

Application Development Guide

Tmax 6

TMAXSOFT

Copyright

Copyright 2018. TmaxSoft Co., Ltd. All Rights Reserved.

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.

Trademarks

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

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, netsnmp, 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

Document History

Product Version	Guide Version	Date	Remarks
Tmax 6	2.1.1	2018-06-11	-

Contents

1. Introduction to Tmax Applications	1
1.1. Overview	1
1.2. Structure	1
1.3. Characteristics	2
2. Client Programs	5
2.1. Characteristics and Components	5
2.2. Development Environments and Tools	5
2.3. Program Flow	6
2.4. Compiling a Program	9
2.5. Starting and Terminating a Process	11
2.5.1. sdlc	12
2.5.2. fdlc	13
3. Server Programs	15
3.1. Characteristics and Components	15
3.1.1. TCS	16
3.1.2. UCS	17
3.2. Development Environments and Tools	19
3.3. Program Flow	19
3.3.1. TCS	20
3.3.2. UCS	21
3.4. Compiling a Program	24
3.4.1. Compiling a TCS Server Program	24
3.4.2. UCS	28
3.5. Creating and Terminating a Process	29
4. Communication Mode	31
4.1. Overview	31
4.2. Synchronous Communication	31
4.3. Asynchronous Communication	32
4.4. Interactive Communication	33
4.4.1. Events Related to Interactive Communication	35
5. Buffer Types	37
5.1. Overview	37
5.2. Buffer Types	37
5.3. Managing a Buffer	38
5.3.1. Struct Buffers	39
5.3.2. Field Buffers	39
6. Transactions	41
6.1. Overview	41

6.2. Distributed Transaction	41
6.2.1. XA Mode	45
6.2.2. Non-XA Mode	46
6.3. Transaction Errors	47
6.3.1. TX Error	48
6.3.2. XA Error	48
7. Multithreading and Multicontexting	49
7.1. Overview	49
7.2. Client Program	49
7.2.1. Program Flow	49
7.2.2. Implementation	51
7.2.3. Program Example	52
7.3. Server Program	58
7.3.1. Overview	58
7.3.2. Program Flow	59
7.3.3. Implementation	62
7.3.4. Service Processing Program Example	64
7.3.5. Context-Sharing Program Example	66
8. Security System	70
8.1. Overview	70
8.2. Level 1 Security (System Access Control)	70
8.3. Level 2 (User Authentication)	71
8.4. Level 3 (Service Access Control)	72
9. Client API	75
9.1. Overview	75
9.2. Connection and Disconnection	78
9.2.1. tpstart	78
9.2.2. tpend	81
9.3. Synchronous Communication	83
9.3.1. tpcall	83
9.4. Asynchronous Communication	87
9.4.1. tpacall	87
9.4.2. tpgetrply	90
9.4.3. tpcancel	93
9.5. Interactive Communication	95
9.5.1. tpconnect	95
9.5.2. tpsend	98
9.5.3. tprecv	101
9.5.4. tpdicon	105
9.6. Unsolicited Message Processing	107
9.6.1. tpsetunsol	107

9.6.2. tpgetunsol	109
9.7. Timeout Change	112
9.7.1. tpset_timeout	112
9.7.2. tpsetsvctimeout	113
9.8. Buffer Management	114
9.8.1. tpalloc	115
9.8.2. tprealloc	117
9.8.3. tpfree	118
9.8.4. tptypes	120
9.9. Transaction Management	121
9.9.1. tx_begin	121
9.9.2. tx_commit	123
9.9.3. tx_info	125
9.9.4. tx_rollback	127
9.9.5. tx_set_transaction_timeout	129
9.9.6. tx_set_transaction_control	131
9.9.7. tx_set_commit_return	133
9.10. RQ System	136
9.10.1. tpenq	136
9.10.2. tpdeq	139
9.10.3. tpqstat	141
9.10.4. tpextsvcname	143
9.11. Functions using Events	145
9.11.1. tpsubscribe	145
9.11.2. tpunsubscribe	146
9.11.3. tppost	147
9.12. Broadcast and Multicast	149
9.12.1. tpbroadcast	149
9.13. Environment Program	151
9.13.1. WinTmaxAcall	152
9.13.2. WinTmaxAcall2	156
9.13.3. WinTmaxStart	160
9.13.4. WinTmaxEnd	161
9.13.5. WinTmaxSetContext	162
9.13.6. WinTmaxSend	164
9.14. Multithread and Multicontext	168
9.14.1. tpgetctxt	168
9.14.2. tpsetctxt	170
10. Server API	174
10.1. TCS	174
10.1.1. tpreturn	177

10.1.2. tpforward	181
10.1.3. tpsvrinit	183
10.1.4. tpsvrdone	185
10.1.5. tpsvrthrinit	186
10.1.6. tpsvrthrdone	189
10.1.7. tpgetctxt	189
10.1.8. tpsetctxt	189
10.1.9. tpsendtocli	190
10.1.10. tpgetclid	192
10.1.11. tpchkclid	193
10.2. UCS	194
10.2.1. tpschedule	196
10.2.2. tpuschedule	197
10.2.3. tpsetfd	199
10.2.4. tpissetfd	201
10.2.5. tpclrfd	203
10.2.6. tpsavectx	205
10.2.7. tpgetctx	207
10.2.8. tpcancelctx	208
10.2.9. tprelay	209
10.2.10. tpregcb	211
10.2.11. tpunregcb	212
11. Error Handling	214
11.1. Overview	214
11.2. API Level Error Processing	214
11.2.1. tpstrerror	214
11.3. System Level Error Processing	215
11.3.1. Uunixerr	215
11.3.2. Uunix_err	216
11.3.3. Ustrerror	216
11.3.4. tmaxoserrno	217
11.4. Debug	218
11.4.1. Debug CLH	218
11.4.2. Debug Library	218
12. Examples	220
12.1. Programs for Each Communication Type	220
12.1.1. Synchronous Communication	220
12.1.2. Asynchronous Communication	223
12.1.3. Interactive Communication	226
12.2. Global Transaction Programs	232
12.3. Database Programs	238

12.3.1. Oracle Insert Program	238
12.3.2. Oracle Select Program	243
12.3.3. Informix Insert Program	249
12.3.4. Informix Select Program	256
12.3.5. DB2 Program	263
12.4. Database Integration Programs	271
12.4.1. Synchronous Mode (Homogeneous Database)	271
12.4.2. Synchronous Mode (Heterogeneous Database)	277
12.4.3. Asynchronous Mode (Homogeneous Database)	280
12.4.4. Interactive Mode (Homogeneous Database)	286
12.5. Programs Using TIP	294
12.5.1. TIP Structure	294
12.5.2. TIP Usage	297
12.5.3. TIP Usage Example	299
12.5.4. Program for Checking System Environment Information	307
12.5.5. Program for Checking System Statistical Information	310
12.5.6. Program for Starting and Terminating a Server Process	313
12.6. Local Recursive Calls	318
Appendix A: Configuring Tmax	321
A.1. Configuration File	321
A.1.1. DOMAIN Section	322
A.1.2. NODE Section	322
A.1.3. SVRGROUP Section	323
A.1.4. SERVER Section	324
A.1.5. SERVICE Section	324
A.2. Makefile	325

1. Introduction to Tmax Applications

This chapter gives an overview of Tmax application programs and describes their structure and characteristics.

1.1. Overview

Tmax application programs are client/server programs developed in an open environment that uses Tmax as middleware. The emergence of high-performance PCs and the development of programming technologies have brought about changes from central computing to client/server environments, where task processing is divided between clients and servers. Client/server computing has enabled efficient utilization of resources including the use of high-performance PCs, the scaling down of servers, and the expansion of hardware options.

The client/server environment, however, still has the following problems:

- Developers must have in-depth knowledge of hardware, operating systems, and network protocols.
- An increase in the number of users or the amount of data sharply decreases computing speed.
- Excessive network traffic decreases computing speed.
- A distributed environment creates problems related to process management, communication, security, and error handling.

Tmax application programs remedy the drawbacks of client/server programs while maintaining their benefits. Application programs based on Tmax as middleware are used to manage communication programs, processes, and transactions. Tmax functions are divided into server library (libsvr.a) and client library (libcli.a) functions that handle buffers, communication, and transactions. These functions comply with the X/Open Data Transaction Processing (DTP) model, which is the international standard for distributed processing.

1.2. Structure

A Tmax application program consists of a **client program** and a **server program**. It requires an environment configuration to use Tmax.

- Client program

Client programs request services on behalf of users, receive results from a server, and return the results to users in the user-specified format. These programs require the UNIX configuration file (for example, the .profile file). The UNIX configuration file is used to configure the information, such as the address and port number, necessary for clients to connect to Tmax. Client programs refer to relevant binary files when connecting to Tmax and use struct buffers or field buffers with environment variables in the user configuration file.

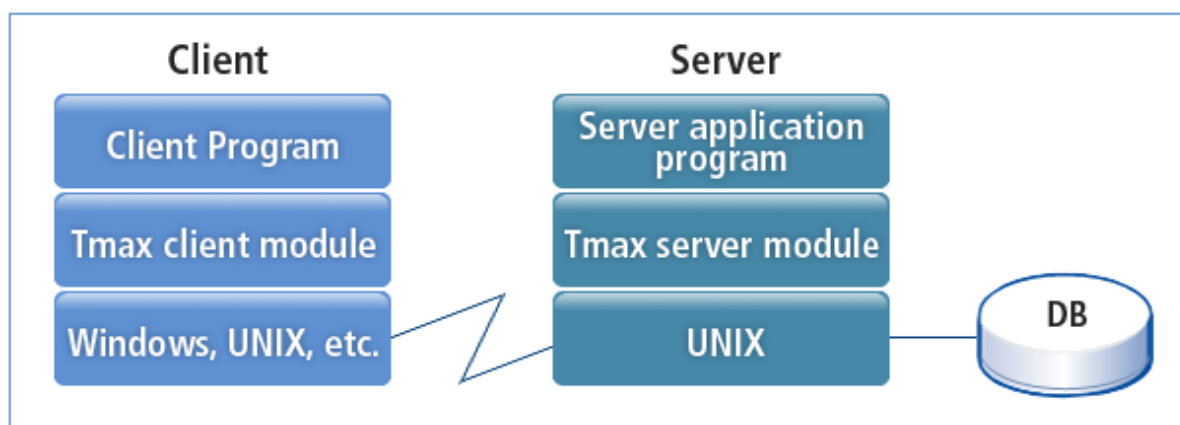
- Server program

Server programs receive requests from clients, control access to system resources, process client requests, and return the results to clients.

Server programs consist of the `main()` task, which is provided and managed by Tmax, and service routines developed by developers. Service routines and service tables created with Tmax utilities are compiled together. If a struct buffer is used, a conversion routine created with a Tmax utility must be run.

Tmax application servers need **server programs** and the **Tmax configuration file**. The Tmax configuration file defines the entire system environment of resources and is written by the administrator. The configuration file is referenced when the Tmax server and server programs are launched. The configuration file is also referenced when service tables are created.

The following figure shows the structure of a Tmax application program.



Tmax Application Program



For more information about client/server programs, refer to [Client Programs](#) and [Server Programs](#).

1.3. Characteristics

Unlike a UNIX program, a Tmax application offers the following features:

- Compliance with the X/Open Distributed Transaction Processing (DTP) model

Routines that control the network use Tmax functions, which comply with the international standard. Programs developed in other middleware that comply with the X/Open DTP model are compatible with Tmax.

- Programs require only service routines that handle client requests.

Clients must develop programs using general C language syntax, which includes `main()`. Server programs only require request-processing routines, which do not include `main()`.

- Service requests can be issued in one of three communication modes.

Synchronous, asynchronous, and interactive communication modes are supported. With these modes, developers can easily develop programs without deep knowledge of communication networks.

The following are descriptions of each mode.

Communication Mode	Description
Synchronous Communication	Waits for a response after requesting a service request (tpcall).
Asynchronous Communication	After requesting a service (tpacall), other tasks are executed until a response (tpgetrply) is received.
Conversational Communication	Sends (tpsend) and receives (tprecv) messages repeatedly after the initial connection is made (tpconnect).



For more information about communication modes, refer to [Communication Mode](#).

- Seven types of buffers can be used.

Tmax supports seven types of buffers for requesting a service. In general, developers find it difficult to guarantee data integrity for data communication between different types of devices. That is because different types of hardware and operating systems support different lengths for a certain type of data and methods of allocating data to the memory. Such differences in processing data may cause a value contained in a message to be recognized as another. To avoid this problem, data must be converted into a standard format so that the data can be recognized to hold the same value regardless of what hardware or operating system is used.

Because Tmax supports various buffers, developers can easily develop programs without deep knowledge of machines, operating systems, and networks. Tmax enhances system performance and reduces network load by providing various buffers including the structure type and the string type.



For more information about buffers, refer to [Buffer Types](#).

- Program development is simple.
 - Developers can develop programs without deep knowledge of hardware and communication networks.
 - Clients can request services without knowing the address of the server that handles the request.
 - Clients can request services only with a name.
- Transaction integrity is guaranteed.

If the range of transactions is set in a program, Tmax guarantees transaction integrity with two-

phase commit (2PC).



2PC handles transactions in two phases (prepare phase and commit phase) when multiple databases are integrated. For more information, refer to [Transactions](#).

2. Client Programs

This chapter describes client program characteristics, components, flow, and development environment.

2.1. Characteristics and Components

Tmax provides functions for the communication network and buffer management, which enable easy development of application programs. These functions are offered as a library and compiled together with application programs. Developers can develop client programs without knowing the origin of services and servers.

Once a client program is written, you need to compile it and create an executable file. In order to compile a client program, the following are required: a client program written by developers, the Tmax client library, a structure file (if a struct buffer is used), and a field table definition file (if a field key buffer is used).

A client program consists of the following components:

- Client program

A client program written by developers.

- Client library

A library (libcli.a / libcli.so) provided by Tmax. It consists of function object codes used for developing client programs.

- Structure file (.s)

A structure file of <filename.s> is required to use structures (STRUCT, X_C_TYPE, or X_COMMON) in a client program. The struct file is precompiled by the sdhc command to create a binary file. The binary file contains information required to convert each struct data into a standard data type. This is used to send/receive data in standard format when a client program is run.

- Field key file (.f)

A field definition file of <filename.f> is required when a field key buffer is used. When the file is compiled by the fdhc command, the field key buffer file creates <field key buffer name_fdl.h> through key mapping and is used as a program. Unlike conventional structure files, users are allowed to modify the field values selectively to prevent resource waste. However, overhead is inevitable during key mapping.

2.2. Development Environments and Tools

Tmax provides the following environment and tools for developing client programs:

- Environment

UNIX, Windows NT, Windows 95/98/2000, Windows 3.1, and MS-DOS.

- Tools

ANSI C, VC++, BC++, VB, VB .Net, C#, Delphi, PowerBuilder, and Embedded VC.



For more information about program development using development tools, refer to *Tmax Programming Guide (4GL)*.

2.3. Program Flow

A client program passes a request from a user to a server, and returns the results from the server to the user.

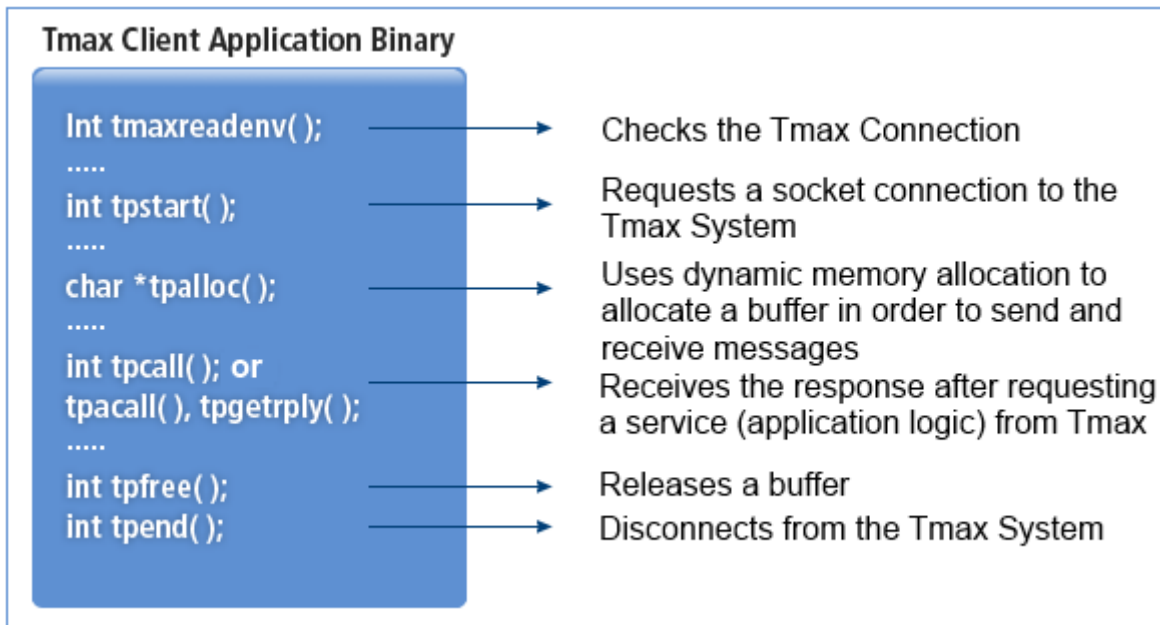
The following are the steps for writing a client program.

```
main()
{
    Allocate a buffer for tpstart.
    Initialize a buffer for tpstart.

    Connect to Tmax.
    Allocate a buffer for data transmission and reception
    while {
        Input user requests into the transmit message buffer.
        Send the transmit message buffer to a server (service request).
        Receive a response from the server with the receive message buffer.
        Show the receive message buffer to users.
    }
    Deallocate the transmit/receive message buffer.

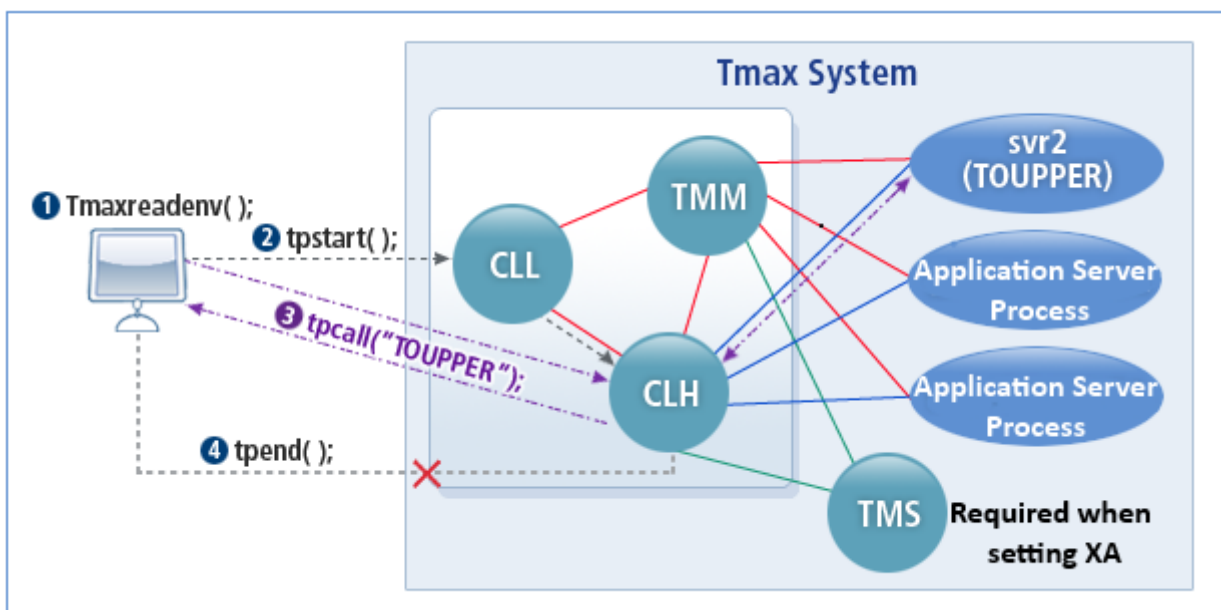
    Disconnect from Tmax.
}
```

The following shows the flow of a client program.



Client Program Flow

The following shows the process of each function in a client program.



Process of Functions in a Client Program



For more information about functions related to communication networks and buffer management for application development, refer to [Client API](#) or *Tmax Reference Guide*.

The following describes the major functions of a client program.

- Functions for Tmax communication environment

Function	Description
tmaxreadenv	<p>Defines Tmax environment variables that are referenced when executing a Tmax client program in a file.</p> <p>When you want a client program to call a specific API, you just need to specify its name and path. When the client program is run, it references the file pointer of the specified file and parses it. The parsed file contains Tmax environment variables TMAX_HOST_ADDR and TMAX_HOST_PORT, which are loaded to memory secured on the execution of the client program. The variables are referenced when the associated API functions are called.</p>

- Functions for connecting/disconnecting a client to/from a server

Function	Description
tpstart	Requests a socket connection through the client listener (CLL) and informs of an accepted connection to the client handler (CLH). TMAX_HOST_ADDR and TMAX_HOST_PORT are used to call tpstart().
tpend	Terminates a socket connection by using CLL.

- Functions for allocating and releasing a buffer

Function	Description
tpalloc	Dynamically allocates the buffer used for passing data between the client and the server through Tmax. The buffer size must be calculated to store data and allocate it. Dynamically allocated memory must be returned with an explicit command.
tpfree	Releases memory that was dynamically allocated by using tpalloc(). If the memory is not returned, garbage (memory leakage) is created.

- Functions for synchronous communication

Function	Description
tpcall	Sends a service request and data to CLH, which checks the requested service, and transfers it to the server process. The client waits until it receives a response for the request.

- Functions for asynchronous communication

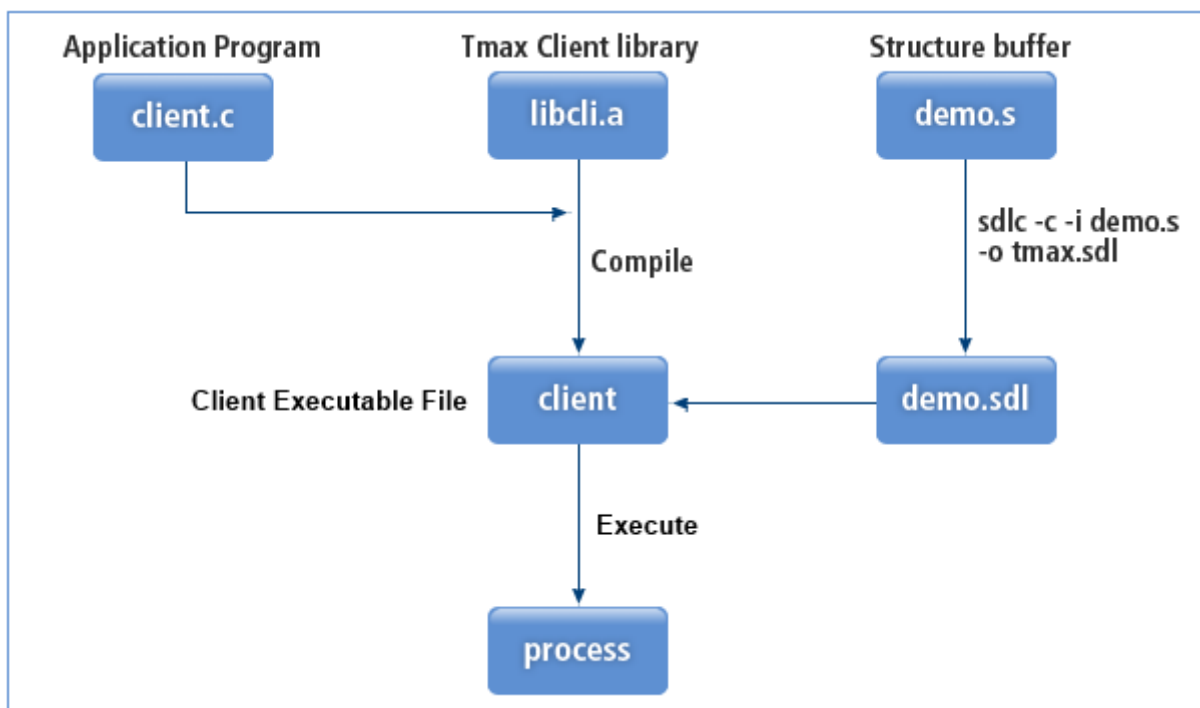
Function	Description
tpacall	<p>Sends a service request and transmitted data to CLH, which checks the requested service, and transfers it to a server application process.</p> <p>The client executes the next logic immediately after calling tpacall() without waiting for a reply.</p>

Function	Description
tpgetrply	Requests reply data from CLH by using the value of tpcall() parameter (cd). If the response data for the corresponding client exists, the data is immediately sent to the client. According to the value in the flags parameter, tpgetrply waits until it receives the reply data, or sends a reception error to a client.

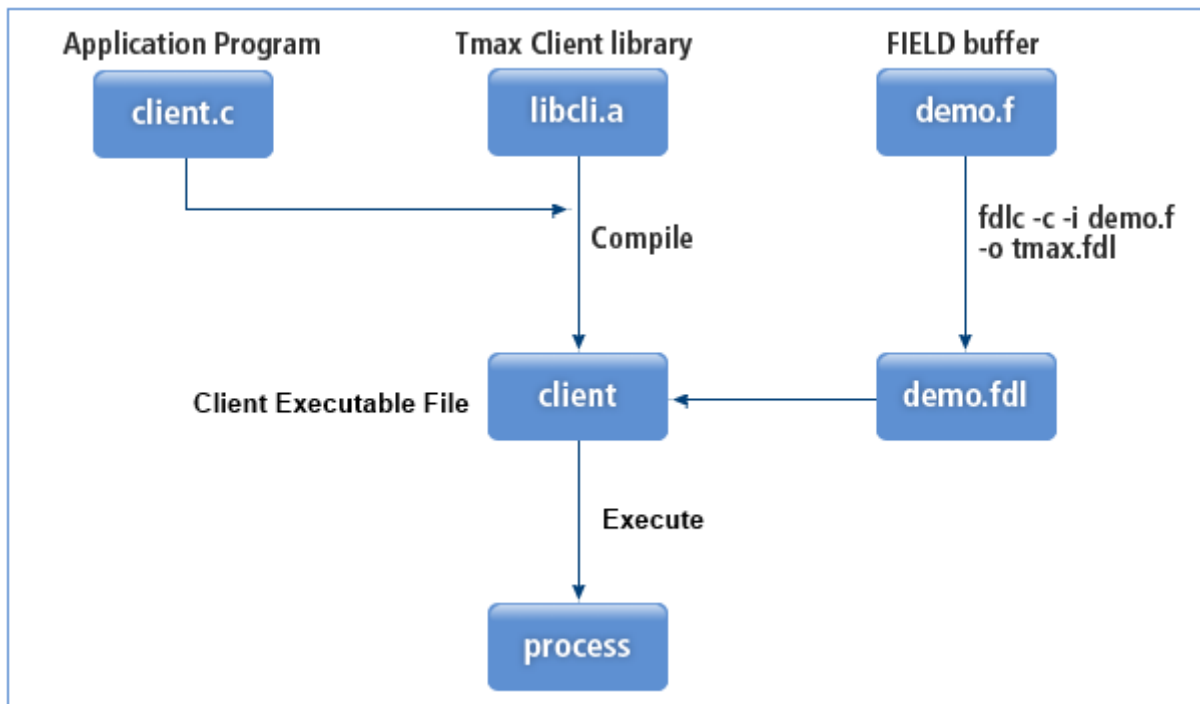
2.4. Compiling a Program

Compile a client program to create an object file.

The following shows the process of compiling a client program by using a struct buffer or field buffer.



Compiling a Client Program Using a Struct Buffer



Compiling a Client Program Using a Field Buffer

The following are the steps for compiling a client program.

1. Create a UNIX configuration file.

Set the environment variables needed to connect to Tmax (.profile, .login, and .cshrc).

```

TMAX_HOST_ADDR = Tmax address (=IP Address)
TMAX_HOST_PORT = Port number (default : 8888)
  
```

2. Write a client program (client.c).

Develop a program that requests a service from a server and receives a reply from the server.

3. Compile the client program with the libraries (libcli.a / libcli.so).
4. Compile the structure header file if you are using a struct buffer. A dummy structure is required if you are using a STRUCT, X_C_TYPE, or X_COMMON buffer. Compile the header file with sdlc to convert it to a standard communication type. The sdl file must be created to execute the client program.

If you are using a field buffer, compile the field key file with fdlc.

The following is an example of a Makefile for a Tmax client program.

```

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

TMAXLIBD = $(TMAXDIR)/lib

#TMAXLIBS is different according to OS.
  
```

```

#Solaris : TMAXLIBS = -lsocket -lnsl -lcli
#Compac, HP, IBM, Linux : TMAXLIBS= -lcli

TMAXLIBS = -lcli

#CFLAGS is different according to OS.
#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)

```

The sample program provided by Tmax executes "make" using the shell script named "compile". To use the shell script, type the following into a command prompt. In "compile", enter the client program name without the file extension.

```
compile c cli (=>client program name without its extension)
```

2.5. Starting and Terminating a Process

To connect to Tmax, you must set environment variables.

Because a client program gets necessary information from environment variables when connecting to Tmax, TMAX_HOST_ADDR and TMAX_HOST_PORT must be defined in the UNIX shell configuration file (.cshrc (C), .profile (korn), and .bash_profile (bash)). To handle potential failures, you can define TMAX_BACKUP_ADDR and TMAX_BACKUP_PORT. To handle potential network failures, you can define TMAX_CONNECT_TIMEOUT.

After Tmax environment variables are set in the UNIX configuration file, you can create client processes. You can start and terminate the client processes in the same way you start and terminate other executable files.

The following describes each environment variable.

Environment Variable	Description
TMAX_HOST_ADDR	<p>IP of the server where Tmax is installed. This variable is very important because connections are reset internally if the service requested by a client is not provided by the machine where the client connected to initially.</p> <p>If the client requests for a transaction processing, the Tmax server that was initially connected to takes over the transaction and oversees 2PC. Therefore, the host address must be set to the server with the most frequently requested services in order to decrease network traffic and reduce response time.</p>
TMAX_HOST_PORT	Port of the server where Tmax is installed. If TPORTNO is not defined in the Tmax configuration file, the default value (8888) is used.
TMAX_BACKUP_ADDR	IP of a backup server. When a client issues a request, a connection is established to a server specified by TMAX_HOST_ADDR. If the server is unavailable, another connection is established to the server specified by TMAX_BACKUP_ADDR. If TMAX_BACKUP_ADDR is not specified, the request fails.
TMAX_BACKUP_PORT	Tmax server port number specified by TMAX_BACKUP_ADDR.
TMAX_CONNECT_TIMEOUT	<p>This value is set to handle network failures. A client waits for a connection until this timeout period expires.</p> <p>If a connection is not established during this timeout period, tpstart() fails. If a network error occurs, tperrono is set to TPETIME.</p> <p>If TMAX_BACKUP_ADDR and TMAX_BACKUP_PORT are specified, tpstart() attempts to connect through the backup host for the timeout period.</p>
SDLFILE	Required for client programs using the struct buffer.
FDLFILE	Required for client programs using the field buffer.



For more information about the sdlc and fdlc commands, refer to *Tmax Reference Guide*.

2.5.1. sdlc

The **SDLFILE** environment variable must be set for a client program that uses the struct buffer. This variable must define the SDL file created by compiling a structure file used in a client program with sdlc. The client can communicate with Tmax server if the SDLFILE contains the structure file during structure communication.

The following command is used to compile the structure file.

```
$ sdlc -c -i Structure file name [ -o sdl file name] [ -h Header file name]
```

Option	Description
-i <i>structure file name</i>	Name of the structure file to compile. One or more file can be defined regardless of the extension. An asterisk (*) can be used as a wild card for any file name. In a structure file, one or more structures must be defined using the "structure name { . . . };" format.
[-o <i>sdl file name</i>]	Name of SDL file to create. It is set to the SDLFILE variable.
[-h <i>header file name</i>]	Creates the structure information in a header file format. If the [-h] option is omitted, <name_sdl.h> is created.

If [-o] option is omitted, the following default SDL file is created.

- If there is only one structure file, the created file name is <structure file name without extension.sdl>.

In the following case, the file is named <demo.sdl>.

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

- If there are multiple structure files, the created file name is <the first structure file name without extension.sdl>.

In the following case, the file is named <stra.sdl>.

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

If the current directory contains a.s, b.s, and c.s files, the file is named <a.sdl>.

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

2.5.2. fdlc

The **FDLFILE** environment variable must be set for a client program using the field buffer. This variable must define the FDL file created by compiling a field buffer file used in a client program with fdlc. The client can communicate with Tmax server if the FDLFILE contains the field information during field buffer communication.

The field buffer file can be compiled with the following command:

```
$ fdlc -c -i Field buffer file name [ -o fdl File name] [ -h Header file name]
```

Option	Description
<i>-i field buffer file name</i>	Name of a field buffer file to compile.
<i>[-o fdl file name]</i>	Name of FDL file to create. It is set to the FDLFILE variable. If the [-o] option is omitted, <txmax.fdl> is created by default.
<i>[-h header file name]</i>	Create the structure information into a header file format. If the [-h] option is omitted, <name_fdl.h> is created.

3. Server Programs

This chapter describes server program characteristics, components, flow, and development environment.

3.1. Characteristics and Components

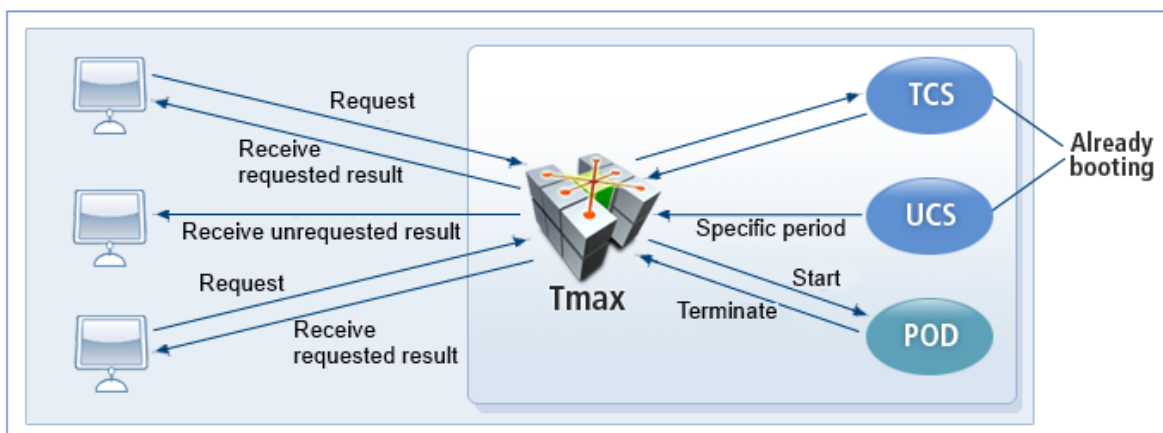
Server programs process user requests and return the results to the client.

Tmax server programs have the following characteristics:

- Program development is simple and quick because programmers only have to develop a service routine.
- A server is composed of one or more services. It can act as a client and issue service requests to another server. Also, a server can make another server manage clients by sending the clients' information and results.
- Because Tmax functions support communication and data conversion, developers do not need to develop the UNIX internal system and protocols. This allows developers to create programs without deep knowledge of network programming.
- In XA mode, developers can ignore database handling because the Tmax routine handles tasks related to connecting to and disconnecting from a database.

A Tmax application server program consists of `main()`, which is provided by Tmax, and a service routine you develop. Server programs are created using Tmax functions, which are provided in the form of a library file (`libsvr.a` and `libsvr.so`) and are compiled together with application server programs. The methods for developing and configuring server programs differ according to the processes supported in Tmax.

The following are the server processes supported in Tmax.



Tmax Server Processes

- Tmax Control Server (TCS)

Processes client requests.

- User Control Server (UCS)

TCS is a typical processing type, but UCS is an active processing type. UCS sends data to clients constantly without client requests.

- Realtime Data Processor (RDP)

An improvement of the UCS type process intended for special purposes.

3.1.1. TCS

TCS type server programs consists of the following components:

- Server program

A service routine written by developers. It is used to process client requests. If SQL statements are used, it must be precompiled using a vendor-specific tool.

- Tmax server library (libsvr.a / libsvr.so)

A library provided by Tmax. It includes server main(), tpsvrinit(), tpsvrdone(), and other Tmax functions.

- Service table

A file that lists service names provided by each server. It is created by referring to the Tmax configuration file. The service table is used to find the location of a service routine in a server when services are executed. It is provided by the system administrator.

The following are the steps for creating a service table.

1. Refer to the binary Tmax configuration file (for example, tmconfig) to use the `gst` (generate service table) command.

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

When you execute the `gst` command, a service table is created in the `svct` directory with the name "`server name_ svctab.c`" for each server registered in the `SERVER` clause of the Tmax configuration file. By referencing the `SERVICE` clause, the service table lists the services provided by each server.

2. The Tmax configuration binary file (tmconfig) is created by using the **cfl** command to compile the `sample.m` file written by the system administrator about the entire system.

```
$cfl [-i Tmax configuration file]
```



For more information about cfl and service tables, refer to *Tmax Administrator's Guide*.

- Struct binary table (SDLFILE)

If you want to use struct buffers (STRUCT, X_C_TYPE, and X_COMMON), a structure file in the <xxxx.s> form is required. There are two structure files: the standard communication type (structure file name_sdl.c) and the structure header file (structure file name_sdl.h).

Structure files are compiled with the **sdlc -c** command to create a binary table. The table is used to convert structure type data to/from the standard communication type data.

- Field buffer binary table (FDLFILE)

If you want to use field buffers, a field buffer file in the <xxxx.f> form is required.

Field buffer files are compiled with the **fdlc** command to create a binary table file and a header file. The binary table file matches a field key to data, while the header file (xxxx_fdl.h) matches a field key to a field key name. Unlike conventional structure files, you can specify which fields are to change in order to prevent resource waste. However, it might waste resources if you are using an insufficient number of fields because data values and field key values are managed together.

- Structure-standard buffer conversion/reversion program

To use a struct buffer in a server program, you must compile the structure-standard buffer conversion/reversion program (xxxx_sdl.c and xxxx_sdl.h) and link it to the server program. If you are not using a structure, link TMAXDIR/lib/sdl.o to the server program.

3.1.2. UCS

UCS type server programs consists of the following components:

- main() routine

Handles database connections, disconnections, and command line options.

- Service routine

Processes client requests.

The following is how a service routine is declared.

```
<Service Name>(TPSVCINFO *msg)
```


Item	Description
Service Name	Up to 63 characters including the NULL character. Do not start with an underscore (_) because names starting with an underscore are internally defined in Tmax.
msg	A service routine receives the TPSVCINFO structure as a parameter and handles the corresponding tasks.

- **usermain() routine**

The `usermain()` routine handles its own tasks when there are no special messages such as a service request or control message. At a specific time, the routine receives commands to control or to execute a service from TMM or CLH. The routine is generally written as an infinite loop because when `usermain()` ends, the corresponding server process also ends. In the loop, you must use one or more scheduling APIs to handle client service requests or TMM and CLH control messages.

`usermain()` can receive command line parameters. These parameters are set in CLOPT of the SERVER clause in the Tmax system environment file. The parameters are the same as the values passed to `tpsvrinit()`. When `usermain()` ends, the `main()` routine executes the `tpsvrdone()` routine and is terminated.

```
int usermain(int argc, char *argv[])
{
    .....
    while(1) {
        .....
        tpschedule(-1);
        .....
    }
    return 0;
}
```

When UCS type server programs do not have a service routine, server processes have similar characteristics as daemon processes.

The following is how a TOUPPER service routine is declared.

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



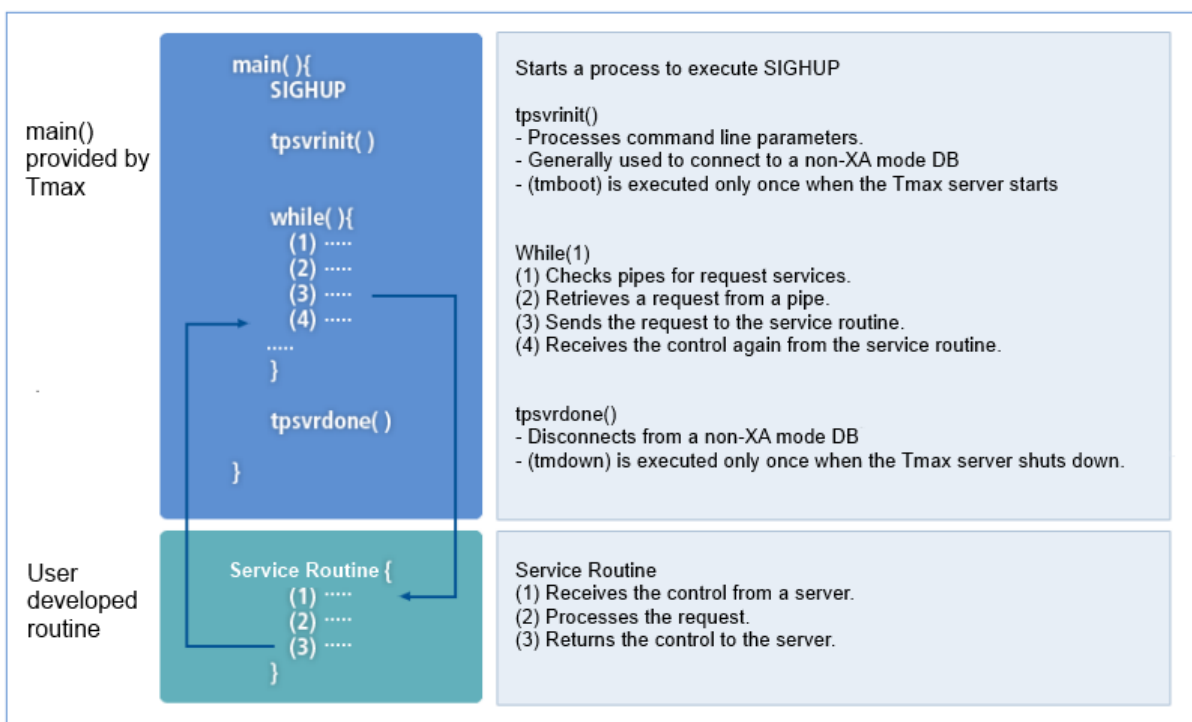
For more information about the SERVER section, refer to *Tmax Administrator's Guide*.

3.2. Development Environments and Tools

Tmax application server programs can run on all UNIX environments, independent of hardware. You can use vi, the UNIX standard editor, to develop server programs.

3.3. Program Flow

The flow of TCS type Tmax server programs is different from the flow of UCS type Tmax server programs. For TCS type server programs, which are the general server programs, all tasks are handled in the service routine. But, for UCS type server programs, various services, which are difficult to implement conventionally, can be provided by using the service routine and `usermain()`. A server program is divided into the **main() routine** provided by Tmax and the **service routine** developed by users.



Server Program's Flow

- **main() routine**

The `main()` routine processes command line arguments. It connects to or disconnects from the database, allocates buffers that are used in the service routine, and calls the service routines that process client requests.

- **Service routine**

The service routine processes client requests. Client requests are first received by the `TPSVCINFO` structure that holds request-related information in the `main()` program. The buffer requested by clients resides in the `TPSVCINFO → DATA` structure. The data from the `TPSVCINFO → DATA` structure is accepted and processed by service routines. After the data is processed, a service routine sends the results to the client program or issues a service processing request to other servers.

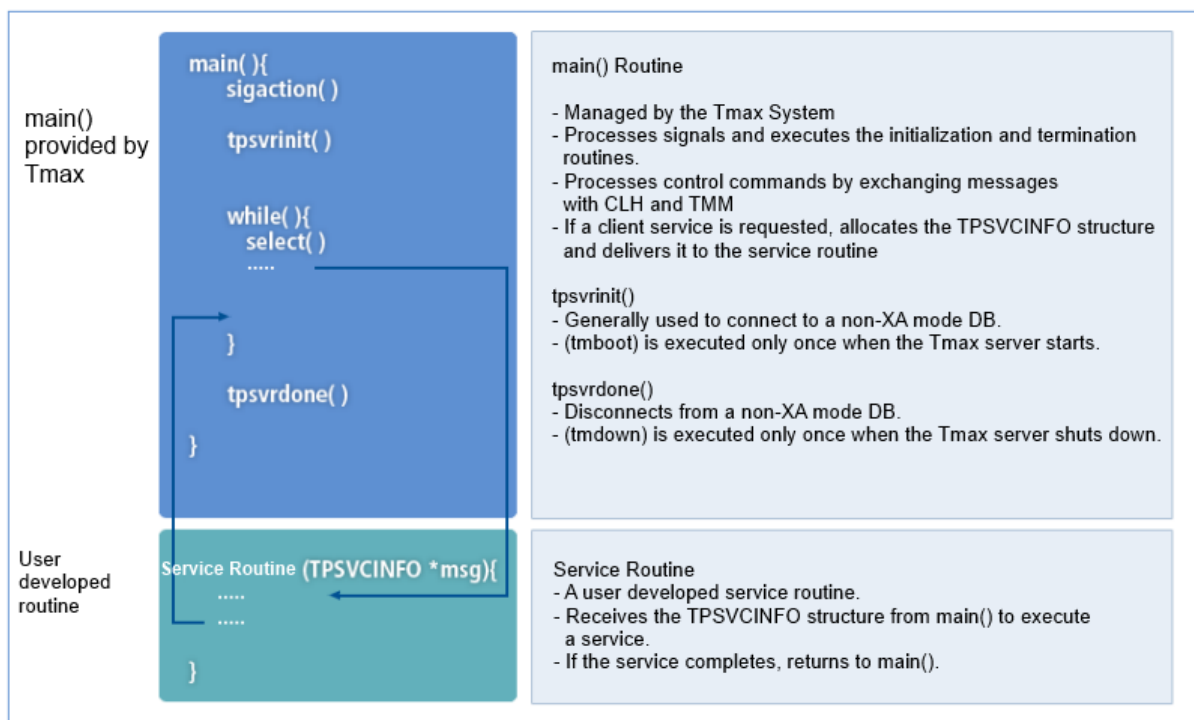


For more information about functions used in a service routine, refer to *Tmax Reference Guide* or *Tmax Programming Guide (UCS)*.

3.3.1. TCS

TCS has the typical three-tier server process. Tmax system receives requested results and returns the results to clients by scheduling client requests to the appropriate process. Developers only have to develop service routines because Tmax provides `main()` for TCS type service programs. TCS type programs are developed using specified service names as functions.

The following shows a `main()` routine and a service routine in TCS.



TCS type Server Program's Flow

- **main() routine**

A command line is used to enter the `main()` routine arguments. It connects to or disconnects from the database, allocates and manages TPSVCINFO struct buffers used in the service routine, waits for messages sent by CLH and TMM, and calls the service routines that process client requests.

- **Service routine**

The service routine processes client requests. It receives the TPSVCINFO structure, which includes service requests of clients and necessary information from `main()`, and returns the results to clients.

The following shows a TCS type program.

```

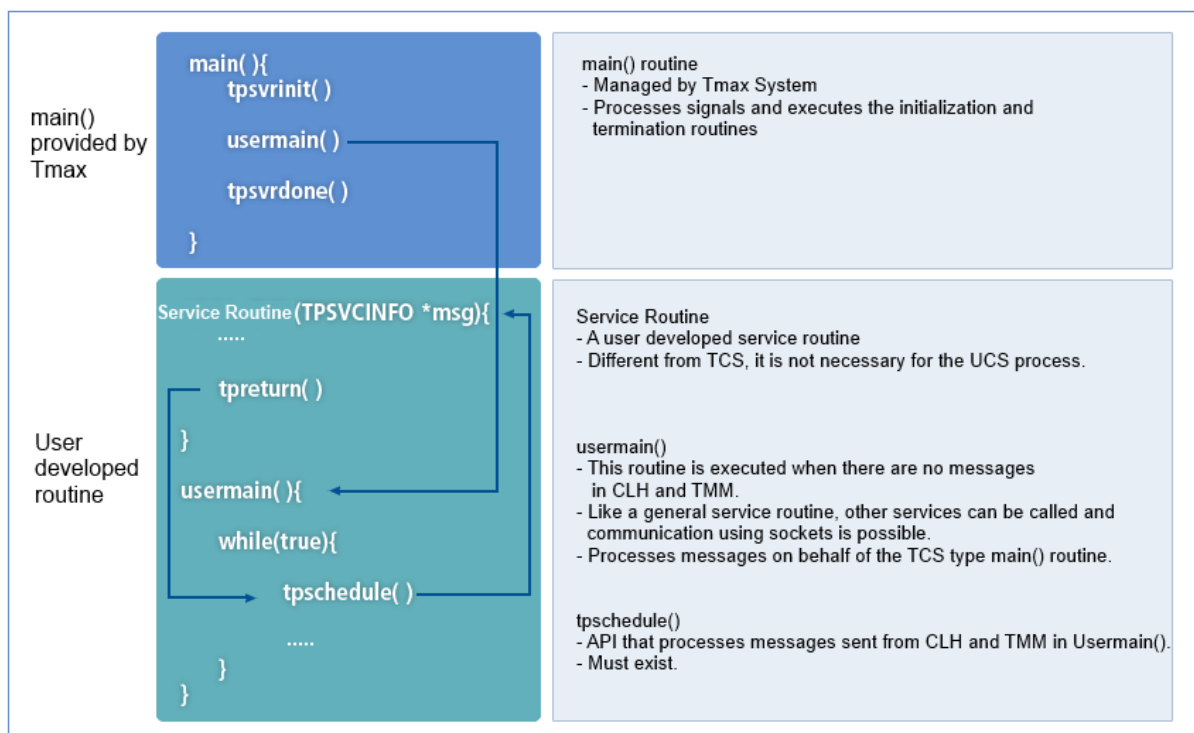
SDLTOUPPER(TPSVCINFO *msg)
{
    ...
    struct kstrdata *stdata;
    stdata = (struct kstrdata *)msg->data;
    ...
    for (i = 0; i < stdata->len; i++)
        stdata->sdata[i] = toupper(stdata->sdata[i]);
    ...
    tpreturn(TPSUCCESS,0,(char *)stdata, sizeof(struct kstrdata), TPNOFLAGS);
}

```

3.3.2. UCS

Compared to TCS, which only processes requests from clients, UCS can provide services without client requests. UCS supports TCS to quickly provide services with more features. A UCS program consists of `usermain()` and a service routine.

The following show a `main()` routine and a service routine in UCS.



UCS Type Server Program's Flow

- **main() routine**

The UCS type `main()` routine is similar to the TCS type `main()` routine, but UCS type `main()` routine has an additional `usermain()` routine.

- **Service routine**

The UCS type service routine is the same as the TCS type service routine. UCS type processes must be linked to the UCS library (`libsvrucs.a`).

- usermain() routine

The UCS type usermain() routine independently processes repeated tasks when there are no other commands. When CLH or TMM sends commands, they are processed by using scheduling API. To terminate UCS type processes with the tmdown command, you must use **tpschedule()** in the loop.

The following shows a usermain() function.

```
...
int usermain(int argc, char *argv[])
{
    ...
    while (1) /* infinte loop */
    {
        clid = tpgetcli();
        ret = tpsendtocli(clid, sndbuf, slen, TPNOFLAGS);
        if (ret < 0)
        {
            error processing routine
        }
        ...
        tpschedule(10); /* UCS process must add this function in a while loop.*/
        /* If tmdown is called, an event is sent to here. */
        ...
    } /* end of while */
}
```

- usermain() handles tasks in a loop. usermain() uses tpsendtocli() to send data to clients and uses tpschedule() to process TCS type services.
- tpschedule() is used only by a UCS server processes.

The function has a timeout parameter, which specifies how long in seconds the function is to await data. Upon receiving the data, the function returns it to the client. If the data does not arrive within time, the function stops waiting.

If the parameter is set to -1, tpschedule() checks for returned data. If no data is received, tpschedule() proceeds to the next step. If the parameter is set to 0, tpschedule() waits for client requests.

When data arrives, the required services are executed automatically and tpschedule() is returned. Therefore, users cannot execute services arbitrarily after data arrives. Services are always carried out by the system, which applies to UCS type service programs.

- If usermain() calls services in asynchronous mode (tpacall), tpregcb() must be used to receive the response.

The following is an example of the Tmax configuration file for using UCS type processes.

```
*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

#Add SVRTYPE=UCS for UCS.
ucssvr1   SVGNAME = svg1, SVRTYPE = UCS
ucssvr2   SVGNAME = svg1, SVRTYPE = UCS
ucssvr3   SVGNAME = svg1, SVRTYPE = UCS

*SERVICE
TOUPPER   SVRNAME = svr1
TOLOWER   SVRNAME = svr1
DUMY1     SVRNAME = ucssvr1
DUMY2     SVRNAME = ucssvr2

```

The following is an example of a UCS program.

```

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

DUMY1(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 is a UCS server process enhanced at the kernel level to send multiple client small, fast-changing data more efficiently and quickly. RDP shows remarkably low process occupancy rate and processing speed when the server sends small data to many clients frequently (more than 10 times in a second). However, each node can have one RDP server. The RDP server differs from the UCS only in terms of the settings in the configuration file and the library used for compilation.

The following is an example of the Tmax configuration file for enabling an RDP process.

```
*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
```



For more information about RDP, refer to *Tmax Administrator's Guide* or *Tmax Programming Guide (UCS)*.

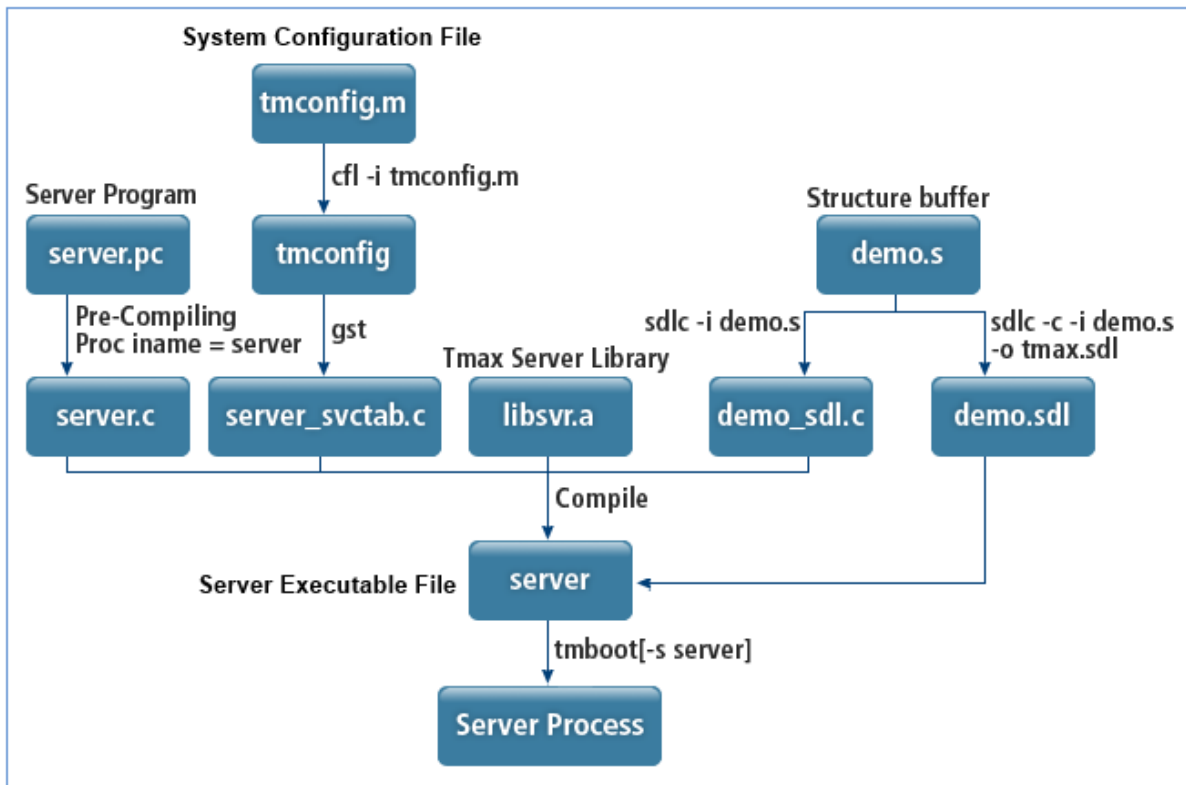
3.4. Compiling a Program

After you write a server program, it must be compiled to create a binary file. For compilation, you need the server program, Tmax server library, and the service table. If a struct buffer is used, you need a structure-standard buffer conversion/reversion program and a structure binary table. If a field buffer is used, you need a field header file and a field buffer binary table. TCS and USC server programs are compiled differently.

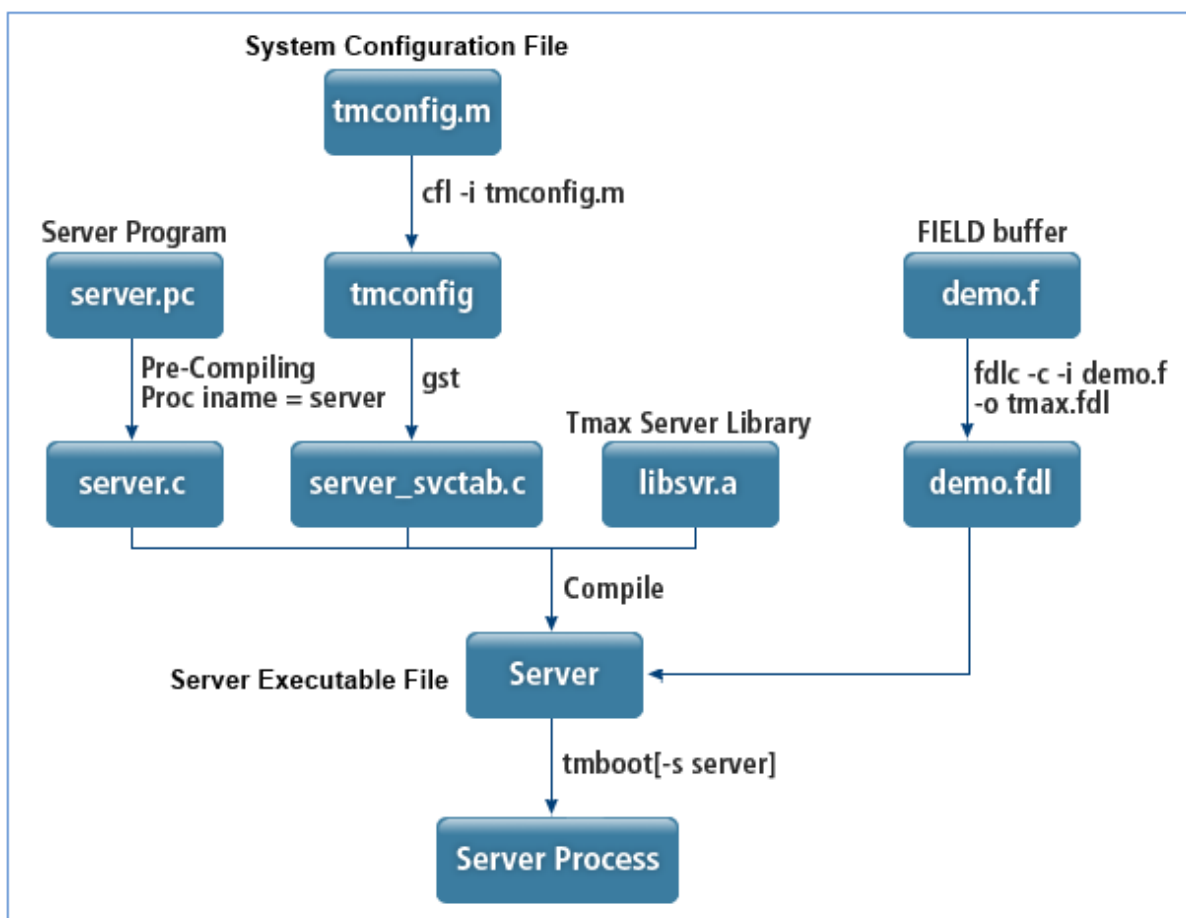
3.4.1. Compiling a TCS Server Program

In general, you can compile a C program by using the shell installed with Tmax or the mksvr utility.

The following shows the process of compiling a server program using a struct buffer or field buffer.



Compiling a Server Program Using a Struct buffer



Compiling a Server Program Using a Field Buffer

The following are the steps for compiling a TCS server program in Linux. Each development environment (32 or 64 bit) and platform requires different flags and libraries.

1. Compile the server program to create an object file.

The server program must include the header files provided by Tmax. If necessary, the program must include the structure file or field buffer header files. If the program contains SQL statements, the program must be precompiled with a tool provided by the corresponding database vendor.

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

2. Compile the structure file if a struct buffer is used.

Compilation involves two phases. First, use `sdlc` to create the structure-standard buffer conversion/reversion program. Second, compile the program to create an object file. If a structure file is not used, use **TMAXDIR/lib/sdl.o**.

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

3. Compile the service table provided by the system administrator to create an object file.

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

4. Link the object files created in the previous steps to the server library provided by Tmax to create an executable server program.

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

Using a Shell program to compile a server program

When Tmax is installed, sample programs, four Makefiles, and a shell program (Makefile.c, Makefile.sdl, Makefile.pc, Makefile.psdl, and compile) are created in the sample server program directory.

The following is a Makefile (Makefile.sdl) of a Tmax server program that uses a struct buffer but not a database. The name of the program is included in `$COMP_TARGET`, which is passed when a shell program compiler is used.

```
# Server makefile  
  
TARGET = $(COMP_TARGET)  
APOBJS = $(TARGET).o  
SDLFILE = demo.s  
  
#Not use Db  
LIBS = -lsvr -lnodb  
#In the case of Solaris, -lsocket -lnsl is added.
```

```

#In the case of Oracle, Informix, Db2, and Sybase,
#-loras, -linfs, -ldb2, and -lsybs are used instead of -lnodb, respectively.

OBSJ    = $(APOBSJ) $(SDLOBJ) $(SVCTOBJ)

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

CFLAGS  = -O -I$(TMAXDIR)
#Different CFLAG is used according to development environments (32 or 64 bit) and platforms.
#Solaris 32bit, Compaq, Linux: CFLAGS = -O -I$(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
#In the 64 bit environment, it is $(TMAXDIR)/lib64.

#
.SUFFIXES : .c

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

#
# server compile
#

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

$(APOBSJ): $(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)

```

The following is an example of using a Makefile. svr1, svr2, svr3, fdlttest, and sdltest are sample programs created during Tmax installation. For more information, refer to the each Makefile.

```

$./compile sdl svr1

```

```
./compile c svr2
./compile c svr3
./compile pc fdlttest
./compile psdl sdltest
```

Using the mksvr utility to compile a server program

If you are using the mksvr utility to compile a server program, you do not have to specify the service name when configuring the Tmax system configuration file or create a service table with the gst command. However, if a structure is used, you must create the structure-standard buffer conversion/reversion program. To connect to a database, you must use a pre-compiled file or specify an RM file.

```
$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 FDLTOpper, FDLTOlower -o svr3 -f svr3.c

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

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



For more information about mksvr, refer to *Tmax Reference Guide*.

3.4.2. UCS

UCS server programs are compiled like TCS server programs. However, USC uses the libsvrucs.a (or libsvrucs.so) server library while TCS uses libsvr.a (or libsvr.so). A USC server program, like a TCS server program, can be compiled with a general C compiler, which can be used through the shell program created when Tmax is installed or the mksvr utility.

The following are the steps for compiling a UCS type server program in Linux. Each development environment (32 or 64 bit) and platform requires different flags and libraries. Because of the similarities in the compilation processes of UCS and TCS, refer to [Compiling a TCS Server Program](#) for more information.

1. Compile the server program to create an object file.

The server program must include the header files provided by Tmax. If necessary, the program

must include the structure file or field buffer header files. If the program contains SQL statements, the program must be precompiled with a tool provided by the corresponding database vendor.

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

2. Compile the structure file if a struct buffer is used.

Compilation involves two phases. First, compile with `sdlc` to create the structure-standard buffer conversion/reversion program. Second, compile the program to create an object file. If a structure file is not used, use **TMAXDIR/lib/sdl.o**.

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

3. Compile the service table provided by the system administrator to create an object file.

The following is an example of when a database is not used. If a database is used, `liboras.so(a)`, `libinfs.so(a)`, `libdb2.so(a)`, or `libsybs.so(a)` is used (depending on the database). If a shell program is used, change **-lsvr** to **-lsvrucs** in the corresponding Makefile. The usage is the same.

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

Using the `mksvr` utility to compile a server program

If the `mksvr` utility is used, service names are not required when writing the Tmax system configuration file and a service table created through `gst` is also not required. However, if a structure is used, the structure-standard buffer conversion/reversion program must be created. If a database is used, a precompiled file must be used or an RM file must be specified.

```
$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
```



For more information about `mksvr`, refer to *Tmax Reference Guide*.

3.5. Creating and Terminating a Process

The system administrator is responsible for creating server processes. Unlike a UNIX executable file, the Tmax application server does not start automatically because a server process references the Tmax configuration file when the server process is created. Therefore, you must create a server

process by using the **tmboot** command and terminate the server process by using the **tmdown** command.

Before creating a Tmax application server process, the Tmax configuration file (for example, sample.m) written by the system administrator must be compiled by using the cfl command. The server process is created by referring to the compiled binary configuration file (for example, tmconfig).

The following are commands used to create and terminate a process.

- Creating a process

- The following command creates the Tmax system process and all server processes registered in the binary Tmax configuration file.

```
$tmboot [-f Binary configuration file]
```

- The following command creates specific server processes.

```
$tmboot [-s Server program name]
```

- Terminating a process

- The following command terminates the Tmax system process and all server processes registered in the binary Tmax configuration file.

```
$tmdown [-f Binary configuration file]
```

- The following command terminates specific server processes.

```
$tmdown [-s Server program name]
```



For more information about commands, refer to *Tmax Reference Guide*.

4. Communication Mode

This chapter describes the communication modes supported by Tmax.

4.1. Overview

Tmax supports three client-to-server communication modes: **synchronous**, **asynchronous**, and **interactive**.

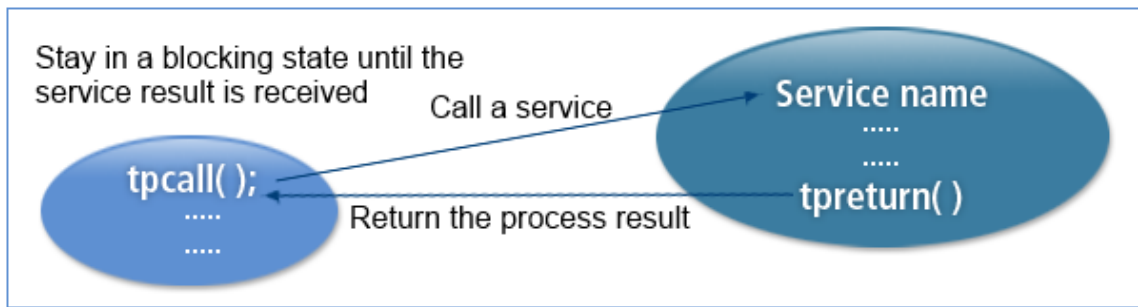
- Synchronous/Asynchronous communication
 - Synchronous communication is most widely used. During synchronous communication, the client sends one request and awaits a reply one at a time, and the server similarly processes one request one at a time before moving onto another.
 - Services are requested by `tpcall()` and `tpacall()`. The transmit/receive buffer must be allocated by `tpalloc()` before data is transmitted. The service routine on the server side is completed by `tpreturn()` or `tpforward()`.
 - In synchronous mode, a client sends a request to the server by calling `tpcall()` and waits in the blocking status for the response from the server.

In asynchronous mode, a client issues a request to the server by calling `tpcall()`. The client conducts other tasks without waiting for the response from the server. When the client is ready to receive the response from the server, the client calls `tpgetrply()` and receives the response from the server.

- Interactive communication
 - Unlike synchronous or asynchronous mode, server programs must be developed only with interactive service functions. Interactive mode is rarely used because of its long connection maintenance time.
 - Service requests are initialized by `tpconnect()`. Data is sent by using `tpsend()` and received by using `tprecv()`. The transmit/receive buffer must be allocated by `tpalloc()` before data is sent.
 - In interactive mode, you cannot use `tpforward()`.

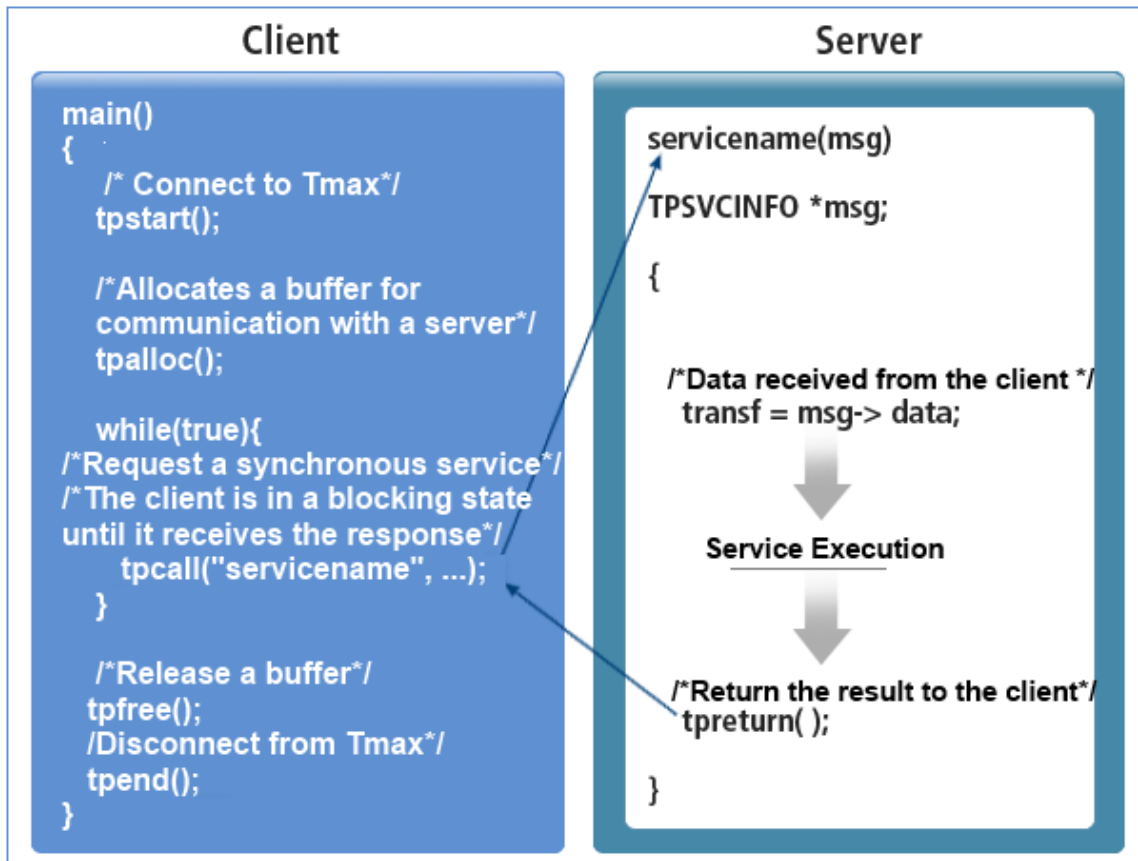
4.2. Synchronous Communication

In synchronous mode, a client requests a service and waits for a response in the blocking status.



Synchronous Communication

The following is how the client and the server communicate in synchronous mode.



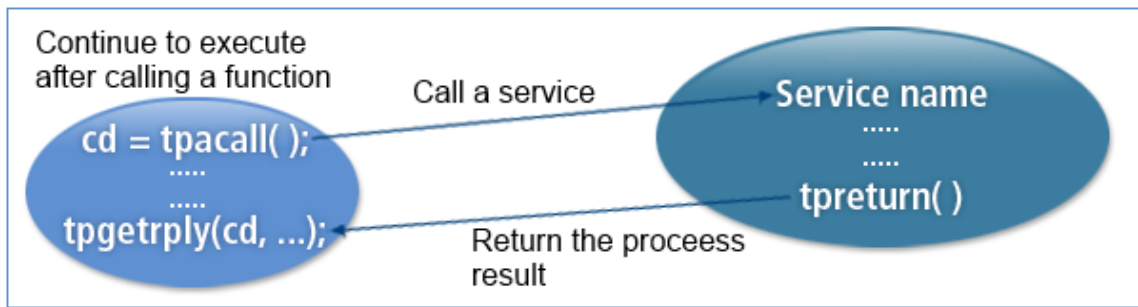
Synchronous Mode



For more information about functions related to synchronous communication, refer to [Synchronous Communication](#) or *Tmax Reference Guide*.

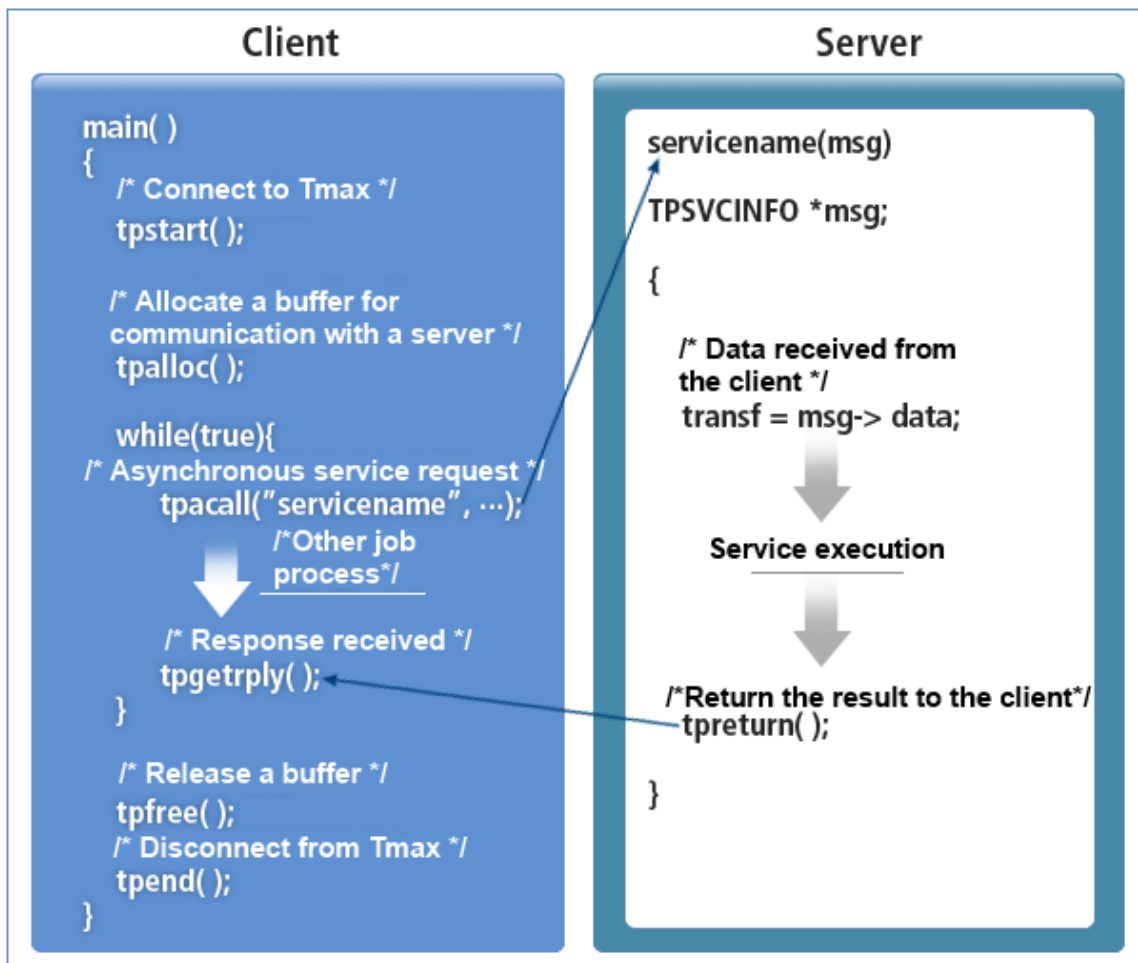
4.3. Asynchronous Communication

In asynchronous mode, the client can request a service and proceed with another task without awaiting a reply. This is because `tpacall()` immediately returns after it is called. However, `tpgetrply()`, which is called to receive a response, waits in the blocking status until a reply arrives or timeout occurs.



Asynchronous Communication

The following is how the client and the server communicate in asynchronous mode.



Asynchronous Mode



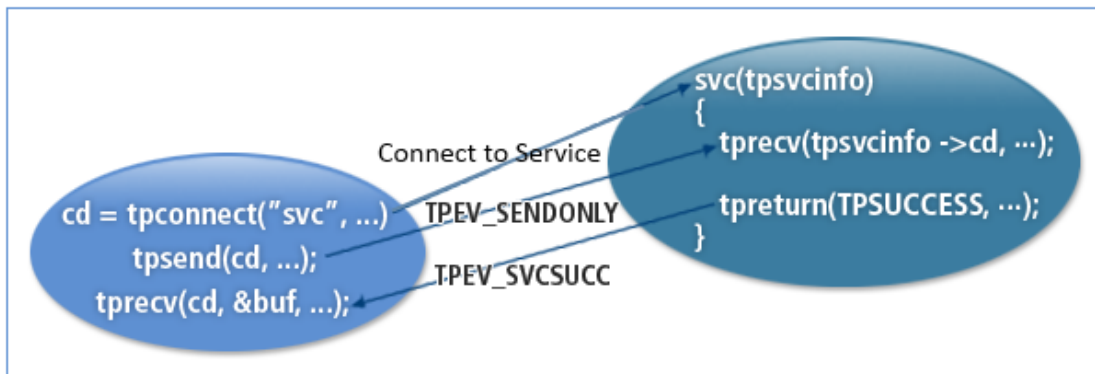
For more information about functions related to asynchronous communication, refer to [Asynchronous Communication](#) or *Tmax Reference Guide*.

4.4. Interactive Communication

During interactive communication, a half-duplex connection is established between the client and server, where messages are sent in only one direction. When a client initially connects to a server, communication control is given to either the client or the server. The side with control can send a

message while the other side can receive a message. Control can be passed by using a flag from the side with control. The other side learns of control change through an event value.

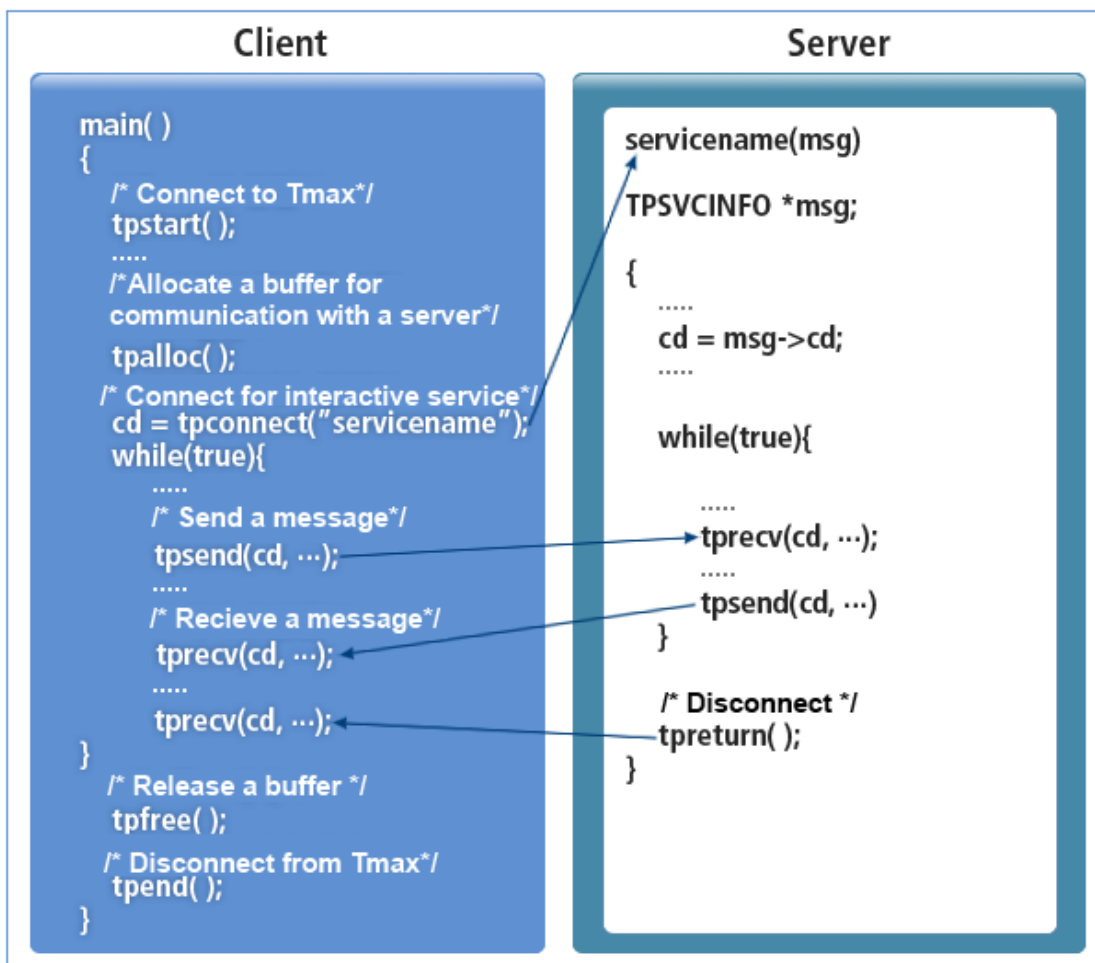
The following is the process of passing communication control and disconnecting a connection.



Interactive Communication

In interactive mode, `tpconnect()` is used to make a connection, `tpsend()` is used to send data, and `tprecv()` is used to receive data. To send data, the client or the server must have connection control, which operates by setting a flag value in `tpconnect()` and `tpsend()`.

The following is how the client and the server communicate in interactive mode.



Interactive Mode

- Interactive communication is initiated by calling **tpconnect()**, which selectively sends data to services and determines which program holds control. If **tpconnect()** is executed successfully, it returns the connection descriptor (cd), which distinguishes message transmissions and receptions from those for other connections.
- Interactive communication is used when additional information is required depending on context. However, compared to other communication modes, an interactive connection stays connected for a long period of time: from the point of connection (**tpconnect()**) to the point of disconnection (**tpdiscon()** or **tpreturn()**). Synchronous/asynchronous communication is recommended because an interactive connection has greater network load.
- **tpreturn()** is generally called in the server to terminate an interactive communication. In this case, the receiving side receives both the data and the **TPEV_SVCSUCC** event. If the control side calls **tpdiscon()**, interactive communication is forcibly terminated. When the function is called, data being transmitted may be lost and transactions being processed are rolled back.
- Only the program that has communication control can give the control to the counterpart using the **TPRECVONLY** flag in **tpsend()**. The counterpart receives data and the **TPEV_SENDOONLY** event and takes control.
- To pass the control to the counterpart, set the flags parameter to **TPRECVONLY** and call **tpconnect()** or **tpsend()**. The **TPEV_SENDOONLY** event occurs to the counterpart to let the counterpart know control was received. After control is passed, data can be sent by calling **tpsend()**.
- Like synchronous or asynchronous communication, interactive mode uses buffers, which must be allocated in advance by **tpalloc()**. All buffers used in Tmax communication must be allocated by **tpalloc()**.



For more information about functions related to interactive communication, refer to [Interactive Communication](#) or *Tmax Reference Guide*.

4.4.1. Events Related to Interactive Communication

The following lists the five events of interactive communication.

Event	Receiving Function	Description
TPEV_SENDOONLY (0x0020)	tprecv()	Indicates the location of the connection control.
TPEV_DISCONIMM (0x0001)	tesend() tprecv() tpreturn()	Abnormal disconnection, which occurs when tpdiscon() is called or tpreturn() is called while sub-services are still open.

Event	Receiving Function	Description
TPEV_SVCERR (0x0002)	tpsend() tprecv()	Service function error, which occurs when the receiver has called tpreturn() without having control or when the argument of tpreturn() is valid but an error is generated.
TPEV_SVCFAIL (0x0004)	tpsend() tprecv()	Service function failed, which occurs when tpreturn() is called by a party that does not have control of the connection or tpreturn() is called with its flags set to TPFAIL or TPEXIT.
TPEV_SVCSUCC (0x0008)	tprecv()	Service function successfully completed. tpreturn() is called with the flag set to TPSUCCES.



For more information about functions, refer to *Tmax Reference Guide*.

5. Buffer Types

This chapter describes the buffer types supported in Tmax.

5.1. Overview

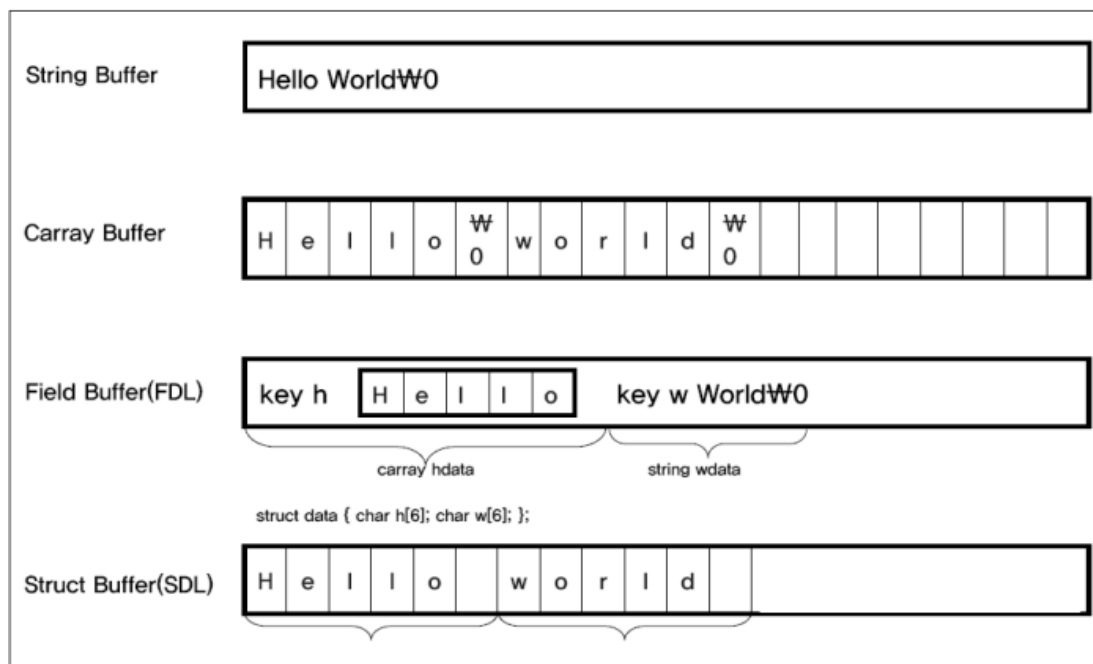
If you are using multiple servers with different hardware devices and operating systems, you encounter an issue when you need to transfer data among the servers: you need in-depth knowledge about all the operating systems and data conversion process.

To avoid this problem, the Tmax system supports various types of standard buffers. The buffers bring the following benefits:

- Developing flexible business programs using various buffers.
- Saving development time that could be otherwise spent on data conversion.
- Suffering less network overhead that arises from converting data into character strings. With the Tmax buffers, you can send and receive the data whose type is specified before a transaction begins.

5.2. Buffer Types

To decrease network overhead and enhance the development process, the following buffers are supported.



Tmax Communication Buffer Types

- String buffer

Used for strings that end with NULL. The buffer length is not required. This buffer is platform-neutral.

- CARRAY and X_OCTET buffers

Used for strings with a specified data transmission length (usually binary data). This buffer is platform-neutral.

- Struct and X_C_TYPE buffers

Used for structure data in C (language). This buffer can have all primitive types, declared structs, and arrayed of structs as members.

- X_COMMON buffer

A C program structure that can have only char, int, and long as members.

- Field buffer

Used for data with a field key. This buffer can have all primitive types as members. It is used to pass flexible data types because it supports various data access methods and conversion APIs.

5.3. Managing a Buffer

To guarantee buffer integrity, a **standard communication buffer** is used, which sets the standards for the data type and memory allocation. Before it is sent, data is converted to the standard communication format, and before it is received, the data is converted back into the original format. In this way, structure and string buffers are passed for communication between heterogeneous machines.

Tmax uses **sdlc** to convert data to the standard format. On the client side, the sdc program generates a conversion information table. On the server side, the sdc program creates a conversion/reversion program for standard communication. The client encrypts data to a standard recognized by the server. A program on the server decrypts the data and then encrypts the result to the client in the standard type. Throughout this process, struct buffers or field buffers that have string or primitive data types can be used without any constraints.

The STRUCT, X_C_TYPE, and X_COMMON struct buffers must be converted to standard communication types and added during compilation. STRING, CARRAY, and X_OCTET buffers do not need to be converted to standard buffer types because they allow string type communication between heterogeneous machines.



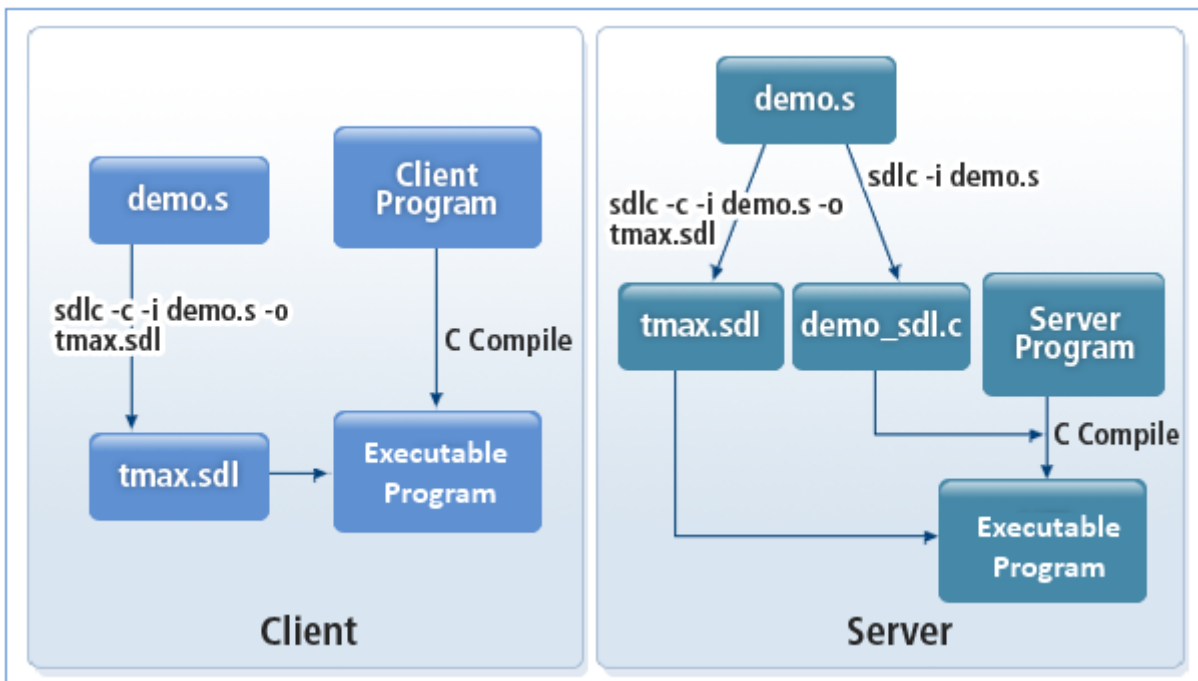
For more information about buffers and related functions, refer to [Buffer Management](#) or *Tmax Reference Guide*.

5.3.1. Struct Buffers

Developers write the structures they want to use in a struct file (xxx.s), which is referenced to create the conversion table and conversion program. A struct file cannot include commands such as `typedef` and `include`. All primitive and structure types can be members of a struct.

To use the struct buffer, you must use the **sdlc** program to create a conversion table and the conversion/reversion program that convert the struct buffer to and from the standard communication buffer. The information table is automatically used when the structure is used, and the conversion/reversion program is compiled and used together with the service routine.

The following is the process of compiling a server/client program when developing an application that uses a struct buffer (STRUCT, X_C_TYPE, and X_COMMON). demo.s is the struct buffer.

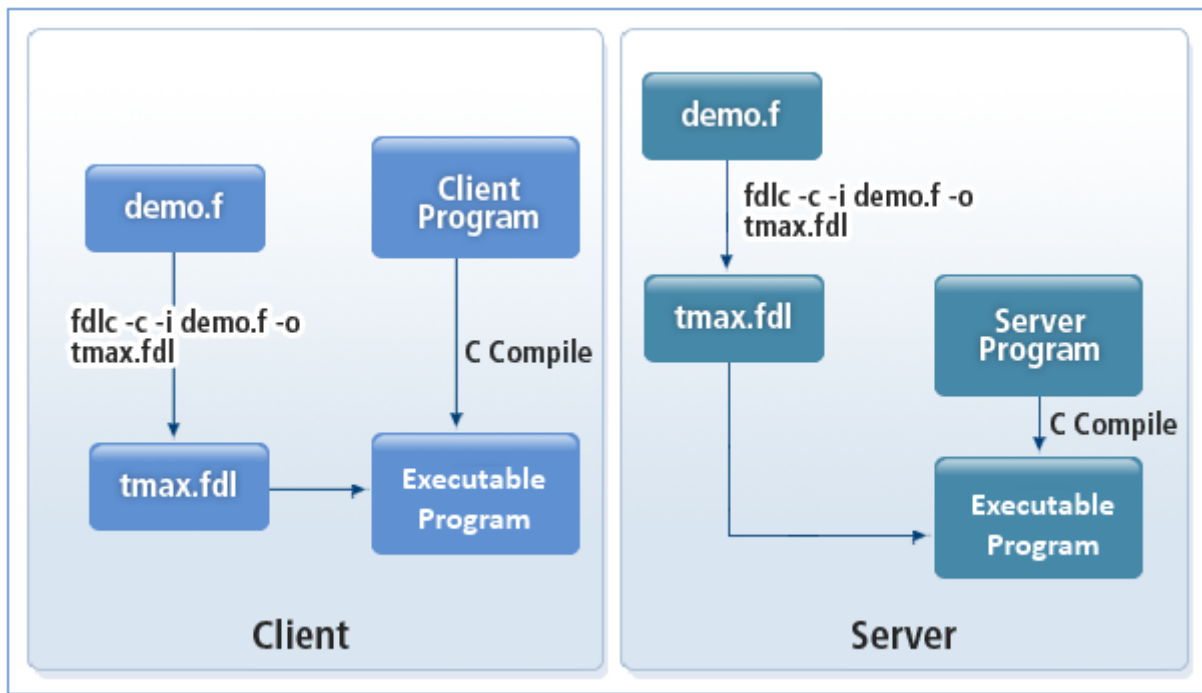


Compiling an Application Program that Uses a Struct Buffer

5.3.2. Field Buffers

Developers write field buffers they want to use in a field buffer file (xxx.f), which is referenced to create the conversion table. To convert a field buffer to the standard communication format, you must use the **fdlc** program to create the conversion table, which is automatically used by the system when the field buffer is used.

The following is the process of compiling a server/client program when developing an application that uses a field buffer. demo.f is the field buffer.



Compiling an Application Program that Uses a Field Buffer

6. Transactions

This chapter outlines the concept and process of a transaction and describes some transaction errors.

6.1. Overview

A transaction is a unit of work that changes a resource from a valid state to another.

Tmax supports local and global transactions.

- Local Transaction

A local transaction is a transaction that involves one resource manager (database).

- Global Transaction

A global transaction involves a logical unit consisting of more than one resource manager (database) and more than one physical site. All transactions in the Tmax system are handled as global transactions.

Transaction processing in the Tmax system preserves ACID (Atomicity, Consistency, Isolation, and Durability) properties and implements the TX standard of the X/Open DTP model.

The following are the ACID properties.

- Atomicity

A transaction is either completed or not (all or nothing).

- Consistency

A completed transaction changes a shared resource from one valid state to another.

- Isolation

The execution result of a transaction that involves a shared resource is visible only after it is committed.

- Durability

The results of committed transactions are permanent even if a system or a media failure occurs.

6.2. Distributed Transaction

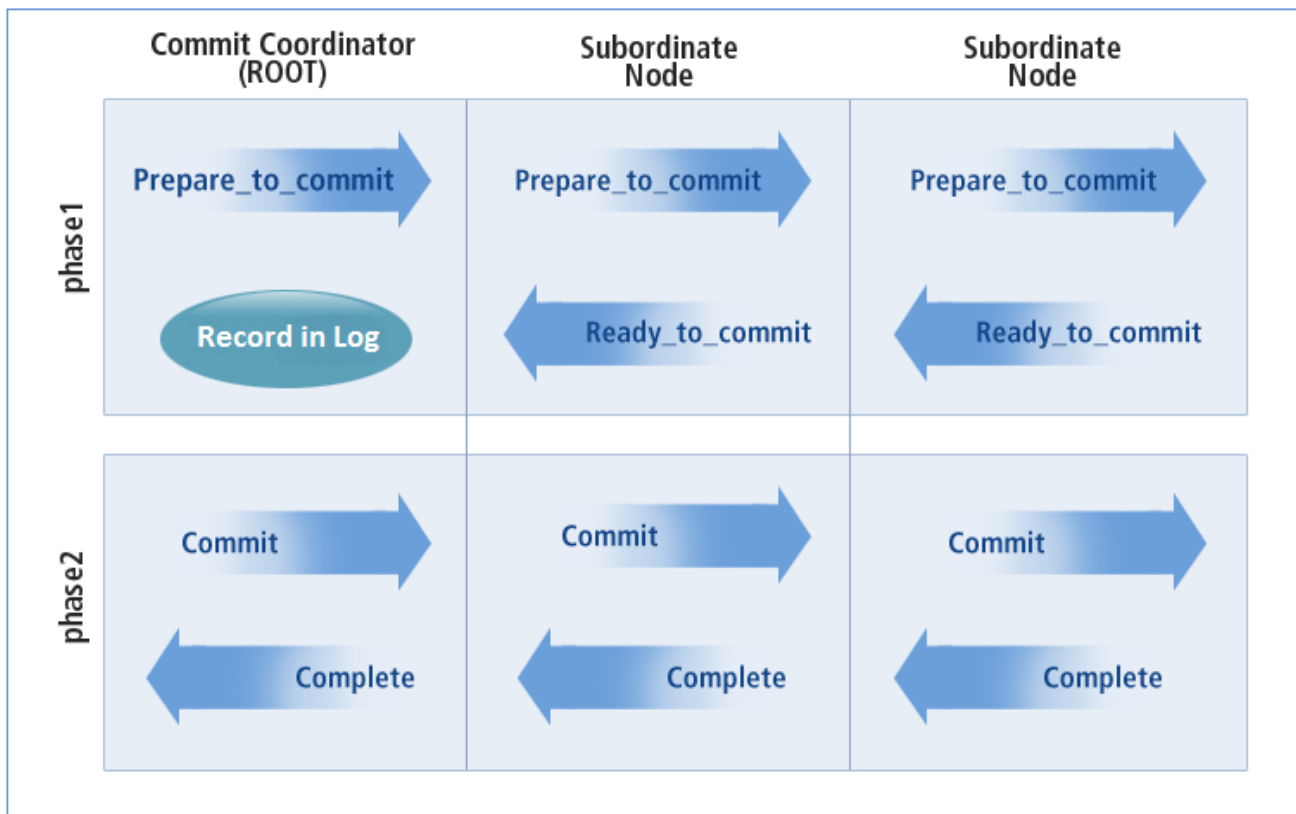
Global transactions that involve two or more databases use the two-phase commit (2PC) protocol to guarantee ACID properties of the transaction. The following describes the two phases of 2PC.

- Prepare Phase

The commit manager checks whether the databases participating in the transaction are ready to commit. When all databases are ready, a ready signal is sent.

- Commit Phase

When all databases send a ready signal, the transaction is committed. If a database fails to send a ready signal, the transaction is rolled back.



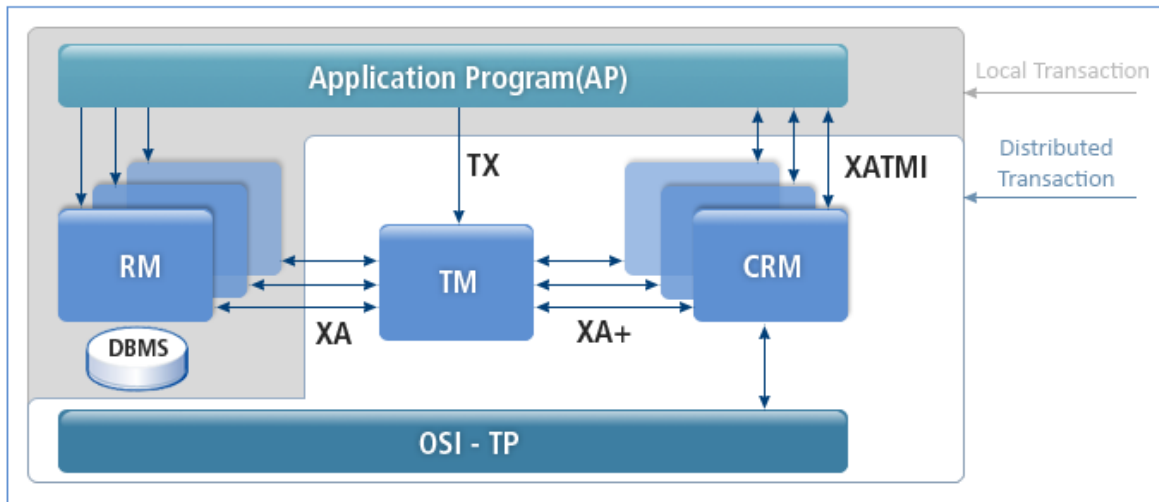
Two-phase Commit (2PC) Protocol

During a distributed transaction processing, the 2PC protocol ensures data integrity. Also, a number of functions such as tx_begin, tx_commit, and tx_rollback enable global transaction processing. In addition, a multi-thread transaction manager ensures improved resource efficiency and dynamic error logging. Dynamic error logging enables a quick response to errors and supports reliability through recovery and rollback. All transactions are monitored through a single interface to facilitate easy scheduling and management of transactions.

Processing Mechanism

The Tmax system uses a set of standard functions, the Application-to-Transaction Monitor Interface (ATMI), which implements the X/Open distributed transaction model of transaction processing.

The following is the of X/Open DTP structure.



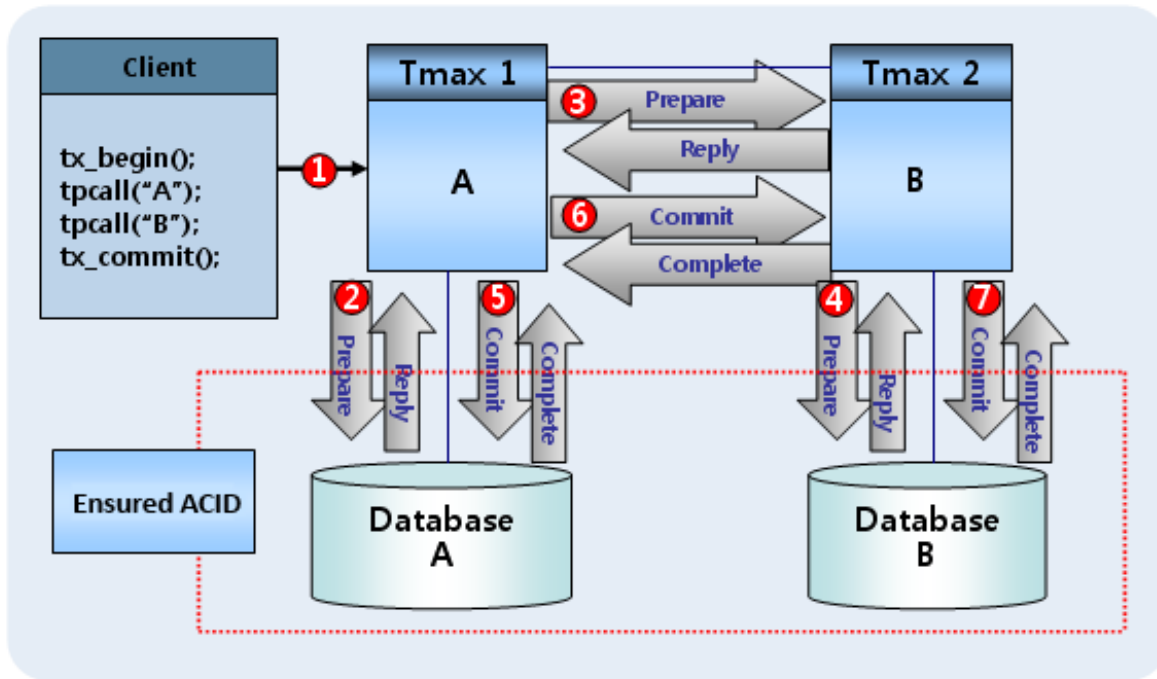
X/Open DTP Structure

The following describes X/Open DTP components.

Classification	Description
AP	The application program (AP) defines the boundary of a distributed transaction.
RM	The resource manager (RM) provides access to resources, such as databases.
TM	The transaction manager (TM) assigns identifiers to transactions by each database, manages execution of transactions, and recovers a transaction in case it completes or fails.
CRM	The communication resource manager (CRM) controls communication between applications that have operations distributed among different computers or databases from different vendors.
OSI-TP	The open system interconnection transaction processing (OSI-TP) manages communication between TMs.

X/OPEN ATMI processes transactions that occur between heterogeneous DBMSs that comply with X/OPEN DTP by grouping them.

The following is how distributed transactions are processed.



The Process of Distributed Transactions

A transaction starts when **tx_begin()** is called and ends when **tx_commit()** or **tx_rollback()** is called.

`tx_begin()` begins a transaction. The calling process becomes transaction publisher that is responsible for closing the transaction by using `tx_commit()` or `tx_rollback()`. Processes that are part of a transaction demarcated by `tx_begin()` and `tx_commit()` or `tx_rollback()` are called transaction participants, which can affect the results of the transaction when they return values with `tpreturn()`. You cannot call `tx_begin()` to start a transaction if another transaction is being processed, which causes failure and sets `tperrno` to `TPEPROTO`. Before calling another transaction, the transaction must be completed first.

A client participating in a transaction can exclude a server from the transaction by setting flags to `TPNOTRAN` when calling `tpcall()` or `tpacall()`. An excluded server cannot affect the result of the transaction being processed.

You can start an **explicit transaction** by issuing transaction functions such as `tx_begin` and `tx_commit` or an **implicit transaction** by configuring a database in `SVRGROUP` of the Tmax configuration file.

Checklist for DBMS Integration

The following is a DBMS integration checklist.

- Check that the DBMS vendor's client module is installed.
- Check restrictions on DBMS settings such as maximum session count for database and client.
- Get ready development tools module needed to create a Tmax server application
- Check the XA and non-XA settings.

The use of XA mode must be determined in the application design phase. XA and non-XA modes

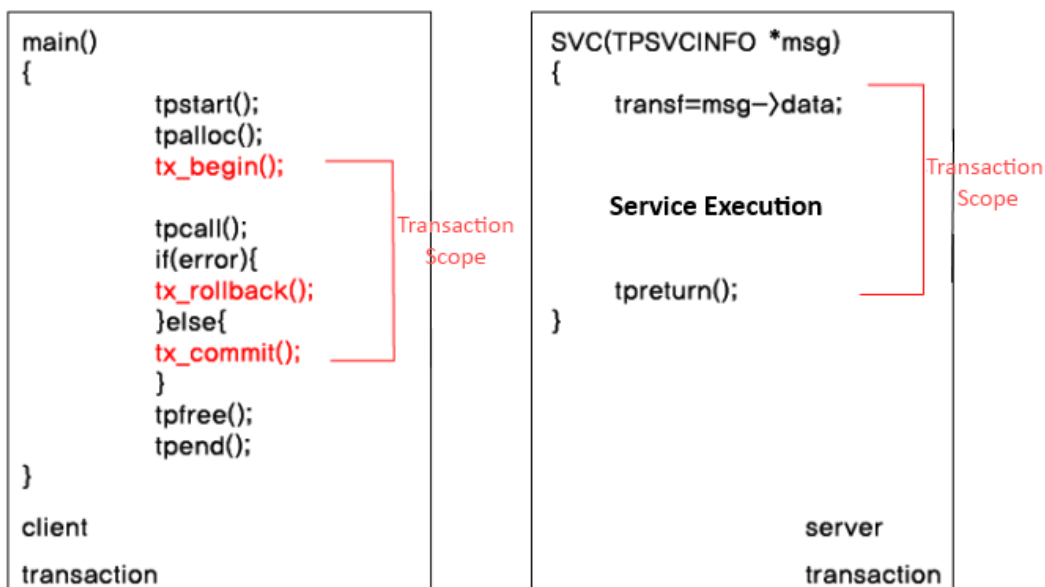
differ in terms of database access and distributed transaction processing methods as follows:

Mode	Description
XA	Settings of the Tmax configuration file is used to access a DBMS. X/OPEN ATMI functions are used to commit or roll back XA transactions, and the commit or rollback is completed through the transaction management server (TMS).
Non-XA	DBMS connections, transaction commits, and rollbacks are processed via the developer's ESQL statement.

6.2.1. XA Mode

In XA mode, the transaction management server (TMS) provided by Tmax manages transactions.

All transactions are handled as global transactions and are processed with the 2PC protocol to ensure data integrity. To connect and disconnect an RM, you must configure some settings in the Tmax configuration file. Transactions are managed using the Tmax API, which is necessary when processing databases in a distributed environment. The following are steps for executing an XA transaction.



XA Mode

To connect to and disconnnt a database, `tpsvrinit()` and `tpsvrdone()` are not required. As in non-XA mode, connection to the database is made with `tmboot`, and TMS, which manages commits and rollbacks of global transactions, also connects to the database. Any start, commit, rollback, or rollback cancellation calls of a global transaction must be made via the TX API of the Tmax system or by configuring the Tmax configuration file settings. To use the XA option, go to the SVRGROUP section in the configuration file. Then, check the name of a server group that contains the server application to connect to DBMS and specify the XA option for the server group.

The following is an example of specifying XA mode in the SVRGROUP section.

```
*SVRGROUP
```

```
svg1      NODENAME = tmax,  
          DBNAME = ORACLE,  
          OPENINFO = "ORACLE_XA+Acc=P/scott/tiger+SesTm=600",  
          TMSNAME = svg1_tms  
  
*SERVER  
svr1      SVGNAME  = svg1
```



For more information about server group configuration environment, refer to *Tmax Administrator's Guide*.

6.2.2. Non-XA Mode

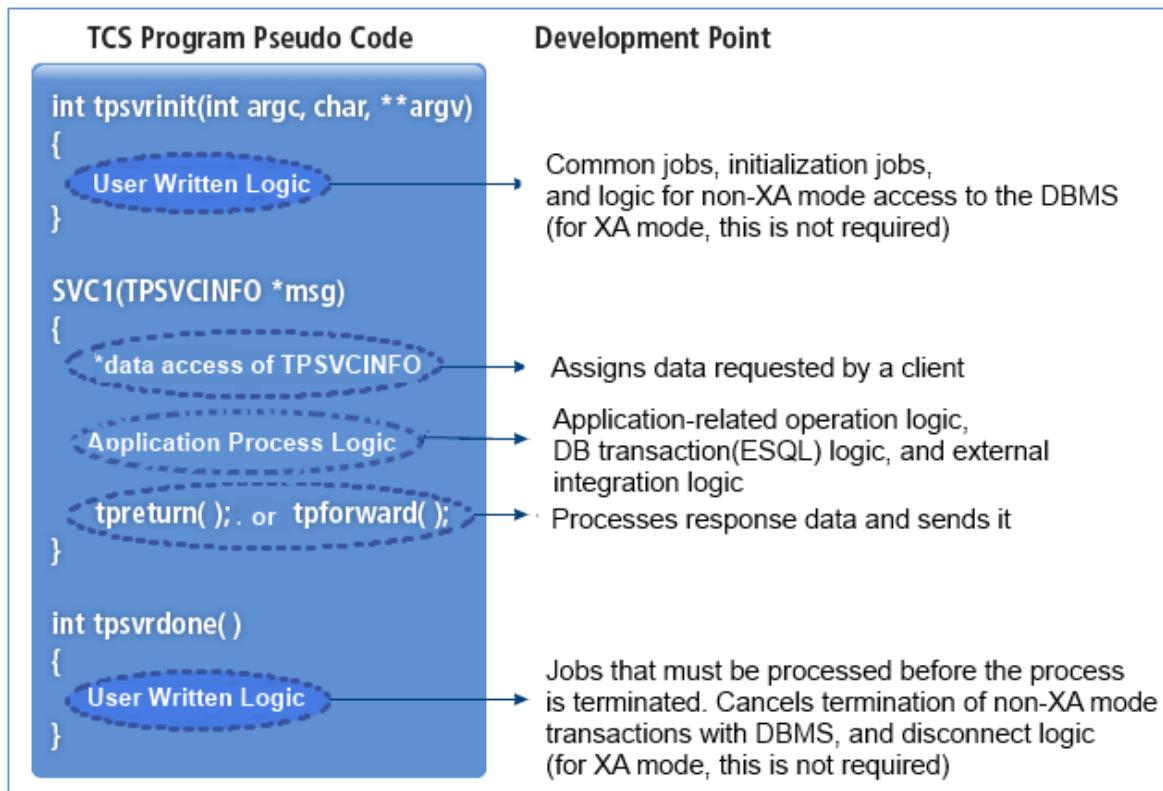
In non-XA mode, explicit transaction functions are enabled and global transactions are disabled in a distributed environment. The Tmax system issues explicit transaction functions, and global transactions are disabled in a distributed environment. To enable non-XA mode, you must connect an application to an RM and declare a transaction for the application.

In non-XA mode, a server program uses native SQL and has control over transaction involving the RDBMS, which cannot participate in global transactions supported by the Tmax system. Server programs running in non-XA mode are suitable for executing simple select queries or time-consuming batch server programs. In non-XA mode, you can define transaction boundary and control but only for direction connection between a client application and RDBMS, which does not involve the Tmax system.

The following describes how non-XA mode differs from XA mode.

- An RDBMS is accessed through the client application's SQL.
- The server can request a transaction but the client cannot. It is recommended to issue `tpsvrinit()` and `tpsvrdone()` to connect the server to the database or disconnect it.

The following are steps for executing a non-XA transaction.



Non-XA Mode

1. Write logic in `tpsvrinit()` to access the DBMS.
2. Write explicit logic to roll back and release the DBMS in `tpsvrdone()`.
3. When a Tmax server application is booted through `tmboot` or `tmboot -S svrname`, a connection is established between the DBMS and the Tmax server application process.
4. After a connection is established, the service logic can send a transaction request, commit, or roll back the transaction via ESQL.

The following is an example of specifying non-XA mode in the SVRGROUP section.

```
*SVRGROUP
svg1 NODENAME = tmax

*SERVER
svr1 SVGNAME = svg1
```



For more information about transaction-related functions, refer to [Transaction Management](#) or *Tmax Reference Guide*.

6.3. Transaction Errors

Transactions errors involve TX and XA.

TX errors are defined in <usrinc/tx.h>, and XA errors are listed by database vendor. For Oracle's XA errors, refer to <\$ORACLE_HOME/rdbms/demo/xa.h>.

6.3.1. TX Error

The following describes transaction errors.

Error Code	Description
TX_NOT_SUPPORTED(1)	Transactions are not supported in this mode.
TX_OK(0)	This transaction has been completed successfully.
TX_OUTSIDE(-1)	A local transaction is being processed.
TX_ROLLBACK(-2)	commit cannot be executed. The transaction is rolled back.
TX_MIXED(-3)	This transaction is partly committed and partly rolled back.
TX_HAZARD(-4)	This transaction has not been completed successfully.
TX_PROTOCOL_ERROR(-5)	This transaction has been abnormally called.
TX_ERROR(-6)	An error has occurred in the database.
TX_FAIL(-7)	The database has failed.
TX_EINVAL(-8)	Invalid parameters have been received.
TX_COMMITTED(-9)	This transaction has been committed independently to the database.
TX_NO_BEGIN(-10)	This transaction has been committed, but a new transaction cannot begin.

6.3.2. XA Error

The following describes XA errors.

Error Code	Description
XA_OK(0)	This transaction has been successfully completed.
XAER_RMERR(-3)	An error has occurred with the resource manager specified in the OPENINFO section.
XAER_NOTA(-4)	The given XID is invalid.
XAER_INVAL(-5)	Invalid parameters have been received.
XAER_PROTO(-6)	The routine has been invoked in an improper context.
XAER_RMFAIL(-7)	Failed to connect to database.
XAER_DUPID(-8)	A duplicate XID exists.
XAER_OUTSIDE(-9)	The database is working outside a local transaction.

7. Multithreading and Multicontexting

This chapter describes how to use multithreaded and multicontexted Tmax applications.

7.1. Overview

The Tmax system supports a kernel-level thread package to enable multithreading and multicontexting. However, you must consider logic when writing a program that creates and removes a thread. Multithreaded and multicontexting applications are supported in C but not in COBOL.



The multithread and multicontext functions are available for the client library in Tmax version 3.8.15 or later and for the server library in Tmax 5 SP2 or later.

7.2. Client Program

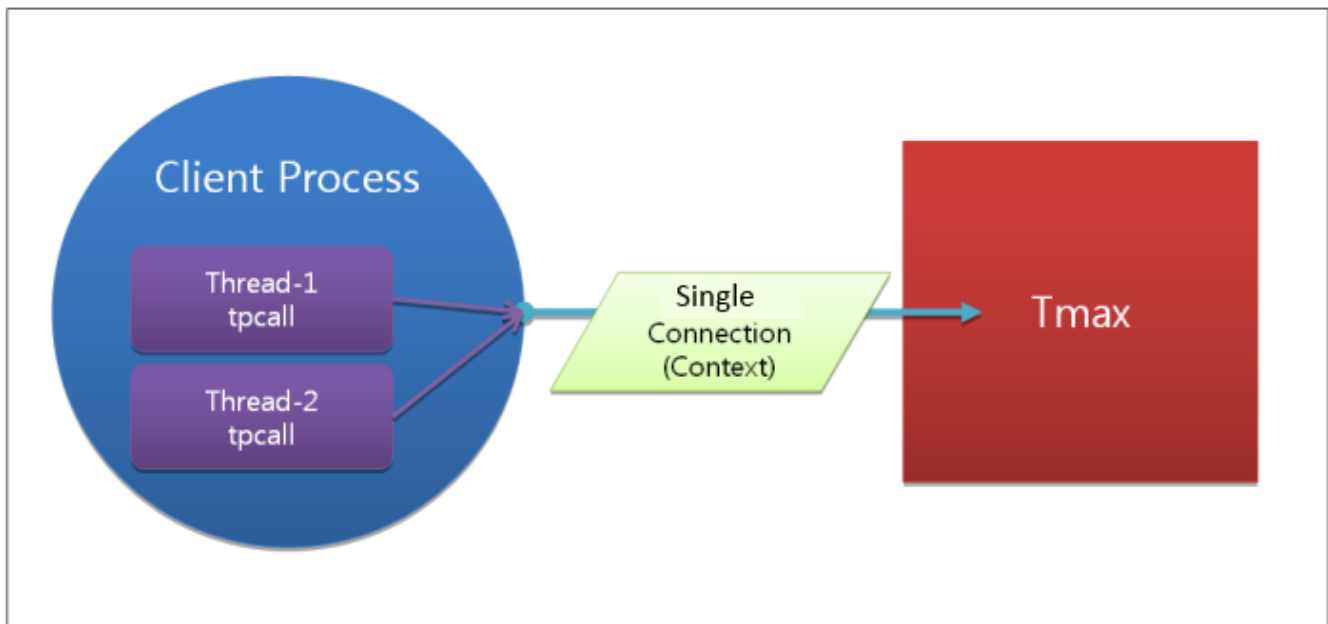
This section describes the program flow, implementation method, and examples of multithreaded and multicontexted server programs.

7.2.1. Program Flow

7.2.1.1. Multithreading

A multithreaded process has more than one execution unit. A multithreaded Tmax application enables one process to make multiple service requests concurrently.

The following figure shows the program process of a multithreaded client application, which can make two concurrent service requests.

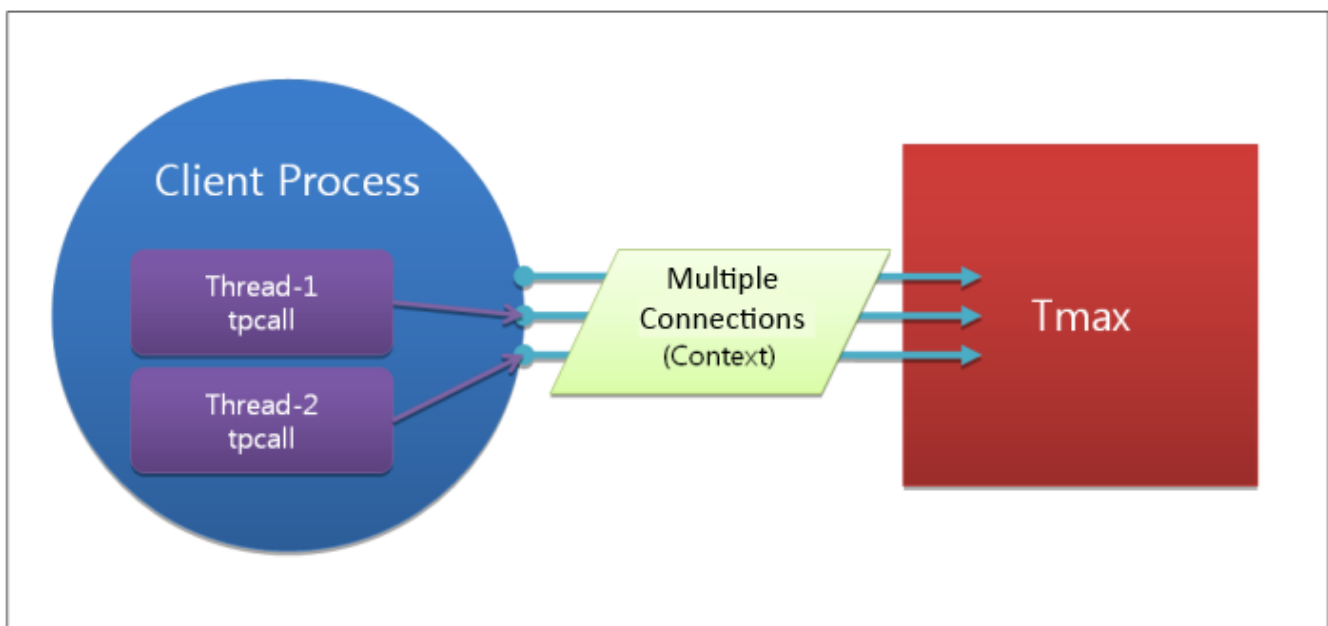


Multithreaded Tmax Client Application

7.2.1.2. Multicontexting

Multicontexted programming allows a single client to make multiple connections to the Tmax system.

The following figure shows the program flow of a multi-context client application program. One client has multiple contexts and each context is connected to the Tmax system for transferring data back and forth with the system. In a multicontexted environment, the Tmax system is considered a single user.



Tmax Client Multicontext Application

7.2.2. Implementation

The following routines must be included in a program to use multithreading and multicontexting.

1. Start statement
2. Implementation statement
3. End statement



For more information about the functions used to implement a client program, refer to *Tmax Reference Guide*.

Start Statement

The following function is used to start a multithreaded and multicontexted program.

- **tpstart()**

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

tpstart() creates a connection to the Tmax system. The first argument, TPSTART_T, has flags which can be set to **TPMultiContextS** or **TPSINGLECONTEXT** to enable a multi-context or single-context connection to the Tmax system. If the flag is not specified, TPSTART_T is set to TPSINGLECONTEXT by default.

After specifying the mode, the client continues to run in that mode.



For more information about tpstart(), refer to *Tmax Reference Guide*.

- **tpsetctxt()**

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

This function is used to configure the current context. Using this function differs for the client and server programs. For more information about the function, refer to [tpsetctxt](#).

Implementation Statement

You can implement a multithreaded or multicontexted application by using the ATMI functions. To use the functions successfully, you must check that you are applying them to the current context.

- Synchronous Communication

During synchronous communication between the Tmax system and the client, you can enable

multithreading and the multicontexting as long as a thread does not have TPINVALIDCONTEXT, which indicates that the current context is freed by another thread.

Let's suppose that thread1 invokes `tpstart()` and gets context1, and thread2 invokes `tpsetctxt()` and shares context1 with thread1. In this case, if thread1 finishes a task and invokes `tpend()`, context1 is deleted from memory. As a result, thread2 that shared context1 is left with TPINVALIDCONTEXT.

In single-context mode, the client can automatically start a connection to Tmax system by calling **`tpcall()`** without using **`tpstart()`**. However, in multicontext mode, the client must issue `tpstart()` to use other APIs such as `tpcall()`.

- Asynchronous Communication

In multicontext mode, after a thread calls **`tpacall()`**, another thread can access the results by calling **`tpgetreply()`** if the context is shared by the two threads. `tpacall()` must be called before `tpgetreply()`. The priority of the two threads must be explicitly defined in order to get a valid result.

Similar to synchronous communication, the process is successfully completed as long as the current context is not TPINVALIDCONTEXT.

- Transaction

If a thread starts a transaction, transactions of other threads that share the context of the thread become a single transaction. Similar to asynchronous communication, priority must be explicitly defined. In addition, the process is successfully completed as long as the current context is not TPINVALIDCONTEXT.

End Statement

To end multithreaded and multicontexted mode, use **`tpend()`**.

```
int tpend()
```

If `tpend()` is not used, the information of the context and associated threads is not deleted from memory, which can cause a problem. To avoid this issue, use `tpend()` after using a context.

7.2.3. Program Example

The following are examples of a client program, a server program, and a Makefile.

Client Program

The following is an example of a client program.

```
/*
 * Multi-thread/Multi-context Sample Program
 */
```

```

/*                                                                    */
/* TmaxSoft Co. / QA                                                */
/* remarks: TOUPPER of Tmax must be started already.                */
/*****                                                                */

#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      /* The number of maximum contexts*/
#define MAX_CALL          1
#define NUM_THREADS       2        /* The number of threads that must be created at once*/
#define NUM_CONTEXTS      40       /* The number of contexts to be created in a thread*/

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

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

#ifdef _MULTI_CONXTXT_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);
    }
#endif
}

```

```

        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);
}
/*****
/*                                */
/*****

```

Server Program

The following is an example of a server program.

```

#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);
}

```

Makefile

The following is an example of a Makefile used to build a client program.

```

TARGET = $(COMP_TARGET)
APOBJS = $(TARGET).o

TMAXLIBD = $(TMAXDIR)/lib64

TMAXLIBS = -lcli
#TMAXLIBS = /home/ancestor/tmax/lib/libclid.a

```



```

#In case of multi_thread / multi_context
CFLAGS = -q64 -O -I$(TMAXDIR) -D _MULTI_THREAD_TEST_ -D _MULTI_CONXTXT_TEST_

#In case of single_thread / multi_context
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. Server Program

This section describes the program flow, implementation method, and examples of multithreaded and multicontexted server programs.

7.3.1. Overview

Tmax supports multithreaded and multicontexted TCS server libraries. The UCS and RDP libraries do not support multithreading and multicontexting functions.

The multithreaded and multicontexted server library enables a server process to handle service requests from multiple threads and allows multiple threads to share a single context to run a service.

The standard Tmax server library does not support multicontexting. Therefore, from a user-created thread, you cannot use APIs provided by the Tmax server library.

The following must be completed to use the multithreaded and multicontexted server library.

- Writing the server application code using the associated APIs.
- Linking libsvmt,so or tmaxsvrmt.dll, which is a multithreaded and multicontexted server library when you are building a server program.
- Setting the multithreading and multicontexting server settings in the **SERVER** section of the Tmax configuration file.
 - Relevant settings: SVRTYPE, MINTHR, MAXTHR, STACKSIZE, and CPC.

Multithreaded programming provides many benefits. However, you must consider concurrency and performance when writing a program to avoid low concurrency and performance issues. To use multithreaded programming to your advantage, you need to understand both the benefits and limits of multithreaded programming.

The following are advantages and disadvantages of using multithreaded and multicontexted server libraries.

- **Advantages**

- You can write simple codes and intuitive programming scripts.
- You can involve less server processes.

- **Disadvantages**

- It is relatively difficult to manage concurrency and write code.
- It is relatively difficult to debug errors.
- If you are porting a program from another platform, you must check whether the program is thread-safe.
- If you are connecting to RM, you must check whether it is supported for a multithreaded program.



These benefits and limits must be considered with care before adopting multithreaded programming. If running any server program causes an issue, check logs. For more information about the logs, refer to *Tmax Error Message Reference Guide*.

7.3.2. Program Flow

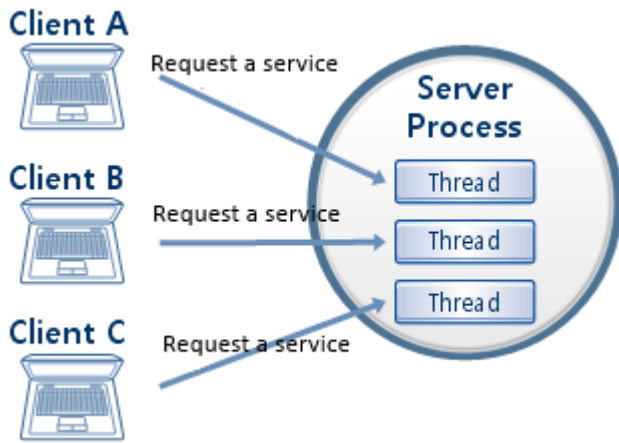
7.3.2.1. Multithreading

Multiple threads can exist in a single process to handle multiple service requests. In the multithread and multicontext server library, a thread is classified as a **Service thread** or a **User-created thread**.

Service Threads

A service thread is generated and managed by the multithreaded server library. It creates a specific number of service threads to process multiple service requests concurrently as set in the configuration file. The service threads are managed in the thread pool.

You can configure the thread pool with MINTH and MAXTHR settings of the configuration file. When a server process is booted, the minimum number of threads is created by default. If the number of threads becomes less than minimum number, additional threads are created to maintain the minimum number. If the number of service requests increases and no idle threads exist in the thread pool, additional threads are created up to the maximum number to process the services.



Tmax Server Multithread Application

When it receives service requests, the server process handles each service request in a separate thread independently and returns the results to the client.

The basic operations of service routines and regular server programs are the same. Therefore, you can write a service routine as you do from a regular server library. However, note that all service routines must be thread safe to avoid one service routine being executed by multiple threads.

The following are the steps for writing a server program flow. All steps but step five are automatically carried out by the server library.

1. At startup, a server process calls `tpsvrinit()`.
2. A thread pool is configured and service threads are created as set in the configuration file.
3. Each thread created initially calls the `tpsvrthrinit()` function one time.
4. Service threads wait in the thread pool. When a service request is received, an idle thread executes the service routine.
5. A service routine is executed by using the jobs described in [Implementation Statement](#).
6. After calling the `tpreturn()` and `tpforward()` functions, the thread returns the process results to a client and waits in the thread pool.
7. If the server process is terminated, all service threads created execute the `tpsvrthrdone()` function and are terminated.
8. The server process calls the `tpsvrdone()` function and is terminated.

User Threads

User threads can be created in a service or initialization routine such as `tpsvrinit()` or `tpsvrthrinit()`. They have their own starting routine and are not in the thread pool that the server library manages. Therefore, user threads do not handle any service request. After a user thread is created, you must keep in mind when the thread is created and when it is to expire.

User threads created by `tpsvrinit()`, `tpsvrthrinit()`, or a service routine function work independently from the multithreaded and multicontexted server library and therefore have no context. The user threads can be used in place of, together with, or independently from service routines. Before

creating user threads, refer to [Restrictions and Considerations](#).

The following are the steps for program execution flow when a user thread is sharing the context of a server thread. Steps one and eight are executed automatically by the server library.

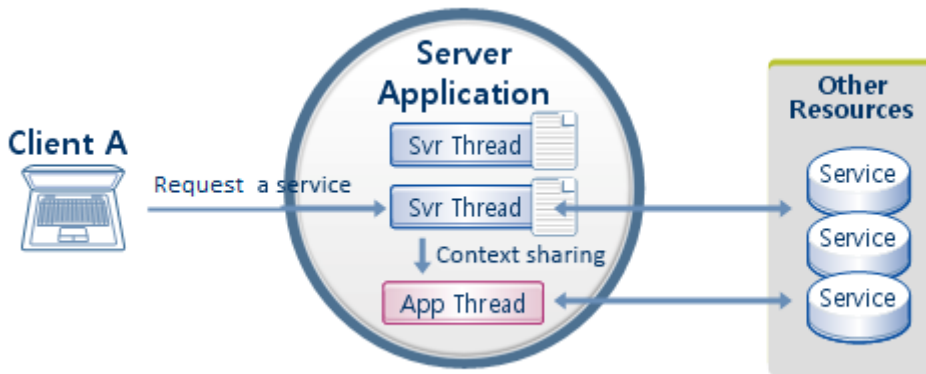
1. A service request is received, and a service thread performs a service routine.
2. The service routine calls `tpgetctxt()` to retrieve its context ID.
3. An existing user thread is used or a new user thread is created in the service routine.
4. The user thread receives the context ID from the service thread and calls `tpsetctxt()` to share the context.
5. Each thread performs its defined routine. Both the user thread and the service thread can call ATMI APIs such as `tpcall()` and `tpacall()`.
6. Before a service thread calls `tpreturn()` or `tpforward()`, the following must be completed.
 - Terminating all synchronous, asynchronous, and interactive communications.
 - Committing or rolling back an ongoing transaction.
 - Calling `tpsetctxt()` from the user thread to stop sharing the context
7. The user thread checks its termination time.
8. The service thread executes `tpreturn()` and is returned to the thread pool to wait for the next service request.

7.3.2.2. Multicontexting

A context in a server library is a set of data required to process a service request. A multithreaded environment requires multicontexting so that each thread can process a service request independently.

Each service thread in the thread pool has its own context by default. In a multicontexted environment, multiple threads can share a single context. However, user threads do not have their own context. For a user thread to call Tmax APIs such as `tpcall()`, the user thread must share the context of a service thread by calling `tpgetctxt()` or `tpsetctxt()`. If a user thread calls a Tmax API without sharing a context, the call fails and returns the TPEPROTO error code.

In the multithreaded and multicontexted server library, a context includes data about transactions and communication mode such as synchronous, asynchronous, and interactive communication. Such data is shared when the context of a service thread is shared with a user thread.



Tmax Server Multicontext Application

7.3.3. Implementation

The following APIs can be used when you are writing multithreaded and multicontexted server programs.

Related API

The following are the APIs that can be used to write a server program.

- **tpsvrthrinit**

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

Available only in a multithreaded and multicontexted server, `tpsvrthrinit` initializes a service thread managed by the thread pool after the `tpsvrinit` function is called. For more information about `tpsvrthrinit()`, refer to [tpsvrthrinit](#).

- **tpsvrthrdone**

```
int tpsvrthrdone()
```

Available only in a multithreaded and multicontexted server. As a server process is terminated, before calling `tpsvrdone`, `tpsvrthrdone` terminates service threads. For more information about `tpsvrthrdone()`, refer to [tpsvrthrdone](#).

- **tpsetctxt**

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

`tpsetctxt` specifies a context for use. The syntax of this function varies with the client program and the server program. For more information about `tpsetctxt()`, refer to [tpsetctxt](#).

- **tpgetctxt**

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

tpgetctxt returns as the first parameter the context ID set for the thread that called this function. For more information about tpgetctxt(), refer to [tpgetctxt](#).



More APIs are available in the server library. For more information about APIs used to implement a program, refer to *Tmax Reference Guide*.

Restrictions and Considerations

There are some restrictions and considerations you must keep in mind when developing a server program.

The following restrictions must be considered when APIs such as **tpsetctxt()** and **tpgetctxt()** are called to be used between a service thread and a user thread.

- User thread creation and termination times must be carefully and logically specified. If a service routine only creates a user thread but does not delete it, calling the service routine can result in a large number of threads, which negatively affects system performance.
- Because it does not have a context by default, a user thread must share the context of a service thread by calling tpsetctxt() to communicate with Tmax.
- The tpreturn() or tpforward() function must be called by a service thread. User threads cannot call tpreturn().
- When tpreturn() or tpforward() is called, if any synchronous, asynchronous, or interactive communication is incomplete, the TPESVCERR error code is returned to the client that has called the function and the service fails.
- The context that is used by a user thread must be returned to the service thread before tpreturn() or tpforward() is called. You must use tpsetctxt() to set the context to TPNULLEXCONTEXT so a user thread releases the context it is sharing before calling tpreturn(). Alternatively, you can specify a different context to be used by the user thread so that it does not share the context with the service thread that calls tpreturn(). Otherwise, tpreturn() returns the TPESVCERR error code to the client and the service fails.
- The tpsetctxt() function cannot be called from a service thread. If it is called from a service thread, the call returns the TPEPROTO error code and the service fails.
- A thread that has been started in one of the threads that share a context can be committed or rolled back from only one of them irrespective of where the transaction began.

The following are other considerations.

- When a service timeout occurs in a multithreaded or multicontexted server, the server process is immediately terminated. Other service requests currently being processed are also canceled

because it is difficult to determine if a thread was suspended due to a timeout while running a service. For example, if a thread is suspended after occupying synchronization resources or receiving dynamically allocated memory, resolving such an issue requires complex code and abnormal operations may occur.

- The `tpstart()` and `tpend()` functions used by clients cannot be used.
- Only services provided by its own server process can be called in synchronous and asynchronous communications. In this situation, you must prepare for a deadlock resolution process (for example, setting a service timeout for related services) in case a deadlock occurs.

7.3.4. Service Processing Program Example

The following are examples of a client program, a server program, and a Makefile.

Server Program

The following is an example of a server program.

```
#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;
    }

```

Makefile

The following is an example of a server Makefile.

```

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

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

$(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)

```


Configuration

The following is an example of a configuration file.

```
*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. Context-Sharing Program Example

In this example, a client calls the service named MSERVICE. A service routine creates a user thread, which shares the context of the service thread and simultaneously requests a service through tpcall().

Server Program

The following is an example of a server program.

```
#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.ctxid, 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->ctxid, 0) == -1) {
        printf("tpsetctx(%d) failed, [tperrno:%d]", param->ctxid, tperrno);
        return NULL;
    }

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

    if (tpsetctx(TPNULLCONTEXT, 0) == -1) {
        printf("tpsetctx(TPNULLCONTEXT) 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;
}

```

Makefile

Refer to [Makefile](#).

Configuration File

The following is an example of a configuration file.

```
*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
```

8. Security System

This chapter describes the security system provided by Tmax.

8.1. Overview

Tmax provides a three-level security system: system access control, user authentication, and service access control. The security level can be set with the **SECURITY** item in the DOMAIN section of the Tmax configuration file. The following values are available.

Value	Description
DOMAIN_SEC	System access control
USER_AUTH	User authentication
ACL MANDATORY	Service access control
NO_SECURITY	No security



For more information about creating each file (group, user, and acl) and the relevant commands, refer to *Tmax Reference Guide*.

8.2. Level 1 Security (System Access Control)

The level-one security function, system access control, restricts a client's access to the Tmax system. The client must enter a password required to access the Tmax system, which is used as the password for the account defined in **OWNER** of the DOMAIN section.

Before registering the password in Tmax, the client must set the password by using the **mkpw** utility. After system access control is set, the client must register the password with **dompwd** of the TPSTART_T structure when tpstart() is called. The client must enter the correct password to log in to the Tmax system. If **SECURITY** is set to NO_SECURITY, the client does not need to set a value for dompwd, to which NULL is passed.

The following is a Tmax configuration file that uses system access control.

```
#For system authentication, set SECURITY to "DOMAIN_SEC".
*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

```

The following is an example of specifying a password by using dompwd of the TPSTART_T structure.

```

...
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. Level 2 (User Authentication)

The level-two security, user authentication, allows only an authorized user to access the Tmax system.

In order to connect to Tmax by calling tpstart(), a client must register **usrname** and **usrpwd** of the TPSTART_T structure. **usrname** is the user name of the account authenticated by Tmax and **usrpwd** is the account password. A username and password must be specified by using the **mkpw** utility. When user authentication is enabled, the client can log into the system only with a valid user name and password.

Because user authentication includes system access control, the client must register a valid dompwd by using the mkpwd utility.



The password file must be created and updated before Tmax is configured.

The following is a Tmax configuration file that uses user authentication.

```

#In case of User Authentication, set "USER_AUTH" to the SECURITY item.
*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
```

The following is an example of setting **usrname** and **usrpwd** of the TPSTART_T structure.

```
...
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. Level 3 (Service Access Control)

The level-three security, service access control, restricts an authorized client's access to services.

Service access control classifies users into groups. A service is available only to users who belong to a group that is authorized for the service. Therefore, service access control requires a group file, one or more user files that belong to the group file, and an **ACL** file, which is used for associating a user group with certain access permissions to use services. Before using service access control, you must set the **SECURITY** item to **ACL** or **MANDATORY**.

Service access control includes both system access control and user authentication security measures. Therefore, if the parameter in SECURITY is set to ACL (or MANDATORY), a client must pass the level one and two security measures by using the dompwd, username, and usrpwd to connect to the Tmax system and gain permission to access services.

The following is a Tmax configuration file that uses service access control.

```
*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

```

<Group File>

```

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

```

<User File>

```

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

```

<ACL File>

```

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

```


<Client Program>

```
int main()
{
    ...
    strcpy(tpinfo->usrname, "starbj1");
    strcpy(tpinfo->dompwd, "starbj0");
    strcpy(tpinfo->usrpwd, "starbj1");
    ...
    if(tpcall("TOUPPER1", sndbuf, 0, &rcvbuf, &rcvlen, 0)==-1){
        error routine..
    }
}
```

9. Client API

This chapter describes the APIs used in client program development.

9.1. Overview

The following is a list of functions available for client program development. For more information about the functions, refer to *Tmax Reference Guide*.

- Connection and Disconnection

Function	Description
tpstart	Connects client to the Tmax system.
tpend	Disconnects client from the Tmax system.

- Synchronous Communication

Function	Description
tpcall	Sends a service request through synchronous communication and waits for a reply.

- Asynchronous Communication

Function	Description
tpacall	Sends a service request message and directly returns without waiting for a reply.
tpgetrply	Receives a reply for the service that was requested through the <code>tpacall()</code> function.
tpcancel	Cancels a server or client reply.

- Interactive Communication

Function	Description
tpconnect	Establishes a connection and starts communication.
tpsend	Sends a message from the connection controller.
tprecv	Receives a message from the connection controller.
tpdiscon	Server and client close the interactive communication connection.

- Unsolicited Message Processing

Function	Description
tpsetunsol	Used by the client to set the routine that processes unsolicited messages.
tpgetunsol	Processes unsolicited messages.

- Timeout Change

Function	Description
tpset_timeout	Used by the client and server to set the blocking timeout.
tpsetsvctimeout	Called in a service to set the service timeout.

- Buffer Management

Function	Description
tpalloc	Used by the client and server to allocate a buffer.
tprealloc	Used by the client and server to reallocate a buffer.
tpfree	Used by the client and server to release the memory allocated to the buffer.
tptypes	Used by the client and server to provide information about the buffer type and sub-type.

- Transaction Management

Function	Description
tx_begin	Used by the client and server to start a global transaction.
tx_commit	Used by the client and server to commit a transaction.
tx_rollback	Used by the client and server to roll back a transaction
tx_set_transaction_timeout	Used by the client and server to set the transaction_timeout property with a timeout value.
tx_set_transaction_control	Used by the client and server to set the transaction_control property with a control value.
tx_set_commit_return	Used by the client and server to set the commit_return property.
tx_info	Used by the client and server to return the global transaction information.

- RQ System

Function	Description
tpenq	Used by the client and server to store data in RQ.
tpdeq	Used by the client and server to load data from RQ.

Function	Description
tpqstat	Used by the client and server to request the statistics of the data stored in RQ.
tpextsvcname	Used by the client and server to extract the service name of data that is read from the RQ using tpdeq.

- Functions using Events

Function	Description
tpsubscribe	Used by the client and server to register a request for the message of a specific event.
tpunsubscribe	Used by the client and server to unregister the request for the message of a specific event.
tppost	Used by the client and server to generate a specific event and send a message.

- Broadcast and Multicast

Function	Description
tpbroadcast	Used by the client and server to send an unsolicited message to other clients.

- Windows Environment Programming

- tmaxmt.dll

Function	Description
WinTmaxAcall	Connects to Tmax to request a service, replies to the Windows procedure with the result, and then closes the connection.
WinTmaxAcall2	Connects to Tmax to request a service and receives the reply with a callback function.

- WinTmax.dll

Function	Description
WinTmaxStart	Creates a thread to process a message and initializes the used memory.
WinTmaxEnd	Closes the thread that processed a message and releases the used memory.
WinTmaxSetContext	Sets the Windows and message number to send data received by Tmax.
WinTmaxSend	Allows a client to call a service and process other jobs immediately.

- Multithread and Multicontext

Function	Description
tpgetctxt	Returns the current context.
tpsetctxt	Sets the current context.

9.2. Connection and Disconnection

The following describes functions that are used to connect to Tmax and disconnect from Tmax.

9.2.1. tpstart

Connects a client to a Tmax system. Before using ATMI functions to process service requests or transactions, a client must use `tpstart()` to connect to a Tmax system.

Before `tpstart()` is called, if another ATMI function (`tpalloc()` or `tpcall()`) is called, `tpstart(NULL)` is called internally. To disable this action, set the `TMAX_ACTIVATE_AUTO_TPSTART` environment variable to N. In such a case, if another ATMI function is called, a TPEPROTO error is returned without calling `tpstart(NULL)` internally.

To connect to a Tmax system by using `tpstart()`, the IP and port number of the server where the Tmax system is installed are required. If `tpstart()` is returned successfully, the client can send an initial service request or can define a transaction. In this case, if `tpstart()` is called again, a TPEPROTO error occurs.

The following describes environment variables required to find server information.

Variable	Description
<code>TMAX_HOST_ADDR</code>	IP address of a node to which a client is to be connected. This variable is used for a client to be internally connected to a server system when <code>tpstart()</code> is called.
<code>TMAX_HOST_PORT</code>	<p>Port number of a node to which a client is to be connected. This variable is used together with a <code>TMAX_HOST_ADDR</code> for a client to be internally connected to a server system when <code>tpstart()</code> is called.</p> <p>The port number must be defined in a <code>TPORTNO</code> in a Tmax configuration file. When both a client and a server reside in a node, it is more efficient to use a domain socket rather than a TCP/IP socket when processing a client request. In this case, replace the <code>TPORTNO</code> value with <code>PATHDIR</code>. Refer to the <code>TPORNO</code> item in the <code>DOMAIN</code> and <code>NODE</code> sections of a Tmax configuration file.</p>

Variable	Description
TMAX_BACKUP_ADDR	Alternative Tmax system node to use when the node specified by TMAX_HOST_ADDR fails. The client tries to connect to the node specified by TMAX_HOST_ADDR, and, if the attempt fails, to the node specified by TMAX_BACKUP_ADDR.
TMAX_BACKUP_PORT	Tmax system port number of the node with a TMAX_BACKUP_ADDR address.
TMAX_CONNECT_TIMEOUT	Timeout value for connecting to a Tmax system in microseconds. (Example: 3.5)

- Prototype

```
#include <atmi.h>
int tpstart (TPSTART_T *tpinfo )
```

- Parameters

A tpinfo is a pointer for a TPSTART_T structure. This structure uses a TPSTART buffer type, which must be allocated by a tpalloc() before tpstart() is called. After tpstart() has been called, the allocated buffer must be freed by using tpfree(). When connecting to the system, a client transfers necessary information with the tpinfo parameter, which contains client information, unsolicited message processing methods, and security information.

You can set tpinfo to NULL. In this case, cltid, dompwd, username, and usrpwd are given zero-length strings, the Tmax security features is not enabled, and flags are not selected.

The following shows a TPSTART_T structure.

```
struct TPSTART_T{
    char cltid[MAXIDENT+2]; /* Client name (tpbroadcast())*/
    char dompwd[MAX_PASSWD_LENGTH+2]; /*Password for system access security*/
    char username[MAXIDENT+2]; /*Account for user authentication security*/
    char usrpwd[MAX_PASSWD_LENGTH+2]; /*Password for user authentication security*/
    int flags; /*Unrequested message type and system access method*/
} ;
```

Member	Description
cltid	Character string ending with a NULL value that can be up to 30 characters long. A cltid is a name defined by an application program and is used to specify a client to which tpbroadcast() sends unsolicited messages.
dompwd	Password used for system access control of the Tmax security system. It registers a password for an account set to an OWNER item of the DOMAIN section of the Tmax configuration file.

Member	Description
username	User name used for user authentication of the Tmax security system. A username must be an account name registered in the passwd file of a Tmax system.
usrpwd	Password for an account. When user authentication is required, a client must register a username and usrpwd to connect to the Tmax system and access services. For information about security settings, refer to the SECURITY item in the DOMAIN section of the source config file.
flags	<p>Used to determine how to handle unsolicited notifications and how to access the system.</p> <p>The following values are available.</p> <ul style="list-style-type: none"> • TPUNSOL_POLL: receives unrequested messages. • TPUNSOL_HND: sets a function that receives unrequested messages. For more information, refer to tpsetunsol. • TPUNSOL_IGN: ignores unsolicited messages. (Default)

• Return Values

Value	Description
0 or 1	A function call was successful. 0 is returned to the primary host and 1 to a backup host.
1	A function call failed. A tperrno is set to an error code.

• Errors

The following is a list of error codes to one of which tperrno is set to when tpstart() fails.

Error Code	Description
[TPEINVAL]	An invalid parameter. For example, a tpinfo is NULL or not a pointer for a TPSTART_T.
[TPEITYPE]	A tpinfo is not a pointer for a TPSTART_T structure.
[TPEPROTO]	A tpstart() was called from an invalid state. For example, a tpstart() was called in a server program, or it was called after a connection was already established.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred. Environment variables may be invalid. For example, a connection failed because a TMAX_HOST_ADDR or TMAX_HOST_PORT was invalid.

• Examples

```

#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();
}

```

- Related Functions

tpend()

9.2.2. tpend

Disconnects a client from a Tmax system. If a client is in a transaction mode, a transaction will be rolled back automatically. If a tpend() is returned successfully, a caller will not communicate with any other programs and will not participate in any transactions. Maintained interactive connections will be terminated immediately.

If a tpend() is called more than once (if called again after already being disconnected from a Tmax system), a -1 value will be returned without affecting a system.

- Prototype

```

# include <atmi.h>
int tpend (void)

```

- Return Values

Value	Description
-1	A function call failed. tpperrno is set to an error code.

- Errors

When a tpend() fails to execute, a tpperrno will be set to one of the following values:

Error Code	Description
[TPEPROTO]	tpend() was called from an invalid state. For example, a caller is a server or a tpend() was called after being disconnected.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

```
#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();
}
```

- Related Functions

tpstart()

9.3. Synchronous Communication

The following describes functions that are used for synchronous communication.

9.3.1. tpcall

Sends a service request to a svc service named through synchronous communication and receives a reply. It works similar to calling a **tpgetrply()** successively after calling a **tpacall()**.

If tx_time expires after tx_begin() is called, calling tpcall() occurs the TPETIME error.

- Prototype

```
# include <atmi.h>
int tpcall (char *svc, char *idata, long ilen, char **odata, long *olen,
           long flags)
```

- Parameters

Parameter	Description
svc	Service that is requested, which must be one provided by a Tmax application server program.
idata	Pointer to service request data. The buffer must have been allocated by a tpalloc(). The type and subtype of an idata must be supported by the service specified by svc.
ilen	Data length. <ul style="list-style-type: none">• If an idata points to a variable-length buffer type, such as STRING, STRUCT, X_COMMON, or X_C_TYPE, an ilen is ignored and set to 0 by default.• If an idata points to a fixed-length buffer type, such as X_OCTET, CARRAY, or MULTI STRUCTURE, an ilen cannot be set to 0.• If an idata is NULL, ilen is ignored.
*odata	Pointer to reply data with the length of a value specified by *olen. *odata must have been allocated by using tpalloc(). If the same buffer is used to both send and receive data, *odata must be set to the idata address. Before tpcall() is completed, the reply buffer size of *odata and the value received by *olen are compared to determine whether to change the reply buffer size. If the reply buffer is smaller than the value of *olen, the reply buffer size is enlarged. Otherwise, the reply buffer size is unchanged.

Parameter	Description
*odata	<p>If tpcall() is called when idata and *odata are using the same buffer, the idata address becomes invalid if the reply buffer is too big or if any other occasion causes invalidation.</p> <p>If a *olen is returned as 0, no data is received and *odata and the buffer it refers to are not changed. If *odata or *olen is NULL, it is an error.</p>
*olen	Reply data length in a *odata.
flags	<p>Communication mode option.</p> <p>The following values are available.</p> <ul style="list-style-type: none"> • TPNOTRAN <p>If a program calls tpcall() to request the svc service with the flag set to TPNOTRAN in transaction mode, the service is executed outside the transaction. Therefore, to call a service that does not support transactions, set the flag to TPNOTRAN to call a tpcall() in transaction mode. When tpcall() is called in transaction mode, it is subject to a transaction timeout (TXTIME) even if the flag is set to TPNOTRAN. For example, a tpacall with a TPNOTRAN flag fails after a transaction times out without calling a service. If a service called with the flag set to TPNOTRAN fails, there is no effect on the current transaction of the caller.</p> • TPNOCHANGE <p>If a TPNOBLOCK flag is set and a blocking condition (for example, the internal buffer is full with messages to send) occurs, a request fails. A TPNOCHANGE is applied only to a Tx in a tpcall(). If a tpacall() is called without setting a TPNOBLOCK flag and a blocking condition occurs, a caller must wait until it is unblocked or until a transaction or blocking timeout occurs.</p> • TPNOTIME <p>A caller awaits a reply indefinitely and a blocking timeout is ignored. Timeout will occur even if a tpcall() is called during a transaction timeout.</p> • TPSIGRSTRT <p>Allows signal interrupts. If a system function is interrupted by a signal, the system function will be executed again. If a signal interrupt occurs without the flag set to TPSIGRSTRT, a function fails and tperrno is set to TPGOTSIG.</p>

- Return Values

Value	Description
1	A function call was successful.
-1	A function call failed, and a tperrno is set to an error code.

- Errors

If a tpcall() fails, a tperrno will be set to one of the following values:

Error Code	Description
[TPEINVAL]	Invalid parameter or flag. For example, svc is NULL or data points to a buffer that was not allocated by a talloc().
[TPENOENT]	The specified svc does not exist.
[TPEITYPE]	The svc does not support the idata type or subtype.
[TPEOTYPE]	<p>A function caller does not know the type or subtype of a reply buffer, or flags were set to a TPNOCHANGE but the type or the subtype of the buffer pointed to by *odata does not match that of the reply buffer. In this case, the data in *odata and *olen are not changed.</p> <p>If a caller requests a service in transaction mode, the transaction is rolled back because a reply is ignored.</p>
[TPETRAN]	A xa_start failed due to an error in a database while invoking a transaction service.
[TPETIME]	<p>A transaction timeout occurred when a caller was in transaction mode, and the transaction was rolled back. If a caller is not in transaction mode and if neither TPNOTIME nor TPNOBLOCK is specified, a blocking timeout occurs. In that case, *data and *len are not changed. If a transaction timeout occurs, any attempt to receive a new service request or await a reply causes a [TPETIME] error and fails until the transaction is rolled back.</p>
[TPESVCFAIL]	<p>A service routine sent a reply by calling a tpreturn() with a TPFAIL because of an application error. Service reply data can be accessed through *odata.</p> <p>Communication may be attempted before a transaction is rolled back due to a transaction timeout. Such a communication attempt may succeed or fail. For the communication to succeed, you must specify TPNOTRAN. All operations performed while a caller is in transaction mode are rolled back when the transaction is complete.</p>
[TPESVCERR]	An error occurred during the execution of a service routine or a tpreturn() (for example, a wrong argument was passed). If this error occurs, no reply data is returned, and neither *odata nor *olen is changed.

Error Code	Description
[TPESVCERR]	<p>This error occurs when a buffer that was not allocated by a tpalloc is used, when the Tmax header of an allocated buffer is affected by an invalid pointer such as memcpy, when a call descriptor of a tpacall or a tpconnect is returned, or when a service includes invalid interactive data while in recv mode. A client will receive this error when attempting to execute a tpreturn.</p> <p>When a TPEV_DISCOMN event occurs, which is equivalent to a TPEV_DISCOMN that is returned to a service program after a client forcefully disconnects a conversation, a client receives a tperrno set to TPESVCERR through a tpreturn of a service.</p> <p>If the caller is running a transaction, further communication may be attempted before a transaction is rolled back due to a transaction timeout. Such a communication attempt may succeed or fail. For the communications to succeed, you must specify TPNOTRAN. All operations performed while a caller is in transaction mode are rolled back when a transaction is complete.</p> <p>A SVCTIMEOUT can be specified for each service. If a service execution time exceeds a specified limit, service execution stops and returns a TPESVCERR. If a SVCTIMEOUT occurs, a tpsvctimeout() is called. Operations such as buffer lock releases and logging can be performed during a tpsvctimeout() as needed.</p>
[TPEBLOCK]	Blocking occurred while the flag is set to TPNOBLOCK.
[TPGOTSIG]	A signal was received while the flag is not set to TPSIGRSTRT.
[TPEPROTO]	A tpcall() was called in an invalid state.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

```
#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();
}

```

- Related Functions

tpalloc(), tpacall(), tpgetrply(), tpreturn()

9.4. Asynchronous Communication

The following describes functions that are used for asynchronous communication.

9.4.1. tpacall

Sends a service request to a **svc** service through asynchronous communication and is returned immediately without awaiting a reply. A response can be obtained using **tpgetrply()** or canceled using **tpcancel()**.

If tx_time expires after tx_begin() is called, tpacall() succeeds only when flags is set to TPNOTRAN or TPNOREPLY; otherwise, the TPETIME error occurs.

- Prototype

```

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

```

- Parameters

Parameter	Description
svc	Service to call. Must be provided by a Tmax application server program.
data	Pointer to a buffer allocated by a tpalloc().
len	Length of data to be sent. <ul style="list-style-type: none"> • If data points to a variable-length buffer type, such as STRING, STRUCT, X_COMMON, or X_C_TYPE, len is ignored and 0 is used.

Parameter	Description
len	<ul style="list-style-type: none"> • If data points to a fixed-length buffer type, such as X_OCTET, CARRAY, or MULTI STRUCTURE, len cannot be 0. • If data is NULL, len is ignored and a service request is received without data. The data type and subtype must be supported by svc. If a service request is sent in transaction mode, a reply must be received.
flags	<p>Calling mode option.</p> <p>The following values are available.</p> <ul style="list-style-type: none"> • TPBLOCK <p>If a tpacall is used without a flag, a normal reply is returned even if a called service does not exist in svc or if an invalid result is returned. An error is returned when a tpgetrply is called. If a TPBLOCK is used to call a tpacall(), the service state can be checked.</p> • TPNOTRAN <p>If a service does not support transactions in transaction mode, you must set the flag to TPNOTRAN to call tpacall(). If a caller requests the svc service by setting a flag in transaction mode, the service is executed outside the transaction. When tpacall() is called in transaction mode, the call is affected by a transaction timeout even if TPNOTRAN is specified. For example, a tpacall with the flag set to TPNOTRAN will fail after a transaction times out without calling a service. If a service called with the flag set to TPNOTRAN fails, there is no effect on the current transaction of the caller.</p> • TPNOREPLY <p>If tpacall() is used to send a service request, tpacall is returned immediately without awaiting a reply. A client can retrieve results by using tpgetrply() through the descriptor returned by tpacall(). If the flag is set to TPNOREPLY, tpacall() does not await a reply and returns 0 if a service is called successfully. If a caller is in transaction mode, you must specify TPNOREPLY together with TPNOTRAN. When TPNOREPLY is specified, you must specify TPBLOCK if you want to check a service state. If TPBLOCK is not specified, an error is returned even if a service is NRDY.</p> • TPNOBLOCK <p>If there is a blocking condition such as when an internal buffer is full of messages to send, this option causes a request to fail. If a tpacall() is called without setting the flag to TPNOBLOCK and if a blocking condition occurs, a caller must wait until a transaction or a blocking timeout occurs.</p>

Parameter	Description
flags	<ul style="list-style-type: none"> • TPNOTIME A caller must wait indefinitely until a reply is received. Blocking timeouts are ignored. A timeout will occur even if a <code>tpacall()</code> is called during a transaction timeout. • TPSIGRSTRT Allows signal interrupts. If a system function call is interrupted by a signal, it will be re-executed. If a signal interrupt occurs without this flag set, a function will fail and <code>tperrno</code> will be set to <code>TPGOTSIG</code>.
flags	Flags are listed in the following table.

- Return Values

Value	Description
Descriptor	A function call is successful. A returned descriptor is used to get a service reply.
-1	A function call failed, and <code>tperrno</code> is set to an error code.

- Errors

If a `tpacall()` fails, a `tperrno` will be set to one of the following values:

Error Code	Description
[TPEINVAL]	Invalid parameter. For example, when a <code>svc</code> is <code>NULL</code> , data will point to a buffer that was not allocated by a <code>tpalloc()</code> , or a flag will be invalid.
[TPENOENT]	The specified <code>svc</code> does not exist.
[TPEITYPE]	The data type or subtype is not supported by the <code>svc</code> . For a struct, this error occurs when the struct is not declared in a <code>SDLFILE</code> file.
[TPELIMIT]	The maximum number of unprocessed asynchronous service requests has been reached. A request was not sent.
[TPETIME]	A transaction timeout occurred when a caller was in transaction mode, and a transaction was rolled back. If a caller is not in transaction mode, and if neither <code>TPNOTIME</code> nor <code>TPNOBLOCK</code> is specified, a blocking timeout occurs. In that case, the data in <code>*data</code> and <code>*len</code> are not changed. If a transaction timeout occurs, any attempt to receive a new service request or await a reply causes a [TPETIME] error and fails until the transaction is rolled back.
[TPEBLOCK]	Blocking occurred while the flag is set to <code>TPNOBLOCK</code> .
[TPGOTSIG]	A signal was received while the flag is not set to <code>TPSIGRSTRT</code> .
[TPEPROTO]	A <code>tpacall()</code> was called in an invalid state. For example, a <code>TPNOREPLY</code> was called in a transaction mode without setting a <code>TPNOTRAN</code> .

Error Code	Description
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

```
#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();
}
```

- Related Functions

tpalloc(), tpacall(), tpcancel(), tpgetrply()

9.4.2. tpgetrply

Used to get a reply for an asynchronous service request made through a **tpacall()**. This function can be used on both a client and a server.

- Prototype

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

- Parameters

Parameter	Description
cd	Call descriptor returned by a tpacall(). In general, tpgetrply waits until a reply for a cd is received or until a timeout occurs. In general, cd becomes invalid after tpgetrply receives a valid reply.
*data	Pointer to a buffer allocated by a talloc().
len	<p>Length of data received from a tpgetrply(). If necessary, the buffer size can be increased to store a reply.</p> <p>You can change the size of *data for various reasons such as when the buffer is smaller than the received data. Before tpgetrply is called, if a len is greater than the total size of a buffer, the buffer size will be set to the value of len. If len is returned as 0, no reply is received and buffers pointed to by a *data and len are not changed.</p> <p>If a *data or a len is NULL, it is an error.</p>
flags	<p>The following values are available.</p> <ul style="list-style-type: none"> • TPGETANY <p> Ignores the cd value specified for a reply and returns a reply that can be received. In general, tpgetrply() waits until a reply is received. If TPGETANY is not specified, *cd becomes null without additional settings. If TPGETANY is specified, cd can be used for an error response. If an error occurs before a response is returned, cd is set to 0. This does not affect a current transaction for a caller unless there are additional settings.</p> • TPNOCHANGE <p> The buffer type that *data points to cannot be changed. If the types of the reply buffer and *data do not match, the *data type is changed to the reply buffer type. If TPNOCHANGE is specified, the buffer type is not changed. The buffer type and subtype of a reply buffer must be the same as those of *data.</p> • TPNOBLOCK <p> Does not wait for a response, but returns a valid response if available. If a TPNOBLOCK flag is not set, and there is no available response, a caller waits until a reply is received or until a transaction or a blocking timeout occurs.</p> • TPNOTIME <p> A caller waits indefinitely until a reply is received. Blocking timeouts are ignored and timeout will occur even if a tpgetrply() is called during a transaction timeout.</p>

Parameter	Description
flags	<ul style="list-style-type: none"> TPSIGRSTRT <p>Allows signal interrupts. If a system function call is interrupted by a signal, it is re-executed. If a signal interrupt occurs without this flag specified, the function will fail and a tperrno will be set to TPGOTSIG.</p>

- Return Values

Value	Description
1	A function call was successful. When a tpgetrply() returns successfully or a tperrno is [TPESVCFAIL], the global variable tpurcode returned by a tpreturn() will be set to a value defined in the application.
-1	A function call failed. tperrno is set to an error code.

- Errors

If a tpgetrply() fails, a tperrno will be set to one of the following values:

Error Code	Description
[TPEINVAL]	An invalid parameter. For example, a cd, data, *data, or a len is NULL or a flags is invalid. If a cd is not NULL, it is valid even if an error occurs, and a function will continue to wait for a response.
[TPEBADDESC]	An invalid cd.
[TPEOTYPE]	<p>A caller does not know the type or subtype of a response buffer.</p> <p>In this case, flags are set to TPNOCHANGE but the type or subtype of *data does not match that of the response buffer, so the data in *data and *len are not changed. If the caller received a response in a transaction mode, the transaction will be rolled back because the response will be ignored.</p>
[TPETIME]	A transaction timeout occurred when a caller is in a transaction mode, and the transaction is rolled back. If a caller is not in a transaction mode, a block timeout will occur if both a TPNOTIME and a TPNOBLOCK are not set. In such cases, the data in a *data and a *len are not changed. If a transaction timeout occurs, any new service requests and processes waiting for a response will fail due to a [TPETIME] error until the transaction is rolled back.
[TPESVCFAIL]	<p>A service routine sent a reply by calling a tpreturn() with a TPFAIL because an error occurred in an application program. Service reply data can be accessed through a *data. If a caller is in a transaction mode, the transaction will be rolled back.</p> <p>When a transaction timeout occurs, other communication may be attempted before a transaction is rolled back. For such attempts to be successful, a TPNOTRAN must be set. All operations performed while a caller is in a transaction mode are rolled back when the transaction is complete.</p>

Error Code	Description
[TPEBLOCK]	Blocking occurred while a TPNOBLOCK was set. A cd is valid.
[TPGOTSIG]	A signal was received while a TPSIGRSTRT was not set.
[TPEPROTO]	A tpcall() was called in an invalid state.
[TPETRAN]	A xa_start failed due to an error in the database while invoking a transaction service.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

```
#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();
}
```

- Related Functions

tpacall(), tpalloc(), tpreturn()

9.4.3. tpcancel

Cancels a server or client response. It cancels the caller description, cd, returned by a **tpacall()**.

- Prototype

```
# include <atmi.h>
int tpcancel (int cd)
```

- Parameters

Parameter	Description
cd	A target to be canceled, which is set by a caller descriptor returned by tpacall(). Services related to global transactions cannot be canceled. If a service response is canceled, cd is nullified and all responses received through cd are ignored.

- Return Values

Value	Description
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a tpcancel() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPEBADDESC]	An invalid cd.
[TPETRAN]	A cd is related to a caller's global transaction. It is still valid and the caller's current transaction is not affected.
[TPEPROTO]	A tpcanceled() was called from an invalid state.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

```
#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]);          /* Cancels the second response. */
    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();
}

```

- Related Functions

tpacall()

9.5. Interactive Communication

The following describes functions that are used for interactive communication.

9.5.1. tpconnect

Allows a program to communicate with an interactive service svc. Communication is half-duplex, which only allows one process to either receive or send a message but not both at the same time. During a connection configuration process, a function caller can deliver data to a service routine. Interactive services receives **data** and **len** using a TPVCINFO structure, so calling a **tprecv()** is unnecessary to receive data delivered by tpconnect(). This function can be used on both a client and a server.

- Prototype

```

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

```

- Parameters

Parameter	Description
svc	Service name of an interactive service.
data	A buffer must be allocated by a tpalloc() for a caller to deliver data.

Parameter	Description
len	<p>Length of data to be sent.</p> <ul style="list-style-type: none"> • If data points to a buffer type that does not require a specified length, such as a STRING, STRUCT, X_COMMON, or X_C_TYPE, a len will be ignored and 0 will be used. • If data points to a buffer type that requires a specified length, such as a X_OCTET, CARRAY, or MULTI STRUCTURE, a len cannot be 0. <p>If data is NULL, a len will be ignored and a service request will be received without data. The data type and subtype must be supported by a svc.</p>
flags	<p>The following values are available.</p> <ul style="list-style-type: none"> • TPNOTRAN <p>If a tpconnect() caller requests a svc service by setting a TPNOTRAN flag in a transaction mode, the svc service will be excluded from a transaction mode and then executed. If svc does not support transactions in a transaction mode and tpconnect() is called in transaction mode, a flag must be set to TPNOTRAN. If tpconnect() is called in a transaction mode, it is affected by a transaction timeout even if TPNOTRAN is set. If a service called with the flag set to TPNOTRAN fails, there is no effect on the current transaction of the caller.</p> • TPSENDONLY <p>After a connection is made, a caller can only send data and only the requested services can be performed. The caller gets the communication control first. The flag must be set to either TPSENDONLY or TPRECVONLY.</p> • TPRECVONLY <p>After a connection is made, a caller can only receive data and a requested service will send data, so at first, a requested service only has communication control. Either a TPSENDONLY or TPRECVONLY must be set.</p> • TPNOTIME <p>A function caller must wait indefinitely until a response is received. Blocking timeouts are ignored. If a tpacall() is called during a transaction timeout, a transaction timeout will be applied.</p> • TPSIGRSTRT <p>Allows signal interrupts. If a system function is interrupted by a signal, the system function will be executed again. If a signal interrupt occurs without this flag, the function will fail and a tperrno will be set to TPGOTSIG.</p>

- Return Values

Value	Description
Descriptor	A function call was successful and a descriptor used for a following connection is returned.
-1	A function call failed. A tperrno is set to an error code.

- Errors

If a function call fails, a caller's transaction will not be affected unless there is a specific direction. When a tpconnect() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPEINVAL]	An invalid parameter. For example, a svc is NULL, data points to a buffer that was not allocated by a tpalloc(), or a flag is invalid.
[TPENOENT]	Cannot request a service because a service corresponding to a svc does not exist.
[TPEITYPE]	A svc does not support a data type or subtype.
[TPELIMIT]	A maximum number of connections has been reached. Cannot request a service.
[TPETRAN]	A xa_start failed because a problem occurred in a database when a transaction service was requested.
[TPETIME]	<p>A timeout occurred. If a transaction timeout occurs when a function caller is in a transaction mode, the transaction will be rolled back. If the function caller is not in a transaction mode, a block timeout will occur if neither a TPNOTIME nor a TPNOBLOCK is set.</p> <p>In these cases, *data content and *len are not changed. If a transaction timeout occurs, new service requests and processes waiting for a response fail with the [TPETIME] error until the transaction is rolled back.</p>
[TPEBLOCK]	A blocking state occurred when a TPNOBLOCK was set.
[TPGOTSIG]	A signal was received when a TPSIGRSTRT was not set.
[TPEPROTO]	A tpconnect() was called from an invalid state.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

```
#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,TPRECVOONLY);
    if (cd==-1) { error processing }
    data process....

    ret=tprecv(cd, &buf, &len, TPNOFLAGS, revent);
    if (ret==-1) { error processing }
    tpfree(buf);
    tpend();
}

```

- Related Functions

tpalloc(), tpdiscon(), tprecv(), tpsend()

9.5.2. tpsend

Available in a server and a client. This function sends data to a partner program in an interactive communication. A caller must have communication control.

- Prototype

```

# include <atmi.h>
int tpsend (int cd, char *data, long len, long flags, long *revent)

```

- Parameters

Parameter	Description
cd	Sets a connection to receive data. It is a descriptor returned by a tpconnect() or a TPSVCINFO parameter.
data	A buffer allocated by a tpalloc() . If no application data is received (for example, only a communication control was passed without any data), data can be NULL. The data type and subtype must be recognizable by a connected partner.
len	A buffer length to receive. If data points to a buffer that does not need to be specified, a len will be ignored and 0 will be used. If data points to a buffer which must be specified, a len cannot be 0.

Parameter	Description
flags	<p>The following values are available.</p> <ul style="list-style-type: none"> • TPNOBLOCK <p>If a blocking situation occurs, for example if an internal buffer is filled with messages to be sent, data and events will not be sent. If a <code>tpsend()</code> is called without setting a TPNOBLOCK flag or if a blocking state occurs, a caller will wait until either a transaction or a block timeout occurs or a state is resolved.</p> <ul style="list-style-type: none"> • TPNOTIME <p>A function caller waits for a response and infinitely ignores a block timeout. If a <code>tpsend()</code> is used within a transaction timeout, the transaction time will still apply.</p> <ul style="list-style-type: none"> • TPRECVONLY <p>A caller sends a communication control to a partner after receiving data. A caller cannot call a <code>tpsend()</code> until it receives a communication control again. A communication partner will receive a TPEV_SENDOONLY event, which means a communication control will be received when receiving data using a <code>tprecv()</code>. A receiver cannot call a <code>tprecv()</code> until receiving a communication control from a partner.</p> <ul style="list-style-type: none"> • TPSIGRSTRT <p>Allows signal interrupts. If a system function is interrupted by a signal, the system function will be executed again. If a signal interrupt occurs without this flag, a function will fail and a <code>tperrno</code> will be set to TPGOTSIG.</p>
revent	<p>If an event exists for a descriptor <code>cd</code>, a <code>tpsend()</code> will fail and data will not be received. The event type will be returned as a <code>revent</code>.</p> <p>The events that can be delivered to <code>revent</code> are as follows:</p> <ul style="list-style-type: none"> • TPEV_DISCONIMM <p>A communication starter used a <code>tpdiscon()</code> to forcefully terminate a connection. This event is received by a communication subordinate and is returned when a connection is terminated due to a communication error such as server, node, or network error.</p> <ul style="list-style-type: none"> • TREV_SVCERR <p>This event is received by a communication starter and notifies that a communication subordinator performed a <code>tpreturn()</code> without a communication control being in a TPEV_SVCFAIL state.</p>

Parameter	Description
revent	<ul style="list-style-type: none"> TREV_SVCFAIL <p>This event is received by a communication starter and notifies that a communication subordinator performed a tpreturn() without a communication control, and that a tpreturn() was called as a TPFAIL without data and that it was performed as a TPFAIL rval and had data with a NULL value.</p>

- Return Values

If a revent is a TREV_SVCFAIL, a tpurcode global variable is set to a rcode value, which is delivered when calling a tpreturn().

Value	Description
0	A function call was successful.
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a tpsend() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPEINVAL]	An invalid parameter. For example, a svc is NULL, data points to a buffer that was not allocated by a tmalloc(), or a flag is invalid.
[TPEBADDESC]	An invalid cd.
[TPETIME]	<p>A timeout occurred. If a transaction timeout occurs when a function caller is in a transaction mode, a transaction will be rolled back. If a function caller is not in a transaction mode, a block timeout will occur if neither a TPNOTIME nor a TPNOBLOCK is set.</p> <p>In these cases, a *data content and a *len will not be changed. If a transaction timeout occurs, new service requests and processes waiting for a response will fail with a [TPETIME] error until a transaction is rolled back.</p>
[TPEEVENT]	An event occurred. If an error occurs, data will not be sent and an event type will be returned as a revent.
[TPEBLOCK]	A blocking state occurred when a TPNOBLOCK was set.
[TPGOTSIG]	A signal that is received when a TPSIGRSTRT was not set.
[TPEPROTO]	A tpsend() was called from an invalid state. For example, a tpsend() was called in a receiver mode.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

```
#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();
}
```

- Related Functions

tpalloc(), tpconnect(), tpdiscon(), tprecv(), tpreturn()

9.5.3. tprecv

Available in a server and a client. This function receives messages in an interactive communication. This is used to receive data sent from a partner in an interactive communication. A tprecv() can be used only by a program (on a server or a client) which does not have communication control.

- Prototype

```
# include <atmi.h>
int tprecv (int cd, char **data, long *len, long flags, long *revent)
```

- Parameters

Parameter	Description
cd	Sets a connection to receive data. It is a descriptor returned by a tpconnect() or a TPSVCINFO parameter.
data	Pointer to a buffer allocated by a tpalloc(). If a function successfully returns, a *data will point to received data.
len	<p>The length of data to be sent.</p> <p>If the value of a len is bigger than the total size of a buffer before calling it, the size of the buffer will be set to the value of len.</p> <p>If the value of a len is 0, no data will be received and neither *data nor a buffer, which a *data points to, will be changed.</p>
flags	<p>The following values are available.</p> <ul style="list-style-type: none"> TPNOCHANGE <p>If a received response buffer and a buffer type that a *data points to are not the same, the buffer type of a *data will be changed to a received response buffer type in a scope that a receiver can recognize. If a TPNOCHANGE flag is set, the buffer type that a *data points to will not be changed. A received response buffer type and subtype must be the same with those of the buffer that a *data points to.</p> TPNOBLOCK <p>Does not wait for data. If there is receivable data the data will be returned. If a TPNOBLOCK flag is not set, and there is no receivable data, a caller will wait for data.</p> TPNOTIME <p>A function caller waits infinitely until a response is received, and ignores block timeouts. If a tprecv() is used within a transaction timeout, a transaction time will be applied.</p> TPSIGRSTRT <p>Allows signal interrupts. If a system function is interrupted by a signal, the system function will be executed again. If a signal interrupt occurs without this flag, a function will fail and a tperrno will be set to TPGOTSIG.</p>

Parameter	Description
revent	<p>The event types returned to a revent are as follows:</p> <ul style="list-style-type: none"> • TPEV_DISCONIMM <p>A communication starter used a <code>tpdiscon()</code> to forcefully terminate a connection. This event is received by a communication subordinate. This event is also returned when a connection is terminated due to a communication error such as a server, node, or network error. Any data being sent may be lost. If two programs participate in the same transaction, the transaction will be rolled back. a <code>cd</code> used by an interactive communication will not be valid.</p> • TPEV_SENDOONLY <p>A connected partner program gave up on a communication control. A TPEV_SENDOONLY event receiver can send data but cannot receive any data until a receiver returns a control.</p> • TPEV_SVCERR <p>This event notifies a communication starter of an error that occurred while a communication subordinator executes a <code>tpreturn()</code>. The error can occur if an invalid parameter is passed to a <code>tpreturn()</code>, or a <code>tpreturn()</code> is called while a service maintains a connection to another subordinator. In these cases, a return code or a part of data cannot be used. An interactive connection will be terminated a <code>cd</code> will not be valid. If this event occurs during a receiver's transaction, the transaction will be rolled back.</p> • TPEV_SVCFAIL <p>This event notifies a communication starter of information that a service of a communication subordinator was terminated due to a failure. A <code>tpreturn()</code> was called using a <code>TPFAIL</code> as an argument. If a communication subordinator service has a communication control when calling a <code>tpreturn()</code>, a service cannot send data to a connected partner. A server process terminates an interactive connection when a service is terminated. Therefore, a <code>cd</code> will not be valid anymore. If this event occurs to a receiver transaction process, the transaction will be rolled back.</p> • TPEV_SVCSUCC <p>This event notifies a communication starter that a service of a communication subordinator was terminated successfully. A <code>tpreturn()</code> was called using a <code>TPSUCCESS</code> as an argument.</p>

- Return Values

If a revent value is TREV_SVCSUCC or TREV_SVCFAIL, a **tpurcode** global variable that is delivered by a tpreturn() will be defined by an application. Otherwise, a -1 will be returned and a tperrno will be set to the value corresponding to a state. If an event has no error, a tprecv() will return a -1 and a tperrno will be set to [TPEEVENT].

- Errors

When a tprecv() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPEBADDESC]	An invalid cd.
[TPEBLOCK]	A blocking state occurred when a TPNOBLOCK was set.
[TPEEVENT]	An event occurred. Use a revent to find the event type.
[TPEINVAL]	An invalid parameter. For example, a svc is NULL, a *data points to a buffer that was not allocated by a talloc(), or a flag is invalid.
[TPEOS]	An operating system error occurred.
[TPEOTYPE]	<p>The type or subtype of an entered buffer is unknown to a caller, or a TPNOCHANGE flag was set but the type or subtype of a buffer pointed to by a *data did not match the type or subtype of an entered buffer. In this case, both the contents of a *data and a *len are not changed. If interactive communication is a part of a transaction, the transaction will be rolled back because a reply will be ignored.</p> <p>If an error occurred, the event for a cd will be ignored and an interactive communication state cannot be guaranteed. Therefore a caller must terminate an interactive communication.</p>
[TPEPROTO]	A tprecv() was called from an invalid state. For example, a tprecv() was called in a sender mode.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPETIME]	<p>A timeout occurred. If a transaction timeout occurs when a function caller is in a transaction mode, a transaction will be rolled back. If a function caller is not in a transaction mode, a block timeout will occur if neither a TPNOTIME nor a TPNOBLOCK are set.</p> <p>In these cases, a *data content and a *len are not changed. If a transaction timeout occurs, new service requests and processes waiting for a response will fail with a [TPETIME] error until a transaction is rolled back.</p>
[TPGOTSIG]	A signal was received when a TPSIGRSTRT was not set.

- Examples

```
#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();
}

```

- Related Functions

tpalloc(), tpconnect(), tpdiscn(), tpsend()

9.5.4. tpdiscn

Terminates an interactive communications connection. If a service is connected by a **tpconnect()**, in extremely rare cases the connection will immediately terminate and a **TPEV_DISCONIMM** event will occur for the connection partner. This function can be used on both a client and a server.

A **tpdiscn()** can be called only by the side that started an interactive communication. The service that provides a descriptor cannot call a **tpdiscn()**. A program that communicates with an interactive service can terminate communication. To ensure correct results, a **tpreturn()** must be called to terminate the connection.

A **tpdiscn()** terminates a connection forcibly, which may result in data that did reach a target to become lost. It can be called while a connected program participates in a caller's transaction. In this case a transaction is canceled and data may be lost. A function caller does not need to have communication control.

- Prototype


```
# include <atmi.h>
int tpdicon (int cd)
```

- Parameters

Parameter	Description
cd	A descriptor returned by a tpconnect() .

- Return Values

Value	Description
-1	A function call failed. A tperno is set to an error code.

- Errors

When a tpdicon() fails to execute, a tperno will be set to one of the following values:

Error Code	Description
[TPEBADDESC]	An invalid cd, or a cd is already in use by an interactive service.
[TPETIME]	A timeout occurred due to an invalid cd.
[TPEPROTO]	A tpdicon() was called from an invalid state.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

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

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

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

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

    data process....
    cd=tpconnect("SERVICE",buf,0,TPRECVONLY);

    if (cd==-1) { error processing }
```

```

data process....
ret=tprecv(cd, (char **)&buf, &len, TPNOFLAGS, &revent);
if (ret==-1 && revent != TPEV_SENDOONLY && revent != TPEV_SVCSUCC)
{ error processing }
printf("received data = %s\n", buf);
if (atoi(buf)>90) {
    ret=tpdiscon(cd);
    if (ret==-1) {error processing }
    tpfree(buf);
    tpend();
    exit(1);
}

data process....
tpfree(buf);
tpend();
}

```

- Related Functions

tpconnect(), tprecv(), tpreturn(), tpsend()

9.6. Unsolicited Message Processing

These functions are used to send messages unilaterally or to process messages received unilaterally. Not all clients can always receive unsolicited messages (tpbroadcast()) sent by a server. To receive an unsolicited message, the client must be connected to Tmax. When connecting to Tmax, the client must notify the server that it can receive unsolicited messages.

For Tmax to handle data that is sent from a server unilaterally, the flags field of the TPSTART_T structure, which is used when calling tpstart(), must be set to TPUNSOL_POLL or TPUNSOL_HND.

9.6.1. tpsetunsol

Available in a client. This function sets a routine that processes unrequested and received messages. How a system receives unrequested messages is determined by each application and can be changed by each client.

Any unrequested messages received by Tmax libraries before a tpsetunsol() is called will be ignored. A tpsetunsol() that is called by a NULL function pointer is also ignored. The function pointer delivered by a call must be suitable for a parameter definition.

- Prototype

```

# include <atmi.h>
Unsolfunc *tpsetunsol (void ( *disp ) ( char *data, long len, long flags ) )

```

- Parameters

Parameter	Description
data	Points to a received type buffer. If there is no data, a DATA can be NULL. If a buffer type or subtype of a DATA are unknown by a client, data cannot be recognized. An application cannot delete a DATA, instead a system deletes it and nullify a data area to return.
len	Data length
flags	Not currently supported.

- Return Values

Value	Description
Pointer / NULL	<p>A function call was a success.</p> <ul style="list-style-type: none"> Pointer: Returns the pointer of a routine for an unrequested message process that was previously set. NULL: A message process function was not previously set. A return will be successful.
TPUNSOLERR	A function call failed. An error code will be set in tperrno.

- Errors

When a `tpsetunsol()` fails to execute, a `tperrno` will be set to one of the following values:

Error Code	Description
[TPEPROTO]	A <code>tpsetunsol()</code> was called from an invalid state.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information will be recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

```
#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();
}

```

- Related Functions

tpstart(), tpend(), tpgetunsol()

9.6.2. tpgetunsol

Processes a message that was received unilaterally without a client request. The message is sent through a **tpbroadcast()**, **tpsendtocli()**, **tppost()** from a sender.

All messages received before calling a tpgetunsol() will be ignored. To receive unrequested messages through a tpgetunsol(), a TPUNSOL_POLL or a TPUNSOL_HND must be set when connecting to a Tmax system through a tpstart(). If a tpgetunsol() is called from a program, a client will receive an unrequested message from a server because a flag has been changed to TPUNSOL_POLL internally even though the flag of a tpstart() was initially set to TPUNSOL_IGN.

- Prototype

```

#include <tmaxapi.h>
int tpgetunsol (int type, char **data, long *len, long flags)

```

- Parameters

Parameter	Description
type	A message type delivered from a server. The types are UNSOL_TPPOST, UNSOL_TPBROADCAST, and UNSOL_TPSENDTOCLI.
data	A pointer to a delivered message. If this is a buffer type or subtype that is not known by a client, "data" cannot be used.
len	The total length of a message.

Parameter	Description
flags	<p>Determines whether to block a message.</p> <p>The following values are available.</p> <ul style="list-style-type: none"> • TPBLOCK <p>A caller waits for a reply in a blocking state.</p> • TPNOCHANGE <p>In general, if a received response buffer and the buffer type pointed to by a *data do not match, the *data buffer type will be changed to a received response buffer in a scope that a receiver can recognize. If this is set, the buffer type pointed to by a *data cannot be changed. A received response buffer type and subtype must match those of a buffer pointed to by a *data.</p> • TPNOTIME <p>A function caller must wait indefinitely until a response is received. Blocking a timeout will be ignored. If a tpgetrply() is called during a transaction timeout, a transaction timeout will be applied.</p> • TPSIGRSTRT <p>Allows signal interrupts. If a system function is interrupted by a signal, the system function will be executed again. If a signal interrupt occurs without this flag, a function will fail and tperrno will be set to TPGOTSIG.</p> • TPGETANY <p>Ignores a cd as an input value and returns any receivable responses. A cd will become a caller descriptor for a returned response. If there is no response, a tpgetrply() will wait until a response is received.</p>

- Return Values

Value	Description
1	A function call was successful.
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a tpgetunsol() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPEPROTO]	A tpgetunsol() was called from an invalid state. For example, it was called from a server.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information will be recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

```
#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();
}
```

- Related Functions

tpbroadcast(), tpsetunsol(), tpstart(), tpend()

9.7. Timeout Change

The following describes functions that are used to change timeout.

9.7.1. tpset_timeout

Available in a server and a client. This function sets a block timeout, a service limited time set in a server. If a timeout is set using a tpset_timeout(), the response for a service request will wait during the specified amount of time. If a response is not received during the specified amount of time, a timeout error will occur and the service request will be returned as a failure without waiting for a response.

A tpset_timeout() applies to service requests that were received after this function was called. The function is valid until a tpset_timeout() is called again, or a client or server process is terminated. If a tpset_timeout() is not used, a **BLOCKTIME** set in a Tmax configuration file will be used as a block timeout.

- Prototype

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

- Parameters

Parameter	Description
sec	Sets a block time in seconds.

- Return Values

Value	Description
0	A function call was successful.
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a tpset_timeout() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

```

#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

Available in a server. This function sets a service timeout in a server. If a service time is set using a `tpsetsvctimeout()`, a service request will wait for a response during the time set by this function. If a response was not received during the set time, a timeout error will occur and a service request will be returned as a service failure without waiting for a response.

- Prototype

```

#include <tmaxapi.h>
int tpsetsvctimeout (int sec, long flags)

```

- Parameters

Parameter	Description
sec	Sets a service timeout in seconds.
flags	Not currently supported.

- Return Values

Value	Description
0	A function call was successful.
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a tpsetsvctimeout() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

```
#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. Buffer Management

Because memory allocation methods and data types differ according to hardware and operating system, received data may have different data values. Therefore, in a general client/server environment, heterogeneous systems communicate through conversion, which creates network overhead.

The following describes APIs that are used for buffer management.

9.8.1. talloc

Allocates a typed buffer. A buffer cannot be set by malloc(), realloc(), or free() of C library. For example, the buffer allocated with talloc() cannot be deleted through free().The talloc function allocates a typed buffer in servers and clients. It cannot be used along with malloc(), realloc(), or free() of C library. For example, a buffer allocated by talloc() cannot be released by free().In this case, **tfree()** must be used.

The talloc function allocates a buffer of the type specified by "type", and it returns the pointer to the buffer. The buffer subtype and size can also be specified for some types. Some types of buffers require initialization, so talloc() initializes and returns pointers to these buffers. The buffer returned to the caller can be used immediately. If talloc() fails to initialize the buffer to 0, the allocated buffer becomes free.

- Prototype

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

- Parameters

Parameter	Description
type	Buffer type. Input options: <ul style="list-style-type: none">• STRING: sends string-type data that ends with NULL.• CARRAY, X_OCTET: sends character-type data with a specified length.• STRUCT, X_C_TYPE: sends C-language structure type data.• X_COMMON: is used for C structures that allow only char, int, and long data types.• FDL (FIELD buffer): stores data with an identifier and a corresponding value.
subtype	For STRUCT, X_C_TYPE, and X_COMMON type buffers, a subtype must be set. Only the first 8 bytes of type and the first 16 bytes of subtype are used. Any remaining data is truncated. If the specified buffer type does not use a subtype, the subtype is ignored and NULL is used. The default allocated buffer size is greater than 1024 bytes.
size	If type is CARRY or X_OCTET, size must be specified. size is not required for other buffer types. If size is 0, the default size of each buffer is used. The default size of STRING, STRUCT, X_C_TYPE, X_COMMON is 1024 bytes. The default size of CARRY is 0, but the size must be greater than 0 when allocating a buffer.

- Return Values

Value	Description
Buffer Pointer	A function call was successful and a proper buffer pointer is returned.
NULL	A function call failed. A tperrno is set to an error code.

- Errors

When a tpalloc() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPEINVAL]	An invalid parameter. For example, a type is NULL.
[TPENOENT]	Unknown type or subtype. For example, in a STRUCT buffer type the subtype (tag name of the structure) does not exist in SDLFILE.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred. Memory cannot be allocated.
[TPEOTYPE]	Buffer type is STRUCT but the server is compiled without a structure file.

- Examples

```
#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();
}
```

- Related Functions

tpfree(), tprealloc(), tptypes()

9.8.2. tprealloc

Reallocates a buffer pointed to by a ptr in bytes, and returns a pointer for a new buffer (when a buffer is changed).

Like a tmalloc(), a buffer size must be larger than the default size (1024 bytes). A buffer type is kept the same even after it is reallocated. When a function successfully returns a pointer, the returned pointer is used to refer to a reallocated buffer. At this time, a ptr cannot be used any more. If a reallocated buffer size is smaller than a previous buffer size, the original ptr content cannot be ensured.

Some buffer types need to be initialized to be used. A tprealloc() re-initializes a re-allocated buffer and then returns it. Therefore, a buffer returned to a caller can be used immediately. If it fails to reinitialize a buffer, a tprealloc() will return a NULL and data in a buffer pointed to by a ptr will not be valid.

- Prototype

```
# include <atmi.h>
char * tprealloc (char *ptr, long size)
```

- Parameters

Parameter	Description
ptr	A pointer to a buffer to be allocated.
size	The size of a buffer to be allocated.

- Return Values

Value	Description
Buffer Pointer	A function call was successful and a proper buffer pointer is returned.
NULL	A function call failed. A tperno is set to an error code.

- Errors

When a tprealloc() fails to execute, a tperno will be set to one of the following values:

Error Code	Description
[TPEINVAL]	An invalid parameter. For example, a buffer pointed to by a ptr is not the one allocated by a tmalloc().
[TPEPROTO]	A tprealloc() was called from an invalid state.
[TPENOENT]	A buffer pointed to by a ptr was not allocated by a tmalloc().
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.

Error Code	Description
[TPEOS]	An operating system error occurred.

- Examples

```
#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 }
}
```

- Related Functions

tpalloc(), tpfree(), tptypes()



This function cannot be used together with a **malloc()**, **realloc()**, or **free()** from a C library. For example, it is impossible to free a buffer allocated by a **tprealloc()** by using a **free()**.

9.8.3. tpfree

Releases memory that is allocated to a typed buffer by a **tpalloc()** or a **tprealloc()**. This function can be used on both a client and a server.

- Prototype

```
# include <atmi.h>
void tpfree(char *ptr)
```

- Parameters

Parameter	Description
ptr	<p>A pointer to a buffer allocated by a tpalloc() or a tprealloc().</p> <p>If a ptr is NULL, nothing will happen. If a ptr points to a non-typed buffer or a buffer already released by a tpfree(), nothing will happen.</p>

Parameter	Description
ptr	<p>If a ptr points to a buffer sent to a service routine, a tpfree() will not free the buffer but instead will return it as it is. To free some buffer types, related data or status information must first be removed. A tpfree() removes related information before freeing these types of buffers.</p> <p>Once a tpfree() returns, a ptr cannot be transferred to a XATMI routine as a parameter, and it cannot be used in any other way.</p>

- Return Values

A tpfree() function does not return any value to a function caller.

- Examples

```
#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();
}
```

- Related Functions

tpalloc(), tprealloc()



This function cannot be used together with a malloc(), a realloc(), or a free() from a C library. For example, it is impossible to free a buffer allocated by a tpalloc() by using a free().

9.8.4. tptypes

Provides information about the type and subtype of a buffer. This function is available both in a server and a client. A tptypes() function receives a pointer for a data buffer, and returns the type and subtype of the buffer.

- Prototype

```
#include <atmi.h>
long tptypes (char *ptr, char *type, char *subtype)
```

- Parameters

Parameter	Description
ptr	Must indicate a buffer allocated by a tpalloc().
type, subtype	Types and subtypes will have type and subtype names of each buffer respectively for the string they indicate if they are not NULL. If names have a maximum length (8 characters for a type, and 16 characters for a subtype), a string could not end with a NULL. If there is no subtype, the arrangement indicated by a subtype will contain a NULL string. Only the first 8 bytes of a type and 16 bytes of a subtype have valid values.

- Return Values

Value	Description
Buffer Size	A function call was successful and a proper buffer pointer is returned.
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a tptypes() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPEINVAL]	An invalid parameter. For example, a buffer pointed to by a ptr is not allocated by a tpalloc().
[TPEPROTO]	A tptypes() was called from an invalid state.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

```
#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();
}

```

- Related Functions

tpalloc(), tpfree(), tprealloc()

9.9. Transaction Management

The following describes functions that are used for transaction management.

9.9.1. tx_begin

Begins a global transaction, for which a function caller becomes a transaction mode. This function is available for both servers and clients. To start a transaction, a calling process must be first connected to a resource manager with a **tx_open()**, then, it will return a [TX_PROTOCOL_ERROR]. A tx_begin() function will fail if a caller is already in a transaction mode or a tx_open() is not called first.

Once a transaction begins, a calling process must call a **tx_commit()** or a **tx_rollback()** in order to complete a current transaction. There are also chaining transactions that do not need to call a tx_begin() directly in order to begin. For further information, see [tx_commit](#) and [tx_rollback](#).

- Prototype

```

#include <tx.h>
int tx_begin (void)

```

- Return Values

Value	Description
TX_OK	A function call was successful.
Negative Value	A function call failed.

- Errors

When a tx_begin() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TX_OUTSIDE]	A transaction manager cannot start a global transaction because a current calling process is participating in external global transactions. The global transaction can be started after all outside jobs are complete. This error does not affect the participating transactions.
[TX_PROTOCOL_ERROR]	A tx_begin() was called from an invalid state. For example, a caller is already in a transaction mode. This error does not affect current transactions.
[TX_ERROR]	A transaction manager or a resource manager momentarily encountered an error while starting a transaction. When this error is returned, a caller is not in a transaction mode. Exact causes of the error depend on product characteristics.
[TX_FAIL]	A transaction manager or a resource manager encountered a critical error. The manager can no longer execute jobs for an application program. When this error is returned, a caller is not in a transaction mode. Exact causes of the error differ depending on product characteristics.

- Examples

```
#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();
}

```

- Related Functions

tx_commit(), tx_open(), tx_rollback(), tx_set_transaction_timeout()

9.9.2. tx_commit

Commits a global transaction. This function is available for both servers and clients.

When a transaction_control property value is TX_UNCHAINED, a caller will not be in a transaction mode anymore when a tx_commit() returns. However, if the value is TX_CHAINED, the caller will remain in the transaction mode for a new transaction when the tx_commit() returns. For detailed information, see [tx_set_transaction_control](#).

- Prototype

```

# include <tx.h>
int tx_commit(void)

```

- Return Values

Value	Description
TX_OK	A function call was successful.
Negative Value	A function call failed. A tperrno is set to an error code.

- Errors

When a tx_commit() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TX_NO_BEGIN]	Occurs only when a transaction_control property is a TX_CHAINED. A transaction was successfully committed, but new transactions cannot be started, and a caller will no longer be in a transaction mode.
[TX_ROLLBACK]	A transaction is rolled back. If a transaction_control property is TX_CHAINED, a new transaction will begin.
[TX_ROLLBACK_NO_BEGIN]	Occurs only when a transaction_control property is TX_CHAINED. A transaction is rolled back, but new transactions cannot be started, and a caller is no longer in a transaction mode.
[TX_HAZARD]	Portions of a transaction may be committed while others may be rolled back due to an error. If a transaction_control property is TX_CHAINED, a new transaction will be started.
[TX_HAZARD_NO_BEGIN]	Occurs only when a transaction_control property is TX_CHAINED. Portions of a transaction are committed while others are rolled back. A new transaction cannot be started and a caller will no longer be in a transaction mode.
[TX_PROTOCOL_ERROR]	A tx_commit() was called from an invalid state. For example, a caller was not in a transaction mode. a caller's status is not affected by the transaction.
[TX_FAIL]	A transaction manager or a resource manager encountered a critical error. A manager can no longer execute jobs for an application program. Exact causes of the error differ depending on product characteristics. The caller's status related to the transaction is unknown.

- Examples

```
#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();
}

```

- Related Functions

tx_begin(), tx_set_commit_return(), tx_set_transaction_control(), tx_set_transaction_timeout()

9.9.3. tx_info

Returns global transaction information. This function is available for both servers and clients. It notifies global transaction information through a structure indicated by info. It also returns a value informing whether a caller is in a transaction mode or not.

- Prototype

```

#include <tx.h>
int tx_info (TXINFO *info)

```

- Parameters

If info is not NULL, a TXINFO structure indicated by info will be global transaction information.

A TXINFO structure is composed as follows:

```

struct TXINFO {
    XID                xid;
    COMMIT_RETURN      when_return;
    TRANSACTION_CONTROL transaction_control;
    TRANSACTION_TIMEOUT transaction_timeout;
    TRANSACTION_STATE  transaction_state;
};

```

If a tx_info() is called in a transaction mode, a xid will be the current transaction branch ID, and a transaction_state will be the current transaction state. If a caller is not in a transaction mode, a

xid will be a NULL XID (See <tx.h> for further information).

In addition, regardless of whether a caller is in transaction mode or not, when_return, transaction_control, and transaction_timeout will contain the current settings of a commit_return, transaction_control property, and a transaction timeout value in seconds.

A returned transaction timeout value is used when a next transaction begins. It may not be a timeout value for a caller's current global transaction, because the caller might have changed the transaction timeout by calling a tx_set_transaction_timeout() after a current transaction began. If info is NULL, a TXINFO structure will not be returned.

Continuous tx_info() calling in the same global transaction ensures provision of a XID of the same gtrid (global transaction identifier). However, it does not ensure a same bqual (local transaction identifier). Thus, XIDs may not be the same.

- Return Values

Value	Description
1	A caller is in a transaction mode and a function call was successful.
0	A caller is not in a transaction mode and a function call was successful.
Negative Value	A function call failed. A tperrno is set to an error code.

- Errors

When a tx_info() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TX_PROTOCOL_ERROR]	A tx_info() was called from an invalid state. For example, a tx_open() was not yet called.
[TX_FAIL]	A transaction manager encountered a critical error. The manager can no longer execute jobs for an application program. Exact causes of the error differ depending on product characteristics.

- Examples

```
#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();
}

```

- Related Functions

tx_open(), tx_set_commit_return(), tx_set_transaction_control(), tx_set_transaction_timeout()

9.9.4. tx_rollback

Used to roll back a global transaction. This function is available for both servers and clients.

If a transaction_control property is TX_UNCHAINED, a caller will not be in a transaction mode any more when a tx_rollback() returns. If a transaction_control property is TX_CHAINED, however, a caller will remain in a transaction mode for a new transaction when a tx_rollback() returns. For detailed information about a transaction_control property, see [tx_set_transaction_control](#).

- Prototype

```

#include <tx.h>
int tx_rollback(void)

```

- Return Values

Value	Description
TX_OK	A function call was successful.
Negative Value	A function call failed. A tperno is set to an error code.

- Errors

When a tx_rollback() fails to execute, a tperno will be set to one of the following values:

Error Code	Description
[TX_NO_BEGIN]	Occurs only when a transaction_control property is TX_CHAINED. New transactions cannot be started, and a caller is no longer in a transaction mode.
[TX_MIXED]	Portions of a transaction may be committed while others may be rolled back due to an error. If a transaction_control property is TX_CHAINED, a new transaction will be started.
[TX_MIXED_NO_BEGIN]	Occurs only when a transaction_control property is TX_CHAINED. Portions of a transaction are committed while others are rolled back. A new transaction cannot be started and a caller will no longer be in a transaction mode.
[TX_HAZARD]	Portions of a transaction may be committed while others may be rolled back due to an error. If a transaction_control property is TX_CHAINED, a new transaction will be started.
[TX_HAZARD_NO_BEGIN]	Occurs only when a transaction_control property is TX_CHAINED. Portions of a transaction are committed while others are rolled back. A new transaction cannot be started and a caller will no longer be in a transaction mode.
[TX_COMMITTED]	A transaction was committed independently. If a transaction_control property is TX_CHAINED, a new transaction will be started.
[TX_COMMITTED_NO_BEGIN]	Occurs only when a transaction_control property is TX_CHAINED. A transaction was successfully committed, but new transactions cannot be started, and a caller will no longer be in a transaction mode.
[TX_PROTOCOL_ERROR]	A tx_rollback() was called from an invalid state. For example, a caller was not in a transaction mode. A caller's status is not affected by a transaction.
[TX_FAIL]	A transaction manager or a resource manager encountered a critical error. A manager can no longer execute jobs for an application program. Exact causes of an error differ depending on product characteristics. A caller's status related to a transaction is unknown.

- Examples

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

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

```

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();
}

```

- Related Functions

tx_begin(), tx_set_transaction_control(), tx_set_transaction_timeout()

9.9.5. tx_set_transaction_timeout

Sets a transaction_timeout property for a timeout value. This function is available for both servers and clients. This value indicates the time in which a transaction must be completed before a transaction timeout occurs. In other words, it is the interval between a **tx_begin()** and a **tx_commit()**, or a **tx_begin()** and a **tx_rollback()**.

A tx_set_transaction_timeout() function can be called regardless of whether a function caller is in a transaction mode or not. If a tx_set_transaction_timeout() is called in a transaction mode, a new timeout value will be applied from a following transaction.

- Prototype

```

# include <tx.h>
int tx_set_transaction_timeout (TRANSACTION_TIMEOUT timeout)

```

- Parameters

Parameter	Description
timeout	Specifies a time allowed until a transaction timeout occurs, in seconds. It can be set up to a maximum value of a long type defined for each system.

Parameter	Description
timeout	Default setting of a transaction_timeout is 0, which means there is no timeout limit.

- Return Values

Value	Description
TX_OK	A function call was successful.
Negative Value	A function call failed. A tperrno is set to an error code.

- Errors

When a tx_set_transaction_timeout() fails to execute, tperrno will be set to one of the following values and a transaction_timeout value will be unchanged.

Error Code	Description
[TX_EINVAL]	An invalid timeout value.
[TX_PROTOCOL_ERROR]	A tx_set_transaction_timeout() was called from an invalid state. For example, a tx_open() was not yet called.
[TX_FAIL]	A transaction manager encountered a critical error. A manager can no longer execute jobs for an application program. Exact causes of the error differ depending on product characteristics.

- Examples

```
#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();
}

```

- Related Functions

tx_begin(), tx_commit(), tx_open(), tx_rollback(), tx_info()

9.9.6. tx_set_transaction_control

Sets a transaction_control property with a control value. This function is available for both servers and clients.

This property determines whether to or not start a new transaction before a **tx_commit()** and a **tx_rollback()** returns to a caller. tx_set_transaction_control() can be called regardless of whether an application program is in a transaction mode or not. This setting is valid until it is changed as tx_set_transaction_control() is called again.

- Prototype

```

# include <tx.h>
int tx_set_transaction_control(TRANSACTION_CONTROL control)

```

- Parameters

Values available for control are as follows:

Configuration Value	Description
TX_UNCHAINED	The initial value of transaction_control property. Does not start a new transaction until a tx_commit() or a tx_rollback() is returned to a caller. In this case, a caller must execute a tx_begin() in order to begin a new transaction.
TX_CHAINED	Allows a new transaction to start before a tx_commit() or a tx_rollback() is returned to a caller.

- Return Values

Value	Description
TX_OK	A function call was successful.

Value	Description
Negative Value	A function call failed. A tperrno is set to an error code.

- Errors

When `tx_set_transaction_control()` fails to execute, `tperrno` will be set to one of the following values and a previous `transaction_control` property will be unchanged.

Error Code	Description
[TX_EINVAL]	A control parameter is neither TX_UNCHAINED nor TX_CHAINED.
[TX_PROTOCOL_ERROR]	A <code>tx_set_transaction_control()</code> was called from an invalid state. For example, a <code>tx_open()</code> was not yet called.
[TX_FAIL]	A transaction manager encountered a critical error. The manager can no longer execute jobs for an application program. Exact causes of the error differ depending on product characteristics.

- Examples

```
#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();
}

```

- Related Functions

tx_begin(), tx_commit(), tx_open(), tx_rollback(), tx_info()

9.9.7. tx_set_commit_return

Sets a commit_return property with a when_return value. This function is available for both servers and clients.

This property determines the method how a tx_commit() function returns a control to a caller. A tx_set_commit_return() can be called regardless of whether a function caller is in a transaction mode. This setting is valid until it is changed as a tx_set_commit_return() is called again. Initial setting of a commit_return property depends on the situation when the function is executed.

- Prototype

```

#include <tx.h>
int tx_set_commit_return (COMMIT_RETURN when_return)

```

- Parameters

Values available for when_return are as follows:

Configuration Value	Description
TX_COMMIT_DECISION_LOGGED	This flag causes a tx_commit() to return before the second phase of a Two-Phase Commit (2PC) protocol is completed after it was logged in a first phase. This allows a tx_commit() to respond to a caller more quickly. However, there is a danger that a transaction may have a heuristic result. In this case, a caller cannot understand the situation generated by the code returned from a tx_commit().

Configuration Value	Description
TX_COMMIT_DECISION_LOGGED	In normal cases, a transaction participant scheduled to commit a transaction in a first phase can normally commit a transaction in a second phase. In abnormal cases such as a network failure or a node fail lasting too long, it may be impossible to complete a second phase and a heuristic result may be resulted. A transaction manager can select this property not to be supported, using a parameter. At this time, a tx_commit() returns [TX_NOT_SUPPORTED], indicating that this property is not supported.
TX_COMMIT_COMPLETED	This flag causes a tx_commit() to return after a 2PC protocol is completed. This setting shows a function caller return code indicating that a transaction produced a heuristic result or a probability. A transaction manager can select this property not to be supported, using a parameter. At this time a tx_commit() returns a [TX_NOT_SUPPORTED], indicating that this property is not supported.

- Return Values

When successfully completed, a tx_set_commit_return() returns a [TX_OK] of a value other than a negative value.

If a when_return is set to a TX_COMMIT_COMPLETED or a TX_COMMIT_DECISION_LOGGED, a function will return a [TX_NOT_SUPPORTED] instead of a negative value and a previously used commit_return property will still be valid. A transaction manager must set a when_return to either a TX_COMMIT_COMPLETED or a TX_COMMIT_DECISION_LOGGED.

- Errors

When a tmax_chk_conn() fails to execute, a tperrno will be set to one of the following values, and a commit_return property will be unchanged.

Error Code	Description
[TX_EINVAL]	A when_return is not set to TX_COMMIT_COMPLETED or TX_COMMIT_DECISION_LOGGED.
[TX_PROTOCOL_ERROR]	A tx_set_commit_return() was called from an invalid state. For example, a tx_open() was not yet called.
[TX_FAIL]	A transaction manager or a resource manager encountered a critical error. The managers could no longer execute jobs for an application program. Exact causes of the error differ depending on product characteristics. A caller's status, related to the transaction, is unknown.

- Examples

```

#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();
}

```

- Related Functions

tx_commit(), tx_open(), tx_info()

9.10. RQ System

This section describes functions that are used by RQ. The Tmax system provides the RQ system for reliable queues. ENQSVR works as a queue management process to enqueue/dequeue data to/from RQ. This function cannot be used in Windows NT or 2000.

Even when services cannot be provided due to a system failure or error, data stored in the queue can guarantee its integrity. If multiple servers are established in a WAN environment or the load is large, RQ can be used to process applications with guaranteed data integrity.

9.10.1. tpenq

The Tmax system ensures the integrity of data stored in a RQ even when a system fault or error causes an out of service status. A tpenq() saves data to a RQ. If a system crashes, it will continue processing data once the system recovers.

If a service is requested through a **tpcall()** or a **tpacall()**, requests will be queued if there are other requests waiting to be processed. If a system error or failure causes a system shutdown, queued data will be lost. To prevent data loss and to improve data consistency, a tpenq() saves data in a RQ for each service request.

Although a tpenq() is executed in a transaction mode, it is excluded from transaction modes. If an error occurs while this function is being executed in a transaction mode, a transaction will not be affected.

- Prototype

```
# include <tmaxapi.h>
int tpenq (char *qname, char *svc, char *data, long len, long flags)
```

- Parameters

Parameter	Description
qname	A RQ that saves data. A name must be specified and registered in a configuration file.
svc	Stores data to a RQ and immediately requests a service unless a svc name is NULL. If the svc name is NULL, data is stored in the RQ but a service will not be performed. In this case, a function caller must request the service again using a tpdeq(). If a system fault occurs when there is no service named svc or a service result has not been received after a service was performed, the data will be stored internally in a Fail Queue. This data must be re-requested using a tpdeq() or processed as an error.
data	This is a pointer to a buffer allocated by a tpalloc() unless the value is NULL,. The type and subtype of data must be supported by a svc.

Parameter	Description
len	<p>The length of data to be sent.</p> <ul style="list-style-type: none"> • If data points to a buffer type that does not require a specified length, such as a STRING, STRUCT, X_COMMON, or X_C_TYPE, a len will be ignored and 0 will be used. • If data points to a buffer type that requires a specified length, such as X_OCTET, CARRAY, or MULTI STRUCTURE, a len cannot be 0. • If data is NULL, len is ignored and a service request will be received without data.
flags	<p>The following values are available.</p> <ul style="list-style-type: none"> • TPRQS <p>When a svc is not NULL, a function caller will request a svc service and store service results in a RQ. To receive service results, a tpdeq() must be called. If a svc is NULL, data will be stored in a RQ but a service will not be performed.</p> • TPNOREPLY <p>When a svc is not NULL, a function caller will request a svc service and will not store results in a RQ. If a svc is NULL, data will be stored in a RQ but a service will not be performed.</p> • TPFUNC <p>Manages RQ data by service. If a TPFUNC is not set, data stored by a tpenq() will be de-queued when it is initially stored. If data must be de-queued before it is stored, a TPFUNC must be set when calling a tpenq().</p> • 0 (zero) <p>Stores service results to a client buffer of a Tmax system that is connected to a function caller. Service is requested through a RQ, but the results are saved to a client's buffer similarly to a tpcall() function. Once this flag is set, a 0 (zero) must also be set when calling a tpdeq() in order to receive service results.</p>

- Return Values

Value	Description
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a tpenq() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPEINVAL]	An invalid parameter. For example, a svc is NULL, data points to a buffer that was not allocated by a tpalloc(), or a flag is invalid.
[TPENOENT]	A qname does not exist.
[TPEQFULL]	A maximum queue length reached due to continuous service results.
[TPGOTSIG]	A signal that is received when a TPSIGRSTRT is not set.
[TPEPROTO]	A tpenq() was called from an invalid state.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

```
#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();
}
```

- Related Functions

tpdeq(), tpqstat()

9.10.2. tpdeq

Loads data from a RQ. It receives the result of a service request or sets a service name to NULL and then reads stored data by using a **tpenq()** function. However, if a TPNOREPLY is set when a tpenq() is called, a service cannot receive any results, so even if an error occurs while executing a tpdeq() in a transaction mode, a transaction will not be affected.

- Prototype

```
# include <tmaxapi.h>
int tpdeq (char *qname, char *svc, char **data, long *len, long flags)
```

- Parameters

Parameter	Description
qname	A RQ that saves data. A name must be specified and registered in a configuration file.
svc	<p>A svc must be set to a service name when a tpenq() is called. In other words, if a tpenq() is called with a service name, a service will automatically be requested and the results will be stored in a RQ. In order to receive service results, the same service name must be specified.</p> <p>If the service name is NULL when a tpenq() is called, the same service name must be entered for a tpdeq_ctl(). If the service name is NULL, all data accumulated in a queue can be loaded individually regardless of the service name.</p> <p>To call a tpdeq() for data saved in a fail queue due to an error or a system failure, a svc must be set to _rq_sub_queue_name[TMAX_FAIL_QUEUE].</p>
*data	A pointer to a buffer allocated by a tpalloc(). When a function is successfully returned, received data will be stored in a *data.
len	<p>Size of the data received by a tpdeq(). A tpdeq() can increase a buffer size if a reply is larger than the buffer.</p> <p>Returned results are stored in a *data, and a len will be the size of received data. A *data can be modified if the size of received data is too large. If a len is larger than the total size of a previously called buffer, a larger buffer size will be allocated. If a len returns a 0, no data will be received and nothing will be changed in the buffer indicated by a *data and a len.</p> <p>If a *data or a len is NULL, an error will occur.</p>
flags	<p>The following values are available.</p> <ul style="list-style-type: none">• TPRQS <p>Loads service results from a reply queue (RQ).</p>

Parameter	Description
flags	<ul style="list-style-type: none"> • TPFUNC Manages RQ data by service. If a TPFUNC is not set, data stored by a tpenq() will be dequeued when it is initially stored. If data must be dequeued before it is stored, a TPFUNC must be set when calling a tpenq(). • TPBLOCK Waits until a message is returned during a block timeout. • TPNOTIME If a TPNOTIME and a TPBLOCK are both set, a function will wait until a reply is returned regardless of a block timeout. • 0 (zero) Loads data from the buffer of a client that a function caller is connected to.

- Return Values

Value	Description
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a tpdeq() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPEINVAL]	An invalid parameter. For example, a svc is NULL, data points to a buffer that was not allocated by a talloc(), or a flag is invalid.
[TPGOTSIG]	A signal that is received when a TPSIGRSTRT was not set.
[TPEMATCH]	When a service name is wrong or there is no data to be dequeued, no data will be found for a given key.
[TPENOENT]	An invalid qname.
[TPEPROTO]	A tpdeq() was called from an invalid state.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

```

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

void main(int argc, char *argv[])
{
    int ret;
    char *buf;
    long len;
    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

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

    ret=tpenq("RQ", "SERVICE", (char *)buf, strlen(buf), TPRS);
    if (ret==-1) { error processing }
    data process....

    ret=tpdeq("RQ", "SERVICE", (char **)&buf, (long *)&len, TPRS);
    if (ret==-1) { error processing }
    data process....

    tpfree(buf);
    tpend();
}

```

- Related Functions

tpenq(), tpqstat()

9.10.3. tpqstat

Used to get the data statistics accumulated in a current RQ. This function is available both in a server and a client. A RQ internally consists of three queues: a _fail queue, a _request queue, and a _reply queue. A tpqstat () can get the data statistics stored in each of these queues, by using a flag value.

- Prototype

```

# include <tmaxapi.h>
int tpqstat (char *qname, long flags)

```

- Parameters

Parameter	Description
qname	A RQ that saves data. A name must be specified and registered in a configuration file.

Parameter	Description
flags	<p>A target data type.</p> <p>The following values are available.</p> <ul style="list-style-type: none"> 0(TMAX_ANY_QUEUE): Retrieves the statistics about all data stored in a _fail queue, _request queue, and _reply queue. 1(TMAX_FAIL_QUEUE): Retrieves the data statistics stored in a _fail queue. 2(TMAX_REQ_QUEUE): Retrieves the data statistics stored in a _request queue. 3(TMAX_RPLY_QUEUE): Retrieves the data statistics stored in a _reply queue.

- Return Values

Value	Description
-1	A function call was successful. A tperrno is set to an error code.

- Errors

When a tpqstat() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPEINVAL]	An invalid parameter. For example, when a qname is NULL, there will be no queue with a corresponding name, or a flag will be invalid.
[TPEPROTO]	A tpqstat() was called from an invalid state.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

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

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

    if (argc!=2) { error processing }
    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) {error processing }

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

```

strcpy(buf, argv[1]);

data process...

ret=tpenq("RQ", NULL, (char *)buf, strlen(buf), TPRS);
if (ret==-1) {error processing }
printf("qstat :");
for (i=0;i<4;i++) {
    ret=tpqstat("rq", i);
    if (ret==-1) {error processing }
    printf("  %d",ret);          /* qstat :  1  0  0  1 */
}
printf("\n");
data process...

ret=tpenq("RQ", "SERVICE", (char *)buf, strlen(buf), TPRS);
if (ret==-1) {error processing }
printf("qstat :");

for (i=0;i<4;i++) {
    ret=tpqstat("rq", i);
    if (ret==-1) {error processing }
    printf("  %d",ret);          /* qstat :  2  0  0  2 */
}
printf("\n");
tpfree((char *)buf);
tpend();
}

```

- Related Functions

tpenq(), tpdeq()

9.10.4. tpextsvcname

Extracts a service name from the data read from a RQ by using a **tpdeq()**. This function is generally used to read data stored in a Fail Queue and can be used on both a client and a server.

- Prototype

```

# include <tmaxapi.h>
int tpextsvcname (char *data, char *svc)

```

- Parameters

Parameter	Description
data	A pointer allocated by a tmalloc() that stores read data from a RQ through a tpdeq().
svc	A pointer that receives the service name of data.

- Return Values

Value	Description
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a `tpextsvcname()` fails to execute, a `tperrno` will be set to one of the following values:

Error Code	Description
[TPEINVAL]	An invalid parameter. For example, a <code>svc</code> is <code>NULL</code> , <code>data</code> points to a buffer that was not allocated by a <code>tpalloc()</code> , or a flag is invalid.
[TPESYSTEM]	A <code>Tmax</code> system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

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

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

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

    ret=tpenq("RQ", "SERVICE", (char *)buf, strlen(buf), TPRQS);
    if (ret==-1) { error processing }
    data process....

    ret=tpdeq("RQ", "SERVICE", (char **)&buf, (long *)&len, TPRQS);
    if (ret==-1) { error processing }

    ret=tpextsvcname(buf, svc_name);
    if (ret==-1) { error processing }
    printf("svc name : %s    ",svc_name);
    data process....

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

- Related Functions

tpenq(), tpdeq()

9.11. Functions using Events

Tmax can send messages to service routines and multiple clients. Developers can specify any client and service routine to register, cancel, or generate an event. Client and service routines can register in each event redundantly or register in multiple events simultaneously.



Occurred events' data and results are not included in the transaction.

9.11.1. tpsubscribe

Subscribes to an event or set of events named by an eventname. It is used to request a notification for an event when an event is posted. This function is available for both servers and clients.

- Prototype

```
# include <tmaxapi.h>
long tpsubscribe(char *eventname, char *filter, TPEVCTL *ctl, long flags)
```

- Parameters

Parameter	Description
eventname	A string ending with NULL that has up to a 63-character length. A wildcard character or partial-matching is not supported. Only a name that matches an entire string can be registered.
filter	Reserved for future-use. NULL must be set in this parameter.
ctl	A structure that receives a message when an event occurs. It works differently depending on the subject of a tpsubscribe(). If a client used a tpsubscribe() , a ctl must always be NULL and a message will be delivered to a client as an unsolicited data type. Afterwards, the client will handle the received data using a tpsubscribe() or a tpgetunsol() .
flags	Currently, only a TPNOTIME can be used.

If a **tpsubscribe()** is executed by a server, a ctl must not be NULL and it must be configured as follows:

```
struct tpevctl {
    long ctl_flags;
    long post_flags;
    char svc[XATMI_SERVICE_NAME_LENGTH];
    char qname[RQ_NAME_LENGTH];
};
```



```
};
typedef struct tpevctl TPEVCTL;
```

Member	Description
ctl_flags	Not currently supported. Set to 0.
post_flags	Not currently supported. Set to 0.
qname	When using a qname, a message will be queued in a RQ via a tpenq (qname, NULL, data, len, TPNOFLAGS).
svc	When using a svc, a message will be sent to a server in a similar manner to a tpacall (svc, data, len, TPNOREPLY) and a response from the server will be ignored. Only one qname or svc can be used at a time.

- Return Values

Value	Description
descriptor	A function call was successful and a descriptor to be used for a tpunsubscribe() is returned.
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a tpunsubscribe() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPESYSTEM]	A Tmax system error occurred. Detailed error information will be recorded in a system log file. For a client program, a network error is the most common error.
[TPEOS]	An operating system error occurred. Memory cannot be allocated.
[TPEINVAL]	An invalid parameter.
[TPENOENT]	A RQ or a server corresponding to a specified qname or svc of a TPEVCTL structure does not exist.
[TPEPROTO]	A ctl is not NULL in a client and a server.
[TPETIME]	A timeout occurred.

- Related Functions

tpsetunsol(), tpsetunsol_flag(), tpgetunsol(), tpacall(), tpenq()

9.11.2. tpunsubscribe

Removes an event subscription or a set of event subscriptions registered by a **tpsubscribe()** from a

Tmax system's list of subscriptions. If all subscriptions are removed, a Tmax system will delete the table containing the events.

- Prototype

```
# include <tmaxapi.h>
int tpunsubscribe(long sd, long flags)
```

- Parameters

Parameter	Description
sd	A return value received when registering for a subscription with a <code>tpsubscribe()</code> .
flags	Currently, only a <code>TPNOTIME</code> can be used.

- Return Values

Value	Description
Positive Value	A function call was successful.
Negative Value	A function call failed. A <code>tperrno</code> is set to an error code.

- Errors

When a `tpunsubscribe()` fails to execute, a `tperrno` will be set to one of the following values:

Error Code	Description
[TPESYSTEM]	A Tmax system error occurred. Detailed error information will be recorded in a system log file. For a client program, a network error is the most common error.
[TPEOS]	An operating system error occurred. Memory cannot be allocated.
[TPEINVAL]	An invalid parameter.
[TPETIME]	A timeout occurred.

- Related Functions

`tpsetunsol()`, `tpsetunsol_flag()`, `tpgetunsol()`, `tpacall()`, `tpenq()`

9.11.3. `tppost`

Generates a specific event from a server or client and delivers a message. A `tppost()` notifies the occurrence of an event to all clients and server processes that registered for the event by using the eventname in a **`tpsubscribe()`**. If required, a message can be delivered.

- Prototype

```
# include <tmaxapi.h>
int tppost(char *eventname, char *data, long len, long flags)
```

- Parameters

Parameter	Description
eventname	A string that ends with a NULL, and has up to a 63-character length. Wildcard or partial-matching is not supported.
data	A pointer to a buffer for messages to be sent. Must be a buffer allocated by a tppalloc().
len	The length of a buffer to be sent. <ul style="list-style-type: none">• If data indicates a buffer that does not require a data length to be provided, a len will be ignored (0 is used by default).• If data indicates a buffer that requires a data length, a len must not be 0.• If data is NULL, a len will be ignored.
flags	Currently, only a TPNOTIME can be used.

- Return Values

Value	Description
Positive Value	A function call was successful.
Negative Value	A function call failed. A tperrno is set to an error code.

- Errors

When a tppost() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPESYSTEM]	A Tmax system error occurred. Detailed error information will be recorded in a system log file. For a client program, a network error is the most common error.
[TPEOS]	An operating system error occurred.
[TPEINVAL]	An invalid parameter.
[TPENOENT]	When using a tpsubscribe(), this error indicates that a RQ, a server corresponding to a qname, or a svc of a TPEVCTL structure does not exist.
[TPEPROTO]	When a tpsubscribe() is executed in a client, a ctl is not NULL. When a tpsubscribe() is executed in a server, a ctl is NULL.

Error Code	Description
[TPETIME]	A timeout occurred.

- Related Functions

tpsetunsol(), tpsetunsol_flag(), tpgetunsol(), tpacall(), tpenq()

9.12. Broadcast and Multicast

Tmax can send data to clients in various ways. When sending data to multiple clients for management or development, Tmax can manage all client IDs and send messages to each client. This method, however, is not efficient when there are many concurrent users in the system.

If clients can be divided into several groups, an API can be used to send messages to all related clients. Tmax implements broadcast and multicast by utilizing client access information or events. Both the broadcast and multicast APIs can broadcast and multicast, but they are named so to distinguish them.

9.12.1. tpbroadcast

Broadcasts notifications to clients registered in the Tmax system. To receive a notification, a client must be connected to the Tmax system using **tpstart()**, the client name and flags must be defined correctly, and flag values of the TPSTART_T structure used for tpstart() must be set to TPUNSOL_POLL or TPUNSOL_HND. This function can be used on both the client and the server.

- Prototype

```
# include <atmi.h>
int tpbroadcast (char *nodename, char *username, char *cltname,
                char *data, long len, long flags)
```

- Parameters

Parameter	Description
nodename, username, cltname	Logical names used to select target clients. The name length must be 63 characters or less. Wildcards (?, *, etc.) can be used to specify names. NULL can also be used as a wildcard corresponding to all clients. A string argument with a length of 0 corresponds only to a client name with a string length of 0. cltname is the client name used, which is registered when the client first accesses the Tmax system, when using the tpstart() function.
data	A buffer must be allocated through tpalloc() in advance.

Parameter	Description
len	<p>Length of data to receive.</p> <ul style="list-style-type: none"> For STRING, STRUCT, X_COMMON, and X_C_TYPE buffers, which require no length specification, len is ignored and 0 is used by default. len is also ignored if data is NULL.
flags	<p>The following values are available.</p> <ul style="list-style-type: none"> TPNOBLOCK <p>If a blocking condition (for example, the internal buffer is filled with messages to be transmitted) is encountered, the request is not sent.</p> <ul style="list-style-type: none"> TPNOTIME <p>Function caller must wait indefinitely until a response is received. Blocking timeout is ignored. If tpbroadcast() is called during a transaction timeout, transaction timeout applies.</p> <ul style="list-style-type: none"> TPSIGRSTRT <p>Allows signal interrupts. If a system function is interrupted by a signal, the system function is executed again. If the signal interrupt occurs without this flag, the function fails and tperrno is set to TPGOTSIG.</p>

- Return Values

Value	Description
1	A function call was successful.
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a **tpbroadcast()** fails to execute and no messages can be transmitted to a client, a tperrno will be set to one of the following values:

Error Code	Description
[TPEINVAL]	An invalid parameter. For example, an identifier is too long or a flag is invalid. If a nodename is incorrect, a tpbroadcast() will fail and return a TPEINVAL. However, if a username or cltname is invalid, a message will not be transmitted, but a function will be considered to have executed successfully.
[TPETIME]	Blocking timeout occurred while TPNOBLOCK or TPNOTIME is not set.
[TPEBLOCK]	A blocking state occurred when a TPNOBLOCK was set.
[TPGOTSIG]	A signal that is received when a TPSIGRSTRT is not set.

Error Code	Description
[TPEPROTO]	A tpbroadcast() was called from an invalid state.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

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

    tpinfo = (TPSTART_T *)tpalloc("TPSTART", NULL, sizeof(TPSTART_T));
    if (tpinfo==NULL) { error processing }
    strcpy(tpinfo->cltname, "cli1");
    strcpy(tpinfo->usrname, "navis");
    ret=tpstart(tpinfo);
    if (ret==-1) { error processing }

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

    tpbroadcast("tmax", NULL, NULL, buf, 0, TPNOFLAGS);
    data process....

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

- Related Functions

tpalloc(), tpend(), tpstart()

9.13. Environment Program

To develop services for general users, the client program is generally developed in Windows. For development in Windows, Tmax provides interfaces for various development tools. Tmax also provides various types of libraries for developer convenience. According to the library, APIs can be added or replaced.



Tmax supports the following development tools: PowerBuilder, Delphi, VC++, BC++, VB, VB .Net, C#, and Net. For more information about each development tool, refer to *Tmax Programming Guide (4GL)*.

The following describes the supported libraries.

Library	Description
tmax.lib (dll)	Used for VC++, Delphi, and C#. (cdecl)
tmax4gl. lib (dll)	Used for BC++, PowerBuilder, VB, and VB.Net. (stdcall)
tmaxmt. lib (dll)	Supports multi-thread type. (cdecl)
Wintmax. lib (dll)	Supports multi-windows. (cdecl)
tmaxce.lib (dll)	Supports Windows ce. (cdecl)

- tmaxmt.dll

APIs were added to process message-driven type service requests, which are used in the Windows programming environment. The two APIs only differ in whether the message is returned to Windows or to the specified Callback function. Whenever an API is run, a thread is created to process the message. tmaxpi.h is used as the header file.

- WinTmaxAcall()
- WinTmaxAcall2()

- WinTmax.dll

APIs were added to process message-driven type service requests, which are used in the Windows programming environment. Messages are processed by independent threads, so WinTmaxStart() or WinTmaxEnd() is used instead of tpstart() or tpend().

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

WinTmax.dll works similarly to tmaxmt.dll but it does not connect to/disconnect from the Tmax system automatically. Also, a thread processes all messages instead of creating a thread for each API. If Windows is not set separately to process error messages and unrequested messages, the messages are ignored.

For debugging and management, logs can be left in the file specified in the MAX_DEBUG environment variable. WinTmax.h is used as the header file.

9.13.1. WinTmaxAcall

Available in a client in a Windows system environment. This function performs the same functions as a **tpacall()** in a multi-threaded environment. It creates a new thread and invokes a **tpstart()** → **tpacall()** → **tpgetrply()** in the thread. After calling a tpgetrply(), it invokes a SendMessage(wHandle, msgType, (UINT) &MSG, tperrno).

- Prototype

```
# include <tmaxapi.h>
int WinTmaxAcall(TPSTART_T *sinfo, HANDLE wHandle, unsigned int msgtype,
                 char *svc, char *sndbuf, int len, int flags)
```

- Parameters

Parameter	Description
sinfo	The structure that is used when sending client information to a Tmax system. Identical to the parameter of a tpstart().
wHandle	A Windows Handler used to receive messages.
msgtype	An arrival message. Generally, a WM_USER is freely defined by a developer. Specify a service name registered in a svc Tmax configuration file.
svc	A service to send a request.
sndbuf	Data to be sent when calling a service. If not null, it must use a buffer allocated using a tpalloc().
len	The length of data to be sent. It must be specified for CARRAY, X_OCTET, and Structure array types.
flags	<p>Identical to the flags of a tpacall().</p> <p>The following values are available.</p> <ul style="list-style-type: none">• TPBLOCK <p>If a tpacall was used without flags, a normal result will be returned even if a called service does not exist in a svc or an invalid result is returned. If a tpacall() is called using a TPBLOCK, a service state can be checked if it is normal or not.</p> <ul style="list-style-type: none">• TPNOTRAN <p>If a service does not support transactions in a transaction mode, flags must be set to a TPNOTRAN for a tpacall() to be called in the transaction mode. If a tpacall() caller requests a svc service by setting a TPNOTRAN in a transaction mode, a svc service will be executed by being excluded from the transaction mode. When calling a tpacall() in a transaction mode, a call will still be affected by a transaction timeout even if a tpacall() is set to TPNOTRAN. In other words, calling a tpacall that is set as a TPNOTRAN after a transaction timeout will fail except when calling a tpacall that has been set as a TPNOTRAN TPNOREPLY. If a service with a flag set to a TPNOTRAN fails, this does not affect a caller's transaction.</p>

Parameter	Description
flags	<ul style="list-style-type: none"> • TPNOREPLY If a <code>tpacall()</code> is used to send a service request, a <code>tpacall</code> will return immediately without waiting for a response. A client can retrieve results by using a <code>tpgetrply()</code> with a descriptor returned by a <code>tpacall()</code>. If a <code>TPNOREPLY</code> is set, a response for a service request will not be received. If set to a <code>TPNOREPLY</code>, a <code>tpacall()</code> will return a 0 if a service is called normally. If a function caller is in a transaction mode, a <code>TPNOREPLY</code> must be set along with a <code>TPNOTRAN</code>. In case of a <code>TPNOREPLY</code>, to check if a service state is normal or not, a <code>TPBLOCK</code> also must be set. If a <code>TPBLOCK</code> is not set, an error will not be returned even a service is <code>NRDY</code>. • TPNOBLOCK Upon encountering a blocking condition with this flag, for example, an internal buffer is filled with messages to be transmitted, a service request will fail. When calling a <code>tpacall()</code> function without setting a <code>TPNOBLOCK</code> flag, if a blocking condition is encountered, a function caller must wait until blocking is released or a timeout (transaction timeout or blocking timeout) occurs. • TPNOTIME A function caller must wait indefinitely until a response is received and blocking timeouts are ignored. If a <code>tpacall()</code> is called during a transaction timeout, a transaction timeout will be applied. • TPSIGRSTRT Allows signal interrupts. If a system function is interrupted by a signal, the system function will be executed again. If a signal interrupt occurs without this flag, a function will fail and a <code>tperrno</code> will be set to a <code>TPGOTSIG</code>.

- Return Values

Value	Description
Descriptor	A function call was successful. A returned descriptor is used to receive a response for a sent service request.
-1	A function call failed. A <code>tperrno</code> is set to an error code.

- Errors

When a `WinTmaxAcall()` fails to execute, a `tperrno` will be set to one of the following values:

Error Code	Description
[TPEINVAL]	An invalid parameter. For example, a svc is NULL, data points to a buffer that was not allocated by a tpalloc(), or a flag is invalid.
[TPENOENT]	A specified svc does not exist.
[TPEITYPE]	A svc does not support a data type or subtype. For a structure, this error occurs when the structure is not declared in a SDLFILE file.
[TPELIMIT]	A maximum number of unprocessed asynchronous service requests was reached. Service requests from a caller cannot be sent.
[TPETIME]	<p>A timeout occurred. If a transaction timeout occurs when a function caller is in a transaction mode, a transaction will be rolled back. If a function caller is not in a transaction mode, a block timeout will occur if neither a TPNOTIME nor a TPNOBLOCK is set.</p> <p>In these cases, a *data content and a *len will not be changed. If a transaction timeout occurs, new service requests and processes waiting for a response will fail with a [TPETIME] error until a transaction is rolled back.</p>
[TPEBLOCK]	A blocking state occurred when a TPNOBLOCK was set.
[TPGOTSIG]	A signal that is received when a TPSIGRSTRT is not set.
[TPEPROTO]	A WinTmaxAcall() was called from an invalid state. For example, a TPNOREPLY was called in a transaction mode, but a TPNOTRAN was not set.
[TPESYSTEM]	A Tmax system error occurred.
[TPEOS]	An operating system error occurred.

- Example

```

...
#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 = tpalloc("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;
}

```

- Related Functions

tpacall(), WinTmaxAcall2()

9.13.2. WinTmaxAcall2

Available in a client in a Windows system environment. This function performs the same function as a **tpacall()** in a multi thread environment. It issues a new thread and invokes a **tpstart()** → **tpacall()** → **tpgetrply()** in the thread. After calling a tpgetrply(), it delivers received data to a specified Callback function.

- Prototype

```

# include <tmaxapi.h>
int WinTmaxAcall2(TPSTART_T *sinfo, WinTmaxCallback fn, char *svc,
    char *sndbuf, int len, int flags)

```

- Parameters

Parameter	Description
sinfo	The structure used when sending information about a client to a Tmax system. Identical to the parameters of a tpstart().
fn	Sets a Callback function to receive a response for a service request.
svc	Sets a service name registered in a Tmax environment configuration file.
sndbuf	Data to be sent when calling a service. If not null, it must use a buffer allocated through a tpalloc().
len	The length of data to be sent. It must be specified for CARRAY, X_OCTET, and Structure array types.
flags	<p>Identical to the flags of a tpacall().</p> <p>The following values are available.</p> <ul style="list-style-type: none"> • TPBLOCK <p>If tpacall was used without flags, a normal result is returned even if a called service does not exist in a svc or an invalid result is returned. If a tpacall() is called using a TPBLOCK, a service state can be checked if it is normal or not.</p> • TPNOTRAN <p>If a service does not support transactions in a transaction mode, flags must be set to a TPNOTRAN for a tpacall() to be called in the transaction mode. If a tpacall() caller requests a svc service by setting a TPNOTRAN in a transaction mode, a svc service will be executed by being excluded from the transaction mode. When calling a tpacall() in a transaction mode, a call will still affected by a transaction timeout even if a tpacall() is set to TPNOTRAN. If a service with a flag set to a TPNOTRAN fails, this does not affect a caller's transaction.</p> • TPNOREPLY <p>If a tpacall() is used to send a service request, a tpacall will return immediately without waiting for a response. A client can retrieve results by using a tpgetrply() with a descriptor returned by a tpacall().</p> <p>If a TPNOREPLY is set, a response for a service request will not be received. If set to a TPNOREPLY, a tpacall() will return a 0 if a service is called normally. If a function caller is in a transaction mode, a TPNOREPLY must be set along with a TPNOTRAN. In case of a TPNOREPLY, to check if a service state is normal or not, a TPBLOCK also must be set. If a TPBLOCK is not set, an error will not be returned even a service is NRDY.</p>

Parameter	Description
flags	<ul style="list-style-type: none"> • TPNOBLOCK Upon encountering a blocking condition with this flag, for example, an internal buffer is filled with messages to be transmitted, a service request will fail. When calling a <code>tpacall()</code> function without setting a TPNOBLOCK flag, if a blocking condition is encountered, a function caller must wait until blocking is released or a timeout (transaction timeout or blocking timeout) occurs. • TPNOTIME A function caller must wait indefinitely until a response is received and blocking timeouts are ignored. If a <code>tpacall()</code> is called during a transaction timeout, a transaction timeout will be applied. • TPSIGRSTRT Allows signal interrupts. If a system function is interrupted by a signal, the system function will be executed again. If a signal interrupt occurs without this flag, a function will fail and a <code>tperrno</code> will be set to a <code>TPGOTSIG</code>.

- Return Values

Value	Description
1	A function call was successful.
-1	A function call failed. A <code>tperrno</code> is set to an error code.

- Errors

When a `WinTmaxAcall2()` fails to execute, a `tperrno` will be set to one of the following values:

Error Code	Description
[TPEINVAL]	An invalid parameter. For example, a <code>svc</code> is <code>NULL</code> , <code>data</code> points to a buffer that was not allocated by a <code>tpalloc()</code> , or a flag is invalid.
[TPENOENT]	A specified <code>svc</code> does not exist.
[TPEITYPE]	A <code>svc</code> does not support a data type or subtype. For a structure, this error occurs when the structure is not declared in a <code>SDLFILE</code> file.
[TPELIMIT]	A maximum number of unprocessed asynchronous service requests was reached. Service requests from a caller cannot be sent.
[TPETRAN]	A <code>svc</code> does not support transactions, and a TPNOTRAN was not set.

Error Code	Description
[TPETIME]	<p>A timeout occurred. If a transaction timeout occurs when a function caller is in a transaction mode, a transaction will be rolled back. If a function caller is not in a transaction mode, a block timeout will occur if neither a TPNOTIME nor a TPNOBLOCK is set.</p> <p>In these cases, a *data content and a *len will not be changed. If a transaction timeout occurs, new service requests and processes waiting for a response will fail with a [TPETIME] error until a transaction is rolled back.</p>
[TPEBLOCK]	A blocking state occurred when a TPNOBLOCK was set.
[TPGOTSIG]	A signal that is received when a TPSIGRSTRT is not set.
[TPEPROTO]	A WinTmaxAcall2() was called from an invalid state. For example, a TPNOREPLY was called in a transaction mode, but a TPNOTRAN was not set.
[TPESYSTEM]	A Tmax system error occurred.
[TPEOS]	An operating system error occurred.

- Examples

```
#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;
    }
}

```

- Related Functions

tpacall(),WinTmaxAcall()

9.13.3. WinTmaxStart

Available in a client in a Windows system environment. This function is used to connect to a Tmax system in a multi-Windows environment. This function is identical to a **tpstart()**. When using multiple threads, connections must be respectively established per thread through a WinTmaxStart().

- Prototype

```

# include <WinTmax.h>
int WinTmaxStart(TPSTART_T *tpinfo)

```

- Parameters

Refer to [tpstart](#) for more information about a TPSTART_T.

- Return Values

Value	Description
1	A function call was successful.
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a `WinTmaxStart()` fails to execute, a `tperrno` will be set to one of the following values:

Error Code	Description
[TPEINVAL]	An invalid parameter. For example, a <code>tpinfo</code> is NULL or not a pointer for a <code>TPSTART_T</code> .
[TPEITYPE]	A <code>tpinfo</code> is not a pointer for a <code>TPSTART_T</code> structure.
[TPEPROTO]	A <code>WinTmaxStart()</code> was called in an inappropriate condition. For example, a <code>WinTmaxStart()</code> was called in a server program, or it was called after a connection was already established.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred. Environment variables may be invalid. For example, a connection failed because a <code>TMAX_BACKUP_ADDR</code> or <code>TMAX_BACKUP_PORT</code> was invalid.

- Related Functions

`tpstart()`, `WinTmaxEnd()`, `WinTmaxSend()`, `WinTmaxSetContext()`

9.13.4. WinTmaxEnd

Available in a client in a Windows system environment. This function closes a connection to a Tmax system. It functions identically to a **`tpend()`**. In a multi-threaded environment, a client is connected to each thread via a **`WinTmaxStart()`**, so a `WinTmaxEnd()` must be used for each thread to close the connection.

- Prototype

```
# include <WinTmax.h>
int WinTmaxEnd(void)
```

- Return Values

Value	Description
1	A function call was successful.
-1	A function call failed. A <code>tperrno</code> is set to an error code.

- Errors

When a `WinTmaxEnd()` fails to execute, a `tperrno` will be set to one of the following values:

Error Code	Description
[TPETIME]	Cannot access a critical section due to an internal system error. Check the system status.
[TPEPROTO]	A WinTmaxEnd() was called from an invalid state. For example, a WinTmaxEnd() was called after a connection was already closed.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Related Functions

tpend(), WinTmaxStart()

9.13.5. WinTmaxSetContext

Available in a client in a Windows system environment. This function controls window handles and message types.

This function, used in a Windows environment, manages specified Windows handles and message types by storing them in a vacant slot of a library. Using this information, when a working thread receives a reply, it will send data to the specified Windows in the specified type. The following is the data transmission format:

```
SendMessage(winhandle, msgType, (UINT) &msg, callRet)
```

In a Windows environment, a WARM corresponds to a msg and a LARA corresponds to a callRet. A msg is a TPSVCINFO structure and a callRet indicates a result of the process. If a proper message is received, a callRet is 0, and if an improper message is received, a callRet is -1.

For example, if a service is processed as atpreturn (TPSUCCESS, ...), or a service fails with a tpreturn (TPFAIL, ...), or an unrequested message is received, a callRet becomes 0, since it will be considered as a proper message was received. However, if a synchronous or conversational message is delivered, a callRet will be -1 because those message types cannot be used in a multi Window environment. If a callRet value is -1, a tperrno value can be checked to know the error cause.

- Prototype

```
# include <WinTmax.h>
int WinTmaxSetContext(void *winhandle, unsigned int msgType, int slot)
```

- Parameters

Parameter	Description
winhandle	A windows handle to process received data.
msgType	A message type.
slot	<p>Indicates a slot used to allocate a specified Windows handle and message type. The maximum available number of slots is 256, and 0 and 1 are internally specified in a system for a default display and error respectively. Thus, if an error occurs while receiving data or a Windows for display is not specified, a default Windows is used.</p> <p>The default slots can be redefined by a user. In case of an unrequested message, a user defined Windows will not exists, so messages are delivered to the number 0 default display Windows.</p> <p>The following values are available.</p> <ul style="list-style-type: none"> • -1: Allocates a specified Windows handle and message type by automatically searching for vacant slots in a system. • A value greater than or equal to 0: Allocates a specified Windows and message type to a slot specified with a given index.

- Return Values

Value	Description
index	A function call was successful and an index for a slot was returned. As well, an index is used as a first parameter of a WinTmaxSend().
-1	A function call failed. A tperrno is set to an error code.

- ErrorsWhen a WinTmaxSetContext() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPESYSTEM]	A Tmax system error occurred.
[TPEOS]	An operating system error occurred.

- Examples

```

...
int CTMaxGwView::Connect()
{
    CString szTemp, Fname;
    WinTmaxEnd();
    int Ret = tmaxreadenv(TMAXINI, "TMAX117");
    if(Ret<0)
    {
        szTemp.Format("tmaxreadenv error");
        LogDisplay2(2, (char *)(const char *)szTemp, szTemp.GetLength());
    }
}

```

```

        return FALSE;
    }
    if (WinTmaxStart((TPSTART_T *)NULL) == -1) {
        szTemp.Format("WinTmaxStart error = [%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;
}

```

- Related Functions

WinTmaxStart(), WinTmaxEnd(), WinTmaxSend()

9.13.6. WinTmaxSend

Available in a client in a Window system environment. This function sends data and requests a service in a multi-windows environment.

- Prototype

```

#include <WinTmax.h>
int WinTmaxSend(int recvContext, char *svc, char *data, long len, long flags)

```

- Parameters

Parameter	Description
recvContext	Sets a Windows for receiving a response from a Tmax system. It is the return value of a WinTmaxSetContext().
svc	A service name.
data	A buffer allocated through a tpalloc(). The data used to request a service is stored.
len	Data length. The length must be correctly defined when using a CARRAY or a multiple Structure buffer.
flags	Flag values are listed in the following table.

A WinTmaxSend() works similarly to a tpacall() function. It requests a service and is returned immediately without waiting for a response. A response is processed by the thread that was created when calling a WinTmaxStart(), and the response result will be sent to a Windows set by a WinTmaxSetContext(), so a separate API is not provided to receive the response.

The following example finds a vacant slot to store (hwd, 0x300) and returns an index. The return value a rc of a WinTmaxSetContext() is used as a rcvContext parameter of a WinTmaxSend(). If a response is received after calling a WinTmaxSend(rc, svc, data, len, 0), a 0 x 300 message and a response result will be sent to a hwd windows.

```
rc = WinTmaxSetContext(hwd, 0x300, -1);
int nSendResult = WindTmaxSend(rc, (LPSTR)(LPCSTR)strService, (Char*)tpbuf, nLen, TPNOFLAGS);
/*
    The internal thread processes the response message and sends the result to the Windows
    SendMessage(hwd, 0x300, (UINT) &msg, callRet);
*/
```

A Windows that receives a message will receive a response data through a WPARAM and a response result through a LPARAM. If a callRet is 0, it means the response will be normal, and -1 means that an error occurred. The cause of an error can be known through a tperrno value.

A WinTmaxSend() does not support transactions due to its structure and is not affected by a BLOCKTIME in a Tmax configuration file or a tpsettimeout(). However, when calling a WinTmaxSend(), if a TPBLOCK flag is given, a BLOCKTIME will be applied according to a corresponding flag.

The following are possible flag values.

Flag	Description
TPBLOCK	<p>If a WinTmaxSend() function is used without this flag, a normal result will be returned even if a called service did not exist in a svc or an invalid result was returned. Errors can be checked at the time of receiving a result.</p> <p>Using this flag, it is possible to check whether a service status is normal or not normal at the time of calling a function. In other words, a WinTmaxSend() checks if a request service can be processed normally within a BLOCKTIME and returns the result. If a service cannot be processed, an error will be stored in a tperrno and a -1 will be returned.</p> <p>If it is impossible to check if a requested service can be processed within a BLOCKTIME, a TPETIME error will be returned. In this case, a client cannot know whether a requested service is processed or not, so an error routine must be written carefully. If a request must be sent again, an error routine must be to check if a previous request was processed.</p>
TPNOREPLY	<p>Immediately returns a service request sent with a WinTmaxSend() function without waiting for a response. A worker thread receives the result and sends it to the specified Windows. However, a TPNOREPLY flag will be set to not receive a response for the service.</p>

Flag	Description
TPNOTRAN	<p>If a WinTmaxSend() function caller requests a svc service by setting this flag in a transaction mode, the svc service will be executed while being excluded from the transaction mode.</p> <p>If a svc does not support transactions in a transaction mode, a flag must be set to a TPNOTRAN when a WinTmaxSend() function is called in the transaction mode. If called in the transaction mode, the WinTmaxSend() function will still be affected by a transaction timeout even though a service flag was set to TPNOTRAN. If the service called with a TPNOTRAN fails, it will not affect a caller's transaction.</p>
TPNOBLOCK	Upon encountering a blocking condition with this flag set (for example, an internal buffer is filled with messages to be transmitted), a service request will fail. When calling a WinTmaxSend() function without setting a TPNOBLOCK flag and encountering a blocking condition, a function caller will wait until blocking is released or a timeout (transaction timeout or blocking timeout) occurs.
TPNOTIME	A function caller must wait indefinitely until a response is received and blocking timeouts are ignored. If a WinTmaxSend() is called during a transaction timeout, a transaction timeout will be applied.
TPSIGRSTRT	Allows signal interrupts. If a system function is interrupted by a signal, the system function will be executed again. If a signal interrupt occurs without this flag, a function will fail and a tperrno will be set to a TPGOTSIG.

- Return Values

Value	Description
Descriptor	A function call was successful and a descriptor is returned. Currently this descriptor is not used.
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a WinTmaxSend() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPENOENT]	A specified svc does not exist.
[TPEITYPE]	A svc does not support a data type or subtype. For a structure, this error occurs when the structure is not declared in a SDLFILE file.
[TPELIMIT]	A maximum number of unprocessed asynchronous service requests was reached. Service requests from a caller cannot be sent.

Error Code	Description
[TPETIME]	<p>A timeout occurred. If a transaction timeout occurs when a function caller is in a transaction mode, a transaction will be rolled back. If a function caller is not in a transaction mode, a block timeout will occur if neither a TPNOTIME nor a TPNOBLOCK is set.</p> <p>In these cases, a *data content and a *len will not be changed. If a transaction timeout occurs, new service requests and processes waiting for a response will fail with a [TPETIME] error until a transaction is rolled back.</p>
[TPEBLOCK]	A blocking state occurred when a TPNOBLOCK was set.
[TPGOTSIG]	A signal that is received when a TPSIGRSTRT is not set.
[TPEPROTO]	A WinTmaxSend() was called from an invalid state. For example, a TPNOREPLY was called in a transaction mode, but a TPNOTRAN was not set.
[TPESYSTEM]	A Tmax system error occurred.
[TPECLOSE]	Disconnected from a Tmax system due to various reasons (for example, network issues).
[TPEOS]	An operating system error occurred.

If a LPARAM of a message sent to Windows is -1, a tperrno will be set to one of the following values:

Error Code	Description
[TPEBADDESC]	Occurs when a valid descriptor cannot be found when a response message is received.
[TPEOTYPE]	Occurs when the buffer types of client and server programs do not match when a response message is received.
[TPEPROTO]	A WinTmaxStart() or a WinTmaxEnd() was called from an invalid state.
[TPESYSTEM]	A Tmax system error occurred.
[TPECLOSE]	Disconnected from a Tmax system due to various reasons (for example, network issues).
[TPEOS]	An operating system error occurred.

- Examples

```
...
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);
    ...
}

```

- Related Functions

WinTmaxStart(),WinTmaxEnd()

9.14. Multithread and Multicontext

Describes functions that are related to multithread/multicontext. Multithread and multicontext server libraries are different from client libraries in the multicontext type.

9.14.1. tpgetctxt

Returns a context ID, which is set in a thread that calls a function, as its first parameter.

In a multithread/multicontext server, a tpgetctxt returns a value of 1 or greater when the context set in a thread is valid. A tpgetctxt returns a TPNULLEXCONTEXT(-2) when a context is invalid or not set. A context is valid only while a service thread is processing a service request. If a tpreturn() is called after completing a service request, a context will no longer be valid and a user created thread can no longer use the context.

A tpgetctxt() is used differently in client and server programs. The following information describes the function by separating examples by server and client.



A MultiThread/MultiContext server does not support a Singlecontext.

- Prototype

```

#include <usrinc/atmi.h>
int tpgetctxt(int *ctxtid, long flags)

```

- Parameters

Parameter	Description
ctxtid	Retrieves a current context at the time a function is called. <ul style="list-style-type: none"> • multicontext: Returns a value of 1 or greater. • singlecontext: Returns 0.
flags	Not currently supported. Set to TPNOFLAGS.

- Return Values

Value	Description
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a tpgetctxt() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPEINVAL]	An invalid parameter. For example, a first parameter is a pointer or a second parameter is not set to 0.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples - Client Program

```
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;
}
```

- Examples - Server Program


```

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

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

    printf("MSERVICE service is started!");
    tpsetctxt(&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;
}

```

- Related Functions

tpsetctxt()

9.14.2. tpsetctxt

Sets a current context. The way the function is used differs in a client program and a server program as follows:

- Client Program

A client can assign a different previously created context to a current client using a function. Most ATMI functions are based on per-context. To know the current context of a client, the return value of a `tpgetctxt()` can be checked.

A client can use multiple contexts, but only one context is used at a time. For example, if a `tpacall()` is called in `context1`, `tpgetrply()` must be called in `context1` even though another context has been used.

- **Server Program**

A service thread processes a service by getting a context, but a user created thread does not have its own context. Most ATMI functions can work after getting a context. Therefore a user created thread must share a context with a service thread if required. A user created thread can share a context with other service threads by using a `tpsetctxt()`.

A user created thread that calls a `tpsetctxt()` shares context information with a service thread. For example, if a service thread calls a `tpacall()`, after that it is possible for a user created thread to receive a response for a request through a `tpgetrply()`.

A `tpsetctxt()` cannot be used in a service thread. A service thread has its own context by default, and cannot replace the context with other one. Therefore, if a service thread calls a `tpsetctxt()`, an error will be returned with a `TPEPROTO` error code.

If a user created thread shares a context with a service thread through this function, the user created thread must call a `tpsetctxt(TPNULLEXCONTEXT)` before the service thread calls a `tpreturn()`. In other words, at the time that the service thread calls `tpreturn()`, the context of the service thread must be changed not to be shared by other user created threads. If this is not kept, `tpreturn()` will fail and a `TPESVCERR` error code will be returned to the client. Therefore, the process flow between threads must be controlled through synchronization. The `ctxtid` parameter of a `tpsetctxt()` uses the Context-ID which that is retrieved by calling a `tpgetctxt()` from a service thread.

With consideration of the different ways how a function is written depending on the program, the basic information of a `tpsetctxt()` is as follows:

- **Prototype**

```
#include <usrinc/atmi.h>
int tpsetctxt(int ctxtid, long flags)
```

- **Parameters**

Parameter	Description
ctxtid	Sets a current context at the time point of a function call. In a client program, new context IDs created using a <code>tpstart()</code> can be used. In a server program, the Context-ID of a service thread can be used. Other contexts such as a <code>TPNULLEXCONTEXT</code> can be used but not a <code>TPINVALIDCONTEXT</code> .

Parameter	Description
flags	Not currently supported. Set to TPNOFLAGS.

- Return Values

Value	Description
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a tpsetctxt() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPEPROTO]	A function was called from an invalid state. For example, a service thread called a function or a Context ID was delivered as a parameter that was invalid in a server program.
[TPEINVAL]	An invalid parameter. A ctxid is set to 0 or TPINVALIDCONTEXT, or flags are set to non-zero. In a client program, a tpsetctxt() is called before the program calls a tpstart(). A buffer flag is not set to TPMULTICONTEXTS when this function is called before the tpstart() is called. This error occurs when a context ID is set to TPINVALIDCONTEXT or 0.
[TPENOENT]	A ctxtid value is not a configurable context.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples - Client Program

```
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;
}
```

- Examples - Server Program

```
typedef param {
    int ctxtid;
    TPSVCINFO *svcinfo;
} param_t;

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

    printf("MSERVICE service is started!");
    tpgetctx(&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 (tpsetctx(param->ctxtid, TPNOFLAGS) == -1) {
        printf("tpsetctx(%d) failed, [tperrno:%d]", param->ctxtid, 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;
}
```

- Related Functions

tpgetctx()

10. Server API

This chapter describes functions that are used in a Tmax server and functions that are required to write a server program in ATMI. A server can act as a client, so functions that are used in a client also can be used.

10.1. TCS

The following describe the functions that are used in a TCS type server program.

- Service completion functions

`tpreturn()` sends a response to a client. `tpforward()` forwards a request to another server and ends the service.

Function	Description
<code>tpreturn</code>	Sends a response to a service request and ends the service routine.
<code>tpforward</code>	Forwards a request to another server to process a service.

- Server initialization and termination functions

Opens or closes a database that is connected to the application, and provides a function to process command line options. This subroutine is provided by default.

Function	Description
<code>tpsvrinit</code>	Initializes a server.
<code>tpsvrdone</code>	Ends a server process.

- Multithread/Multicontext

Function	Description
<code>tpsvrthrinit</code>	Provides an initialization function to each thread when a service thread is created in a STD_MT type server. Supported in Tmax 5 SP2 and later.
<code>tpsvthrdone</code>	Called when a service thread is terminated in a STD_MT type server. Supported in Tmax 5 SP2 and later.
<code>tpgetctxt</code>	Returns the currently set context ID as the first parameter of the thread that called this function.
<code>tpsetctxt</code>	Sets the context of the thread that called this function as the ID of the first parameter.

- Unrequested message functions

A server can send unrequested messages to clients unilaterally in Tmax. This feature is used

when there is information that all clients connected to Tmax must be notified of. The clients that can receive the unrequested message must be connected to Tmax with flags set.

Function	Description
tpsendtocli	A server sends a message, which was registered in advance by clients, to the clients automatically without client request.
tpgetclid	Returns the ID of the client that is connected to the Tmax system. The ID is used for tpsendtocli().
tpchkclid	Checks if the client that corresponds to the client ID of the node where the server process resides is connected.



For more information about each function, refer to *Tmax Reference Guide*.

Service Routine Parameters

A server program consists of main(), which is provided by Tmax, and service routines. main() consists of routines that process database connections, disconnections, and command line options. Service routines receive and process requests from a client.

The server main() receives a request from a client and calls the corresponding service routine with a TPSVCINFO structure to process the request. The TPSVCINFO structure has data to be processed and information about the client that sent the request.

The TPSVCINFO structure is defined in the atmi.h header file. The following are the components of the TPSVCINFO structure.

```
#define XATMI_SERVICE_NAME_LENGTH 16

struct tpsvcinfo {
    char name[XATMI_SERVICE_NAME_LENGTH]; /* Requested service name */
    char *data;                          /* Request data */
    long len;                            /* Request data length */
    long flags;                          /* Service property */
    Int cd;                              /* Connection descriptor */
};
typedef struct tpsvcinfo TPSVCINFO
```

Member	Description
name	Name of the service routine requested by a client.
data	Buffer to receive requested data from a client. This is allocated in advance with tpalloc() in the server main().
len	Length of the requested data.

Member	Description
flags	Notifies a service if it is in a transaction state or the caller requires a response. For example, if flags is TPTRAN, the service is in transaction mode. If flags is TPNOTRAN, the service can participate in the current transaction.
cd	Connection Descriptor. Sets the clients the response must be sent to. The server allocates the buffer in advance in main() to process the service requested by a client. It is recommended to use data in TPSVCINFO when communicating through tpreturn() or tpforward(). When accessing data in TPSVCINFO, the buffer types of the server program and the client program must be the same.

Client Program

The following is an example client program.

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

Server Program

The following is an example service routine. main() is provided by Tmax.

```
#include <usrinc/atmi.h>
. . .
SEL_SVC(TPSVCINFO *msg)
/* A structure that contains client information and client request*/
{
    struct strdata *svcddata;
    /* Change the data type to match the buffer type */
    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

Means the completion of a service routine. It works the same as a return sentence in the C language. When a tpreturn() is called, a service routine is returned to a Tmax system. To return a service routine correctly to a Tmax system, call a tpreturn() in a service routine.

A tpreturn() function sends a reply message for a service. If a program to receive a reply is waiting with a **tpcall()**, **tpgetrply()**, or **tprecv()**, the reply will be transferred through a receiver's buffer after a tpreturn() is successfully called.

A tpreturn() also allows interactive services to terminate interactive communication. A service routine cannot call a tpdicon() directly. To ensure a correct result, a program connected to an interactive service must not to call a tpdicon(). Rather, it must wait for a completion notification from an interactive service, for example, an event like a TPEV_SVCSUCC or a TPEV_SVCFAIL transmitting with a tpreturn().

If a service routine is in a transaction mode, and a client or service that called a service does not start a transaction explicitly (if tx_begin is not used), then a tpreturn will be committed or rolled back as a part of a transaction when tpreturn is TPSUCCESS. A service can be called to multiply as a part of a same transaction (global transaction), therefore, it is will not be completely committed or rolled back until a transaction beginner calls either a tx_commit or a tx_rollback to complete a transaction.

A tpreturn() function must be called after all replies are received from services requested by a service routine. Otherwise, a [TPESVCERR] error or a TPEV_SVCERR event will be returned to the program that communicated with a service routine that depended on the service characteristics. Replies that are not received are automatically ignored by a Tmax system. In addition, the descriptors used for these replies will also be invalidated.

A tpreturn() function must be called after all connections that began from a service used for interactive communication are ended. Otherwise, either a [TPESVCERR] error or a TPEV_SVCERR event will be returned to the program that communicated with a service routine that depended on the service characteristics. In addition, a forcible disconnection event, such as a TPEV_DISCONIMM, will be transferred to a service and all items that are connected to the service.

In an interactive communication, if a service routine does not have communication control when it calls a tpreturn(), two results may happen:

1. If a service routine calls a tpreturn() with a rval of a TPFAIL and data is NULL, a TPEV_SVCFAIL event will be transferred to a communication starter.
2. When a tpreturn() is called in other ways, a TPEV_SVCERR event will be transferred to a communication starter. Because an interactive service must have only one interactive communication that does not start from a service, a Tmax system knows the descriptor to which data or an event that is to be transmitted. For this reason, a descriptor is not transferred to a tpreturn() as a parameter.

- Prototype

```
# include <atmi.h>
void tpreturn (int  rval, long  rcode, char  *data, long  len, long  flags)
```


- Parameters

Parameter	Description
rval	<p>The following values are available.</p> <ul style="list-style-type: none"> TPSUCCESS <p>The service is successfully completed. If data exists and no error occurs during tpreturn() execution, the data will be transmitted. If the caller is in a transaction mode, a part of this transaction will be determined as it can be committed. It allows the other services belonging to the transaction to be committed if all of them are successfully completed and ready for commit. But the transaction will be rolled back if any of them fails. Note that calling the tpreturn() does not mean to complete the entire transaction. In addition, if there is any waiting reply or inter active connection or a job performed in the service tries to roll back the transaction, the message will be sent as a failure even though the caller has returned TPSUCCESS. That is, the receiver of the reply will receive [TPESVCERR] error or TPEV_SVCERR event. Note that, if transaction is rolled back in a service routine, rval is set to TPFAIL. If TPSUCCESS returns in an interactive, TPEV_SVCSUCC even will occur.</p> TPFAIL <p>The service is ended due to an error in an application program. The error is returned to the program that receives the reply. That is, the call to receive the reply fails and the receiver receives [TPSVCFAIL] value or TPEV_SVCFAIL event. This value cannot sent data. If the caller is in a transaction mode and is autotransaction, tpreturn() will roll back the transaction. The transaction might have already been determined to be rolled back.</p> <ul style="list-style-type: none"> TPEXIT <p>When a service is returned, this flag is used to forcibly terminate the server process. The process closed via tpexit() can be restarted through TMM.</p> TPDOWN <p>Similar to TPEXIT, but the process terminated through TPDOWN is not restarted using TMM.</p> <ul style="list-style-type: none"> TMSUCCESS <p>Same as TPSUCCESS.</p>

Parameter	Description
rval	<ul style="list-style-type: none"> TMFAIL <p>Same as TPFAIL.</p> <p>Other values are considered as TPFAIL.</p>
rcode	<p>The return code a rcode defines in an application program can be transmitted to a program that receives a service reply. This code will be transmitted, regardless of a rval value, as long as a reply can be transmitted to a client successfully. The client is considered to have received a reply successfully in the following cases: A receiving call was a success or a reply was returned through a [TPSVCFAIL], or the client received either a TPEV_SVCSUCC or a TPEV_SVCFAIL event.</p> <p>A rcode value is transferred to a receiver with a tpurcode global variable.</p>
data	<p>Reply data to be sent. If it is not NULL, it must indicate a buffer that was allocated by tmalloc() before. If the buffer is the same as the one transferred to the service routine, the Tmax systems will handle it.</p> <p>Therefore, the service routine writer does not need to take care of whether to or not to free the buffer. Indeed, if a user attempts to free the buffer, the attempts will fail. However, if the buffer transferred with tpreturn() is not the same as the one transferred along with the service request, the tpreturn() will free the buffer.</p>
len	<p>Data length to be sent.</p> <p>If data indicates a buffer that does not require specifying the length, the len will be ignored (0 is used in general). But if data indicates a buffer that needs to specify the length, the len must not be 0. If data is NULL, the len is ignored. In this case, if the program that calls the service is waiting for a reply, the reply without any data will be transmitted. If no reply is expected, the tpreturn() will free the data and returns without transmitting any reply, accordingly.</p>
flags	<p>Not currently supported. Set to 0.</p> <p>If the service is an interactive type, data would not be transferred in the following two cases.</p> <ul style="list-style-type: none"> The interactive service was already disconnected when tpreturn() is called. That is, the caller receives TPEV_DISCONIMM event. In this case, the tpreturn() simply ends the service routine and if it is in a transaction mode, it will roll back the current transaction. In this case, the caller's data cannot be transferred. If the caller does not have the communication control, either TPEV_SVCFAIL or TPEV_SVCERR event will be transferred to the communication starter as mentioned earlier.

Parameter	Description
flags	Regardless of the event that the communication starter receives, no data is transferred. If the communication starter receives the TPEV_SVCFAIL event, the return code could be used as a tpurcode variable of the starter.

- Return Values

The service routine does not return any value to the caller, that is, the Tmax systems. It is a rule that the service routine returns with tpreturn(). If the service routine returns using return sentence of C language, rather than using the tpreturn() , the server will return a service error to the service requester. In addition, connection that has been kept for interactive communication is disconnected forcibly and all the replies that are waiting asynchronously waiting are ignored.

If the server is in a transaction mode, the transaction will be rolled back. In addition, if tpreturn() is used outside the service routine, it will simply return without performing anything.

- Errors

Since tpreturn() ends a service routine, if an error occurs while a parameter is being processed, it cannot be not transferred to the service routine, the caller. Errors are sent as follows:

Classification	Description
Synchronous and Asynchronous communication	tperrno is set to [TPESVCERR] for a program that receives a service result through tpcall() or tpgetrply().
Interactive communication	Generates a TPEV_SVCERR event for the program that uses tpsend() or tprecv().

- Examples

```
#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);
    }
}

```

- Related Functions

tpalloc(), tpcall(), tpconnect(), tpdiscon(), tpgetrply(), tprecv(), tpsend()

10.1.2. tpforward

Ends its own service processing, and forwards a client's request to a svc service routine.

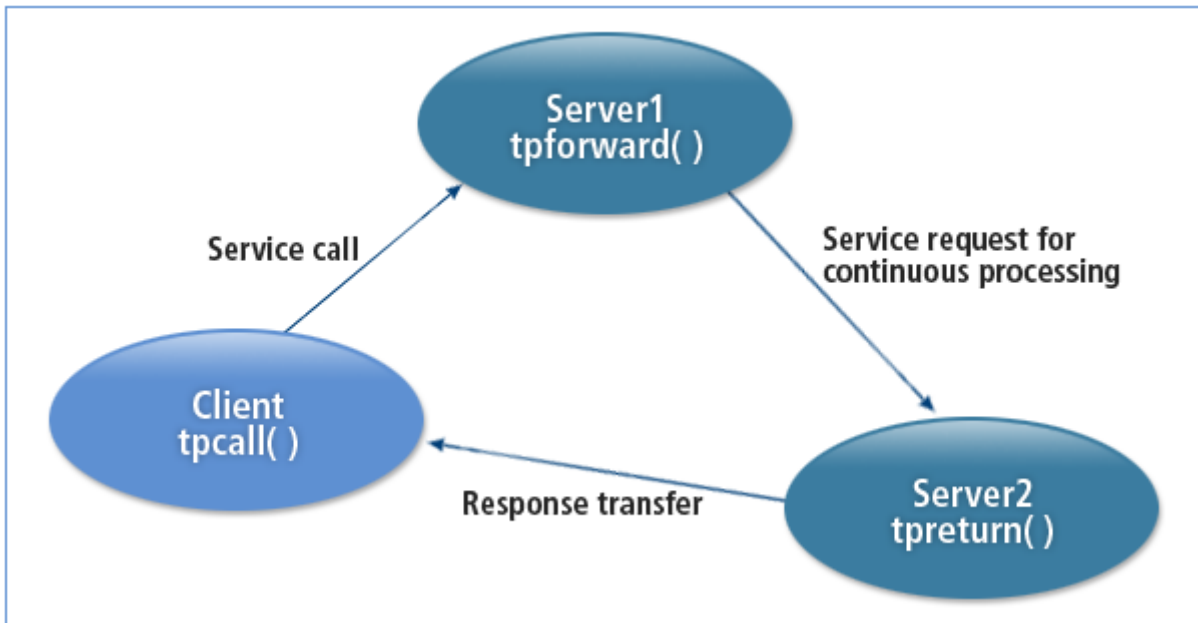
This function is called at the end of a service routine, which acts like a **tpreturn()**. Like a **tpreturn()**, a **tpforward()** must be called in a service routine controlled by a Tmax system to be correctly returned to the system.

This function forwards a service request to a service named svc using data. The service routine that forwards the request will not receive a reply. After forwarding the request, a service routine returns to the Tmax system. Then, the service can perform other operations. Because a **tpforward()** does not expect any response from a requester, it can forward a service request to any service without any particular errors.

If a service routine is in a transaction mode, the transaction can be completed only when the transaction originator completes the transaction by executing either a **tx_commit()** or a **tx_rollback()**. In other words, like a **tpreturn()**, a **tpforward()** does not complete a transaction. If a transaction is originated in a service routine that uses a **tx_begin()**, the transaction must be completed before a **tpforward()** is called, either by using a **tx_commit()** or a **tx_rollback()**. This means that all services connected by a **tpforward()** must be or not be in a transaction mode. The finally forwarded server process sends a reply to a client that requested the service first, using a **tpreturn()**. In other words, a **tpforward()** shifts the responsibility of sending a reply for a waiting requester, to another server process. This service is available even in between nodes.

A **tpforward()** must be called after a service routine receives replies for all services requested. The descriptors of unreceived replies are invalidated and a forward request is not transmitted. A **tpforward()** cannot be called in a interactive communication.

The following shows the **tpforward** function flow:



tpforward

- Prototype

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

- Parameters

Parameter	Description
svc	The name of a service to receive a buffer.
data	<p>If data is NULL, it must indicate a buffer that has been allocated with a tpalloc().</p> <p>If a buffer is the same as the one transmitted to a service routine, ta Tmax system must handle this buffer. If a service routine writer attempts to release the buffer, this attempt will fail. However, if the buffer transmitted to a tpforward() is not the same as the one transferred at the time of calling the service, a tpforward() will release the buffer.</p>
len	<p>A len is the length of data to be transmitted. If data indicates a buffer that does not require a specific length (for example, a STRUCT Type buffer), len will be ignored and is set to 0. If data is NULL, len will be ignored and a request that has a data length of 0 will be transmitted.</p> <p>A service routine writer cannot regain control after calling a tpforward(), blocking transmission in the form that a TPSIGRSTRT is defined implicitly is used. Therefore, if a signal is generated during a tpforward() operation to stop an operation, processing will continued later. If a blocking condition is encountered, it will wait until a timeout occurs and then sends a service request.</p>
flags	Not currently supported. Set to TPNOFLAGS.

- Return Values

A service routine does not return any value to a Tmax system, that is, a caller. In other words, a service routine will be declared as void.

- Errors

When a `tpforward()` fails to execute, a `tperrno` will be set to one of the following values:

Error Code	Description
[TPESVCERR]	Occurs when an invalid buffer is used; a <code>tpacall()/tpconnect()</code> returns a <code>cd</code> ; a <code>tpforward()</code> was used instead of a <code>tpreturn()</code> during interactive communication; a <code>TPEV_DISCONIMM</code> event occurred; or an XA operation (<code>tx_begin()</code> , <code>tx_rollback()</code> , and a <code>tx_commit()</code>) failed in a transaction mode.
[TPETIME]	Occurs when a transaction timeout occurs during a service routine or while a service request is being transmitted.

- Examples

```
#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);
}
```

- Related Functions

`tpalloc()`, `tpconnect()`, `tpreturn()`

10.1.3. tpsvrinit

Processes an initialization of a service routine. This function is called for initializing the main procedure of a Tmax application server program. This routine is called after a server process is executed but no service request has been processed. It is possible to execute Tmax communications or define a transaction in the `tpsvrinit()` routine.

If the application program does not provide `tpsvrinit()`, default routine provided by the Tmax is called instead. If the server belongs to a server group that processes transactions, `tpsvrinit()` calls **tx_open()** and **userlog()**, to inform that the server has started successfully.

Command line parameters (CLOPT) of an application program can be transferred to the server, to be processed by `tpsvrinit()`. See CLOPT item of SERVER section of tmax configuration file. The parameters are transferred through `argc` and `argv`.

As **getopt()** is used in the Tmax server main, `optarg`, `optind` and `opterr` are used for parameter parsing and error detection in the `tpsvrinit()`.

- Prototype

```
# include <tmaxapi.h>
int tpsvrinit (int argc, char **argv)
```

- Parameters

Parameter	Description
argc	The number of command line parameters.
argv	A command line parameter.

- Return values

Value	Description
0	A function call was successful.
Negative number	A function call failed. No service requests are received and a server process is terminated without generating an error.

- Examples

```
#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=tpgetcliid();
if (cd==-1) { error processing }
ret=tpsendtocli(cd, buf, 0, TPNOFLAGS);
if (ret==-1) { error processing }

data process....
printf(" Service end\n");
EXEC SQL COMMIT WORK RELEASE;
tpreturn(TPSUCCESS, buf, strlen(buf), 0);
}

```

- Related Functions

tx_open(), tpsvrdone()

10.1.4. tpsvrdone

Sets a routine to be executed when a UCS server process is terminated. Tmax application server program's main() provided by the Tmax systems calls tpsvrdone() before ending a process after finishing all service request handling. When this routine is executed, a server process will still be a part of Tmax but the process will not provide service. It is possible to execute a Tmax communication or define a transaction in the tpsvrdone() routine.

If the tpsvrdone() is keeping an interactive connection, is waiting for asynchronous replies, or returns during a transaction mode, Tmax will disconnect the interactive connection, ignore the asynchronous replies that have been waiting, and stop the transaction. Then, the server will be terminated immediately.

If a program does not provide a tpsvrdone() routine, a default routine provided by Tmax can be called instead. If a server belongs to a server group that processes transactions, the default tpsvrdone() routine will call **tx_close()** and **userlog()** to notify that a server will be terminated. If either tpreturn() or tpforward() are called in tpsvrdone(), a routine will simply return without performing any operations.

- Prototype

```

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

```

- Return Values

A tpsvrdone() is a function used to perform necessary jobs before a developer ends a server process. Due to this, it does not have a return value and an error cannot be generated.

- Examples

```
#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;
}
```

- Related Functions

tx_close(), tpsvrinit()

10.1.5. tpsvrthrinit

Available only in multithread and multicontext servers. A Tmax server provides a tpsvrinit function that performs an initialization process when a server process starts. Multithread and multicontext servers provide an initialization function for each thread when a thread is created for service threads managed by a thread pool after a tpsvrinit function is called.

A thread pool works according to MINTHR and MAXTHR fields, so when a server process initially starts, the same number of service threads will be created as the number set in MINTHR and they will call a tpsvrthrinit(). After that, if there are no idle service threads in a thread pool, service threads will be created as needed up to a value set as a MAXTHR. If MINTHR is 0, no service threads will be created when a process starts, so only a tpsvrinit() will be called and the process will wait until a service request is received.

This function is processed after a `tpsvrinit()` is called and before each thread handles a service request. The same parameters with those delivered to a `tpsvrinit()` are delivered. These parameters are set in a CLOP field in the SERVER section of a configuration file. When writing this function, a user must remember that the parameters delivered to a `spsvrinit()` and a `tpsvrthrinithrinit()` must be the same. For more information, refer to [tpsvrinit](#).

- **Prototype**

```
# include <tmaxapi.h>
int tpsvrthrinithrinit (int argc, char **argv)
```

- **Parameters**

Parameter	Description
argc	The number of command line parameters.
argv	A command line parameter.

- **Return Values**

When performing an initialization using a `tpsvrthrinithrinit()`, if a process fails, a user will return -1. After a server process calls a `tpsvrthrinithrinit()`, if -1 is returned, the server process will cancel the starting process and will be terminated.

Value	Description
0	A function call was successful.
Negative number	A function call failed.

- **Examples**

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

void tpsvrthrinithrinit(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 = tpgetctx(&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 (tpsetctx(param->ctxtid, TPNOFLAGS) == -1) {
                printf("tpsetctx(%d) failed, [tperrno:%d]", param->ctxtid,
                    tperrno);
                return NULL;
            }
        }

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

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

```

- Related Functions

tpsvrthrdone()

10.1.6. tpsvrthrdone

Available only in multithread and multicontext servers. Multithread and multicontext servers terminate service threads before performing a tpsvrdone when a server process is terminated. If this function is defined, a service thread will call this function automatically when it is terminated. Developers need to write a routine to process required jobs before a thread is terminated. In this function, a Tmax communication or a transaction can be performed. If a function is returned without completing jobs, all incomplete jobs will be ignored when a thread is terminated.

A client can allocate other previously created context to a current client using a function. Most ATMI functions are per-context based. A client can use multiple contexts, but only one context is used at a time. For example, if context1 calls a tpacall(), context1 must be set as the current context at the moment of calling a tpgetrply() to perform a tpgetrply() normally even when other context was used. For more information, refer to [tpsvrdone](#).

- Prototype

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

- Return Values

A tpsvrthrdone() is written by a developer to perform necessary jobs before terminating a server process. This function neither has any return value nor generates an error.

- Examples

Refer to the example in [tpsvrthrinit](#).

- Related Functions

tpsvrthrinit()

10.1.7. tpgetctxt

This function returns the context ID, which is currently set in a thread, as the first parameter. This function is used differently in the client and server. For more information, refer to [tpgetctxt](#).

10.1.8. tpsetctxt

This function sets the current context. This function is used differently in the client and server. For more information, refer to [tpsetctxt](#).

10.1.9. tpsendtocli

Available only in a server. This function sends an unrequested message to a specified client. While a **tpbroadcast()** function sends an unrequested message to any client connected to a Tmax systems, this function sends a message only to a client that has requested a service provided by a server process.

- Prototype

```
# include <tmaxapi.h>
int tpsendtocli (int clid, char *data, long len, long flags)
```

- Parameters

Parameter	Description
clid	A unique client number obtained with a tpgetclid().
data	A buffer allocated by a tpalloc(). If data indicates a buffer that does not require a specific length, a len is ignored (0 is used in general). If data indicates a buffer that needs to have a length specified, a len must not be 0. In addition, when data is NULL, a len will be ignored.
len	A buffer length to be transmitted.
flags	<p>Operation method.</p> <p>The following values are available.</p> <ul style="list-style-type: none">• TPNOFLAG(0) <p>Messages must be received by a client. However it might take a long time for a client to receive a requested result if a client cannot process received messages fast enough.</p> <ul style="list-style-type: none">• TPUDP <p>This flag does not mean that data communication with a client is UDP. When a caller transmits data, it is impossible to transfer the data because the internal buffer that the data is transmitted to is filled with messages to be transferred. This flag means that data may be discarded in that situation. In other words, it means that data might be lost similarly to a communication in a UDP.</p> <ul style="list-style-type: none">• TPFLOWCONTROL <p>Checks the status of a client to decide whether another request message can be sent or not. If there are too many accumulated messages, a tpsendtocli() will return a value of -1 and a tperrno will be set to TPEQFULL. This flag is used to reduce the load of a Tmax system.</p>

- Return Values

Value	Description
1	A function call was successful.
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a `tpsendcli()` fails to execute, a `tperrno` will be set to one of the following values.

Error Code	Description
[TPEBADDESC]	An invalid clid.
[TPEPROTO]	A <code>tpsendcli()</code> was called from an invalid state.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information will be recorded in a system log file.
[TPEOS]	An operating system error occurred.
[TPEQFULL]	A duplicate message exists.

- Examples

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

SERVICE(TPSVCINFO *msg)
{
    int ret, clid;
    char *buf;

    buf = (char *)tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
    strcpy(buf, msg->data);
    data process...
    clid = tpgetclid();
    if (clid==-1) { error processing }

    ret=tpsendtocli(clid, (char *)buf, 0, 0);
    if (ret==-1) { error processing }
    tpreturn(TPSUCCESS, 0, 0, 0);
}
```

- Related Functions

`tpbroadcast()`

10.1.10. tpgetclid

Retrieves the ID of a client connected to a Tmax system. This ID is a unique number in a domain system. In other words, even though a domain system is built up with multiple nodes, unique numbers are given to each client. This function is available only for a server. An ID is retrieved so that other functions, such as a **tpsendtocli()** function, can use it for sending a message to a client.

- Prototype

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

- Return Values

Value	Description
A positive integer	A function call was successful and a positive integer value that corresponds to a client number is returned.
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a tpgetclid() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPESYSTEM]	A tpgetclid() was called from an invalid state. For example, it was used in a client program.
[TPEOS]	An operating system error occurred.

- Examples

```
#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);  
}
```

- Related Functions

tpsendtcli()

10.1.11. tpchkclid

Checks if a client corresponding to an ID is connected to a node where a server process resides in. When developing a RDP type server program, if a service routine stores a connected client ID and a usermain() routine sends a message to a tpsendtcli(), unnecessary errors can be prevented.



If a client is not directly connected to the node where a server process resides in a RDP type, a tpsendtcli() cannot be used.

- Prototype

```
#include <tmaxapi.h>  
int tpchkclid(int clid)
```

- Parameters

Parameter	Description
clid	A unique client number obtained with a tpgetclid() .

- Return Values

Value	Description
-2	A client is not the client that is connected to a local node.
-1	A client is not connected.
1	A client is connected normally.

- Errors

When a tpchkclid() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPEINVAL]	A client is not connected in a local node or an invalid client number was entered.
[TPENOREADY]	A client is not normally connected to a server.

- Examples

```
int _discon(char **buf)
{
    int clid, n;
    clid = tpgetclid();
    n = tpchkclid(clid);
    if (n < 0) {
        printf("Invalid Client\n");
        return -1;
    }
    ...
}
```

- Related Functions

tpgetclid()

10.2. UCS

The following describes the functions that are used in a UCS type server program.

- API Schedule related API

In a UCS type, the user-developed usermain() routine is used like the main() routine so the usermain() routine must handle various messages received from TMM and CLH using a scheduling API.

Function	Description
tpschedule	Checks if a message is received in CLH or in a user-defined FD within a specified amount of time (sec).
tpuschedule	Checks if a message is received in CLH or in a user-defined FD within a specified amount of time (microsec).

- Socket FD related macros

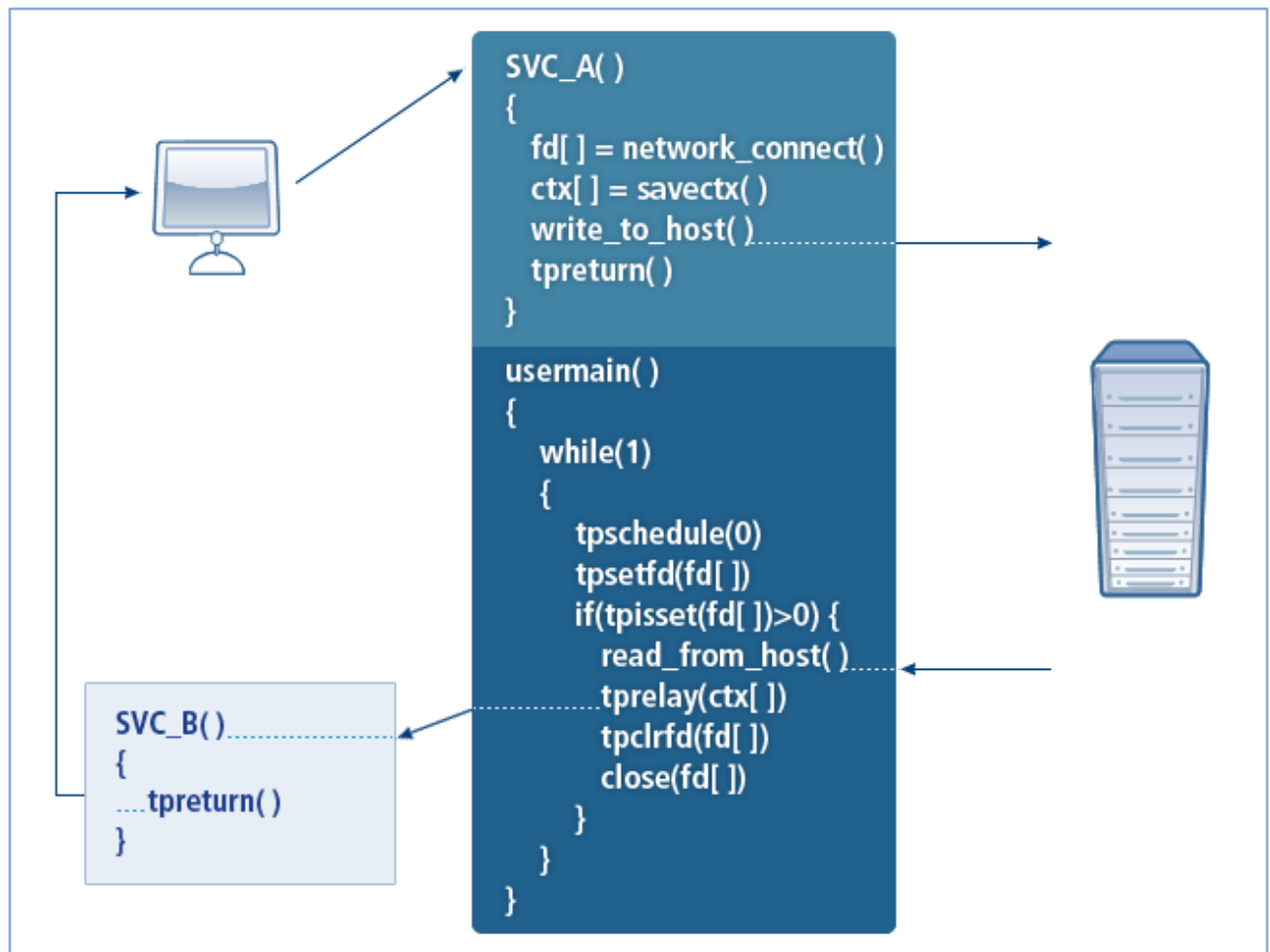
Socket related macros are used to use a user socket FD in a USC scheduler such as tpschedule(). These macros are similar to FD_SET, FD_CLR, and FD_ISSET, which are used in a regular network program.

Function	Description
tpsetfd	Registers the socket FD in an external socket scheduler of the UCS process.
tpissetfd	Checks if a message is received in the FD.
tpclrfd	Releases the FD from a USC scheduler.

- Service Forwarding

When operating with a general host system, a server program, instead of a client, opens a socket to send and receive data. If the connection to an external host system is unstable or the service runtime is long, the server program enters a blocking state and cannot perform additional services.

To resolve this situation, a service routine stores only the client information after receiving a client request. The service routine then sends the request to an external host. The `usermain()` routine is used to send the response to another service routine with the stored client information. If the service routine sends the result value to a client, a server program can perform all processes without being blocked.



UCS Type Service Forwarding

Function	Description
tpsavctx	Stores the client information in a server library.
tpgetctx	Stores the CTX_T structure value of the server library to a user variable.
tpcancelctx	Deletes the CTX_T structure content from a server library.
tprelay	Sends client maintenance to another service routine with the client information/transaction information, which was obtained using <code>tpgetctx()</code> or <code>tpsavctx()</code> . Similar to <code>tpforward()</code> , which is used in a TCS type server program.

- Asynchronous communication in the `usermain()` routine

If asynchronous communication that requires a long service time is used in the `usermain()` routine, `tpgetreply()` may cause blocking, which prolongs the schedule. If the service result is processed through a callback function instead of `tpgetreply()`, the result value can be processed without affecting the schedule. However, if asynchronous communication occurs every time in the `usermain()`, which has a short loop time, the maximum number of asynchronous services may be exceeded, which generates an error.

Function	Description
tpregcb	Sets a callback function that processes asynchronous service requests.
tpunregcb	Releases the set callback function.



For more information about each function, refer to *Tmax Reference Guide*.

10.2.1. tpschedule

Waits for data to be received in a UCS server process. This function is available only in a UCS server process. It sleeps until a maximum timeout value and returns data immediately when data is received.

This function is returned after a corresponding service is automatically executed when data is received. Therefore, a user must not execute any service after data is received.



Services including UCS services are performed by systems unconditionally.

- Prototype

```
#include <ucs.h>
int tpschedule(int timeout)
```

- Parameters

Parameter	Description
timeout	The amount of time to wait in seconds. <ul style="list-style-type: none">• -1: Only checks whether data arrives or not and then returns immediately.• 0: Waits indefinitely until data arrives.

- Return Values

Value	Description
Positive Integer	A function was performed successfully so data will be received.

Value	Description
-1	Data was not received until a timeout or an error occurred due to a function execution failure. If data was not received until a timeout, -1 will be returned and the number 13 error (TPETIME) will be set in a tperrno. In other cases, an error code will be set in a tperrno.

- Errors

When a tpschedule() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPESYSTEM]	A Tmax system error occurred. Detailed error information will be recorded in a system log file.
[TPEOS]	An operating system error occurred.
[TPETIME]	Data was not received before a timeout.

- Examples

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

- Related Functions

tpsleeep(), tp_sleep(), tp_usleep()

10.2.2. tpuschedule

Waits for data to be received in a UCS server process. It is available only in UCS server processes. A tpuschedule() will sleep until a maximum timeout value is reached and will return data immediately when data is received.

- Prototype

```
#include <ucs.h>
int tpuschedule (int timeout)
```

- Parameters

Parameter	Description
timeout	The amount of time to wait in microseconds. <ul style="list-style-type: none">• -1: Only checks whether data arrives or not and then returns immediately.• 0: Waits indefinitely until data arrives.

- Return Values

Value	Description
0	Data was not received before a timeout.
Positive integer	Data was received before a timeout.
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a tpuschedule() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPESYSTEM]	A Tmax system error occurred. Detailed error information is recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

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

```
}
```

- Related Functions

tpschedule()

10.2.3. tpsetfd

Registers a socket FD in an external socket scheduler of a UCS process. This is used to turn on a socket FD, which uses a UCS type process. A UCS scheduler tests a message received in a socket FD as well as messages received in a TMM and a CLH. If a message is received in a user defined socket, a tpschedule() will return a normal result (UCS_USER_MSG) without a separate process. To know in which socket a message has been received, a tpistfd() must be used.

- Prototype

```
#include <ucs.h>
int tpsetfd (int fd)
```

- Parameters

Parameter	Description
fd	Sets a socket FD to be registered.

- Return Values

Value	Description
1	A function call was successful.
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a tpsetfd() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPESYSTEM]	A Tmax system error occurred. Detailed error information will be recorded in the system log file.
[TPEOS]	An operating system error occurred.

- Examples

```
#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)){
        /* Reads a buffer from a socket */
        fd_read(newfd, buf, 1024);
        ret = tpcall("SERVICE", (char *)buf, sizeof(buf), (char **)&buf,
                    (long *)&rln, TPNOFLAGS);
        if (ret == -1){
            error processing
        }
        ...
        tpclrfd(newfd);
        close(newfd);
    }
    ...
}
return 1;
}

```

- Related Functions

tpclrfd(), tpissetfd()

10.2.4. tpissetfd

Used to check if data is received in a socket FD in a UCS process. It is used for scheduling an external socket in a UCS server process.

- Prototype

```

#include <ucs.h>
int tpissetfd (int fd)

```

- Parameters

Parameter	Description
fd	An internal FD of a fdset to be tested.

- Return Values

Value	Description
A positive number	A message was received.
0	A message was not received.

Value	Description
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a tpissetfd() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPESYSTEM]	A Tmax system error occurred. Detailed error information will be recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

```
#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)){
                /* Reads a buffer from a socket */
                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;
}

```

- Related Functions

tpissetfd(), tpsetfd()

10.2.5. tpclrfd

Turns off a socket FD in an internal fdset of a UCS process. It is used for scheduling an external socket in a UCS server process.

- Prototype

```

#include <ucs.h>
int tpclrfd (int fd)

```

- Parameters

Parameter	Description
fd	The socket of an internal fdset to be turned off.

- Return Values

Value	Description
1	A function call was successful.
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a tpcrlfd() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPESYSTEM]	A Tmax system error occurred. Detailed error information will be recorded in a system log file.
[TPEOS]an	An operating system error occurred.

- Examples

```
#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)){
        /* Reads a buffer from a socket */
        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;
}

```

- Related Functions

tpissetfd()

10.2.6. tpsavctx

Manages client information in a UCS process. This function is used along with a **tprelay()**, which forwards a request to another service. It works in the same way as a general service program, which calls for other services as a **tpforward()**. Consequently, a called service sends a processing result to a client.

A **tpsavctx()** function can be used to communicate with an external process that has heterogeneous protocols that are time-consuming and can block channels.

The function can be used in the following format:

```

Client → svc1 → svc2(service, tpsavctx) → External Channel
Client ← svc3 ← svc2(usermain, tprelay) ← External Channel

```

1. A client makes a service request to svc1.
2. svc1 calls svc2 using a **tpforward (...TPNOREPLY)**.
3. svc2 is a service which runs in a UCS process, and it calls a **tpsavctx()** in a service routine to save

client information to communicate with an external system.

4. A result is sent to a usermain and forwarded to svc3 via a `tprelay()`. svc3 considers that svc2 has called it via a `tpforward()`, so it finally sends a result to the client.

In this process, because svc1 calls a service via a `tpforward` with a flag set to `TPNOREPLY`, it can prevent channel congestion. This enables a large numbers of clients to be handled with a small number of processes. Additionally, a single UCS process can act as both a sending and receiving process. It can organize a relatively simple system with efficient system management.

- Prototype

```
#include <ucs.h>
CTX_T * tpsavctx(void)
```

- Return Values

Value	Description
CTX_T	A function call was successful.
NULL	A function call failed. A <code>tperrno</code> is set to an error code.

- Errors

When a `tpsavctx()` fails to execute, a `tperrno` will be set to one of the following values:

Error Code	Description
[TPEPROTO]	A <code>tpsavctx()</code> must be used in a service routine. Otherwise, <code>TPEPROTO</code> will be returned. Accordingly, it cannot be used with a <code>tpsvrinit()</code> or a <code>tpsvrdone()</code> .
[TPESYSTEM]	A <code>Tmax</code> system error occurred. Detailed error information will be recorded in a system log file. The error occurred during memory allocation.

- Examples

```
...
#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);
}

```

- Related Functions

tpreturn(), tpforward(), tprelay()

10.2.7. tpgetctx

Copies current client information to a CTX_T structure that was declared and allocated by a user. If a tpgetctx() is used, and if a tprelay() is not used to use information, a client will keep waiting for a response even after a service routine completes.

The information obtained by a tpgetctx() cannot be canceled with a tpcancelctx(), so a tprelay() must be used only in a service routine.

- Prototype

```

#include <tmaxapi.h>
int tpgetctx (CTX_T *ctxp)

```

- Parameters

Parameter	Description
ctxp	Receives client information that was stored with a tpsavectx(), to a CTX_T structure.

- Examples

```
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

Cancels a corresponding structure content among information saved using a tpsavectx(). Even when a tprelay() is not performed, if a service routine is terminated, a result will be returned normally.

A tpgetctx() can be used only in a service routine.

- Prototype

```
#include <ucs.h>
int tpcancelctx(CTX_T *ctxp);
```

- Parameters

Parameter	Description
ctxp	Deletes a CTX_T structure saved in a library.

The following is a CTX_T structure definition:

```
typedef struct {
    int    version[4];
    char   data[CTX_USR_SIZE - 16];
} CTX_T;
```

- Examples

```
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

Available only in a UCS server process. It can be used in multiple nodes. When a client requests a service, this function requests another service with information about the client. Since the service called by the function notices that it was called by the client, not the function, it returns a result to the client.

A service execution result can be sent to a client that called for a request, so a fast response can be induced with a simple structure in a UCS process. In general, this function is useful when processing a service, because it is integrated with an external application which is a program routine that can obtain results after calling a service two or three times.

If a server process is terminated after saving client information using a `tpsavctx()` or a `tpgetctx()` but before a request is sent to another service using a `tprelay()`, an error response will be sent to a service caller automatically. For more information about error responses, refer to the `CTX_EREPly` option in the `SERVER` section of an environment configuration. These operations are supported for Tmax versions v5.0 SP2 or later. In earlier versions, an error response will not be sent to a service caller.

- Prototype

```
#include <ucs.h>  
int tprelay(char *svc, char *data, long len, long flags, CTX_T *ctxp);
```

- Parameters

Parameter	Description
svc	A service name registered in a Tmax configuration file.
data	Data to be transmitted when a service is called. If data is not NULL, it must indicate a buffer that has been allocated with a <code>tpalloc()</code> .
len	The length of data to be sent. It must be specified for <code>CARRAY</code> , <code>X_OCTET</code> , and <code>Structure array Types</code> .
flags	Not currently supported. Set to <code>TPNOFLAGS</code> .
ctxp	An information structure retrieved through a <code>tpgetctx()</code> or a <code>tpsavctx()</code> .

- Return Values

Value	Description
1	A function call was successful.
-1	A function call failed. A <code>tperrno</code> is set to an error code.

- Errors

When a `tprelay()` fails to execute, a `tperrno` will be set to one of the following values:

Error Code	Description
[TPEINVAL]	An invalid parameter. For example, a <code>ctx</code> is NULL or an incorrect buffer was used.
[TPESYSTEM]	A Tmax system error occurred. Detailed error information will be recorded in a system log file.

- Examples

```
...
#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);
}

```

- Related Functions

tpreturn(), tpforward()

10.2.10. tpregcb

Sets a routine that receives a response for an asynchronous UCS request from a server. This routine is used when a UCS type process receives a response from a server program. It is used instead of a **tpgetrply()** in a UCS server process.

- Prototype

```

#include <ucs.h>
int tpregcb (UcsCallback)

```

- Parameters

Parameter	Description
UcsCallback	Sets a Callback function that handles a response for an asynchronous request in a UCS.

- Values

Return Value	Description
1	A function call was successful.
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a tpregcb() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPESYSTEM]	A Tmax system error occurred. Detailed error information will be recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

```

...
#include <stdio.h>

```

```

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

void reply_receive(UCSMSGINFO *reply);
DUMMY(TPSVCINFO *msg)
{
    data process ...
}

int usermain(int argc, char *argv[])
{
    int ret;
    char *buf

    ret = tpregcb(reply_receive);
    if (ret == -1){ error processing }
    buf=tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
    data process...
    while(1)
    {
        ...
        tpschedule(3);
        cd = tpacall("SERVICE", buf, strlen(buf), TPNOFLAGS);
        if (cd < 0) { error processing }
        ...
    }
}

void reply_receive(UCSMSGINFO *reply)
{
    printf("data....%s\n", reply->data);
}

```

- Related Functions

tpunregcb()

10.2.11. tpunregcb

Used to reset a routine that receives a response for an asynchronous request. It is used in a UCS server process.

- Prototype

```

#include <ucs.h>
int tpunregcb (void)

```

- Values

Value	Description
1	A function call was successful.
-1	A function call failed. A tperrno is set to an error code.

- Errors

When a tpunregcb() fails to execute, a tperrno will be set to one of the following values:

Error Code	Description
[TPESYSTEM]	A Tmax system error occurred. Detailed error information will be recorded in a system log file.
[TPEOS]	An operating system error occurred.

- Examples

```
...
#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);
}
```

- Related Functions

tpregcb()

11. Error Handling

This chapter describes error handling and debugging.

11.1. Overview

Tmax APIs set an error number based on the situation in which an error occurs. Error messages help you find the cause of the errors. In order to know the system call level error information, refer to the error messages and APIs defined in `tuxinc/Uunix.h`.

Tmax provides CLH, which displays information about the console, to help find the problems that occur in the Tmax system, not at the application level.



For more information about error messages that can occur during the operation of the Tmax system, refer to *Tmax Error Message Reference Guide*.

11.2. API Level Error Processing

The return value of the Tmax API failure differs depending on the API. The error number of each error situation is set in `tperrno`.

11.2.1. `tpstrerror`

Available in both a server and a client. This function displays a message corresponding to an error number. When an error occurs while using Tmax APIs, a relevant error code will be specified in a `tperrno`, which is a global variable. A `tpstrerror()` displays messages about an error specified in a `tperrno`.

- Prototype

```
# include <atmi.h>
char *tpstrerror (int tperrno)
```

- Parameters

Parameter	Description
<code>tperrno</code>	An error code that will display an error message.

- Return Values

Value	Description
Error Message	A message exists for an error code.
NULL	A message does not exist for an error code.

- Examples

```
#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", tpstrerror(tperrno));
        exit(1);
    }

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

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

11.3. System Level Error Processing

Tmax APIs use many system calls. To check a specific system call error generated due an operating system or platform error or to port error messages to heterogeneous platforms, the error messages can be integrated and managed by using the API explained in the following.

The following is the location of the header file.

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

11.3.1. Uunixerr

Variable that is set to an integrated error number if a system call error occurs.

```
int Uunixerr
```

11.3.2. Uunix_err

Writes the type of system error to a stderr when an ATMI API call fails and a tperrno is set to a TPEOS.

- Prototype

```
# include <Uunix.h>
void Uunix_err (char *msg)
```

- Parameters

Parameter	Description
msg	<p>The messages that follow the name of a system call that failed. Generally, a program name is recorded.</p> <p>One of the following is displayed.</p> <ul style="list-style-type: none">• 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

- Examples

```
ret=tmaxreadenv("NO THAT FILE", "TMAX");
if (ret<0) {
    Uunix_err("myprog");
    exit(1);
}
```

The following is the result of executing the previous example.

```
myprog: UOPEN
```

11.3.3. Ustrerror

Returns an integrated error message for a system error code (errno).

- Prototype

```
# include <Uunix.h>
```

```
char * Ustrerror(int err);
```

- Parameters

Parameter	Description
err	An integrated error number for an error message.

- Return Values

When a system call is succeeded, a pointer to an integrated error message for an errno is returned.

One of the following is returned through a chr * pointer.

- 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

- Examples

```
ret=tmaxreadenv("NO THAT FILE", "TMAX");
if (ret<0)
{
    printf("%d->%s\n", Unixerr, Ustrerror(Unixerr));
    exit(1);
}
```

The following is the result of executing the previous example.

```
11->UOPEN
```

11.3.4. tmaxoserrno

Variable that is set to an integrated error number if a system call error occurs. The error number can be changed if another error occurs, which makes difficult to find a reason. tmaxoserrno has a system error number at the point in time when TPEOS or TPESYSTEM occurs. Windows saves GetLastError() values and Unix saves errno values.

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


11.4. Debug

11.4.1. Debug CLH

If changing clh.dbg, which is located in TMAXDIR/bin, to CLH, all the messages handled in CLH can be checked. The original CLH file must be backed up.

```
/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. Debug Library

The libsvrd.a and libsvr.so libraries exist in the TMAXDIR/lib directory. If these libraries are used instead of libsvr.a and libsvr.so, various data values are displayed on the console screen that help the user grasp the flow in the server library and find the point where the error occurred. For libsvrd.so, only the name must be changed. libsvrd.a must be recompiled.

```
/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
.....
```

12. Examples

This chapter describes examples of Tmax programs developed in various environments.

12.1. Programs for Each Communication Type

This section describes examples of synchronous, asynchronous, and interactive programs.

12.1.1. Synchronous Communication

In the following example, a client copies a string to a STRING buffer and calls a service. The service routine of a server receives the string, converts it to all upper case, and then returns the converted string.

Program Files

- Common program

File	Description
sample.m	Tmax configuration file.

- Client program

File	Description
sync_cli.c	Client program.

- Server program

File	Description
syncsvc.c	Service program that converts a string to all upper case.
Makefile	Tmax makefile that must be modified.

Program Feature

- Client program

Feature	Description
Tmax connection	Basic connection (no client information).
Buffer type	STRING.
Communication type	Synchronous communication using tpcall().

- Server program

Feature	Description
Service	TOUPPERSTR.
Database connection	None.

Tmax Configuration File

The following is an example.

<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
```

Client Program

The following is an example.

<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( );

```

Server program

The following is an example.

<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. Asynchronous Communication

In the following example, a client copies a string to a STRUCT buffer and calls a service. The service routine of a server receives the string, converts it to upper or lower case, and then returns the converted string. The client requests the TOUPPER service through asynchronous communication and then calls the TOWER service through synchronous communication. The client receives the TOWER service result first and then receives the TOUPPER service result.

Program Files

- Common program

File	Description
demo.s	Defines a struct buffer.
sample.m	Tmax configuration file.

- Client program

File	Description
async_cli.c	Client program.

- Server program

File	Description
asynsvc.c	Service program converts a string to upper or lower case.
Makefile	Tmax makefile that must be modified.

Program Feature

- Client program

Feature	Description
Tmax connection	Basic connection.
Buffer type	STRUCT.
Communication type	Synchronous and asynchronous.

- Server program

Feature	Description
Service	TOUPPER, TOWER.

Feature	Description
Database connection	None.
Communication type	Synchronous and asynchronous.

Struct Buffer

The following example is a struct buffer used for asynchronous communication.

<demo.s>

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

Tmax Configuration File

The following is an example.

<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
asynsvc        SVGNAME = svg1, MIN = 1, MAX = 5

*SERVICE
TOUPPER        SVRNAME = asynsvc
TOLOWER        SVRNAME = asynsvc
```

Client Program

The following is an example.

<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, &clen, 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() ;
}

```



```
}
```

Server program

The following is an example.

<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. Interactive Communication

A client receives user input and sends a number through a STRING buffer. The service routine of a server returns customer information with a number that is greater than the sent number in a database table.

The client sends the number along with a communication control to the server through interactive communication. The server reads all database data through a cursor and sends data that meets the condition to the client. The client can check that the data is successfully fetched through

Program Files

- Common program

File	Description
demo.s	Defines a structure.
sample.m	Tmax configuration file.
mktable.sql	Script for creating a database table.
sel.sql	Script for outputting tables and data.

- Client program

File	Description
conv_cli.c	Client program.

- Server program

File	Description
convsvc.pc	Server program.
Makefile	Tmax makefile that must be modified.

Program Feature

- Client program

Feature	Description
Tmax connection	Basic connection.
Buffer type	STRING for transmission and STRUCT for reception.
Communication type	Interactive.

- Server program

Feature	Description
Service	MULTI.
Database connection	Oracle is used.

Struct Buffer

The following example is a struct buffer used for interactive communication.

<demo.s>

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

Tmax Configuration File

The following is an example.

<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
```

The following items are added.

Item	Description
DBNAME	Database name.
OPENINFO	Oracle database connection information.
TMSNAME	Name of the process that handles global transactions.
CONV	Interactive mode server.

Database Script

The following creates an Oracle table.

<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
```

The following outputs the Oracle table and data.

<sel.sql >

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

Client Program

The following is an example.

<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);
    }

    /* connects to Tmax with tpstart() */
    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, TPRECVONLY)) == -1){
        printf("tpconnect failed:CONVER service, tperrno=%d\n", tperrno);
        tpfree(sndbuf);
        tpfree((char *)rcvbuf);
        tpend();
        exit(1);
    }

    /* interactive communication connection, The interaction control is sent
       to a server. */
    printf("tpconnect SUCCESS \"MULTI\" service\n");
    while ( 1 ) { /* receives multiple data */
        printf("tprecv strat\n");
        if( tprecv(cd, (char **) &rcvbuf, &rcvlen, TPNOTIME, &revent) < 0 ) {
            /* If ends with tpreturn() in a server */
            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");
}

```

Server program

The following is an example.

<convsvc.pc>

```

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

EXEC SQL begin declare section; /* Oracle global variables declaration */
    char seq[10];
    struct sel_o *sdbuf;
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 ((sdbuf = (struct sel_o *) tmalloc ("STRUCT", "sel_o", 0)) == NULL) {
        printf("tpalloc failed:\n");
        tpreturn (TPFAIL, -1, NULL, 0, TPNOFLAGS);
    }

    /* declares a cursor for large amount of data */
    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);
    }

    /* sends data while there is no Oracle error */
    while ( sqlca.sqlcode == 0 ){
        EXEC SQL fetch democursor into :sdbuf;

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

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

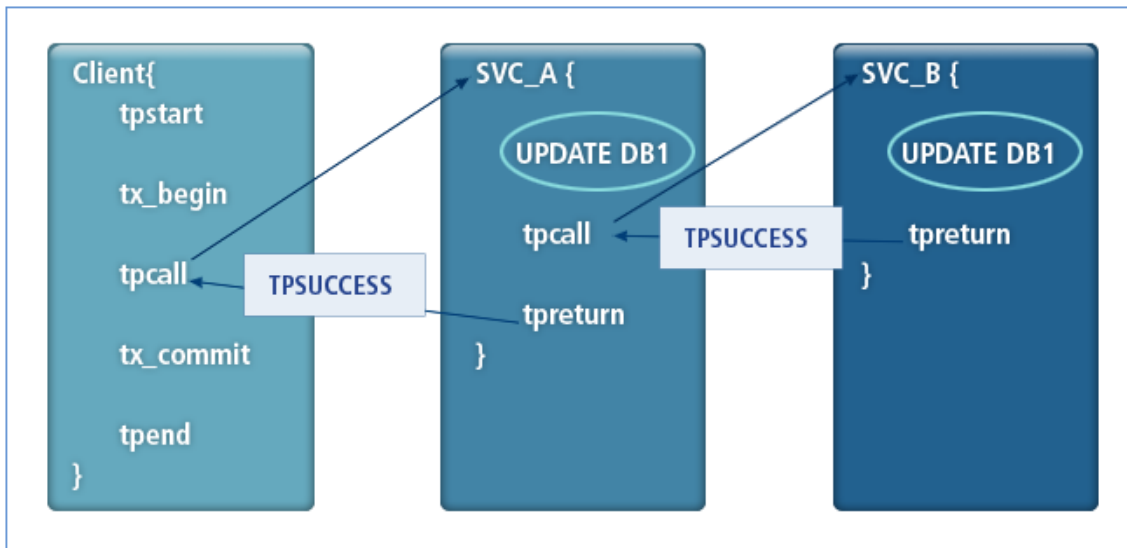
    end_of_fetch:
    exec sql close democursor;
    printf("tpreturn before");
    tpreturn (TPSUCCESS, 0, NULL, 0, TPNOFLAGS);
}

```

12.2. Global Transaction Programs

A global transaction is when multiple resource managers (databases) and physical entities participate in processing one logical unit. Tmax regards all transactions as global transactions, and two-phase commit (2PC) is used for data integrity.

A client receives user input and sends a unique number and data through a struct buffer. A server updates any data that has the unique number and adds it to a table by calling a service that uses another database. If an error occurs, the client can simultaneously roll back both databases because the client specifies the whole process as a single transaction.



Connection to 2 Databases

Program Files

- Common program

File	Description
demo.s	Struct buffer configuration file.
sample.m	Tmax configuration file.
mktable.sql	SQL script for creating a database table.

- Client program

File	Description
client.c	Client program.

- Server program

File	Description
update.pc	Server program that executes UPDATE for a database.

File	Description
insert.pc	Server program that executes INSERT for a database.
Makefile	Tmax makefile that must be modified.

Program Feature

- Client program

Feature	Description
Tmax connection	Basic connection.
Buffer type	STRUCT.
Communication type	Synchronous communication using tpcall().
Transaction handling	Transaction scope is specified by a client.

- Server program

Feature	Description
Server program	2 server programs that use different databases.
Service	UPDATE, INSERT.
Database connection	2 types of Oracle databases.

Struct Buffer

The following example is a struct buffer used for global transactions.

<demo.s>

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

Tmax Configuration File

The following is an example.

<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

```

Database Script

The following creates an Oracle table.

<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

```

Client Program

The following is an example.

<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 = 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);

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();
}

```

Server program

The following example is a server program that executes UPDATE for a database.

<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);
}

```

The following example is a server program that executes INSERT for a database.

<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 *)tpalloc("STRING", NULL, 0);
    if (send == NULL) {
        fprintf(stderr, "tpalloc fail errno = %s\n", tpstrerror(tperrno));
        tpretun(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);
        tpretun(TPFAIL, -1, (char *)NULL, 0, TPNOFLAGS);
    }
    strcpy(send, OKMSG);
    tpretun(TPSUCCESS, 1, (char *)send, strlen(send), TPNOFLAGS);
}

```

12.3. Database Programs

This section describes several examples that illustrate the use of Oracle and Informix databases.

12.3.1. Oracle Insert Program

A client receives user input and calls a service through a struct buffer. A server receives the input and adds it to a corresponding table. If an error occurs, a client can roll back the database by specifying the process as a single transaction.

Program Files

- Common program

File	Description
demo.s	Struct buffer configuration file.
sample.m	Tmax configuration file.
mktable.sql	SQL script for creating a database table.
sel.sql	Script for outputting tables and data.

- Client program

File	Description
oins_cli.c	Client program.

- Server program

File	Description
oinssvc.pc	Oracle source of a service program.
Makefile	Tmax makefile that must be modified.

Program Feature

- Client program

Feature	Description
Tmax connection	Basic connection.
Buffer type	STRUCT.
Communication type	Synchronous communication using tpcall().
Transaction handling	Transaction scope is specified by a client.

- Server program

Feature	Description
Service	ORAINS.
Database connection	Oracle database.

Struct Buffer

The following is an example.

<demo.s>

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

```
};
```

Tmax Configuration File

The following is an example.

<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
```

The following items are added.

Item	Description
DBNAME	Database name.
OPENINFO	Oracle database connection information. CLOSEINFO does not need to be specified for an Oracle database because It is called by tpsvrinfo().
TMSNAME	Name of the process that handles automatic transactions that meet OPENINFO. The service included in svg1 is handled as automatic transitions.

Database Script

The following creates an Oracle table.

<mktable.sql>

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

EOF

The following outputs the Oracle table and data.

<sel.sql>

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

Client Program

The following is an example.

<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);
    }
}
```



```

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

if(tpcall("ORAINS",(char *)sdbuf,0,(char **)&rcvbuf,&rcvlen,TPNOFLAGS)==-1)
{
    printf("tpcall failed:ORA service, tperrno=%d", tperrno);
    printf("sql code=%d\n", tpurcode);
    tx_rollback();
    tpfree ((char *)sdbuf);
    tpfree ((char *)rcvbuf);
    tpend();
    exit(1);
}

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

tpfree ((char *)sdbuf);
tpfree ((char *)rcvbuf);
tpend();
}

```

Server program

The following is an example.

<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");

    /* inserts to a database */
    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 Program

A client receives user input and calls a service through a struct buffer. A server receives all corresponding data and returns it using a structure array. If an error occurs, a client can roll back the database by specifying the process as a single transaction.

Program Files

- Common program

File	Description
demo.s	Struct buffer configuration file.
sample.m	Tmax configuration file.
mktable.sql	SQL script for creating a database table.
sel.sql	Script for outputting tables and data.

- Client program

File	Description
oins_cli.c	Client program.
cdata.c	Function module used by a client.

- Server program

File	Description
oselsvc.pc	Oracle source of a service program.
Makefile	Tmax makefile that must be modified.

Program Feature

- Client program

Feature	Description
Tmax connection	Basic connection.
Service	ORASEL.
Buffer type	STRUCT.
Communication type	Synchronous communication using tpcall().

Feature	Description
Transaction handling	Transaction scope is specified by a client.

- Server program

Feature	Description
Service	ORASEL.
Database connection	Oracle database.
Buffer usage	Buffer size can be changed if necessary.

Struct Buffer

The following is an example.

<demo.s>

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

Tmax Configuration File

The following is an example.

<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
```

The following items are added.

Item	Description
DBNAME	Database name.
OPENINFO	Oracle database connection information. CLOSEINFO does not need to be specified for an Oracle database. Available options are described in the following table.
TMSNAME	Name of the process that handles transactions.
AUTOTRAN	Corresponding service is automatically processed in transaction status.

The following options are available for OPENINFO.

Option	Description
LogDir	Records XA-related log in a specified location. If unspecified, the <xa_NULLdate.trc> file is created in \$ORACLE_HOME/rdbms/log or a current directory.
DbgFl	Level of the debug flag. 0x01 (basic level), 0x04 (OCI level), and other levels can be used.

The following uses the LogDir and DbgFl options for OPENINFO.

```
OPENINFO="Oracle_XA+Acc=P/account/password +SesTm=60+LogDir=/tmp+DbgFl=0x01"
```



To avoid disk full issues, disalbe the degug mode during development.

Database Script

The following creates an Oracle table.

<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_tdb1 on sel_his(account_id);
EOF
```

The following outputs the Oracle table and data.

<sel.sql>

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

Client Program

The following is an example.

<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) { /* connects to Tmax */
        fprintf(stderr, "TPSTART_T(tpinfo) failed -> %s!\n",
            tpstrerror(tperrno)) ;
        exit(1) ;
    }

    /* creates a c struct buffer */
    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;

    /* sets transaction timeout */
    tx_set_transaction_timeout(30);

    /* starts global transactions */
```

```

    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){
        /* request the "ORASEL" service with synchronous communication */
        fprintf(stderr, "Tpcall(SELECT...)error->%s ! ",
            tpstrerror(tperrno)) ;
        tpfree((char *)transf);
        /* cancels a transaction if failed */
        tx_rollback();
        tpend();
        exit(0) ;
    }
    /* commits a transaction if successful */
    if (tx_commit() == -1) {
        fprintf(stderr, "tx_commit() failed ->%s!\n",
            tpstrerror(tperrno)) ;
        tpfree((char *)transf);
        tpend();
        exit(0) ;
    }
    /* Received data is an array of a structure. */
    for (j = 0 ; j < tpurcode ; j++) {
        /* prints data selected by Oracle */
        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);
    }
    /* if there is not selected data or it is the end */
    if (urcode == NOTFOUND) {
        printf("No records selected!\n");
        tpfree((char *)transf);
        tpend();
        return 0;
    }

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

```

Server program

The following is an example.

<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;

    /* transfers contents of the msg buffer to a program variable */
    key = transf->ACCOUNT_ID;

    /* adjusts the size of the transf buffer */
    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
        /* puts data that has account_id greater than the key value sent */
        order by account_id;      /* by a client to a global variable */

    /* sql error check (excludes no data or too many data) */
    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);
    }

    /* puts the number of access to lastno */
    lastno = sqlca.sqlerrd[2];

    /* too many selected data */
    if (sqlca.sqlcode == TOOMANY)
        lastno =rowno;

    /* No records */
    if (lastno == 0)
        transf->ACCOUNT_ID = 0;

    /* puts data selected by Oracle to a buffer to be sent */
    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 Program

A client receives user input and calls a service through a struct buffer. A server receives the input and adds it to a corresponding table. A client can roll back by specifying the process as a single transaction when an error occurs.

Check the following before compiling Informix applications.

1. Unix environment (.profile, .login, and .cshrc)

Set the following items.

```

INFORMIXDIR=/home/informix
INFORMIXSERVER=tnax
ONCONFIG=onconfig
PATH=$INFORMIXDIR/bin: ...
LD_LIBRARY_PATH=/home/informix/lib:/home/informix/lib/esql:
...

```

2. Makefile

Check the following operations and settings.

```

# Server esql makefile

TARGET = <target filename>
APOBJS = $(TARGET).o
SDLFILE = info.s

LIBS    = -lsvr -linfs
# For Solaris, add -lnsl -lsocket.

OBJJS   = $(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 = -lifsq1 -lifasf -lifgen -lifos -lifgls -lm -ldl -lcrypt
          $(INFORMIXDIR)/lib/esql/
checkapi.o -lifglx -lifxa

APPPDIR = $(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) $(APPPDIR)/.
    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
# For Solaris, add -lnsl -lsocket -laio -lelf

```

```

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$(INFOLIBD)
    -L$(INFOELIBDIR) -L
$(INFOELIBDIR) $(INFOLIBS) $(TMAXLIBS)
    mv $(TARGET) $(TMAXDIR)/appbin

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

```

Program Files

- Common program

File	Description
demo.s	Struct buffer configuration file.
sample.m	Tmax configuration file.
mkdb.sql	SQL script for creating a database. XA mode is supported when a database is created in logging mode.
mktable.sql	SQL script for creating a database table.
sel.sql	Script for outputting tables and data.
info.s	SDLFILE.

- Client program

File	Description
client.c	Client program.

- Server program

File	Description
tdbsvr.ec	Server program.
Makefile	Modifies the Makefile provided by Tmax.

Program Feature

- Client program

Feature	Description
Tmax connection	Basic connection.
Buffer type	STRUCT.
Communication type	Synchronous communication using tpcall().
Transaction handling	Transaction scope is specified by a client.

- Server program

Feature	Description
Service	INSERT.
Database connection	Informix database.

Struct Buffer

The following is an example.

<demo.s>

```
struct info {
    char seq[8];
    char data01[128];
    char data02[128];
    char data03[128];
};
```

Tmax Configuration File

The following is an example.

<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

```

The following items are added.

Item	Description
DBNAME	Database name.
OPENINFO, CLOSEINFO	Informix database connection and disconnection information. tpsvrinfo() and tpsvrdone() use the information.
TMSNAME	Name of the process that handles transactions. Automatic transactions that become available due to OPENINFO are handled. The corresponding service included in svg1 is handled in the automatic transaction state.

Database Script

The following creates an Informix table.

<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

```

The following outputs the Informix table and data.

<sel.sql>

```

dbaccess << EOF
database stores7;
select * from testdb1;

```

Client Program

The following is an example.

<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;

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

    /* allocates buffer memory to be used in an application program */
    if ((transf=(struct info *)tpalloc ("STRUCT","info",0))==(struct info *)NULL){
        fprintf(stderr, "tpalloc(struct info, ...) failed ->%s!\n",
            tpstrerror(tperrno)) ;
        tpend();
        exit(1) ;
    }

    /* fills the data fields to be transferred */
    strcpy(transf->seq, "000001");
    strcpy(transf->data01, "Hello");
    strcpy(transf->data02, "World");
    strcpy(transf->data03, "1234");

    /* sets transaction timeout */
    tx_set_transaction_timeout (30);

    /* informs of transaction starts */
    if (tx_begin() < 0) {
        fprintf(stderr, "tx_begin() failed ->%s!\n",
            tpstrerror(tperrno)) ;
        tpfree ((char *)transf);
        tpend( ); exit(0);
    }

    /* calls a service */
    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);
}

/* Transactions are complete. */
if (tx_commit () < 0) {
    fprintf(stderr, "tx_commit() failed ->%s!\n", tpstrerror(tperrno)) ;
    tpfree ((char *)transf);
    tpend( );
    exit(0);
}

tpfree ((char *)transf );
/* disconnects from Tmax */
tpend( ) ;
}

```

Server program

The following is an example.

< tdbsvr.ec>

```

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

EXEC SQL include sqlca.h;

/* a service name */
INSERT(TPSVCINFO *msg)
{
    /* declares a buffer type for the program */
    struct info *INFO;

    /* declares a buffer type for SQL statements */
    EXEC SQL begin declare section;
    varchar seq[8],buf01[128],buf02[128],buf03[128];
    EXEC SQL end declare section;

    /* receives data in a structure format from the message buffer */
    INFO = (struct info *)msg -> data;

    /* copies data received in a structure format to a database buffer */
    strcpy(seq, INFO->seq);
    strcpy(buf01, INFO->data01);
    strcpy(buf02, INFO->data02);
    strcpy(buf03, INFO->data03);

    /* performs an Insert SQL statement */
    EXEC SQL insert into testdb1 (seq,data01,data02,data03)
    values(:seq, :buf01, :buf02, :buf03);

    /* if an error occurs */
    if ( sqlca.sqlcode != 0) {

```

```

        /* informs Insert is failed */
        printf("SQL error => %d !" ,sqlca.sqlcode);
        tpreturn (TPFAIL, sqlca.sqlcode, NULL, 0, TPNOFLAGS);
    }

    /* when Insert successfully completes */
    tpreturn (TPSUCCESS, 0, NULL, 0, TPNOFLAGS);
}

```

12.3.4. Informix Select Program

A client receives user input and calls a service through a struct buffer. A server receives all corresponding data and returns it using a structure array. If an error occurs, a client can roll back the database by specifying the process as a single transaction.

Program Files

- Common program

File	Description
acct.s	SDLFILE.
sample.m	Tmax configuration file.
mkdb.sql	SQL script for creating a database.
mktables.sql	SQL script for creating a database table.
sel.sql	Script for outputting tables and data.

- Client program

File	Description
client.c	Client program.
cdate.c	Function module used by client.c.

- Server program

File	Description
sel_acct.ec	Informix source of a service program.
Makefile	Tmax makefile that must be modified.

Program Feature

- Client program

Feature	Description
Tmax connection	Basic connection.
Buffer type	STRUCT.
Communication type	Synchronous communication using tpcall().
Transaction handling	Transaction scope is specified by a client.

- Server program

Feature	Description
Service	SEL_ACCT.
Database connection	Informix database.
Buffer usage	A buffer can be resized if necessary.

Struct Buffer

The following is an example.

<acct.s>

```
struct stru_acct {
    int  ACCOUNT_ID;
    char PHONE[20];
    char ADDRESS[80];
};
```

Tmax Configuration File

The following is an example.

<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
```

The following items are added.

Item	Description
DBNAME	Database name.
OPENINFO, CLOSEINFO	Informix database connection and disconnection information. tpsvrinfo() and tpsvrdone() use the information.
TMSNAME	Name of the process that handles transactions. Automatic transactions appointed by OPENINFO are handled. The corresponding service included in svg1 is handled in automatic transaction status.

Database Script

The following creates an Informix table.

<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
```

The following outputs the Informix table and data.

<sel.sql>

```
dbaccess << EOF
database stores7;
select * from ACCOUNT;
EOF
```

Client Program

The following is an example.

<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);
    }

    /* repeats the loop as many as times a user wants */
    loop = atoi(argv[1]);

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

    /* sec1 = start time */
    htime(ts, &sec1); key=0;

    /* allocates a message buffer */
    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) { /* starts transactions */
            fprintf(stderr, "tx_begin() failed ->%s!\n", tpstrerror(tperrno));
            tpfree((char*)transf);
        }
    }
}
```

```

        tpend();
        exit(0);
    }

/* calls a select service */
if (tpcall("SEL_ACCT", (char *)transf, 0, (char **)&transf, &nrecv,
    TPNOFLAGS)== -1) {
    /* service error: the message buffer is freed, the transaction is
        cancelled, and the connection is terminated */
    fprintf(stderr,"Tpcall(SELECT...)error->%s! " ,
        tpstrerror(tperrno)) ;
    tpfree ((char *)transf);
    tx_rollback ( ) ;
    tpend( ) ;
    exit ( 1 ) ;
}
urcode = tpurcode;

/* The service is successfully completed.
    The actual resource is changed as a result of the transaction */
if (tx_commit() < 0 ) {
    fprintf(stderr, "tx_commit() failed ->%s!\n", tpstrerror(tperrno)) ;
    tpfree((char *)transf);
    tpend();
    exit(0);
}

/* if data is selected */
if ( urcode != NOTFOUND) {
    narray =urcode;
    /* the last record of selected data */
    key=transf[narray-1].ACCOUNT_ID;
    /* outputs results to a user as many as the number of selected data */
    for ( j = 0 ; j < narray ; j++ ) {
        if ( j == 0)
            printf("%-10s%-14s%\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 */

    /* increases the number of results */
    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);
/* disconnects from Tmax */
tpend ();

/* sec2 = end time */

```

```

    htime(te,&sec2);

    /* calculates processing time for each data */
    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);
}

```

Server program

The following is an example.

<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;

    /* receives client data */
    ACCT_V = (struct stru_acct *) msg->data;

```

```

/* moves an account ID value to be selected to the key */
key = ACCT_V->ACCOUNT_ID;

/* reallocates the size of the client buffer */
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);
}

/* initializes a buffer */
ACCT_V->ACCOUNT_ID = 0;
strcpy(ACCT_V->PHONE, " " );
strcpy(ACCT_V->ADDRESS, " " );

/* extracts phone and address fields from the ACCOUNT table */
EXEC SQL declare CUR_1 cursor for
    select account_id,phone,address
    into :account_id, :pfone, :address
    from ACCOUNT
    where account_id > :key; /* if an account ID is larger than
                                a field key */

/* 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) {
    /* fetches data from the location a cursor points one at a time */
    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 );

    /* increases the number of selected data */
    nfetch++;
    return_code = nfetch

    /* exits from "while"
    if data is selected as many as the number of NFETCH */
    if (nfetch > NFETCH) {
        nfetch = NFETCH;

```

```

        return_code = nfetch;
        break ;
    }
}
/* cursor close */
EXEC SQL close CUR_1;

/* returns the result and its data to a client */
tpreturn ( TPSUCCESS, return_code, (char *)ACCT_V,
          sizeof(struct stru_acct)*nfetch, TPNOFLAGS);
}

```

12.3.5. DB2 Program

A client receives user input, saves EMPNO to a STRING buffer, and calls a service. A server receives all the data and adds it to the table. If an error occurs, a client can roll back the database by specifying the process as a single transaction.

Program Files

- Common program

File	Description
sample.m	Tmax configuration file.
create.ers	SQL script for creating a database table.

- Client program

File	Description
clidb2tx.c	Client program.

- Server program

File	Description
svr_db2.sqc	DB2 source of a service program.
Makefile	Makefile for compiling TMS and the server program.

Program Feature

- Client program

Feature	Description
Tmax connection	Basic connection.
Buffer type	STRING.

Feature	Description
Communication type	Synchronous communication using tpcall().
Transaction handling	Transaction scope is specified by a client.

- Server program

Feature	Description
Service	XASERVICE2.
Database connection	DB2 database.

Considerations before the DB2 integration test

Check the following when integrating to DB2.

1. DB2 client engine (32-bit or 64-bit).
2. DB2 client version.
3. Set the XAOPTION item in the SVRGROUP section of the Tmax configuration file.
 - When the DB2 client version is 8.0 or previous

- When the DB2 client engine is 32-bit

```
XAOPTION = "DYNAMIC"
```

- When the DB2 client engine is 64-bit and OS is Unix or Linux

```
XAOPTION = "DYNAMIC XASWITCH32"
```

- When the DB2 client engine is 64-bit and OS is Windows

```
XAOPTION = "DYNAMIC"
```

- When the DB2 client version is 9.0 or later (regardless of the DB2 client engine and OS)

```
Do not set XAOPTION.
```

4. Link appropriate libraries when compiling TMS and the server program.

- When the DB2 client version is 8.0 or previous

- When the DB2 client engine is 32-bit

```
-ldb2s or $(TMAXDIR)/lib/libdb2s.a
```

- When the DB2 client engine is 64-bit and OS is Unix or Linux

```
-ldb2_64s or $(TMAXDIR)/lib64/libdb2_64s.a
```

- When the DB2 client engine is 64-bit and OS is Windows

```
-ldb2s or $(TMAXDIR)/lib/libdb2s.a
```

- When the DB2 client version is 9.0 or later (regardless of the DB2 client engine and OS)

```
-ldb2s_static or $(TMAXDIR)/lib/libdb2s_static.a
```



When the DB2 client version is 9.0 or later, you can link the library used for 8.0 or previous versions for dynamic registration. However, it is not recommended.

Tmax Configuration File

The following is an example.

<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
```


The following items are added.

Item	Description
DBNAME	Database name.
OPENINFO	DB2 database connection information.
TMSNAME	Name of the process that handles transactions that meet OPENINFO.

Database Script

The following creates a DB2 table.

```
#'db2start' execution
$ db2start

# Creating a database named TPTEST
$ db2 "CREATE DATABASE TPTEST"

# Connecting to the TPTEST database
$ db2 "CONNECT TO TPTEST"

# Creating a table named EMP
$ 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)
);

# Checking the table creation
$ db2 "LIST TABLES"
```

The following shows the DB2 table and sample data.

```
$ 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.
```

Client Program

The following is an example.

<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_I *)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("XASERVICE2 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();
}

```

Server program

The following is an example.

<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);
}

```

Server Makefile

The following is an example.

```

# 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

#OBJJS   = $(APOBJS) $(APOBJS2) $(SVCTOBJ)
OBJJS    = $(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): $(OBJS)
    $(CC) $(CFLAGS) $(LDFLAGS) -L$(TMAXLIBDIR) -o $(TARGET) -L$(DB2LIBDIR) $(DB2LIBS) $(OBJS)
$(TMAXLIBS) $(NSDLOBJ)
    mv $(TARGET) $(TMAXAPPPDIR)
    rm -f $(OBJS)

$(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 $(TMAXAPPPDIR)/$(TARGET) $(TARGET).bnd
```

TMS Makefile

The following is an example.

```
# 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) $(APPDIR)/.

$(APOBJ):
    $(CC) $(CFLAGS) -c dummy.c
#
clean:
    -rm -f *.o core $(APPDIR)/$(TARGET)

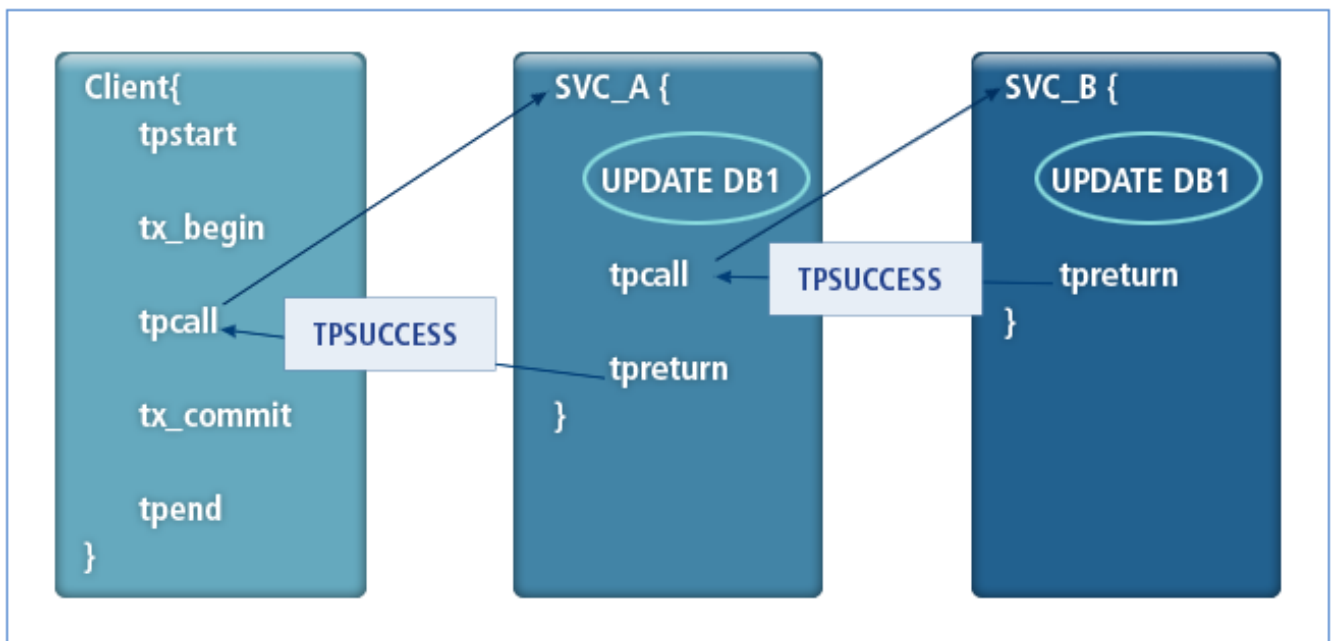
```

12.4. Database Integration Programs

This section describes actual programming that you can use when developing applications. A database can be integrated with homogeneous or heterogeneous databases.

12.4.1. Synchronous Mode (Homogeneous Database)

The following shows a program flow when accessing a homogeneous database in synchronous mode.



Synchronous Mode Flow (Homogeneous Database)

Program Files

- Common program

File	Description
demo.s	SDLFILE.
sample.m	Tmax configuration file.
tmax.env	Configuration file.
mktable.sql	SQL script for creating a database table.

- Client program

File	Description
client.c	Client program.

- Server program

File	Description
update.pc, insert.pc	Server program.
Makefile	Tmax makefile that must be modified.

Program Feature

- Client program

Feature	Description
Tmax connection	Connection with the NULL parameter.
Buffer type	STRUCT.
Subtype	SDL file must be created by using sdlc to compile an input structure file.
Transaction	Transaction is specified by a client.

- Server program

Feature	Description
The number of services	INSERT service is requested from the UPDATE service.
Database connection	Oracle database is used. Database information is specified in the SVRGROUP section of the Tmax configuration file.

Program Environment

Classification	Description
System	SunOS 5.7 32-bit
Database	Oracle 8.0.5

Struct Buffer

The following is an example.

<demo.s>

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

Tmax Configuration File

The following is an example.

<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
```

Configuration File

The following is an example.

<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
```

Database Script

The following creates a database table.

<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  
!
```

Client Program

The following is an example.

<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, 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 = 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();
}

```

Server program

The following example is a server program that performs UPDATE in a database.

<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;
}

```

The following example is a server program that performs INSERT in a database.

<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");

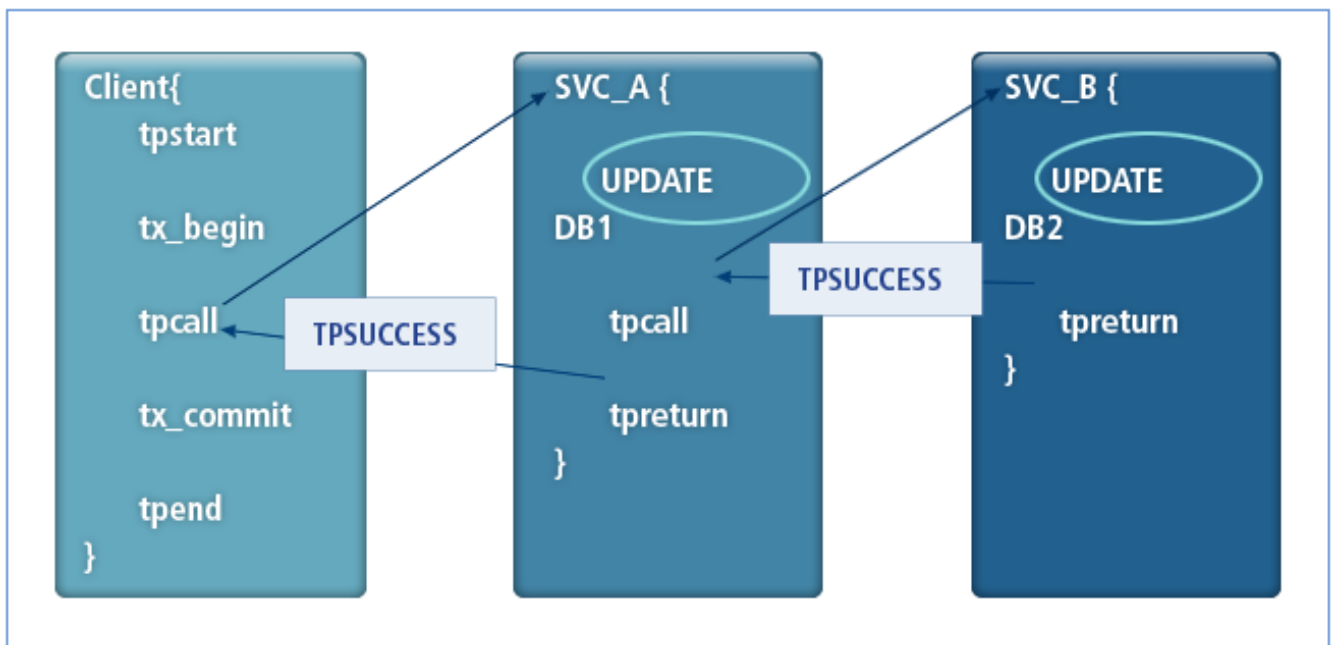
    /* Declare && Open Cursor for Fetch */
    EXEC SQL INSERT INTO ACCOUNT (
        ACCOUNT_ID,
        BRANCH_ID,
        SSN,
        PHONE,
        ADDRESS )
    VALUES (:account_id, :branch_id, :ssn, :phone, :address);

    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. Synchronous Mode (Heterogeneous Database)

The following shows a program flow when accessing a heterogeneous database in synchronous mode.



Synchronous mode Flow (Heterogeneous Database)

Program Files

- Common program

File	Description
demo.s	SDLFILE.
sample.m	Tmax configuration file.
tmax.env	Configuration file.
mktable.sql	SQL script for creating a database table.

- Client program

File	Description
client.c	Client program.

- Server program

File	Description
update.pc, insert.pc	Server program.
Makefile	Tmax makefile that must be modified.



The client and server programs are the same as in [Synchronous Mode \(Homogeneous Database\)](#). For more information about environment settings for multiple nodes, refer to *Tmax Administrator's Guide*.

Program Feature

- Client program

Feature	Description
Tmax connection	Connection with the NULL parameter.
Buffer type	STRUCT.
Subtype	SDL file must be created by using sdlc to compile an input structure file.
Transaction	Transaction is explicitly specified by a client.

- Server program

Feature	Description
The number of services	INSERT service is requested from the UPDATE service.
Database connection	Oracle database is used. Database information is specified in the SVRGROUP section of the Tmax configuration file.

Program Environment

Classification	Description
System	SunOS 5.7 32-bit, SunOS 5.8 32-bit
Database	Oracle 8.0.5

Tmax Configuration File

The following is an example.

<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

```

Configuration File

The following is an example.

<tmax.env>

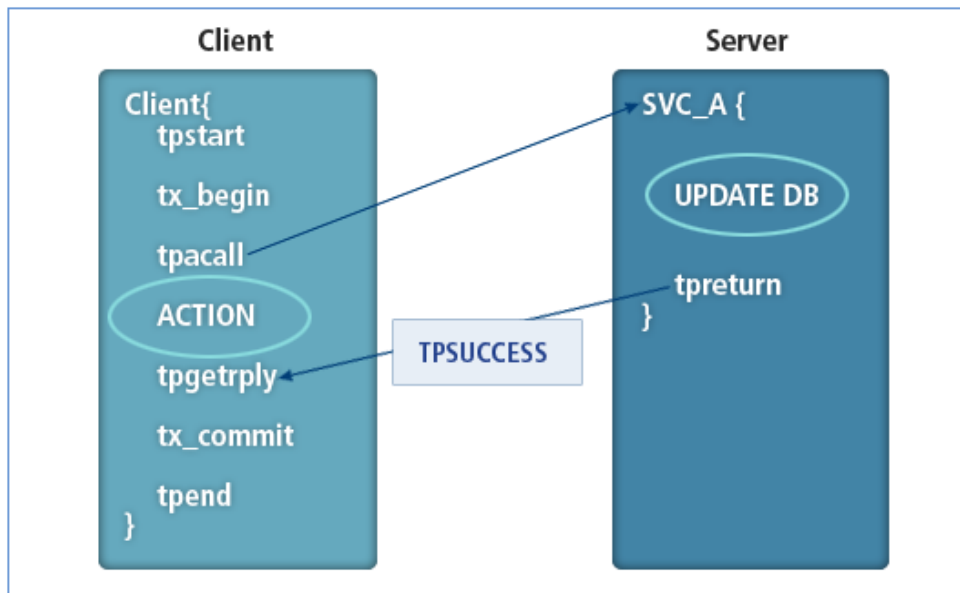
```

[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. Asynchronous Mode (Homogeneous Database)

The following shows a program flow when accessing a homogeneous database in asynchronous mode.



Asynchronous Mode Flow (Homogeneous Database)

Program Files

- Common program

File	Description
demo.s	SDLFILE.
sample.m	Tmax configuration file.
tmax.env	Configuration file.
mktable.sql	SQL script for creating a database table.

- Client program

File	Description
client.c	Client program.

- Server program

File	Description
update.pc	Server program.
Makefile	Tmax makefile that must be modified.

Program Feature

- Client program

Feature	Description
Tmax connection	Connection with the NULL parameter.
Buffer type	STRUCT.
Subtype	SDL file must be created by using sdlc to compile an input structure file.
Transaction	Transaction is specified by a client.

- Server program

Feature	Description
The number of services	INSERT service is requested.
Database connection	Oracle database is used. Database information is specified in the SVRGROUP section of the system configuration file.

Program Environment

Classification	Description
System	SunOS 5.7 32-bit
Database	Oracle 8.0.5

Struct Buffer

The following is an example.

<demo.s>

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

Tmax Configuration File

The following is an example.

<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

```

Configuration File

The following is an example.

<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

```

Database Script

The following creates a database table.

<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

```

Client Program

The following is an example.

<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 = 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);

    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();
}

```

Server program

The following is an example.

<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 *)tpalloc("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");

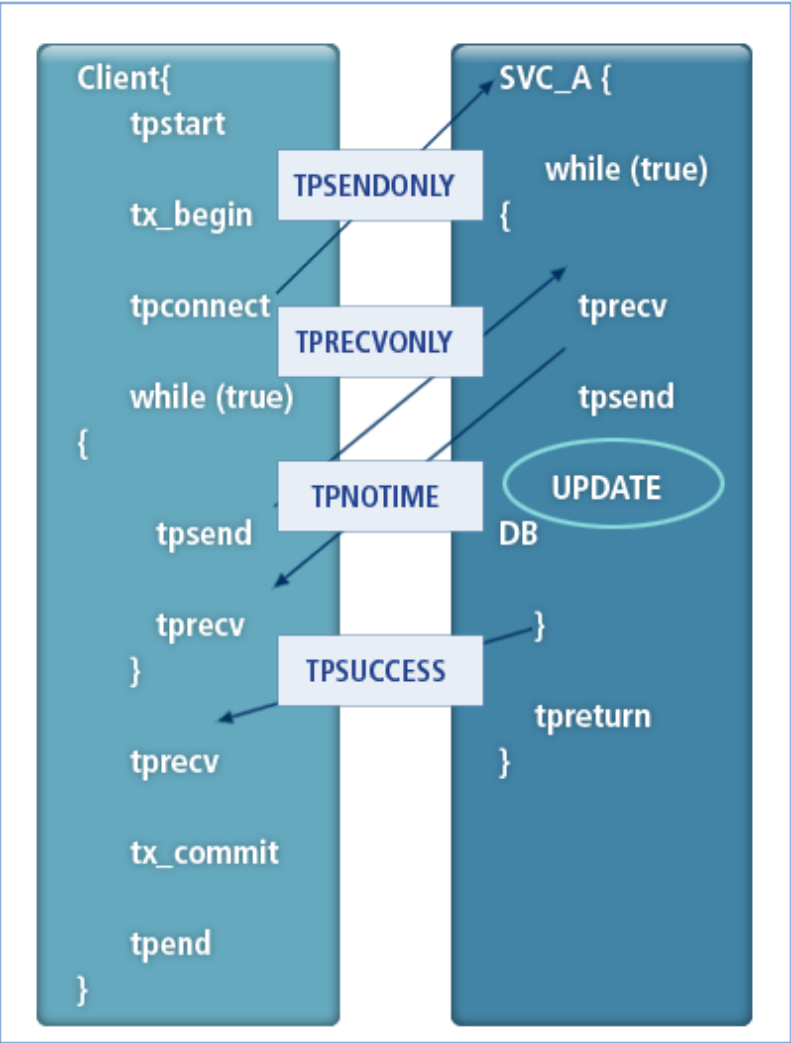
    EXEC SQL UPDATE ACCOUNT
    SET BRANCH_ID = :branch_id,
    PHONE = :phone,
    ADDRESS = :address,
    SSN = :ssn
    WHERE ACCOUNT_ID = :account_id;

    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. Interactive Mode (Homogeneous Database)

The following shows a program flow when accessing a homogeneous database in interactive mode.



Interactive Mode Flow (Homogeneous Database)

Program Files

- Common program

File	Description
demo.s	SDLFILE.
sample.m	Tmax configuration file.
tmax.env	Configuration file.
mktable.sql	SQL script for creating a database table.

- Client program

File	Description
client.c	Client program.

- Server program

File	Description
update.pc	Server program.
Makefile	Tmax makefile that must be modified.

Program Feature

- Client program

Feature	Description
Tmax connection	Connection with the NULL parameter.
Buffer type	STRUCT.
Subtype	SDL file must be created by using sdlc to compile an input structure file. (necessary to run an application)
Transaction	Transaction is specified by a client.

- Server program

Feature	Description
The number of services	UPDATE service is requested.
Database connection	Oracle database is specified. Database information is specified in the SVRGROUP section of the Tmax configuration file.

Program Environment

Classification	Description
System	SunOS 5.7 32-bit
Database	Oracle 8.0.5

Struct Buffer

The following is an example.

<demo.s>

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

Tmax Configuration File

The following is an example.

```
*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
```

Configuration File

The following is an example.

<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
```

Database Script

The following creates a database table.

<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
!
```

Client Program

The following is an example.

<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);
    }

    acntid = 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 = acntid;
sndbuf->branch_id = acntid;
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();
}

```

Server program

The following is an example.

<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. Programs Using TIP

Tmax Information Provider (TIP) is a function process that handles TIPSVC. The following features can be performed using TIP.

- System environment information check: static environment information of a system can be checked.
- System statistical information check: status of each process can be checked while a system is operating.
- System operation management: processes are started or terminated.

12.5.1. TIP Structure

The TIP server has the SYS_SVR server type and is included in the TIP server group. The TIP server receives a request from a client or server, transfers the request to CLH/TMM, and then returns the result to the requester. The TIP server uses field keys to handle the service. The client or server saves data to be requested to a field buffer, sends a request, and then receives the result with the field buffer.

- CHLOG section (log level change)

CHLOG is the section in which the log levels of TMM, CLH, TMS, and SVR are changed. CHLOG performs the same action as **chlog** in tmaxadmin.

When the TIP service is called, TIPSVC is called after the following are set in the field buffer.

Item	Description
TIP_OPERATION (string)	GET.
TIP_SEGMENT (string)	ADMINISTRATION.
TIP_SECTION (string)	CHLOG.
TIP_MODULE (int)	Module to dynamically change log. Options are: <ul style="list-style-type: none">• TIP_TMM• TIP_CLH• TIP_TMS• TIP_SVR
TIP_FLAGS (int)	Flags. Options are: <ul style="list-style-type: none">• TIP_VFLAG• TIP_GFLAG• TIP_NFLAG
TIP_SVRNAME (string)	Server name. Set only when TIP_FLAGS is TIP_VFLAG.

Item	Description
TIP_SVGNAME (string)	Server group name. Set only when TIP_FLAGS is TIP_GFLAG.
TIP_NODENAME (string)	Node name. Set only when TIP_FLAGS is TIP_NFLAG.
TIP_LOGLVL (string)	Log level. Value must be in lowercase letters. The result value is set in TIP_ERROR. Options are: <ul style="list-style-type: none"> • compact • basic • detail • debug1 • debug2 • debug3 • debug4
TIP_ERROR (int)	Error value. Options are: <ul style="list-style-type: none"> • TINESVCFAIL: corresponding service was not handled successfully • TIPEOS: memory allocation failed • TIPEBADFLD: value of TIP_MODULE is not set

- CHTRC section

TMAX_TRACE of TMS and SPR are specified to modify the trace log options in the CHTRC section. CHTRC performs the same action as **chtrc** in tadmin.

When the TIP service is called, TIPSVC is called after setting the following in a field buffer.

Item	Description
TIP_OPERATION (string)	GET.
TIP_SEGMENT (string)	ADMINISTRATION.
TIP_SECTION (string)	CHTRC.
TIP_FLAGS (int)	Flags. Options are: <ul style="list-style-type: none"> • TIP_PFLAG • TIP_VFLAG • TIP_GFLAG • TIP_NFLAG
TIP_SPRI (int)	Sets spri. Set only when TIP_FLAGS is TIP_PFLAG.
TIP_SVGNAME (string)	Server group name. Set only when TIP_FLAGS is TIP_GFLAG.
TIP_NODENAME (string)	Node name. Set only when TIP_FLAGS is TIP_NFLAG.

Item	Description
TIP_SPEC (string)	Filter spec, receiver spec, and trigger spec. The result value is set in TIP_ERROR.
TIP_ERROR (int)	Error value. Options are: <ul style="list-style-type: none"> • TIPSVCFAIL: corresponding service was not successfully handled • TIPEOS: memory allocation is failed • TIPEBADFLD: value of TIP_MODULE is not set

The following must be included in requests to TIP SVC.

- Operation (TIP_OPERATION)

Set Value	Description
GET	To check statistical information and static environment information of a system or to operate and manage a system (BOOT/DOWN).
SET	To change system settings. Currently, only GET is supported.

- Segment (TIP_SEGMENT)

Used to determine which function to execute. The following can be set in the TIP_SEGMENT field.

Set Value	Description
CONFIGURATION	Checks static configuration information of a system.
STATISTICS	Checks statistical information while a system is operating.
ADMINISTRATION	Checks system operation and management (BOOT/DOWN).

- Section (TIP_SECTION)

When TIP_SECTION is set, the following values can be set.

Set Value	Description
CONFIGURATION	DOMAIN, NODE, SVRGROUP, SERVER, SERVICE, ROUTING, RQ, and GATEWAY
STATISTICS	NODE, TPROC, SPR, SERVICE, RQ, TMGW, NTMGW, TMS, TMMS, CLHS, and SERVER (SVR)
ADMINISTRATION	BOOT, DOWN, CHLOG, and CHTRC

- Command (TIP_CMD)

Used only when TIP_SECTION is ADMINISTRATION.

Set Value	Description
TIP_BOOT	Starts the Tmax system.
TIP_DOWN	Terminates the Tmax system.

12.5.2. TIP Usage

The following describes how to use TIP and check an error.

TIP Usage

- Environment setting

A user does not need to write a service because the TIP server is a function process provided by Tmax. However, a user must register the TIP server in a configuration file. The following example is setting an environment.

```
*DOMAIN
res          ..., TIPSVC = TIPSVC

*NODE
tmaxs1      ...

*SVRGROUP
tsvg        ..., SVGTYPE = TIP

*SERVER
TIP         SVGNAME = tsvg, SVRTYPE = SYS_SVR
```

- TIPSVC is registered in the DOMAIN section. If unregistered, TIPSVC is registered by default.
- A TIP server group (SVGTYPE=TIP) is registered in the SVRGROUP section.
- A TIP server (SVRTYPE=SYS_SVR) is registered in the SERVER section.
- System access

A user must set .tpadmin in the username property of the TPSTART_T structure when accessing the Tmax system. username is set only when TIP_SEGMENT is ADMINISTRATION. If username is set incorrectly, the TIPEAUTH error is set in the TIP_ERROR field.

```
strcpy(tpinfo->username, ".tpadmin");
```

- Buffer allocation

A client or a server program must allocate a field key buffer for a request because the TIP server uses field keys to handle a service.

- TIP request items

Item	Description
TIP_OPERATION	Changes and checks system environment information.
TIP_SEGMENT	Checks information for system operation and management.
TIP_SECTION	Detailed property settings in SEGMENT.
TIP_CMD	Set only when TIP_SEGMENT is ADMINISTRATION.
TIP_NODENAME	Set only for multiple nodes. If only a single node exists, TIP_NODENAME is set to a local node by default. If it is incorrectly set, the TPEINVAL error occurs.

- TIPSVC request

After the required properties for TIP are set, set the configured field buffer to sndbuf and use tpcall or tpacall to send the TIPSVC request. Both client and server can request a service, and transactions are not supported.

- Result reception

The service result is saved in a reception field key buffer.

Error Check

- If successfully handled

The TIP_ERROR property of a reception field key buffer is set to 0.

- If an error occurs

Error	Description
TIP_STATUS	Detailed error information set in TIP_ERROR can be checked.
TIP_BADFIELD	Field that caused the error can be checked.
TIP_ERROR	Value greater than 0 is set in TIP_ERROR of a reception field key buffer. It can be checked in /usrinc/tip.h.

The following are the error values that can be set in TIP_ERROR.

Set Value	Description
TIPNOERROR	Error did not occur.
TIPEBADFLD	Invalid field key issued. In general, TIPEBADFLD is set when a field key not compiled by the fdlc utility.
TIPEIMPL	Unavailable feature is requested.
TIPEAUTH	Service not allowed under current privileges.
TIPEOS	OS or system error caused by a memory allocation failure, connection to Tmax system failed, or unstable network status.

Set Value	Description
TIPENOENT	Accessed nonexistent property.
TIPESVCFAIL	tpreturn() is called to TPFail due to a TIP service routine error.

12.5.3. TIP Usage Example

The following examples use TIP.

Configuration File

The following example uses a single node.

<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
```

The following example uses multiple nodes.

<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

```

Field Key Table

The following is an example.

< 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_TIPSVC      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_CLCHKINT	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_EXTCCLHPORT  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_PRE SVC    2      string
TIP_QSIZE      3      int
TIP_FILEPATH   4      string
TIP_BOOT       5      string
TIP_FSYNC      6      int
TIP_BUFFERING  7      int
#TIP_ENQ SVC   8      int
#TIP_FAILINTERVAL 9    int
#TIP_FAILRETRY 10     int
#TIP_FAIL SVC  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_TIME       2      string
TIP_TTIME      3      int
TIP_SVC_TIME   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. Program for Checking System Environment Information

The following example is a client program that checks system environment information.

```

#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);
}

#if 1
    fbprint(rcvbuf);
#endif

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

```

[Result]

The following is the result (Domain Conf) of the previous program.

```

$ 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. Program for Checking System Statistical Information

The following example is a program that checks system statistics.

```

#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_SERVER) {
    printf("out of section [%d - %d]\n", SEC_NODE, SEC_SERVER);
    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();
}

```

[Result]

The following is the result (TMS STATISTICS) of the previous program.

```

$ 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 = xa1
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. Program for Starting and Terminating a Server Process

Example 1

The following program starts and terminates a server process.

< 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();

```

Example 2

The following program changes the log level of a server named 'vr23_stat_ins' to debug4.

```

#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, forkent;

    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->username, ".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;
}

```

[Result] (TIP_SVR, => DEBUG4)

```
$ client xa1 svr23_stat_ins $HOSTNAME 8 1024 7
fkey = 217326601, fname = TIP_ERROR, type = int, value = 0
>>> tadmin (cfg -v)

loglvl = DEBUG4
```

12.6. Local Recursive Calls

When `tpcall()` is executed in a server, the recursive service call feature is added. Only `tpcall()` can make recursive calls through the multicontexting technique in a server. The recursion depth is limited to 8 to prevent an infinite loop.



To use a local recursive call, `-D_MCONTEXT` must be added to `CFLAGS` when a server program is compiled and the `libsvrmc.so` server library must be used instead of `libsvr.so`.

Server program

The following is an example.

```
#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);
        tpreturn(TPFAIL, 0, 0, 0, 0);
    }

    strcat(rcvbuf, "_Success");

    tpreturn(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);
        tpretreturn(TPFAIL, 0, 0, 0, 0);
    }
    strcat(rcvbuf, "_Success");
    tpretreturn(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);
        tpretreturn(TPFAIL, 0, 0, 0, 0);
    }

    strcat(rcvbuf, "_Success");
    tpretreturn(TPSUCCESS,0,(char *)rcvbuf, 0,0);
}

```

Makefile

The following is an example.

<Makefile.c.mc>

```

# Server makefile

TARGET = $(COMP_TARGET)
APOBJS = $(TARGET).o
NSDLOBJ = $(TMAXDIR)/lib64/sdl.o

LIBS = -lsvrmc -lnodb
OBJJS = $(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)

```

Appendix A: Configuring Tmax

This appendix describes the Tmax configuration file.

A.1. Configuration File

The Tmax configuration file enables you to configure Tmax. It is created by a Tmax administrator. This file is used to create a service table and to start Tmax.

The configuration file includes the following sections.

Section	Description	Required
DOMAIN	Configures the entire environment for a single independent Tmax system.	Required
NODE	Configures the environment for each node that comprises the DOMAIN section.	Required
SVRGROUP	Configures server groups and databases.	Required
SERVER	Configures servers.	Required
SERVICE	Configures services.	Required
GATEWAY	Configures gateways between domains.	Optional
ROUTING	Configures data-dependent routings.	Optional
RQ	Configures reliable queues.	Optional

The name of each section starts with an asterisk (*) (for example, *DOMAIN, *NODE) and must start from the first space of a line. In a section, the definition of each item is separated by a comma (,), which indicates that the definition of items continues.



For more information about the Tmax configuration file, refer to *Tmax Administrator's Guide*.

The configuration file is created as a regular text file and compiled using the `cfl` command. The binary file, created after compilation, is referenced to create a service table and to start Tmax.

```
cfl -i Tmax_Configuration_File_Name
```

The following is an example of a Tmax configuration file (items inside angle brackets (< >) can be changed).

```
*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 Section

Configures the Tmax system.

```

*DOMAIN
<resrc_name>  SHMKEY = <UNIQUE IPCKEY>,
              MAXUSER = 256

```

Item	Description
<resrc_name>	Unique name for the Tmax system (Domain). The name must be 63 characters or less.
SHMKEY	Value to internally allocate shared memory. The value must be unique to the entire system and fall within the range of 32,768 to 262,143.
MAXUSER	Maximum number of users that are allowed to concurrently connect to Domain.

A.1.2. NODE Section

Configures the environment for each node that comprises DOMAIN.

```

*NODE
<uname>      TMAXDIR = "<TMAX installed directory>"
              APPDIR = "<APPLICATION DIRECTORY>",
              PATHDIR = "<path directory>"

```

Item	Description
<uname>	Name up to 63 characters in length. It uses the value returned by the uname -n command. The return values of hostname and "uname -n" must be the same.
TMAXDIR	Full path of the Tmax installation directory.
APPDIR	Full path of the Tmax application directory.
PATHDIR	Full path of the directory for internal communication between Tmax processes.

A.1.3. SVRGROUP Section

Configures a group of servers logically related on another. A server group is the basic unit for database utilization, load balancing, and fault tolerance. At least one server group is defined for each server.

```
*SVRGROUP
<svg_name>      NODENAME = <uname> ,
                  OPENINFO = "Oracle_XA+Acc=P/scott/tiger+SesTm=60",
                  DBNAME = ORACLE,
                  TMSNAME = tms_ora
```

Item	Description
<svg_name>	Server group name up to 63 characters in length. It must be unique in the SVRGROUP section.
NODENAME	Name of the node where a server group exists. Since a server group requires specification of a database, the following three items must be specified when a database is used: DBNAME, OPENINFO, and TMSNAME.
DBNAME	Database name (for example, ORACLE).
OPENINFO	Registers information in Tmax to connect to a database. The information can be up to 256 characters in length and the registration varies depending on the database type. <ul style="list-style-type: none"> • Oracle <div>OPENINFO = "Oracle_XA+Acc = P/account/password+SesTm = 60"</div> • Informix <div>OPENINFO = "database name"</div>

Item	Description
CLOSEINFO	<p>Registers information to disconnect the server group from a database. The information can be up to 256 characters in length. The registration process varies depending on the database type.</p> <ul style="list-style-type: none"> • Oracle <div>CLOSEINFO = ""</div> • Informix <div>CLOSEINFO = ""</div>
TMSNAME	TMS process name. The name can be up to 63 characters in length.

A.1.4. SERVER Section

Configures the application servers.

```
*SERVER
<svr_name>          SVGNAME = <sgrpname> , MIN = 1, MAX = 3
```

Item	Description
<svr_name>	Name of the executable file of the server that provides services. The name can be up to 63 characters in length.
SVGNAME	Server group name to which the server process belongs. SVGNAME must be defined in *SVRGROUP.
CLOPT	Parameter that processes the options of a command line. A hyphen (—) is used as a delimiter to separate system options from user options. This line is processed by the tpsvrinit() function.
MIN	Number of servers that start by default when a user wants to start multiple servers.
MAX	Maximum number of servers that can be dynamically started in addition to the default number.
CONV	Indicates whether interactive servers are participating in interactive communication (Default value: N).

A.1.5. SERVICE Section

Configures services.

```
*SERVICE
<svc_name>          SVRNAME = <svrname>
```

Item	Description
<svc_name>	Name of the function called by a client when specified with a name. The name can be up to 63 characters in length and must be unique. Only services registered in the SERVICE section are supported.
SVRNAME	Name of a server that provides specific services.

A.2. Makefile

This section describes TMS Makefile for Oracle and Informix.

TMS Makefile for Oracle

The following is an example of a TMS Makefile (Solaris 32-bit) for Oracle.

```
#
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)
```

TMS Makefile for Informix

The following is an example of a TMS Makefile (Solaris 32-bit) for Informix.

```

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)

```