svg2_tms

*SERVER
update       SVGNAME=svg1
insert       SVGNAME=svg2

*SERVICE
UPDATE       SVRNAME=update
INSERT       SVRNAME=insert
```

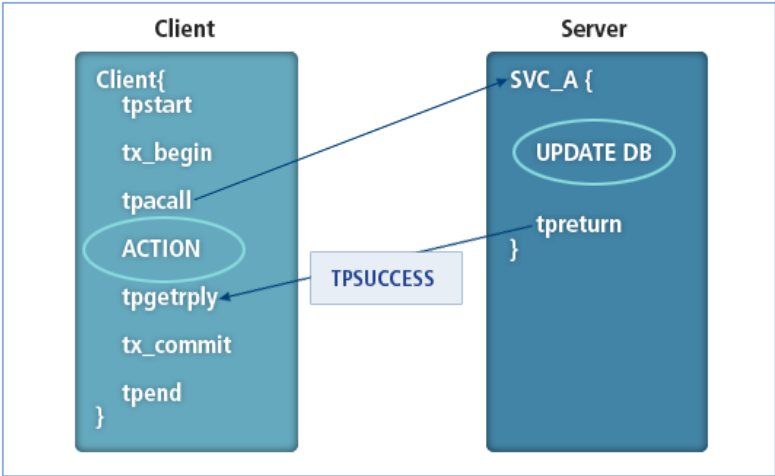
以下は、環境ファイルの例です。

```
[tmax1]
TMAX_HOST_ADDR=192.168.1.39
TMAX_HOST_PORT=8880
SDLFILE=/user/tmax/sample/sdl/tmax.sdl
TMAX_CONNECT_TIMEOUT=5
```

### 12.4.3. 非同期型モード(同機種)

以下は、非同期型モードで同機種のデータベースに接続する場合のプログラムのフローを表した図です。

[図 12.4] 非同期型モードのフロー図(同機種のデータベース接続)



### プログラム構成

● 共通プログラム

プログラム・ファイル	説明
demo.s	SDLFILEです
sample.m	Tmax環境ファイルです
tmax.env	環境ファイルです
mktable.sql	データベース・テーブルを作成するSQLスクリプトです

● クライアント・プログラム

プログラム・ファイル	説明
client.c	クライアント・プログラムです

● サーバー・プログラム

プログラム・ファイル	説明
update.pc	サーバー・プログラムです
Makefile	Tmaxが提供するメイクファイルを変更する必要があります

## プログラムの特徴

- クライアント・プログラム

機能	説明
Tmax接続	NULLパラメータで接続
バッファ・タイプ	STRUCT
サブタイプ	input構造体ファイルをsdhcでコンパイルし、SDLファイルの作成が必要
トランザクション可否	クライアントでトランザクションを指定

- サーバー・プログラム

機能	説明
サービス数	UPDATEサービスを要求
データベース接続	Oracleデータベースを使用し、システム構成ファイルのSVRGROUPセクションにデータベース情報を指定

## プログラム環境

区分	説明
システム	SunOS 5.7 32bit
データベース	Oracle 8.0.5

## 構造体バッファ

以下は、構造体バッファの例です。

<demo.s>

```
struct input {
    int    account_id;
    int    branch_id;
    char   phone[15];
    char   address[61];
};
```



## Tmax環境ファイル

以下は、Tmax環境ファイルの例です。

<sample.m>

```
*DOMAIN
res          SHMKEY=88000, MINCLH=1, MAXCLH=5, TPORTNO=8880, BLOCKTIME=60

*NODE
tmax         TMAXDIR="/user/ tmax ",
             APPDIR="/user/ tmax /appbin",
             PATHDIR = "/user/ tmax /path",
             TLOGDIR = "/user/ tmax /log/tlog",
             ULOGDIR="/user/ tmax /log/ulog",
             SLOGDIR="/user/ tmax /log/slog"

*SVRGROUP
svg1         NODENAME = tmax, DBNAME = ORACLE,
             OPENINFO = "ORACLE_XA+Acc=P/bmt/bmt+SesTm=60",
             TMSNAME = svg1_tms

*SERVER
update       SVGNAME=svg1

*SERVICE
UPDATE       SVRNAME=update
```

以下は、環境ファイルの例です。

<tmax.env>

```
[tmax]
TMAX_HOST_ADDR=192.168.1.39
TMAX_HOST_PORT=8880
SDLFILE=/user/tmax/sample/sdl/tmax.sdl
TMAX_CONNECT_TIMEOUT=5
```

## データベース・スクリプト

以下は、データベースにテーブルを作成するSQLスクリプトの例です。

<mktable.sql>

```
$ORACLE_HOME/bin/sqlplus bmt/bmt <<!
drop table ACCOUNT;
create table ACCOUNT (
    ACCOUNT_ID integer,
```

```

        BRANCH_ID    integer not null,
        SSN          char(13) not null,
        BALANCE       number,
        ACCT_TYPE     char(1),
        LAST_NAME     char(21),
        FIRST_NAME    char(21),
        MID_INIT      char(1),
        PHONE         char(15),
        ADDRESS       char(61),
        CONSTRAINT ACCOUNT_PK PRIMARY KEY (ACCOUNT_ID)
    );
quit
!
```

## クライアント・プログラム

以下は、クライアント・プログラムの例です。

<client.c>

```

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

#define TEMP_PHONE "6283-2114"
#define TEMP_ADDRESS "Korea"

int main(int argc, char *argv[])
{
    struct input *sndbuf;
    char *rcvbuf;
    int acct_id, n, cd, timeout;
    long len;

    if (argc != 2) {
        fprintf(stderr, "Usage:%s account_id \n", argv[0]);
        exit(1);
    }

    acct_id = atoi(argv[1]);
    timeout = 5;

    n = tmaxreadenv("tmax.env", "tmax");
    if (n < 0) {
        fprintf(stderr, "tmaxreadenv fail! tperrno = %d\n", tperrno);
        exit(1);
    }

    n = tpstart((TPSTART_T *)NULL);
    if (n < 0) {
```

```

        fprintf(stderr, "tpstart fail! tperrno = %s\n", tperrno);
        exit(1);
    }

    sndbuf = (struct input *)tpalloc("STRUCT", "input", sizeof(struct input));
    if (sndbuf == NULL) {
        fprintf(stderr, "tpalloc fail: sndbuf tperrno = %d\n", tperrno);
        tpend();
        exit(1);
    }

    rcvbuf = (char *)tpalloc("STRING", NULL, 0);
    if (rcvbuf == NULL) {
        fprintf(stderr, "tpalloc fail: rcvbuf tperrno = %d\n", tperrno);
        tpend();
        exit(1);
    }

    sndbuf->account_id = acnt_id;
    sndbuf->branch_id = acnt_id;
    strcpy(sndbuf->phone, TEMP_PHONE);
    strcpy(sndbuf->address, TEMP_ADDRESS);
    tx_set_transaction_timeout(timeout);
    n = tx_begin();
    if (n < 0)
        fprintf(stderr, "tx begin fail! tperrno = %d\n", tperrno);

    cd = tpacall("UPDATE", (char *)sndbuf, sizeof(struct input), TPNOFLAGS);
    if (cd < 0) {
        fprintf(stderr, "tpacall fail! tperrno = %d\n", tperrno);
        tpend();
        exit(1);
    }

    n = tpgetrply(&cd, (char **)&rcvbuf, (long *)&len, TPNOFLAGS);
    if (n < 0) {
        fprintf(stderr, "tpgetrply fail! tperrno = %d\n", tperrno);
        tpend();
        exit(1);
    }

    n = tx_commit();
    if (n < 0) {
        fprintf(stderr, "tx commit fail! tx error = %d \n", n);
        tx_rollback();
        tpend();
        exit(1);
    }

    printf("rtn msg = %s\n", rcvbuf);
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
}

```

## サーバー・プログラム

以下は、サーバー・プログラムの例です。

<update.pc>

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

#define OKMSG "YOU COMPLETE THE TRANSACTION"

EXEC SQL include sqlca.h;

EXEC SQL BEGIN DECLARE SECTION;
    int      account_id;
    int      branch_id;
    char      ssn[15];
    char      phone[15];
    char      address[61];
EXEC SQL END DECLARE SECTION;

UPDATE(TPSVCINFO *msg)
{
    struct input *rcvbuf;
    int      ret, cd;
    long      acnt_id, rcvlen;
    char      *send;

    rcvbuf = (struct input *) (msg->data);
    send = (char *) tmalloc("STRING", NULL, 0);
    if (send == NULL) {
        fprintf(stderr, "tpalloc fail errno = %s\n", strerror(tperrno));
        tpreturn(TPFAIL, 0, (char *) NULL, 0, TPNOFLAGS);
    }
    account_id = rcvbuf->account_id;
    branch_id = rcvbuf->branch_id;
    strcpy(phone, rcvbuf->phone);
    strcpy(address, rcvbuf->address);
    strcpy(ssn, "1234567");

    EXEC SQL UPDATE ACCOUNT
    SET BRANCH_ID = :branch_id,
    PHONE = :phone,
    ADDRESS = :address,
    SSN = :ssn
    WHERE ACCOUNT_ID = :account_id;

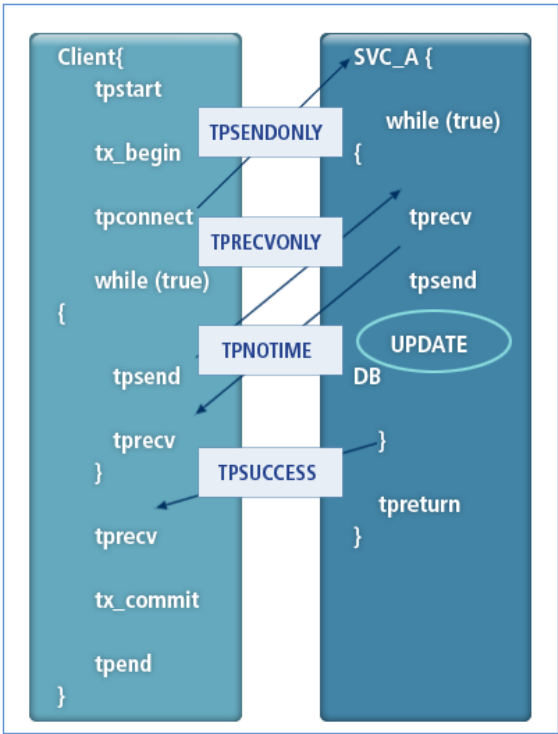
    if (sqlca.sqlcode != 0 && sqlca.sqlcode != 1403 ) {
        fprintf(stderr, "update failed sqlcode = %d\n", sqlca.sqlcode);
        tpreturn(TPFAIL, -1, (char *) NULL, 0, TPNOFLAGS);
    }
}
```

```
strcpy(send, OKMSG);
tpreturn(TPSUCCESS, 1, (char *)send, strlen(send), TPNOFLAGS);
}
```

12.4.4. 会話型モード(同機種)

以下は、会話型モードで同機種のデータベースに接続する場合のプログラムのフローを表した図です。

[図 12.5] 会話型モードのフロー図(同機種のデータベース接続)



プログラム構成

- 共通プログラム

プログラム・ファイル	説明
demo.s	SDLFILEです
sample.m	Tmax環境ファイルです
tmax.env	環境ファイルです
mktable.sql	データベース・テーブルを作成するSQLスクリプトです

- クライアント・プログラム

プログラム・ファイル	説明
client.c	クライアント・プログラムです

- サーバー・プログラム

プログラム・ファイル	説明
update.pc	サーバー・プログラムです
Makefile	Tmaxが提供するメイクファイルを変更する必要があります

## プログラムの特徴

- クライアント・プログラム

機能	説明
Tmax接続	NULLパラメータ接続
バッファ・タイプ	STRUCT
サブタイプ	input構造体ファイルをsdlcでコンパイルし、SDLファイルの作成が必要(アプリケーションを実行させるために必要)
トランザクション可否	クライアントでトランザクション指定

- サーバー・プログラム

機能	説明
サービス数	UPDATEサービスを要求
データベース接続	Oracleデータベースを指定し、Tmax環境ファイルのSVRGROUPセクションにデータベース情報を指定

## プログラム環境

区分	説明
システム	SunOS 5.7 32bit
データベース	Oracle 8.0.5

## 構造体バッファ

以下は、構造体バッファの例です。

<demo.s>

```
struct input {
    int      account_id;
    int      branch_id;
    char      phone[15];
    char      address[61];
};
```

## Tmax環境ファイル

以下は、Tmax環境ファイルの例です。

```
*DOMAIN
res      SHMKEY=88000, MINCLH=1, MAXCLH=5, TPORTNO=8880, BLOCKTIME=60

*NODE
tmax      TMAXDIR="/user/ tmax ",
          APPDIR="/user/ tmax /appbin",
          PATHDIR = "/user/ tmax /path",
          TLOGDIR = "/user/ tmax /log/tlog",
          ULOGDIR="/user/ tmax /log/ulog",
          SLOGDIR="/user/ tmax /log/slog"

*SVRGROUP
svg1      NODENAME = tmax, DBNAME = ORACLE,
          OPENINFO = "ORACLE_XA+Acc=P/bmt/bmt+SesTm=60",
          TMSNAME = svg1_tms

*SERVER
update    SVGNAME=svg1, CONV=YES

*SERVICE
UPDATE    SVRNAME= update
```

以下は、環境ファイルの例です。

<tmax.env>

```
[ tmax ]
TMAX_HOST_ADDR=192.168.1.39
TMAX_HOST_PORT=8880
SDLFILE=/user/tmax/sample/sdl/tmax.sdl
TMAX_CONNECT_TIMEOUT=5
```

## データベース・スクリプト

以下は、データベースにテーブルを作成するSQLスクリプトの例です。

<mktable.sql>

```
$ORACLE_HOME/bin/sqlplus bmt/bmt <<!  
drop table ACCOUNT;  
create table ACCOUNT (  
    ACCOUNT_ID    integer,  
    BRANCH_ID     integer not null,  
    SSN           char(13) not null,  
    BALANCE       number,  
    ACCT_TYPE     char(1),  
    LAST_NAME     char(21),  
    FIRST_NAME    char(21),  
    MID_INIT      char(1),  
    PHONE         char(15),  
    ADDRESS       char(61),  
    CONSTRAINT ACCOUNT_PK PRIMARY KEY(ACCOUNT_ID)  
);  
quit  
!
```

## クライアント・プログラム

以下は、クライアント・プログラムの例です。

<client.c>

```
#include <stdio.h>  
#include <usrinc/atmi.h>  
#include "../sdl/demo.s"  
  
#define TEMP_PHONE "6283-2115"  
#define TEMP_ADDRESS "Korea"  
  
void main(int argc, char *argv[])  
{  
    struct input *sndbuf;  
    char *rcvbuf;  
    int acntid, timeout;  
    long revent, rcvlen;  
    int cd, n;  
  
    if (argc != 2) {  
        fprintf(stderr, "Usage:%s acntid\n", argv[0]);  
    }
```



```

        exit(1);
    }

    acctid = atoi(argv[1]);
    timeout = 5;
    n = tmaxreadenv("tmax.env", "tmax");
    if (n < 0) {
        fprintf(stderr, "tmaxreadenv fail tperrno = %d\n", tperrno);
        exit(1);
    }

    n = tpstart((TPSTART_T *)NULL);
    if (n < 0) {
        fprintf(stderr, "tpstart fail tperrno = %s\n", tperrno);
        exit(1);
    }
    printf("tpstart ok!\n");
    sndbuf = (struct input *)tpalloc("STRUCT", "input", sizeof(struct input));

    if (sndbuf == NULL) {
        fprintf(stderr, "tpalloc fail: sndbuf tperrno = %d\n", tperrno);
        tpend();
        exit(1);
    }

    rcvbuf = (char *)tpalloc("CARRAY", NULL, 0);
    if (rcvbuf == NULL) {
        fprintf(stderr, "tpalloc fail: rcvbuf tperrno = %d\n", tperrno);
        tpend();
        exit(1);
    }

    sndbuf->account_id = acctid;
    sndbuf->branch_id = acctid;
    strcpy(sndbuf->phone, TEMP_PHONE);
    strcpy(sndbuf->address, TEMP_ADDRESS);

    tx_set_transaction_timeout(timeout);
    n = tx_begin();
    if (n < 0)
        fprintf(stderr, "tx begin fail tx error = %d\n", n);
    printf("tx begin ok!\n");

    cd = tpconnect("UPDATE", (char *)sndbuf, 0, TPSENDONLY);
    if (cd < 0) {
        fprintf(stderr, "tpconnect fail tperrno = %d\n", tperrno);
        tpend();
    }

```

```

        exit(1);
    }

    while (1) {
        n = tpsend(cd, (char *)sndbuf, sizeof(struct input), TPRECVONLY,
                  &revent);

        if (n < 0) {
            fprintf(stderr, "tpsend fail revent = 0x%08x\n", revent);
            tx_rollback();
            tpend();
            exit(1);
        }
        printf("tpsend ok\n");

        n = tprecv(cd, (char **)&rcvbuf, (long *)&rcvlen, TPNOTIME, &revent);

        if (n < 0 && revent != TPEV_SENDOONLY) {
            fprintf(stderr, "tprecv fail revent = 0x%08x\n", revent);
            tx_rollback();
            tpend();
            exit(1);
        }
        printf("tprecv ok\n");
        sndbuf->account_id++;

        if (revent != TPEV_SENDOONLY)
            break;
    }
    n = tprecv(cd, (char **)&rcvbuf, (long *)&rcvlen, TPNOTIME, &revent);
    if (n < 0 && revent != TPEV_SVCSUCC) {
        fprintf(stderr, "tprecv fail revent = 0x%08x\n", revent);
        tx_rollback();
        tpend();
        exit(1);
    }
    printf("rcvbuf = [%s]\n", rcvbuf);

    n = tx_commit();
    if (n < 0) {
        fprintf(stderr, "tx commit fail tx error = %d\n", n);
        tx_rollback();
        tpend();
        exit(1);
    }
    printf("tx commit ok!\n");
    printf("rtn msg = %s\n", rcvbuf);
    tpfree((char *)sndbuf);

```

```

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

```

## サーバー・プログラム

以下は、サーバー・プログラムの例です。

<update.pc>

```

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

void _db_work();

#define OKMSG "YOU COMPLETE THE TRANSACTION"

EXEC SQL include sqlca.h;
EXEC SQL BEGIN DECLARE SECTION;
    int      account_id;
    int      branch_id;
    char      ssn[15];
    char      phone[15];
    char      address[61];
EXEC SQL END DECLARE SECTION;

struct input *rcvbuf;

UPDATE(TPSVCINFO *msg)
{
    int      ret, count;
    long      acnt_id;
    long      revent, rcvlen, flag;
    char      *send;

    rcvbuf = (struct input *)tpalloc("STRUCT", "input", 0);
    send = (char *)tpalloc("CARRAY", NULL, 0);

    count = 1;
    flag = 0;

    while (1) {
        ret = tprecv(msg->cd, (char **)&rcvbuf, &rcvlen, TPNOTIME, &revent);
        if (ret < 0 && revent != TPEV_SENDOONLY) {

```

```

        fprintf(stderr, "tprecv fail revent = 0x%08x\n", revent);
        tpreturn(TPFAIL, -1, (char *)rcvbuf, 0, TPNOFLAGS);
    }
    printf("tprecv ok!\n");

    if (count == 10) {
        flag &= ~TPRECVONLY;
        flag |= TPNOTIME;
    }
    else
        flag |= TPRECVONLY;

    ret = tpsend(msg->cd, (char *)send, strlen(send), flag, &revent);
    if (ret < 0) {
        fprintf(stderr, "tpsend fail revent = 0x%08x\n", revent);
        tpreturn(TPFAIL, -1, (char *)NULL, 0, TPNOFLAGS);
    }
    printf("tpsend ok!\n");
    _db_work();

    /* break after 10 iterations */
    if (count == 10)
        break;

    count++;
}

strcpy(send, OKMSG);
printf("tpreturn ok!\n");
tpreturn(TPSUCCESS, 1, (char *)send, strlen(send), TPNOFLAGS);
}

void _db_work() {
    account_id = rcvbuf->account_id;
    branch_id = rcvbuf->branch_id;
    strcpy(phone, rcvbuf->phone);
    strcpy(address, rcvbuf->address);
    strcpy(ssn, "1234567");

    EXEC SQL UPDATE ACCOUNT
    SET BRANCH_ID = :branch_id,
        PHONE = :phone,
        ADDRESS = :address,
        SSN = :ssn
    WHERE ACCOUNT_ID = :account_id;

    if (sqlca.sqlcode != 0 && sqlca.sqlcode != 1403 )

```

```

    {
        fprintf(stderr, "update failed sqlcode = %d\n", sqlca.sqlcode);
        tpreturn(TPFAIL, -1, (char *)NULL, 0, 0);
    }
}

```

## 12.5. TIPを利用したプログラム

Tmaxでは、TIP(Tmax Information Provider)を使用して、システム環境情報、統計情報の確認、システム運用管理など、多様な機能を実行できます。TIPは、TIPSVLCを処理するためにTmaxが提供する機能プロセスで、TIPを使用して以下の機能を実行できます。

- システム環境情報の確認: システムの静的な環境情報を確認します
- システム統計情報の確認: システムの運用中に各プロセスの状態などを確認します
- システム運用管理: 追加でプロセスを起動または終了します

### 12.5.1. TIPの構造

TIPSVLCを処理するための別途の機能プロセスであるTIPサーバーは、SYS\_SVRサーバータイプで、TIPサーバーグループに属します。クライアントあるいはサーバーの要求を受けたTIPサーバーは、その要求をCLH/TMMに転送し、処理結果を再度要求者に戻します。TIPサーバーは、フィールド・キーをパターンにしてサービスを処理します。要求するクライアントやサーバーは、要求しようとするデータをフィールド・バッファーに記載して要求し、その結果をフィールド・バッファーで受け取ります。

- CHLOG section (ログレベルの変更)

CHLOGは、TMM、CLH、TMS、SVRのログレベルを変更できるセクションです。tmadminで**chlog**を実行するのと同じ動作を実行します。

TIPサービスを呼び出す場合、フィールド・バッファーに以下を設定し、TIPSVLCを呼び出します。

項目	説明
TIP_OPERATION (string)	GETと設定します
TIP_SEGMENT (string)	ADMINISTRATIONと設定します
TIP_SECTION (string)	CHLOGと設定します
TIP_MODULE (int)	動的にログを変更したいモジュールを設定する項目で、以下のいずれかを選択します <ul style="list-style-type: none"> <li>– TIP_TMM</li> <li>– TIP_CLH</li> </ul>

項目	説明
	<ul style="list-style-type: none"> <li>– TIP_TMS</li> <li>– TIP_SVR</li> </ul>
TIP_FLAGS (int)	flagsを設定する項目で、以下のいずれかを選択します <ul style="list-style-type: none"> <li>– TIP_VFLAG</li> <li>– TIP_GFLAG</li> <li>– TIP_NFLAG</li> </ul>
TIP_SVRNAME (string)	該当サーバー名を設定する項目で、TIP_FLAGSをTIP_VFLAGと設定した場合のみ設定します
TIP_SVGNAME (string)	該当サーバー・グループ名を設定する項目で、TIP_FLAGSをTIP_GFLAGと設定した場合のみ設定します
TIP_NODENAME (string)	該当ノード名でTIP_FLAGSをTIP_NFLAGと設定した場合のみ設定します
TIP_LOGLVL (string)	変更するログレベルを設定する項目です。以下のいずれかを選択し、必ず小文字で設定します。結果として取得する値は、TIP_ERRORに設定されます <ul style="list-style-type: none"> <li>– compact</li> <li>– basic</li> <li>– detail</li> <li>– debug1</li> <li>– debug2</li> <li>– debug3</li> <li>– debug4</li> </ul>
TIP_ERROR (int)	エラー値が設定される項目で、設定されているエラー値は以下のとおりです <ul style="list-style-type: none"> <li>– TIPSVCFAIL : 該当するサービスを正常に処理できなかった場合</li> <li>– TIPEOS : メモリーの割り当てに失敗した場合</li> <li>– TIPEBADFLD : TIP_MODULE値を設定していない場合</li> </ul>

- CHTRC section

TMSとSPRのTMAX\_TRACEを修正し、trace logオプションを変更できるセクションです。tmadminのchtrcと同一動作を実行します。

TIPサービスを呼び出す場合、フィールド・バッファに以下の内容を設定し、TIPSVCを呼び出します。

項目	説明
TIP_OPERATION (string)	GETと設定します
TIP_SEGMENT (string)	ADMINISTRATIONと設定します
TIP_SECTION (string)	CHTRCに設定します
TIP_FLAGS (int)	フラグをを設定する項目で、以下のいずれかを設定します – TIP_PFLAG – TIP_VFLAG – TIP_GFLAG – TIP_NFLAG
TIP_SPRI (int)	spriを設定する項目で、TIP_FLAGSをTIP_PFLAGと設定した場合のみ設定します
TIP_SVGNAME (string)	該当サーバグループ名を設定する項目で、TIP_FLAGSをTIP_GFLAGと設定した場合のみ設定します
TIP_NODENAME (string)	該当ノード名を設定する項目で、TIP_FLAGSをTIP_NFLAGと設定した場合のみ設定します
TIP_SPEC (string)	変更しようとするfilter spec、receiver spec、trigger specを設定する項目です。結果として受け取る値はTIP_ERRORに設定されます
TIP_ERROR (int)	エラー値が設定されます。設定されるエラー値は以下のとおりです – TIPSVCFAIL：該当サービスを正常に処理できなかった場合 – TIPEOS：メモリーの割り当てに失敗した場合 – TIPEBADFLD：TIP_MODULE値を設定していない場合

以下は、TIPSVCに要求するために含める必要があるデータです。

● 組織(TIP\_OPERATION)

設定値	説明
GET	システムの静的な環境情報、統計情報の確認、BOOT/DOWNシステムを運用管理する場合はGETと設定します
SET	システム設定状態を変更する場合、SETと設定します。現在はGETのみ使用されます

- セグメント(TIP\_SEGMENT)

どんな機能を実行するかを決定するために使用するフィールドです。以下は、TIP\_SEGMENTフィールドに設定できる値です。

設定値	説明
CONFIGURATION	システムの静的な設定情報を確認します
STATISTICS	システムの運用中に統計情報を確認します
ADMINISTRATION	システムの運用管理(起動と終了)を確認します

- セクション(TIP\_SECTION)

TIP\_SECTIONを設定した場合、処理する項目の詳細を決定します。

設定値	説明
CONFIGURATION	DOMAIN, NODE, SVRGROUP, SERVER, SERVICE, ROUTING, RQ, GATEWAY
STATISTICS	NODE, TPROC, SPR, SERVICE, RQ, TMGW, NTMGW, TMS, TMMS, CLHS, SERVER(SVR)
ADMINISTRATION	BOOT, DOWN, CHLOG, CHTRC

- コマンド(TIP\_CMD)

フィールドはTIP\_SECTIONがADMINISTRATIONの場合のみ使用されます。

設定値	説明
TIP_BOOT	Tmaxシステムを起動します
TIP_DOWN	Tmaxシステムを終了します

## 12.5.2. TIPの使用

TIPの使用方法与エラーの確認方法について説明します。

### TIPの使用方法

- 環境設定

TIPサーバーはTmaxが提供する機能プロセスであるため、ユーザーがサービスを作成する必要がありません。但し、環境ファイルにTIPサーバーを登録する必要があります。以下は、環境設定の例です。



```

*DOMAIN
res                ..., TIPSVC = TIPSVC

*NODE
tmaxsl            ...

*SVRGROUP
tsvg              ..., SVGTYPE = TIP

*SERVER
TIP               SVGNAME = tsvg, SVRTYPE = SYS_SVR

```

- DOMAINセクションにはTIPSVCを登録します。登録しない場合、デフォルト値としてTIPSVCが登録されます。
- SVRGROUPセクションにはTIPサーバーグループ(SVGTYPE=TIP)を登録します。
- SERVERセクションにはTIPサーバー(SVRTYPE=SYS\_SVR)を登録します。

#### ● システム接続

Tmaxシステムの接続時、TPSTART\_T構造体のusername項目にtpadminを設定する必要があります。TIP\_SEGMENTがADMINISTRATIONの場合のみusernameを設定し、他の場合は設定しなくても構いません。設定されたusernameが正しくない場合、TIP\_ERRORフィールドにTIPEAUTHエラーが設定されます。

```
strcpy(tpinfo->username, ".tpadmin");
```

#### ● バッファの割り当て

TIPサーバーはフィールド・キーをパターンにしてサービス进行处理するため、要求するクライアントやサーバー・プログラムはフィールド・キー・バッファを割り当てる必要があります。

#### ● TIP要求項目の設定

項目	説明
TIP_OPERATION	システム環境情報の変更及び確認をする場合に設定します
TIP_SEGMENT	システム情報の確認及び運用管理のために設定します
TIP_SECTION	SEGMENTで設定項目に対する詳細処理を設定します
TIP_CMD	TIP_SEGMENTがADMINISTRATIONの場合のみ設定します
TIP_NODENAME	マルチノードの場合のみ設定します。単一ノードの場合、指定しなければデフォルト値としてローカルノードが設定されます。設定が正しくない場合、TPEINVALエラーが発生します

- TIPSVCの要求

TIPの要求項目を設定した場合、設定したフィールド・バッファをsndbufとし、tpcallあるいはtpacallを利用してTIPSVCに要求を行います。クライアントとサーバーですべて要求でき、トランザクションはサポートしません。

- 結果の受信

受信フィールド・キー・バッファにサービス結果が保存されます。

## エラー 確認

- 正常処理された場合

受信フィールド・キー・バッファのTIP\_ERROR項目に0が設定されます。

- エラーが発生した場合

エラー	説明
TIP_STATUS	TIP_ERRORに設定されたエラー内容の詳細を確認できます
TIP_BADFIELD	エラーを引き起こしたフィールドを確認できます
TIP_ERROR	受信フィールド・キー・バッファのTIP_ERROR項目に0より大きい値が設定されます。これは、/usrinc/tip.hで確認できます

以下は、TIP\_ERRORに設定されているエラー値についての説明です。

設定値	説明
TIPNOERROR	エラーが発生しません
TIPEBADFLD	有効でないフィールド・キーが使用されました。一般的にfdlcユーティリティを使用して、コンパイルされていないフィールド・キーが使用された場合、TIP_ERRORはTIPEBADFLDと設定されます
TIPEIMPL	実装されていない機能の要求を行いました
TIPEAUTH	現在の権限では許容されていないサービスです
TIPEOS	運用システムまたはシステムエラーでメモリーの割り当てに失敗した場合、Tmaxシステムに接続が失敗した場合、またはネットワーク状態が不安定な場合などに現れるエラーです
TIPENOENT	存在しない項目にアクセスしました
TIPESVCFAIL	TIPサービスルーチンのエラーが発生し、TPFAILでtpreturn()を呼び出しました

## 12.5.3. TIPの使用例

本節では、TIPを使用した例を説明します。

### 環境ファイル

以下は、単一ノードを使用した例です。

<cfg.m>

```
*DOMAIN
res          SHMKEY=78850,      MAXUSER=200, MINCLH=1, MAXCLH=5,
              TPORTNO=8850,     BLOCKTIME=60, TXTIME=50, RACPORT=3355

*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"

*SVRGROUP
tsvg         NODENAME = "tmaxh4", SVGTYPE=TIP
svg1         NODENAME = "tmaxh4"

*SERVER
TIP          SVGNAME=tsvg, SVRTYPE=SYS_SVR, MIN=1, MAX=1
svr          SVGNAME=svg1, MIN=1

*SERVICE
TOUPPER      SVRNAME=svr
```

以下は、マルチノードを使用した例です。

<cfg.m>

```
*DOMAIN
tmax         SHMKEY=98850,
              TPORTNO=8850,
              BLOCKTIME=60,
              RACPORT=3355,
              MAXUSER=10

*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"

tmaxh2      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"

*SVRGROUP
tsvg        NODENAME = "tmaxh4", SVGTYPE=TIP

svg1        NODENAME = "tmaxh4", COUSIN = "svg2"
svg2        NODENAME = "tmaxh2"

*SERVER
TIP         SVGNAME=tsvg, SVRTYPE=SYS_SVR, MIN=1, MAX=1
svr         SVGNAME=svg1, MIN=1, MAX=5

*SERVICE
TOUPPER     SVRNAME=svr, ROUTING = "rout1"

*ROUTING
rout1       FIELD="STRING", RANGES = "'bbbbbbb'-'ccccccc' : svg1, * :
            svg2

```

## フィールド・キー・テーブル

以下は、フィールド・キー・テーブルの例です。

< tip.f >

```

#
#      common field
#
# name          number  type    flags  comments
*base 16000001
TIP_OPERATION   0       string
TIP_SEGMENT     1       string
TIP_SECTION     2       string

```

```

TIP_NODE      3      string
TIP_OCCURS    4      int
TIP_FLAGS     5      int
TIP_CURSOR    6      string
TIP_SESTM     7      int
TIP_ERROR     8      int
TIP_STATE     9      int
TIP_MORE     10     int
TIP_BADFIELD  11     string
TIP_CMD       12     string
TIP_CLID      13     int
TIP_MSG       14     string

#
#          DOMAIN section fields
#
# name                number  type    flags  comments
*base 16000100
TIP_NAME            0      string
TIP_SHMKEY          1      int
TIP_MINCLH          2      int
TIP_MAXCLH          3      int
TIP_MAXUSER         4      int
TIP_TPORTNO         5      int
TIP_RACPORT         6      int
TIP_MAXSACALL       7      int
TIP_MAXCACALL       8      int
TIP_MAXCONV_NODE    9      int
TIP_MAXCONV_SERVER  10     int
TIP_CMTRET          11     int
TIP_BLOCKTIME       12     int
TIP_TXTIME          13     int
TIP_IDLETIME        14     int
TIP_CLICHKINT       15     int
TIP_NLIVEINQ        16     int
TIP_SECURITY        17     string
TIP_OWNER           18     string
TIP_CPC             19     int
#TIP_LOGINSVC       20     string
#TIP_LOGOUTSVC      21     string
TIP_NCLHCHKTIME     22     int
TIP_DOMAINID        23     int
TIP_IPCPERM         24     int
TIP_MAXNODE         25     int
TIP_MAXSVG          26     int
TIP_MAXSVR          27     int
TIP_MAXSVC          28     int

```

TIP_MAXSPR	29	int	
TIP_MAXTMS	30	int	
TIP_MAXCPC	31	int	
TIP_MAXROUT	32	int	
TIP_MAXROUTSVG	33	int	
TIP_MAXRQ	34	int	
TIP_MAXGW	35	int	
TIP_MAXCOUSIN	36	int	
TIP_MAXCOUSINSVG	37	int	
TIP_MAXBACKUP	38	int	
TIP_MAXBACKUPSVG	39	int	
TIP_MAXTOTALSVG	40	int	
TIP_MAXPROD	41	int	
TIP_MAXFUNC	42	int	
TIP_TXPENDINGTIME	43	int	
TIP_NO	44	int	
TIP_TIPSV	45	string	
TIP_NODECOUNT	46	int	
TIP_SVGCOUNT	47	int	
TIP_SVRCOUNT	48	int	
TIP_SVCCOUNT	49	int	
TIP_COUSIN_COUNT		50	int
TIP_BACKUP_COUNT		51	int
TIP_ROUT_COUNT	52	int	
TIP_STYPE	53	string	
TIP_VERSION	54	string	
TIP_EXPDATE	55	string	
TIP_DOMAINCOUNT	56	int	
TIP_RSVG_GCOUNT	57	int	
TIP_RSVG_COUNT	58	int	
TIP_CSVG_GCOUNT	59	int	
TIP_CSVG_COUNT	60	int	
TIP_BSVG_GCOUNT	61	int	
TIP_BSVG_COUNT	62	int	
TIP_PROD_COUNT	63	int	
TIP_FUNC_COUNT	64	int	
TIP_SHMSIZE	65	int	
TIP_CRYPT	66	string	
TIP_DOMAIN_TMMLOGLVL	67	string	
TIP_DOMAIN_CLHLOGLVL	68	string	
TIP_DOMAIN_TMSLOGLVL	69	string	
TIP_DOMAIN_LOGLVL	70	string	
TIP_DOMAIN_MAXTHREAD	71	int	
#			
#	NODE section fields		
#			

# name	number	type	flags	comments
*base 16000200				
#TIP_NAME	0	string		
TIP_DOMAINNAME	1	string		
#TIP_SHMKEY	2	int		
#TIP_MINCLH	3	int		
#TIP_MAXCLH	4	int		
TIP_CLHQTIMEOUT	5	int		
#TIP_IDLETIME	6	int		
#TIP_CLICKINT	7	int		
#TIP_TPORTNO	8	int		
#TIP_TPORTNO2	9	int		
#TIP_TPORTNO3	10	int		
#TIP_TPORTNO4	11	int		
#TIP_TPORTNO5	12	int		
#TIP_RACPORT	13	int		
#TIP_TMAXPORT	14	string		
TIP_CMPPRPORT	15	string		
TIP_CMPPRSIZE	16	int		
#TIP_MAXUSER	17	int		
TIP_TMAXDIR	18	string		
TIP_TMAXHOME	19	string		
TIP_APPDIR	20	string		
TIP_PATHDIR	21	string		
TIP_TLOGDIR	22	string		
TIP_SLOGDIR	23	string		
TIP_ULOGDIR	24	string		
TIP_ENVFILE	25	string		
#TIP_LOGINSVC	26	string		
#TIP_LOGOUTSVC	27	string		
TIP_IP	28	string		
#TIP_PEER	29	string		
TIP_TMMOPT	30	string		
TIP_CLHOPT	31	string		
#TIP_IPCPERM	32	int		
#TIP_MAXSVG	33	int		
#TIP_MAXSVR	34	int		
#TIP_MAXSPR	35	int		
#TIP_MAXTMS	36	int		
#TIP_MAXCPC	37	int		
TIP_MAXGWSVR	38	int		
TIP_MAXRQSVR	39	int		
TIP_MAXGWCPC	40	int		
TIP_MAXRQCPC	41	int		
TIP_CPORTNO	42	int		
TIP_REALSVR	43	string		
TIP_RSCPC	44	int		

```

TIP_AUTOBACKUP 45      int
TIP_HOSTNAME   46      string
TIP_NODETYPE   47      int
TIP_CPU        48      int
#TIP_MAXRSTART 49      int
#TIP_GPERIOD   50      int
#TIP_RESTART   51      int
TIP_CURCLH     49      int
TIP_LIVETIME   50      string
TIP_NODE_TMMLOGLVL 51      string
TIP_NODE_CLHLOGLVL 52      string
TIP_NODE_TMSLOGLVL 53      string
TIP_NODE_LOGLVL 54      string
TIP_NODE_MAXTHREAD 55      int
TIP_EXTPORT    56      int
TIP_EXTCLHPORT 57      int
TIP_MSGSIZEWARN 58      int
TIP_MSGSIZEMAX 59      int

#
#          SVRGROUP section fields
#
# name          number  type    flags  comments
*base 16000300
#TIP_NAME       0       string
#TIP_NODENAME   1       string
TIP_SVGTYPE     2       string
#TIP_PRODNAME   3       string
TIP_COUSIN      4       string
TIP_BACKUP      5       string
TIP_LOAD        6       int
#TIP_APPDIR     7       string
#TIP_ULOGDIR    8       string
TIP_DBNAME      9       string
TIP_OPENINFO    10      string
TIP_CLOSEINFO   11      string
TIP_MINTMS      12      int
#TIP_MAXTMS     13      int
TIP_TMSNAME     14      string
#TIP_SECURITY   15      string
#TIP_OWNER      16      string
#TIP_ENVFILE    17      string
#TIP_CPC        18      int
TIP_XAOPTION    19      string
TIP_SVG_TMSTYPE 20      string
TIP_SVG_TMSOPT  21      string

```



```

TIP_SVG_TMSTHEADS      22      int
TIP_SVG_TMSLOGLVL      23      string
TIP_SVG_LOGLVL         24      string
TIP_NODENAME           25      string

#
#      SERVER section fields
#
# name          number  type    flags  comments
*base 16000350
#TIP_NAME       0       string
TIP_SVGNAME     1       string
#TIP_NODENAME   2       string
TIP_CLOPT      3       string
TIP_SEQ        4       int
TIP_MIN        5       int
TIP_MAX        6       int
#TIP_ULOGDIR    7       string
TIP_CONV       8       int
TIP_MAXQCOUNT  9       int
TIP_ASQCOUNT  10      int
TIP_MAXRSTART  11      int
TIP_GPERIOD    12      int
TIP_RESTART    13      int
TIP_SVRTYPE    14      string
#TIP_CPC        15      int
TIP_SCHEDULE   16      int
#TIP_MINTHR     17      int
#TIP_MAXTHR     18      int
TIP_TARGET     19      string
TIP_DEPEND     20      string
TIP_CASCADE    21      int
TIP_PROCNAME   22      string
TIP_LIFESPAN   23      string
TIP_DDRI       24      string
TIP_CURSVR     25      int
TIP_SVGNO      26      int
TIP_SVR_LOGLVL 27      string

#
#      SERVICE section fields
#
# name          number  type    flags  comments
*base 16000400
#TIP_NAME       0       string
TIP_SVRNAME     1       string

```

```

TIP_PRI          2      int
TIP_SVCTIME      3      int
TIP_ROUTING      4      string
TIP_EXPORT       5      int
TIP_AUTOTRAN     6      int

#
#          ROUTING section fields
#
# name          number  type    flags  comments
*base 16000425
#TIP_NAME       0      string
TIP_FLDTYPE     1      string
TIP_RANGES      2      string
TIP_SUBTYPE     3      string
TIP_ELEMENT     4      string
TIP_BUFTYPE     5      string
TIP_OFFSET      6      int
TIP_FLDLEN      7      int
#TIP_FLDOFFSET  8      int

#
#          RQ section fields
#
# name          number  type    flags  comments
*base 16000450
#TIP_NAME       0      string
#TIP_SVGNAME    1      string
TIP_PRESVC      2      string
TIP_QSIZE       3      int
TIP_FILEPATH    4      string
TIP_BOOT        5      string
TIP_FSYNC       6      int
TIP_BUFFERING   7      int
#TIP_ENQSVC     8      int
#TIP_FAILINTERVAL 9      int
#TIP_FAILRETRY  10     int
#TIP_FAILSVC    11     string
#TIP_AFTERSVC   12     string

#
#          GATEWAY section fields
#
# name          number  type    flags  comments
*base 16000500

```

```

#TIP_NAME          0      string
TIP_GWTYPE         1      string
TIP_PORTNO        2      int
#TIP_CPC           3      int
TIP_RGWADDR       4      string
TIP_RGWPORTNO     5      int
#TIP_BACKUP        6      string
#TIP_NODENAME      7      string
TIP_KEY           8      string
TIP_BACKUP_RGWADDR 9      string
TIP_BACKUP_RGWPORTNO 10    int
TIP_TIMEOUT       11     int
TIP_DIRECTION     12     string
TIP_MAXINRGW      13     int
TIP_GWOWNER       15     string
TIP_RGWOWNER      16     string
TIP_RGWPASSWD     17     string
TIP_PTIMEOUT      18     int
TIP_PTIMEINT      19     int

#
#      FUNCTION section fields
#
# name          number  type    flags  comments
*base 16000550
#TIP_NAME      0      string
#TIP_SVRNAME   1      string
TIP_FQSTART    2      int
TIP_FQEND      3      int
TIP_ENTRY      4      string

#
#      STATISTICS segment fields
#
# name          number  type    flags  comments
*base 16000600
#TIP_NAME      0      string
TIP_STATUS     1      string
TIP_STIME      2      string
TIP_TTIME      3      int
TIP_SVC_STIME  4      int
TIP_COUNT      5      int
#TIP_NO        6      int
TIP_NUM_FREE   7      int
TIP_NUM_REPLY  8      int
TIP_NUM_FAIL   9      int

```

```

TIP_NUM_REQ      10      int
TIP_ENQ_REQS     11      int
TIP_DEQ_REQS     12      int
TIP_ENQ_REPLYS   13      int
TIP_DEQ_REPLYS   14      int
TIP_CLHNO        15      int
TIP_SVR_NAME     16      string
TIP_SVC_NAME     17      string
TIP_AVERAGE     18      float
TIP_QCOUNT      19      int
TIP_CQCOUNT     20      int
TIP_QAVERAGE    21      float
TIP_MINTIME      22      float
TIP_MAXTIME      23      float
TIP_FAIL_COUNT   24      int
TIP_ERROR_COUNT  25      int
TIP_PID          26      int
TIP_TOTAL_COUNT  27      int
TIP_TOTAL_SVCFAIL_COUNT 28      int
TIP_TOTAL_ERROR_COUNT 29      int
TIP_TOTAL_AVG    30      float
TIP_TOTAL_RUNNING_COUNT 31      int
TIP_TMS_NAME     32      string
TIP_SVG_NAME     33      string
TIP_SPRI         34      int
TIP_TI_THRI      35      int
TIP_TI_AVG       36      float
TIP_TI_XID       37      string
TIP_TI_XA_STATUS 38      string
TIP_GW_NAME      39      string
TIP_GW_NO        40      int
TIP_GW_HOSTN     41      string
TIP_GW_CTYPE     42      string
TIP_GW_CTYPE2    43      string
TIP_GW_IPADDR    44      string
TIP_GW_PORT      45      int
TIP_GW_STATUS    46      string

#
#      ADMIN segment fields
#
# name          number  type    flags  comments
*base 16000650
TIP_IPADDR      0      string
TIP_USERNAME    1      string
TIP_MODULE      2      int
TIP_LOGLVL      3      string

```

```

TIP_SPEC          4          string

#
#          boot time
#
# name            number  type    flags  comments
TIP_BOOTTIME_SEC          5      int
TIP_BOOTTIME_MSEC         6      int

#
#          EXTRA flag fields
#
# name            number  type    flags  comments
*base 16000700
TIP_EXTRA_OPTION          0      int
TIP_SVRI                  1      int
TIP_QPCOUNT               2      int
TIP_EMCOUNT              3      int
TIP_SVR_STATUS            4      string

```

## 12.5.4. システム環境情報照会プログラム

以下は、システムの環境情報を確認するクライアント・プログラムの例です。

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <usrinc/atmi.h>
#include <usrinc/fbuf.h>
#include <usrinc/tip.h>

#define SEC_DOMAIN          1
#define SEC_NODE           2
#define SEC_SVGROUP        3
#define SEC_SERVER         4
#define SEC_SERVICE        5
#define SEC_ROUTING        6
#define SEC_RQ              7
#define SEC_GATEWAY        8

main(int argc, char *argv[])
{
    FBUF      *sndbuf, *rcvbuf;
    TPSTART_T *tpinfo;
    int      i, n, sect;

```

```

long   rcvlen;
char   nodename[NAMELEN];
int     pid, count = 0;

if (argc != 3) {
    printf("Usage: %s section nodename\n", argv[0]);
    printf("section:\n");
    printf("\t1: domain\n");
    printf("\t2: node\n");
    printf("\t3: svrgroup\n");
    printf("\t4: server\n");
    printf("\t5: service\n");
    printf("\t6: routing\n");
    printf("\t7: rq\n");
    printf("\t8: gateway\n");
    exit(1);
}

if (!isdigit(argv[3][0])) {
    printf("fork count must be a digit\n");
    exit(1);
}
count = atoi(argv[3]);

sect = atoi(argv[1]);
if (sect < SEC_DOMAIN || sect > SEC_GATEWAY) {
    printf("out of section [%d - %d]\n", SEC_DOMAIN, SEC_GATEWAY);
    exit(1);
}

strncpy(nodename, argv[2], sizeof(nodename) - 1);

n = tmaxreadenv("tmax.env", "TMAX");
if (n < 0) {
    fprintf(stderr, "can't read env\n");
    exit(1);
}

tpinfo = (TPSTART_T *)tpalloc("TPSTART", NULL, 0);
if (tpinfo == NULL) {
    printf("tpalloc fail tperrno = %d\n", tperrno);
    exit(1);
}
strcpy(tpinfo->username, ".tpadmin");
if (tpstart((TPSTART_T *)tpinfo) == -1){
    printf("tpstart fail [%s]\n", tpstrerror(tperrno));
    exit(1);
}

```

```

}

if ((sndbuf = (FBUF *)tpalloc("FIELD", NULL, 0)) == NULL) {
    printf("tpalloc failed! errno = %d\n", tperrno);
    tpend();
    exit(1);
}

if ((rcvbuf = (FBUF *)tpalloc("FIELD", NULL, 0)) == NULL) {
    printf("tpalloc failed! errno = %d\n", tperrno);
    tpend();
    exit(1);
}

n = fbput(sndbuf, TIP_OPERATION, "GET", 0);
n = fbput(sndbuf, TIP_SEGMENT, "CONFIGURATION", 0);
switch (sect) {
    case SEC_DOMAIN:
        n = fbput(sndbuf, TIP_SECTION, "DOMAIN", 0);
        break;
    case SEC_NODE:
        n = fbput(sndbuf, TIP_SECTION, "NODE", 0);
        break;
    case SEC_SVGROUP:
        n = fbput(sndbuf, TIP_SECTION, "SVGROUP", 0);
        break;
    case SEC_SERVER:
        n = fbput(sndbuf, TIP_SECTION, "SERVER", 0);
        break;
    case SEC_SERVICE:
        n = fbput(sndbuf, TIP_SECTION, "SERVICE", 0);
        break;
    case SEC_ROUTING:
        n = fbput(sndbuf, TIP_SECTION, "ROUTING", 0);
        break;
    case SEC_RQ:
        n = fbput(sndbuf, TIP_SECTION, "RQ", 0);
        break;
    case SEC_GATEWAY:
        n = fbput(sndbuf, TIP_SECTION, "GATEWAY", 0);
        break;
}

n = fbput(sndbuf, TIP_NODENAME, nodename, 0);

n = tpcall("TIP SVC", (char *)sndbuf, 0, (char **)&rcvbuf, &rcvlen,
          TPNOFLAGS);

```

```

        if (n < 0) {
            printf("tpcall fail [%s]\n", tpstrerror(tperrno));
            fbprint(rcvbuf);
            tpfree((char *)sndbuf);
            tpfree((char *)rcvbuf);
            tpend();
            exit(1);
        }

#ifdef 1
        fbprint(rcvbuf);
#endif

        tpfree((char *)sndbuf);
        tpfree((char *)rcvbuf);
        tpend();
    }
}

```

### [結果]

以下は、該当プログラムの結果(Domain Conf)です。

```

$ client 1 tmaxh4
fkey = 217326601, fname = TIP_ERROR, type = int, value = 0
...
fkey = 485762214, fname = TIP_CRYPT, type = string, value = NO
    fkey = 485762215, fname = TIP_DOMAIN_TMMLOGLVL, type = string, value = DEBUG1

    fkey = 485762216, fname = TIP_DOMAIN_CLHLOGLVL, type = string, value = DEBUG2

    fkey = 485762217, fname = TIP_DOMAIN_TMSLOGLVL, type = string, value = DEBUG3

    fkey = 485762218, fname = TIP_DOMAIN_LOGLVL, type = string, value = DEBUG4
    fkey = 217326763, fname = TIP_DOMAIN_MAXTHREAD, type = int, value = 128

```

## 12.5.5. システム統計情報照会プログラム

以下は、システムの統計情報を確認するプログラムの例です。

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <usrinc/atmi.h>
#include <usrinc/fbuf.h>
#include <usrinc/tip.h>

```



```

#define SEC_NODE      1
#define SEC_TPROC     2
#define SEC_SPR       3
#define SEC_SERVICE   4
#define SEC_RQ        5
#define SEC_TMS       6
#define SEC_TMMS      7
#define SEC_CLHS      8
#define SEC_SERVER    9

#define NODE_NAME_SIZE 32

main(int argc, char *argv[])
{
    FBUF      *sndbuf, *rcvbuf;
    TPSTART_T *tpinfo;
    int      i, n, sect;
    long     rcvlen;
    char     nodename[NODE_NAME_SIZE];
    int      stat;

    if (argc != 3) {
        printf("Usage: %s section node\n", argv[0]);
        printf("section:\n");
        printf("\t1: node\n");
        printf("\t2: tproc\n");
        printf("\t3: spr\n");
        printf("\t4: service\n");
        printf("\t5: rq\n");
        printf("\t6: tms\n");
        printf("\t7: tmms\n");
        printf("\t8: clhs\n");
        printf("\t9: server\n");
        exit(1);
    }

    sect = atoi(argv[1]);
    if (sect < SEC_NODE || sect > SEC_CLHS) {
        printf("out of section [%d - %d]\n", SEC_NODE, SEC_TMMS);
        exit(1);
    }

    memset(nodename, 0x00, NODE_NAME_SIZE);
    strncpy(nodename, argv[2], NODE_NAME_SIZE - 1);

    n = tmaxreadenv("tmax.env", "TMAX");

```

```

if (n < 0) {
    fprintf(stderr, "can't read env\n");
    exit(1);
}

tpinfo = (TPSTART_T *)tpalloc("TPSTART", NULL, 0);
if (tpinfo == NULL) {
    printf("tpalloc fail tperrno = %d\n", tperrno);
    exit(1);
}
strcpy(tpinfo->dompwd, "xamt123");

if (tpstart((TPSTART_T *)tpinfo) == -1){
    printf("tpstart fail tperrno = %d\n", tperrno);
    exit(1);
}

if ((sndbuf = (FBUF *)tpalloc("FIELD", NULL, 0)) == NULL) {
    printf("tpalloc failed! errno = %d\n", tperrno);
    tpend();
    exit(1);
}

if ((rcvbuf = (FBUF *)tpalloc("FIELD", NULL, 0)) == NULL) {
    printf("tpalloc failed! errno = %d\n", tperrno);
    tpend();
    exit(1);
}

n = fbput(sndbuf, TIP_OPERATION, "GET", 0);
n = fbput(sndbuf, TIP_SEGMENT, "STATISTICS", 0);
switch (sect) {
    case SEC_NODE:
        n = fbput(sndbuf, TIP_SECTION, "NODE", 0);
        break;
    case SEC_TPROC:
        n = fbput(sndbuf, TIP_SECTION, "TPROC", 0);
        break;
    case SEC_SPR:
        n = fbput(sndbuf, TIP_SECTION, "SPR", 0);
        break;
    case SEC_SERVICE:
        n = fbput(sndbuf, TIP_SECTION, "SERVICE", 0);
        break;
    case SEC_RQ:
        n = fbput(sndbuf, TIP_SECTION, "RQ", 0);
        break;
}

```

```

        case SEC_TMS:
            stat = 1;
            n = fbput(sndbuf, TIP_SECTION, "TMS", 0);
            n = fbput(sndbuf, TIP_EXTRA_OPTION, (char *)&stat, 0);
            break;
        case SEC_TMMS:
            n = fbput(sndbuf, TIP_SECTION, "TMMS", 0);
            break;
        case SEC_CLHS:
            n = fbput(sndbuf, TIP_SECTION, "CLHS", 0);
            break;
        case SEC_SERVER:
            n = fbput(sndbuf, TIP_SECTION, "SERVER", 0);
            break;
    }
    n = fbput(sndbuf, TIP_NODENAME, nodename, 0);

    n = tpcall("TIP SVC", (char *)sndbuf, 0, (char **)&rcvbuf,
              &rcvlen, TPNOFLAGS);
    if (n < 0) {
        printf("tpcall fail [%s]\n", tpstrerror(tperrno));
        fbprint(rcvbuf);
        tpfree((char *)sndbuf);
        tpfree((char *)rcvbuf);
        tpend();
        exit(1);
    }

    fbprint(rcvbuf);

    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
}

```

## [結果]

以下は、該当プログラムの結果(TMS STATISTICS)を表した例です。

```

$ client 3000 1 2
fkey = 217326601, fname = TIP_ERROR, type = int, value = 0
fkey = 485762680, fname = TIP_TMS_NAME, type = string, value = tms_ora2
fkey = 485762681, fname = TIP_SVG_NAME, type = string, value = xal
fkey = 217327226, fname = TIP_SPRI, type = int, value = 0
fkey = 485762649, fname = TIP_STATUS, type = string, value = RUN
fkey = 217327197, fname = TIP_COUNT, type = int, value = 0
fkey = 351544938, fname = TIP_AVERAGE, type = float, value = 0.000000

```

```
fkey = 217327212, fname = TIP_CQCOUNT, type = int, value = 0
fkey = 217327227, fname = TIP_TI_THRI, type = int, value = 1
fkey = 351544956, fname = TIP_TI_AVG, type = float, value = 0.000000
fkey = 485762685, fname = TIP_TI_XID, type = string, value = 00000013664
fkey = 485762686, fname = TIP_TI_XA_STATUS, type = string, value = COMMIT
```

## 12.5.6. サーバー・プロセスの起動および終了プログラム

### 例1

以下は、サーバー・プロセスを追加で起動および終了するプログラムの例です。

< cli.c >

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <usrinc/atmi.h>
#include <usrinc/fbuf.h>
#include <usrinc/tmaxapi.h>
#include <usrinc/tip.h>

#define NODE_NAME_SIZE 32

main(int argc, char *argv[])
{
    FBUF      *sndbuf, *rcvbuf;
    TPSTART_T *tpinfo;
    int      i, n, type, clid, count, flags;
    long     rcvlen;
    char     svrname[TMAX_NAME_SIZE];
    char     svgname[TMAX_NAME_SIZE];
    char     nodename[NODE_NAME_SIZE];
    int      pid, forkcnt;

    if (argc != 6) {
        printf("Usage: %s type svrname count nodename forkcnt\n", argv[0]);
        printf("type 1: BOOT, 2: DOWN, 3: DISCON\n");
        exit(1);
    }

    type = atoi(argv[1]);
    if ((type != 1) && (type != 2) && (type != 3)) {
        printf("couldn't support such a type %d\n", type);
```

```

        exit(1);
    }

    if (strlen(argv[2]) >= TMAX_NAME_SIZE) {
        printf("too large name [%s]\n", argv[1]);
        exit(1);
    }
    strcpy(svrname, argv[2]);
    count = atoi(argv[3]);
    flags = 0;

    strncpy(nodename, argv[4], NODE_NAME_SIZE - 1);
    forkcnt = atoi(argv[5]);

    n = tmaxreadenv("tmax.env", "TMAX");
    if (n < 0) {
        fprintf(stderr, "can't read env\n");
        exit(1);
    }

    tpinfo = (TPSTART_T *)tpalloc("TPSTART", NULL, 0);
    if (tpinfo == NULL) {
        printf("tpalloc fail tperrno = %d\n", tperrno);
        exit(1);
    }
    strcpy(tpinfo->usrname, ".tpadmin");

    for (i = 1; i < forkcnt; i++) {
        if ((pid = fork()) < 0)
            exit(1);
        else if (pid == 0)
            break;
    }

    if (tpstart((TPSTART_T *)tpinfo) == -1){
        printf("tpstart fail tperrno = %d\n", tperrno);
        exit(1);
    }

    if ((sndbuf = (FBUF *)tpalloc("FIELD", NULL, 0)) == NULL) {
        printf("tpalloc failed! errno = %d\n", tperrno);
        tpend();
        exit(1);
    }

    if ((rcvbuf = (FBUF *)tpalloc("FIELD", NULL, 0)) == NULL) {
        printf("tpalloc failed! errno = %d\n", tperrno);

```

```

        tpend();
        exit(1);
    }

    n = fbput(sndbuf, TIP_OPERATION, "GET", 0);
    n = fbput(sndbuf, TIP_SEGMENT, "ADMINISTRATION", 0);
    if (type == 1)
        n = fbput(sndbuf, TIP_CMD, "BOOT", 0);
    else if (type == 2)
        n = fbput(sndbuf, TIP_CMD, "DOWN", 0);
    else
        n = fbput(sndbuf, TIP_CMD, "DISCON", 0);

    if (type == 3) {
        clid = count;          /* at type 3 */
        flags |= TIP_SFLAG;
        n = fbput(sndbuf, TIP_CLID, (char *)&clid, 0);
        n = fbput(sndbuf, TIP_FLAGS, (char *)&flags, 0);
    } else {
        flags |= TIP_SFLAG;
        n = fbput(sndbuf, TIP_SVRNAME, svrname, 0);
        n = fbput(sndbuf, TIP_COUNT, (char *)&count, 0);
        n = fbput(sndbuf, TIP_FLAGS, (char *)&flags, 0);
    }

    n = fbput(sndbuf, TIP_NODENAME, nodename, 0);

    n = tpcall("TIP SVC", (char *)sndbuf, 0, (char **)&rcvbuf,
               &rcvlen, TPNOFLAGS);
    if (n < 0) {
        printf("tpcall failed! errno = %d[%s]\n", tperrno, tpstrerror(tperrno));

        fbprint(rcvbuf);
        tpfree((char *)sndbuf);
        tpfree((char *)rcvbuf);
        tpend();
        exit(1);
    }

    fbprint(rcvbuf);

    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();

```

## 例2

以下は、svr23\_stat\_insというサーバーのログレベルをdebug4に変更する例です。

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

#define NFLAG 32
#define GFLAG 8
#define VFLAG 1024

int case_chlog(int, char *[], FBUF *);

#define NODE_NAME_SIZE 32

main(int argc, char *argv[])
{
    FBUF    *sndbuf, *rcvbuf;
    TPSTART_T *tpinfo;
    int     i, ret, n, type, clid, count, flags = 0;
    long    rcvlen;
    char     svrname[TMAX_NAME_SIZE];
    char     svgname[TMAX_NAME_SIZE];
    char     nodename[NODE_NAME_SIZE];
    int      pid, forkcnt;

    if (argc < 6) {
        printf("Usage: %s svgname svrname nodename [chlogmodule] [flags] [loglvl]\n", argv[0]);
        printf("chlogmodule 1: TIP_TMM, 2: TIP_CLH, 4: TIP_TMS, 8: TIP_SVR\n");

        printf("flags 1: NFLAGS, 2: GFLAGS, 3: VFLAGS\n");
        printf("loglvl : 1: compact, 2: basic, 3: detail, 4: debug1, 5: debug2, 6: debug3, 7: debug4\n");
        exit(1);
    }

    n = tmaxreadenv("tmax.env", "TMAX");
    if (n < 0) {
        fprintf(stderr, "can't read env\n");
    }
}
```

```

        exit(1);
    }

    tpinfo = (TPSTART_T *)tpalloc("TPSTART", NULL, 0);
    if (tpinfo == NULL) {
        printf("tpalloc fail tperrno = %d\n", tperrno);
        exit(1);
    }
    strcpy(tpinfo->usrname, ".tpadmin");
    strcpy(svgname, argv[1]);
    strcpy(svrname, argv[2]);
    strncpy(nodename, argv[3], NODE_NAME_SIZE - 1);

    if (tpstart((TPSTART_T *)tpinfo) == -1){
        printf("tpstart fail tperrno = %d\n", tperrno);
        exit(1);
    }

    if ((sndbuf = (FBUF *)tpalloc("FIELD", NULL, 0)) == NULL) {
        printf("tpalloc failed! errno = %d\n", tperrno);
        tpend();
        exit(1);
    }

    if ((rcvbuf = (FBUF *)tpalloc("FIELD", NULL, 0)) == NULL) {
        printf("tpalloc failed! errno = %d\n", tperrno);
        tpend();
        exit(1);
    }

    ret = case_chlog(argc, argv, sndbuf);
    n = fbput(sndbuf, TIP_OPERATION, "GET", 0);
    n = fbput(sndbuf, TIP_SEGMENT, "ADMINISTRATION", 0);
    n = fbput(sndbuf, TIP_CMD, "CHLOG", 0);
    n = fbput(sndbuf, TIP_NODENAME, nodename, 0);
    n = fbput(sndbuf, TIP_SVGNAME, svgname, 0);
    n = fbput(sndbuf, TIP_SVRNAME, svrname, 0);

    n=tpcall("TIPSVC", (char *)sndbuf, 0, (char **)&rcvbuf, &rcvlen, TPNOFLAGS);

    if (n < 0) {
        printf("tpcall failed! errno = %d[%s]\n", tperrno,
            tpstrerror(tperrno));
        fbprint(rcvbuf);
        tpfree((char *)sndbuf);
        tpfree((char *)rcvbuf);
        tpend();
    }

```



```

        exit(1);
    }

    fbprint(rcvbuf);
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
}

int case_chlog(int argc2, char *argv2[], FBUF *sndbuf)
{
    int  chlogmdl, loglvl, flags, n=0;
    char cloglvl[TMAX_NAME_SIZE];
    const int true = 1, false = 0;

    chlogmdl = atoi(argv2[4]);
    if( (chlogmdl != 1) && (chlogmdl != 2) && (chlogmdl != 4) &&
        (chlogmdl != 8)
    {
        printf("couldn't support such a chlogmdl\n");
        exit(1);
    }

    flags = atoi(argv2[5]);
    if( (flags != NFLAG) && (flags != GFLAG) && (flags != VFLAG) )
    {
        printf("couldn't support such a flags\n");
        exit(1);
    }

    loglvl = atoi(argv2[6]);
    if( (loglvl < 1) || (loglvl > 7) )
    {
        printf("couldn't support such a loglvl\n");
        exit(1);
    }

    switch (loglvl)
    {
        case 1 :
            strcpy(cloglvl, "compact");
            break;
        case 2 :
            strcpy(cloglvl, "basic");
            break;
        case 3 :
            strcpy(cloglvl, "detail");

```

```

        break;
    case 4 :
        strcpy(cloglvl, "debug1");
        break;
    case 5 :
        strcpy(cloglvl, "debug2");
        break;
    case 6 :
        strcpy(cloglvl, "debug3");
        break;
    case 7 :
        strcpy(cloglvl, "debug4");
        break;
}
n = fbput(sndbuf, TIP_MODULE, (char *)&chlogmdl, 0);
n = fbput(sndbuf, TIP_FLAGS, (char *)&flags , 0);
n = fbput(sndbuf, TIP_LOGLVL, cloglvl , 0);
return 1;
}

```

#### [結果] (TIP\_SVR, => DEBUG4)

```

$ client xal svr23_stat_ins $HOSTNAME 8 1024 7
fkey = 217326601, fname = TIP_ERROR, type = int, value = 0
>>> tadmin (cfg -v)

loglvl = DEBUG4

```

## 12.6. 再帰呼び出し(Local recursive call)

サーバーでtpcall()を実行する場合、同じサーバーに存在するサービスの場合にも内部で再帰呼び出しができるように機能が追加されました。これは、サーバーでマルチ・コンテキストの手法を使用して、tpcall()に限って再帰呼び出し(Local Recursive call)が可能となるようにしたものです。無限ループを防ぐために、最大階層は8に制限します。

---

#### 注

再帰呼び出しを使用するためには、サーバー・プログラムをコンパイルする際に必ずCFLAGSに-D\_MCONTEXTを追加し、libsvr.soの代わりにlibsvrmc.soサーバー・ライブラリーを利用してコンパイルする必要があります。

---

## サーバー・プログラム

以下は、サーバー・プログラムの例です。

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <usrinc/atmi.h>

SVC15004_1(TPSVCINFO *msg)
{
    int      i;
    char      *rcvbuf;
    long      rcvlen;

    if ((rcvbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL)
        printf("rcvbuf tpalloc fail[%s]\n", tpstrerror(tperrno));
    if (tpcall("SVC15004_2", msg->data, 0, &rcvbuf, &rcvlen, 0) == -1)
    {
        printf("tpcall fail [%s]\n", tpstrerror(tperrno));
        tpfree((char *)rcvbuf);
        tpreteturn(TPFAIL, 0, 0, 0, 0);
    }

    strcat(rcvbuf, "_Success");

    tpreteturn(TPSUCCESS, 0, (char *)rcvbuf, 0, 0);
}

SVC15004_2(TPSVCINFO *msg)
{
    int      i;
    char      *rcvbuf;
    long      rcvlen;

    if ((rcvbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL)
        printf("rcvbuf tpalloc fail \n");
}

if (tpcall("SVC15004_3", msg->data, 0, &rcvbuf, &rcvlen, 0) == -1)
{
    printf("tpcall fail [%s]\n", tpstrerror(tperrno));
    tpfree((char *)rcvbuf);
    tpreteturn(TPFAIL, 0, 0, 0, 0);
}
strcat(rcvbuf, "_Success");
tpreteturn(TPSUCCESS, 0, (char *)rcvbuf, 0, 0);
}

SVC15004_3(TPSVCINFO *msg)

```

```

{
    int      i;
    char     *rcvbuf;
    long     rcvlen;

    if ((rcvbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL)
        printf("rcvbuf tpalloc fail \n");

    if (tpcall("SVC15004_4", msg->data, 0, &rcvbuf, &rcvlen, 0) == -1)
    {
        printf("tpcall fail [%s]\n", tpstrerror(tperrno));
        tpfree((char *)rcvbuf);
        tpreturn(TPFAIL, 0, 0, 0, 0);
    }

    strcat(rcvbuf, "_Success");
    tpreturn(TPSUCCESS, 0, (char *)rcvbuf, 0, 0);
}

```

## メイクファイル

以下は、メイクファイルの例です。

<Makefile.c.mc>

```

# Server makefile

TARGET   = $(COMP_TARGET)
APOBJS   = $(TARGET).o
NSDLOBJ  = $(TMAXDIR)/lib64/sdl.o

LIBS     = -lsvrmc -lnodb
OBSJ     = $(APOBJS) $(SVCTOBJ)

SVCTOBJ  = $(TARGET)_svctab.o

CFLAGS   = -O -Ae -w +DSblended +DD64 -D_HP -I$(TMAXDIR) -D_MCONTEXT

APPDIR   = $(TMAXDIR)/appbin
SVCTDIR  = $(TMAXDIR)/svct
LIBDIR   = $(TMAXDIR)/lib64

#
.SUFFIXES : .c

.c.o:

```

```

$(CC) $(CFLAGS) -c $<

#
# server compile
#

$(TARGET): $(OBJS)
    $(CC) $(CFLAGS) -L$(LIBDIR) -o $(TARGET) $(OBJS) $(LIBS) $(NSDLOBJ)
    mv $(TARGET) $(APPDIR)/.
    rm -f $(OBJS)

$(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)

```



# 付録 A. Tmax環境設定

本章では、Tmax環境ファイルの構成要素について説明します。

## A.1. 環境ファイル

Tmax環境ファイルはTmaxのシステム構成を設定し、Tmax管理者が作成します。このファイルはサービス・テーブルの作成とTmaxシステムの起動に使われます。

環境ファイルの構成内容は以下のとおりです。

セクション	説明	必須可否
DOMAIN	1つの独立的なTmaxシステムの全体環境を定義します	必須
NODE	DOMAINを構成する各ノードに関連する環境を定義します	必須
SVRGROUP	サーバー・グループおよびデータベース関連項目を定義します	必須
SERVER	サーバー関連項目を定義します	必須
SERVICE	サービス関連項目を定義します	必須
GATEWAY	ドメイン間のゲートウェイ関連項目を定義します	選択
ROUTING	データ依存型ルーティング関連項目を定義します	選択
RQ	信頼性のあるキュー関連項目を定義します	選択

各セクションの名前はアスタリスク(\*)で始め(例: \*DOMAIN, \*NODEなど)、各セクションの名前とセクションの下位エンティティの名前は必ず行の1番目のコマで始めます。1つの下位エンティティに対する定義はカンマ(,)で区切ります。

### 参考

Tmax環境設定の詳細については、『Tmax 運用ガイド』を参照してください。

一般テキスト・ファイルでTmax環境ファイルが作成され、cflコマンドでコンパイルします。コンパイルした後作成されたバイナリーファイルはサービス・テーブルの作成とTmaxシステムが起動するときに参照されます。

```
cfl -i Tmax環境ファイル名
```

以下は、Tmax環境ファイルの例です(「< >」で囲んでいる部分は代替可能)

```

*DOMAIN
<resrc_name>          SHMKEY = <UNIQUE IPCKEY>,
                      MAXUSER = 256

*NODE
<uname>              TMAXDIR = "<TMAX installed directory>"
                      APPDIR = "<APPLICATION DIRECTORY>",
                      PATHDIR = "<path directory>"

*SVRGROUP
<svg_name>           NODENAME = <uname> ,
                      OPENINFO = "Oracle_XA+Acc=P/scott/tiger+SesTm=60"

*SERVER
<svr_name>           SVGNAME = <sgrpname> , MIN = 1, MAX = 3

*SERVICE
<svc_name>           SVRNAME = <svrname>

```

## A.1.1. DOMAINセクション

Tmaxのシステム構成を設定します。

```

*DOMAIN
<resrc_name>          SHMKEY = <UNIQUE IPCKEY>,
                      MAXUSER = 256

```

項目	説明
<resrc_name>	1つのTmaxシステム(ドメイン)の一意の名前で、63字以内で定義します
SHMKEY	内部的に共有メモリーを割り当てるための値で、システム全体で一意の値でなければなりません。32,768から262,143の間で選択します
MAXUSER	ドメインに同時アクセス可能な最大のクライアント数です

## A.1.2. NODEセクション

ドメインを構成する各ノードの情報を設定します。

```

*NODE
<uname>              TMAXDIR = "<TMAX installed directory>"
                      APPDIR = "<APPLICATION DIRECTORY>",
                      PATHDIR = "<path directory>"

```



項目	説明
<uname>	63字以内の文字列で定義されるノード名です。これは、 <b>uname -n</b> コマンドで返される値を使用します。hostnameの戻り値と「uname -n」の戻り値が同じである必要があります
TMAXDIR	Tmaxが存在するディレクトリー情報です
APPDIR	アプリケーションが存在するディレクトリー情報です
PATHDIR	Tmaxプロセスの内部通信のためのディレクトリー情報です

### A.1.3. SVRGROUPセクション

論理的な関連性を持つサーバーの集まりで、データベースの使用や負荷調節、障害対策などの基本単位になります。1つのマシンに対して少なくとも1つのサーバー・グループが定義されています。

```
*SVRGROUP
<svg_name>      NODENAME = <uname> ,
                  OPENINFO = "Oracle_XA+Acc=P/scott/tiger+SesTm=60" ,
                  DBNAME = ORACLE,
                  TMSNAME = tms_ora
```

項目	説明
<svg_name>	サーバー・グループ名として、63字以内の文字列で定義します。 <b>SVRGROUP</b> セクションで一意である必要があります
NODENAME	サーバー・グループが存在するノード名です。サーバー・グループにはデータベースについての定義が含まれます。したがって、データベースを使用する場合、DBNAME、OPENINFO、TMSNAMEの3つの項目を定義する必要があります
DBNAME	使用するデータベース名(例:ORACLE)を定義します
OPENINFO	データベースと接続を確立するための情報を登録します。256字以内の文字列で定義し、データベースの種類に応じて登録方法が異なります <ul style="list-style-type: none"> <li>Oracle <pre>OPENINFO = "Oracle_XA+Acc = P/account/password+SesTm = 60"</pre> </li> <li>Informix <pre>OPENINFO = "データベース名"</pre> </li> </ul>
CLOSEINFO	データベースとの接続を解除するために必要な情報を登録します。256字以内の文字列で定義し、省略可能です。データベースの種類に応じて登録方法が異なります <ul style="list-style-type: none"> <li>Oracle</li> </ul>

項目	説明
	<pre>CLOSEINFO = " "</pre> <p>– Informix</p> <pre>CLOSEINFO = " "</pre>
TMSNAME	TMSプロセス名として、63字以内で定義します

## A.1.4. SERVERセクション

応用サーバーに対する情報を登録します。

```
*SERVER
<svr_name>          SVGNAME = <sgrpname> , MIN = 1, MAX = 3
```

項目	説明
<svr_name>	サービスを提供するサーバーの実行ファイルの名前で、63字以内で定義します
SVGNAME	サーバー・プロセスが属するサーバー・グループ名で、*SVRGROUPに定義されている必要があります
CLOPT	コマンド行のオプションを渡すためのパラメータです。システム・オプションとユーザー・オプションをハイフン(-)で区切ります。この行は、tpsvrinit()関数で処理します
MIN	サーバーを複数動作させる場合、基本的に動作させるサーバー数です
MAX	動的に追加動作できるサーバーの最大数です
CONV	値が「Y」であれば、会話型通信が実行される会話型サーバーであることを示します(基本値:N)

## A.1.5. SERVICEセクション

サービスに対する情報を設定します。

```
*SERVICE
<svc_name>          SVRNAME = <svrname>
```

項目	説明
<svc_name>	クライアントで呼び出す関数名です。63字以内で定義し、一意の名前である必要があります。 <b>SERVICE</b> セクションに登録されているサービスのみ対応できます
SVRNAME	サービスを提供するサーバー名です

## A.2. メイクファイル

本節では、OracleとInfomixのためのTMSメイクファイルの例について説明します。

### Oracle用のTMSメイクファイル

以下は、Oracle用のTMSメイクファイル(solaris 32ビット)の例です。

```
#
include $(ORACLE_HOME)/precomp/lib/env_precomp.mk
ORALIBDIR = $(LIBHOME)
ORALIB = $(PROLDLIBS) $(LIBCLNTSH)

TARGET    = tms_ora
APOBJ     = dummy.o

APPDIR    = $(TMAXDIR)/appbin
TMAXLIBD= $(TMAXDIR)/lib

TMAXLIBS  = -ltms -loras -lsocket -lnsl

all: $(TARGET)

$(TARGET): $(APOBJ)
    $(CC) -L$(TMAXLIBD) -o $(TARGET) -L$(ORALIBDIR) $(ORALIB) $(APOBJ)
    $(TMAXLIBS)
    mv $(TARGET) $(APPDIR)

$(APOBJ):
    $(CC) -c dummy.c

#
clean:
    -rm -f *.o core $(TARGET)
```

### Informix用のTMSメイクファイル

以下は、Informix用のTMSメイクファイル(solaris 32ビット)の例です。

```
TARGET    = tms_info

INFOLIBDIR = ${INFORMIXDIR}/lib
INFOELIBDIR = ${INFORMIXDIR}/esql
INFOLIBD = ${INFORMIXDIR}/lib/esql
INFOLIBS = -lifsq1 -lifasf -lifgen -lifgls -lifos -lnsl -lsocket -laio -elf -lm
```

```
-ldl
    ${INFOLIBDIR}/esql/checkapi.o -lifglx -lifxa

TMAXLIBDIR = $(TMAXDIR)/lib
TMAXLIBS   = -ltms -linfs

$(TARGET):
    $(CC) -o $(TARGET) -L$(TMAXLIBDIR) -L$(INFOLIBD) -L$(INFOLIBDIR)
    -L$(INFOELIBDIR) $(TMAXLIBS) $(INFOLIBS)
    cp $(TARGET) $(TMAXDIR)/appbin

#
clean:
    -rm -f core $(TARGET)
```

# 索引

## シンボル

2PC, 4, 44

## A

ACID, 43

AP, 45

Atomicity, 43

## C

Commit段階, 44

Consistency, 43

CRM, 45

## D

Durability, 43

## E

Explicitトランザクション方法, 46

## I

Implicitトランザクション方法, 46

Isolation, 43

## L

Local recursive call, 356

## M

Multicontext, 54

Multithread, 53

サービススレッド, 65

ユーザー作成スレッド, 67

## O

OSI-TP, 45

## P

Prepare段階, 44

## R

RDP, 16

RM, 45

## S

sdlc, 41

## T

TCS, 16

TIP (Tmax Information Provider), 327

TM, 45

tmaxoserrno, 240

Tmaxアプリケーション, 1

TMS, 47

tpacall, 98

tpalloc, 129

tpbroadcast, 166

tpcall, 93

tpcancel, 105

tpchkclid, 213

tpclrfd, 225

tpconnect, 107

tpdeq, 155

tpdiscon, 118

tpend, 92

tpenq, 152

tpextsvcname, 160

tpforward, 200

tpfree, 133

tpgetclid, 212

tpgetctxt, 186

tpgetrply, 102

tpgetunsol, 123

tpissetfd, 222

tppost, 165

tpqstat, 158

tprealloc, 131

tprecv, 114

tpregcb, 233

tprelay, 231  
tpreturn, 196  
tpsavectx, 227  
tpschedule, 217  
tpsend, 111  
tpsendtocli, 210  
tpset\_timeout, 126  
tpsetctx, 189  
tpsetfd, 220  
tpsetsvctimeout, 127  
tpsetunsol, 121  
tpstart, 88  
tpstrerror, 237  
tpsubscribe, 162  
TPSVCINFO構造体, 194  
tpsvrdone, 205  
tpsvrinit, 203  
tpsvrthrdone, 209  
tpsvrthrinit, 206  
tptypes, 134  
tpunregcb, 235  
tpunsubscribe, 164  
tpuschedule, 218  
tx\_begin, 136  
tx\_commit, 138  
tx\_info, 140  
tx\_rollback, 142  
tx\_set\_commit\_return, 149  
tx\_set\_transaction\_control, 147  
tx\_set\_transaction\_timeout, 145

## U

UCS, 16  
Ustrerror, 240  
Uunix\_err, 239  
Uunixerr, 239

## W

WinTmaxAcall, 170  
WinTmaxAcall2, 174  
WinTmaxEnd, 179  
WinTmaxSend, 182

WinTmaxSetContext, 180  
WinTmaxStart, 178

## X

XAモード, 47

## か

会話型通信, 36  
クライアント・プログラム, 1  
グローバル・トランザクション, 43, 256

## さ

サーバー・プログラム, 2  
サービス・アクセス制御, 82  
システム接続セキュリティ, 79

## た

トランザクション, 43

## は

非XAモード, 48  
非同期型通信, 35  
同期型通信, 33, 34  
標準通信型バッファ, 41

## や

ユーザー認証セキュリティ, 80

## ら

ローカル・トランザクション, 43