# Getting Started Guide

Tmax 6

TMAXSOFT

## Copyright



## Restricted Rights Legend



## Trademarks



## 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

# Glossary

**Two-phase commit (2PC) Protocol**

A two step protocol used to guarantee transaction properties for global transactions related to more than one homogeneous or heterogeneous database. The first step is the prepare phase. The second step is the commit phase.

**Atomicity**

All or nothing. All work in a transaction is performed, or nothing is performed.

**CARRAY and X_OCTET Buffers**

A buffer used to save binary type data that has a specified number of bytes. The length of a buffer must be specified to exchange data.

**Client Handler (CLH)**

A process that mediates between clients and servers, requests a service from a server that handles businesses, connects to a server, and manages the connection.

**Client Listener (CLL)**

A process for connections between clients and Tmax. CLL receives requests from clients by setting the PORT Listener to manage client connections.

**Commit**

As a part of handling transactions, makes tentative changes permanent.

**Consistency**

The successful result of a transaction is updated to shared resources. If the transaction fails, shared resources are kept in their original states.

**Database Access System**

A service that enables multiple database servers to be used with a single consistent method in a distributed environment.

**Data Dependent Routing (DDR)**

A method that distributes loads with data values. If multiple nodes provide the same service, routing is possible for the nodes within the data range.

**Dynamic Load Management (DLM)**

A method that dynamically selects a handling group according to the load ratio. If loads are concentrated at a certain node, Tmax distributes the loads with this method, which dynamically adjusts the loads for each node.

**Distributed Transaction Processing (DTP)**

Multiple Resource Managers (RMs) handle a single transaction.

**Domain Socket**

A method that uses the socket API without any changes and uses a file to enable communication

between processes.

**Downsizing**

The process of changing a centralized mainframe environment to an open distributed system environment.

**Durability**

The result of a transaction is always maintained after it is committed.

**FIELD Buffer**

A buffer used to save field key and data value pairs. All native types can be saved in this buffer.

**Global Transaction**

A complete unit of work that takes place in an environment managed by more than one resource manager.

**Gateway Process (GW)**

Handles inter-domain communication when multiple domains exist.

**Hybrid Messaging System (HMS)**

A Tmax feature that is the communication medium for loosely coupled senders and receivers. It supports the Queue and Topic methods.

**Isolation**

Changes in shared resources affected by a transaction do not influence other transactions before the transaction is committed.

**Middleware**

The system software that provides a single user environment in a distributed computing environment. It connects a network of heterogeneous systems, supports communication between clients and servers, and connects between computers.

**Messaging Oriented Middleware (MOM)**

A service that handles messages by putting them into a queue and provides an asynchronous message management feature.

**Object Request Broker (ORB)**

A service that provides the feature that enables a client object to call a method of a remote server using a software bus called ORB.

**Processing On Demand (POD)**

A server process that starts and handles business logic only when there is a client request.

**Remote Access Control Daemon (RACD)**

Remotely controls all domains in which Tmax is installed.

**Raw Client Agent (RCA)**

An agent that supports multiple ports that efficiently handle processes with the multi-threading

method.

**Rollback**

An operation that recovers the result of a transaction to a previous state due to a transaction failure or a user request.

**RPC System**

A service that synchronously runs a program located in another computer through a network.

**Reliable Queue (RQ)**

Enables data to be maintained and reliably handled by preventing a request from disappearing due to a failure.

**Reliable Queue Server (RQS)**

As a process that manages the disk queue of the Tmax system, it reads/writes from/to a file.

**Simple Client Agent (SCA)**

An agent that supports multiple ports that can handle both non-Tmax clients and Tmax clients.

**System Load Management (SLM)**

A method that distributes loads by using a defined load ratio.

**STRING Buffer**

A buffer used to save a string that ends with NULL. The length of the buffer does not need to be specified.

**STRUCT and X_C_TYPE Buffers**

A buffer used to save a C language struct.

**Tmax Control Server (TCS)**

A server process that handles business logic passively at the request of CLH and returns the results.

**Tmax Information MAP (TIM)**

Core information required to operate the Tmax system. It is created by the TMM service and located in the shared memory managed by Tmax.

**Tmax Information Provider (TIP)**

Checks system environment information and statistics information, and operates and manages the system.

**Transaction Log Manager (TLM)**

Saves transaction logs in tlog before CLH executes commit when a transaction occurs.

**Tmax Administrator (Tmadmin)**

Monitors Tmax-related information and manages changes in the configuration file.

**Tmax Manager (TMM)**

A core process that operates and manages the Tmax system. It manages all shared information of the Tmax system and the following server processes: Client Listener (CLL), Client Handler (CLH), Transaction Management Server (TMS), and Application Program (AP).

**Transaction Management Server (TMS)**

A process that changes databases and handles transactions while operating in a database-related system. It delivers commit/rollback requests from XA services to Resource Manager (RM).

**Transaction Processing Monitor (TP-Monitor)**

Transaction management middleware, which monitors transactions and maintains their consistency.

**Transaction**

A complete unit of work. A single transaction includes multiple tasks.

**User Control Server (UCS)**

A server process that actively handles business logic without a caller request and returns the results. It is a unique feature of Tmax.

**Web Application Server (WAS)**

A service that handles transactions in the web and provides the mutual communication (J2EE) feature between heterogeneous systems.

**Web Transaction (WebT)**

A program that supports the transaction service for Tmax and Java application programs.

**X_COMMON Buffer**

A buffer used to save a C language struct. Only the char, int, and long types can be used as members of the struct.

# 1. Introduction to TP-Monitor

To facilitate a better understanding of Tmax, this chapter describes the concepts and features of middleware and TP-Monitor.

## 1.1. Overview

TP-Monitor (Transaction Processing Monitor) is the transaction management middleware that monitors transactions and maintains their consistency. A transaction is the smallest unit of handling among sessions, systems, and databases that operate via various protocols.

Since Tmax is a product based on TP-Monitor, this chapter first describes middleware and TP-Monitor.

## 1.2. Middleware

Because the centralized mainframe environment has various issues, such as cost and operation, it became necessary to downsize the environment to an open distributed system environment that separates hosts by application.

However, the open distributed environment has issues involving the compatibility and integration between server programs. Additionally, communication between different operating systems has to be managed in multiple distributed systems because applications are handled in a single mainframe. As a result, multiple systems, server programs, and network resources must be used in a single user environment.

The following tables illustrate the issues of the two environments.

- The centralized mainframe environment

| Classification | Issue |
|---|---|
| Cost | • High adoption cost<br>• High maintenance cost |
| Operation | • Does not consider a user environment because the mainframe environment takes precedence over the application process |
| System | • Difficult to communicate between heterogeneous systems<br>• Difficult to migrate a program to another system<br>• Difficult business expansion |

- The open distributed system environment

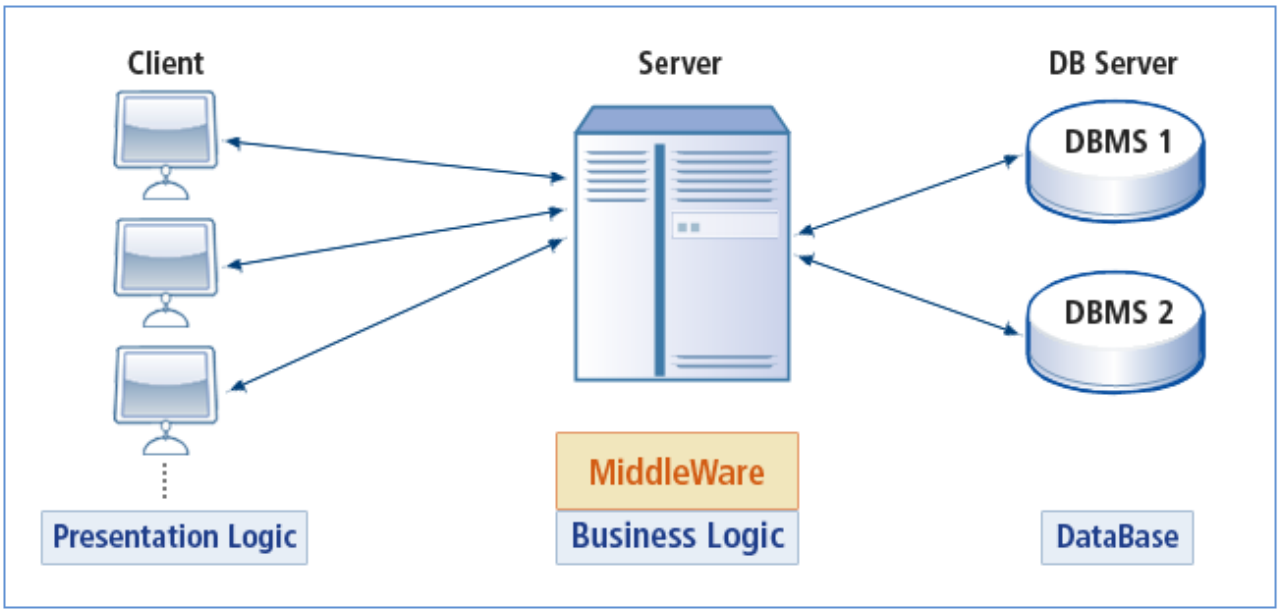
| Classification | Issue |
|---|---|
| Cost | • Requires professional technical skills for network, DBMS, etc. |
| Operation | • Inefficient system operation<br>• Does not consider a user environment<br>• Issues due to the distributed environment<br>• Difficult to manage and monitor systems<br>• Difficult to handle a failure<br>• Different operating methods for multiple servers<br>• Issues due to multiple providers<br>• Performance suffers as the number of users increases<br>• Load differences between servers |
| System | • Difficult to manage processes<br>• Requires multiple communication methods<br>• System security vulnerability<br>• Difficult to handle heterogeneous databases and global transactions<br>• Difficult to migrate a program between heterogeneous servers<br>• Development issues (various OS and development languages)<br>• Provides transaction handling and process management features in the distributed environment with only a single middleware |

Middleware has been developed to solve the open distributed environment issues.

Middleware is system software that provides a single user environment in a distributed computing environment. It connects a network of heterogeneous systems and supports communication between clients and servers, and connections between computers. Middleware enables stable communication in applications and in the operating environment by integrating heterogeneous hardware, protocols, and communication environments.

Clients and database servers do not need to directly communicate with each other because middleware engages in communication between the two systems. Since middleware has the business logic necessary for applications, clients and server programs only need to communicate with middleware. Clients and database servers can be configured into a single system environment through middleware. Middleware guarantees system integration for heterogeneous machines in a multiple database environment, and provides data compatibility and consistency. This environment provides maximum performance for minimal resources.

The following figure shows the client/server environment with middleware.




Middleware Workflow

Middleware products can be divided into six types according to usage and purpose:

- Transaction Processing Monitor (TP-Monitor)

  Handles transactions in a heterogeneous distributed environment and manages various handling processes. Tmax belongs to this TP-Monitor product type.

- Web Application Server (WAS)

  Handles transactions on the web and provides the mutual communication feature (J2EE) to heterogeneous systems.

- Messaging Oriented Middleware (MOM)

  Handles messages by putting them into a queue and provides an asynchronous message management feature.

- Database Access System

  Enables multiple database servers to be used with a single consistent method in a distributed environment.

- RPC System

  Through the network, synchronously runs a program located in another computer.

- Object Request Broker (ORB)

  Provides a feature that enables a client object to call a remote server method using a software bus called ORB.

# 1.3. TP-Monitor

Most application systems are used in a centralized environment based on mainframes. The open distributed system was developed because of the various issues of the centralized environment such as cost and management.

The open distributed system, however, has issues of its own such as system operation and management. Middleware was adopted to resolve these issues. TP-Monitor is middleware that monitors transactions and maintains their consistency. A transaction is the smallest unit of handling among sessions, systems, and databases that operate via various protocols.

Major features of TP-Monitor are as follows:

- Convenient application development

  A complex application process can be developed by focusing on features, not data. It is very difficult to develop an application in the mainframe environment because business logic and data handling logic are developed together. However, if TP-Monitor is used, all middleware, client programs, and database server programs only need business logic, modules provided to users, and the data management feature, respectively. This modularization helps ease application development.

- Efficient application management

  Efficiently manages each application by using TP-Monitor to manage distributed business systems.

- Heterogeneous DBMS resource management

  Manages heterogeneous DBMS resources by integrating and managing DBMS transactions.

- Load balancing

  Distributes loads and supports distributed transactions for optimized resource usage.

- High performance and reliability

  Reduces overhead and response time by managing a large number of clients with limited resources.

# 2. Introduction to Tmax

This chapter describes the Tmax concepts, its architecture, features, characteristics, and issues.

## 2.1. Overview

Tmax stands for Transaction Maximization, which means the maximization of the transaction handling ability. Tmax is a TP-Monitor product that handles transactions (for heterogeneous systems in a distributed environment), distributes loads, and takes a proper action when an error occurs.

Tmax provides an efficient development environment with optimized solutions used in the client/server environment. It also improves performance and handles all failures.

Tmax complies with the X/Open DTP (Distributed Transaction Processing) model, the international standard for distributed transaction processing. Tmax was developed to meet the API, services, and X/Open model transaction model components standards created by the international standard organization OSI (Open Systems Interconnection group). Tmax transparently handles the applications of heterogeneous systems in a distributed environment. It also supports OLTP (On-Line Transaction Processing) and meets ACID (Atomic, Consistent, Isolated, Durable: Transaction Properties) for transaction processing.

Tmax dramatically enhances performance by transparently handling applications. Tmax also provides an efficient development environment for creating new applications by facilitating the processing of mission-critical legacy applications. In all industries, Tmax guarantees system reliability by controlling loads and preventing failure in critical systems in which large numbers of transactions are handled. It can be utilized to develop large-scale OLTP applications and used in various industries (for example, airlines, hotels, hospitals, and banks) and businesses (for example, online jobs, credit card approval, and customer and sales management).

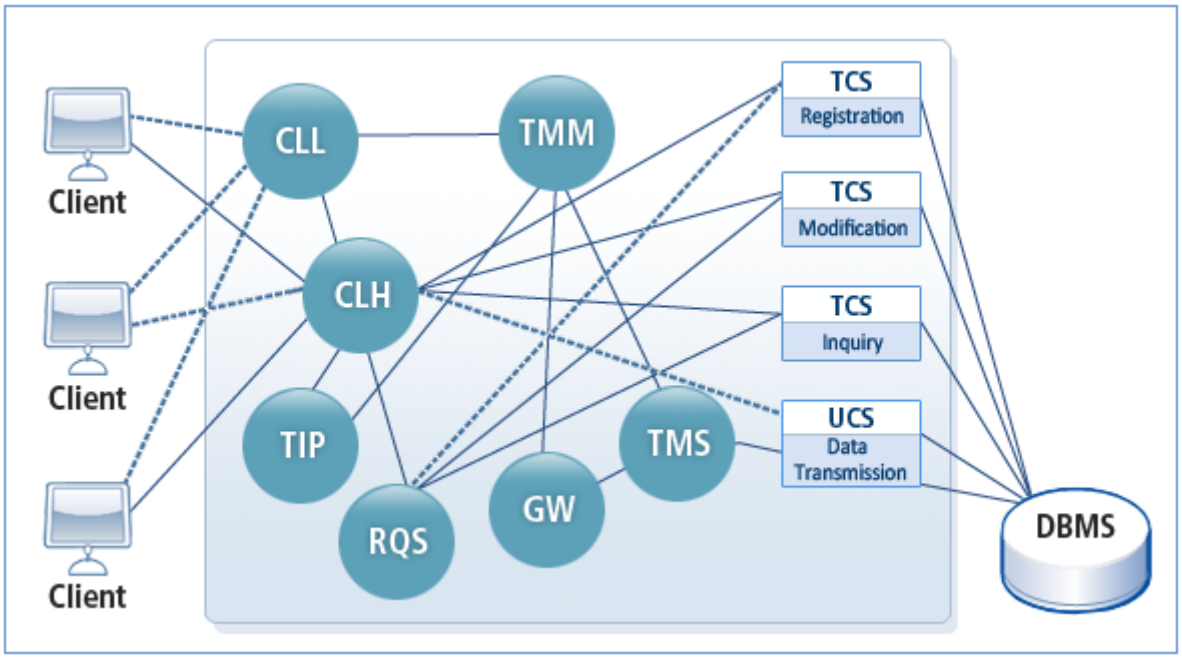
For more information about transactions, refer to "Transactions" in *Tmax Application Development Guide*.

## 2.2. Architecture of Tmax

This section describes the system configuration and features.

### 2.2.1. System Configuration

The following figure shows the system configuration of Tmax.




Tmax Architecture

- Tmax Manager (TMM)

  A core process that operates and manages the Tmax system. TMM manages all shared information in the Tmax system and the following server processes: Client Listener (CLL), Client Handler (CLH), Transaction Management Server (TMS), and Application Program (AP).

  The following are the major features of TMM:

| Major Feature | Description |
|---|---|
| Shared memory allocation | When configured environment information is compiled with the **cfl** command, a binary file is created. TMM loads the binary file into shared memory and manages the Tmax system using the loaded information. |
| Process management | TMM is the main process for the operation and termination of all systems. |
| Log management | TMM manages Tmax system logs (slog) and user logs (ulog). |

- Client Listener (CLL)

  A process for the connection between clients and Tmax. It receives requests from clients by setting PORT Listener for managing client connections.

- Client Handler (CLH)

  Mediates between clients and servers, requests a service from a server that handles tasks, connects to a server, and manages the connection.

  Delivers requests from clients through a function (for example, tpcall) to a proper server, and transmits requests for XID numbering and commit/rollback in the XA service environment.

- Transaction Management Server (TMS)

  Manages databases and handles transactions while operating in a database-related system. Delivers commit/rollback requests of XA services to the Resource Manager (RM).

- Transaction Log Manager (TLM)

  Saves transaction logs in tlog before CLH commits when a transaction occurs.

- Reliable Queue Server (RQS)

  Manages the disk queue of the Tmax system and executes reads/writes from/to a file.

- Gateway Process (GW)

  Handles inter-domain communication in the environment in which multiple domains exist.

- Tmax Administrator (Tmadmin)

  Monitors Tmax-related information and manages changes in the configuration file.

- Remote Access Control Daemon (RACD)

  Remotely controls all domains in which Tmax is installed.

- Tmax Control Server (TCS)

  Handles business logic at the request of CLH and returns the results.

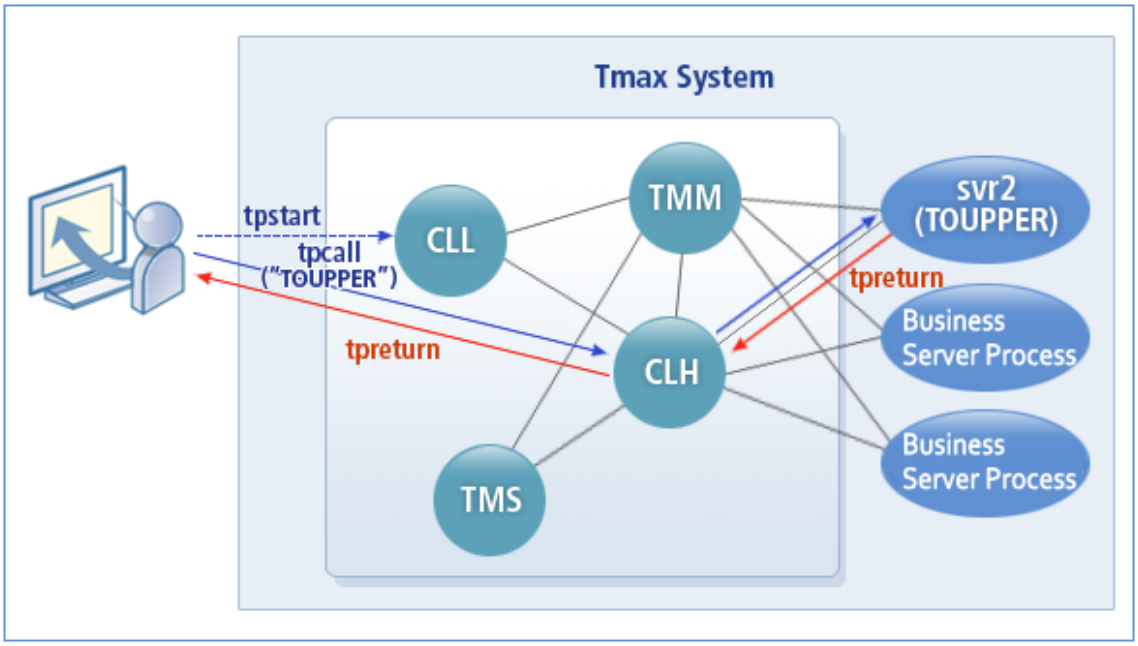
- User Control Server (UCS)

  Handles business logic at the request of CLH and returns the results. A corresponding process maintains control.

- Tmax Information Provider (TIP)

  Checks system environment and statistics information, and operates and manages the system. (boot/down only)

The following figure shows the services performed by the Tmax system.




Services of the Tmax System

The following describes how Tmax system performs a service:

1. When a client executes tpstart, the CLL process handles the connection request to connect to the CLH process.
2. If there is a service request, CLH handles all services.
3. CLH receives a client service request (tpcall), analyzes the service, and maps the service to the proper business server process (svr2).
4. The business server process (svr2) handles the service and then returns the result (tpreturn) to CLH.
5. CLH receives the result and sends it to the client.

## 2.2.2. TIM

Tmax Information MAP (TIM) is core information required to operate the Tmax system. TIM is created by the TMM process, and located in the shared memory managed by Tmax.

TIM can be divided into the following according to the role:

- Information for setting the Tmax system

  Loaded to shared memory and referred to if necessary to manage <tmconfig.m>, the Tmax configuration file.

- Information for operating the Tmax system

  Manages information to operate the Tmax system. The information includes the following: how to respond to a failure that occurs in the system, base information for load balancing, naming service information to access each service if a system consists of multiple pieces of equipment,

and application location.

- Information for application status

  Manages the status (Ready, Not Ready, Running, etc.) of application processes loaded in a system.

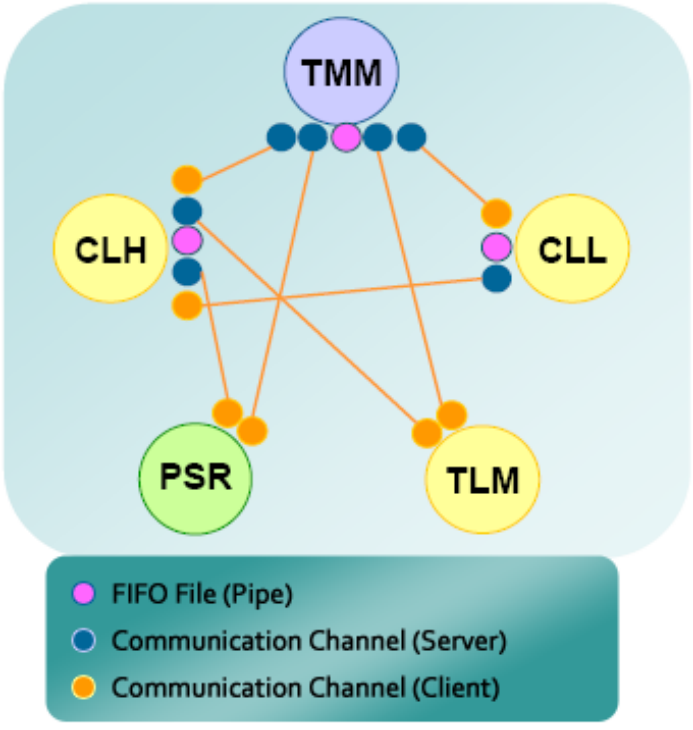
- Information for distributed transactions

  Manages information for operating a database that mediates communication with RM, and manages numbering information for transaction processes.

### 2.2.3. Socket Communication

Tmax uses the UNIX domain socket communication method. This method uses the socket API without any changes, and enables communication between processes by using a file. It is more stable and faster than an external communication method based on ports. Both this method and FIFO (Named Pipe) use a file and manage messages from kernels. However, the socket communication method has a two-way communication feature and unlike FIFO, it is easy to build multiple client/server environments.

The following figure shows the process of UNIX domain socket communication of CLL, CLH, and TLM. The application server (PSR) connects to CLL, CLH, and TLM.




Domain Socket Communication

## 2.3. Features of Tmax

Tmax has the following features:

- Process management

  To provide an optimized environment, Tmax adjusts the number of the server's application handling processes created for each client. Tmax provides a 3-tier based client/server environment.

- Transaction management

  Tmax guarantees data integrity by supporting Two-Phase Commit for distributed transactions. Tmax also facilitates the use of global transactions by providing simple functions such as tx_begin, tx_commit and tx_rollback. It also improves efficiency with the transaction manager that uses multi threads, and guarantees stability with recovery/rollback by rapidly responding to errors with dynamic logging. Transactions can be easily scheduled and managed because all transactions are centrally managed.

- Load balancing

  Tmax provides the load balancing feature with the following 3 methods to increase throughput and reduce handling time.

  - System Load Management (SLM)
  - Data Dependent Routing (DDR)
  - Dynamic Load Management (DLM)

- Failure handling

  Tmax can operate normally by using failover through load balancing and service backup even when a hardware failure occurs. Even if a software failure occurs due to a down server process, services are provided continuously.

- Naming service

  Naming service provides service location information in distributed systems by providing transparency and a name for easy service calls.

- Process control

  Tmax provides three methods of data transmission processes. For more information, refer to Process Control.

  - Tmax Control Server (TCS)
  - User Control Server (UCS)
  - Processing On Demand (POD)

- Reliable Queue (RQ) feature

  Data is maintained and reliably handled with a disk queue that prevents requests from disappearing due to a failure.

- Security feature

Tmax provides a data protection feature based on the Diffie-Hellman algorithm, supports the following 3-step security feature, and includes the UNIX security feature.

  - Step 1: System connection authentication

    A unique password is set for the entire Tmax system (domain). Only clients who registered the password can connect to the Tmax system.

  - Step 2: User authentication

    Tmax services can be used with user IDs that are registered in the Tmax system after authentication.

  - Step 3: Service access authentication

    Services that require special security can be used by users who have the corresponding privileges. Tmax 4.0 and later versions support this.

For more information about security, refer to "Security System" in *Tmax Application Development Guide*.

- Convenient API and various communication methods

  Tmax supports the following communication methods: Synchronous communication, Asynchronous communication, Conversational communication, Request Forwarding, Notify, and Broadcasting. Tmax provides a convenient API for these methods.

- System and resource management
  - The following statuses of the entire system can be monitored: process status, service queuing status, the number of handled services, and the average time of handling a service. System status and queue management system statistics can be analyzed, and a report can be created.
  - Resources are efficiently managed because applications and databases are integrated and managed.
- Multiple domains and various gateway services

  Remote distributed systems can exchange data; heterogeneous platform based systems can be easily integrated; and various gateway modules, such as SNA CICS, SNA IMS, TCP CICS, TCP IMS, and OSI TP, are supported. It is possible to handle a transaction service and route to multiple domains.

- Various client agents

  Tmax provides various agents to easily change a 2-tier system into a 3-tier system.

| Classification | Description |
|---|---|
| Raw Client Agent (RCA) | Supports multiple ports that efficiently handle processes with the multi-threading method. |
| Simple Client Agent (SCA) | Supports multiple ports that can handle both non-Tmax and Tmax clients. |

- Various development methods

| Classification | Description |
|---|---|
| Real Data Processor (RDP) | Supports direct data delivery using UDP communication data, not via the Tmax system. |
| Window control | Handles concurrent bundled data by providing the WinTmax library, the client library for multiple windows settings. |

- Extensibility
  - Integration with the web

    If the client/server environment and the web environment are integrated using Java Applet/Servlet, PHP, etc., response time can be decreased and system performance can improve. For the service integration, Tmax provides **WebT**. For more information about WebT, refer to *Tmax WebT User Guide*.

  - Integration with mainframe

    Host-link enables access to application services in a legacy system, such as IBM mainframe, like those in the client/server environment.

  - Easy to change other middleware

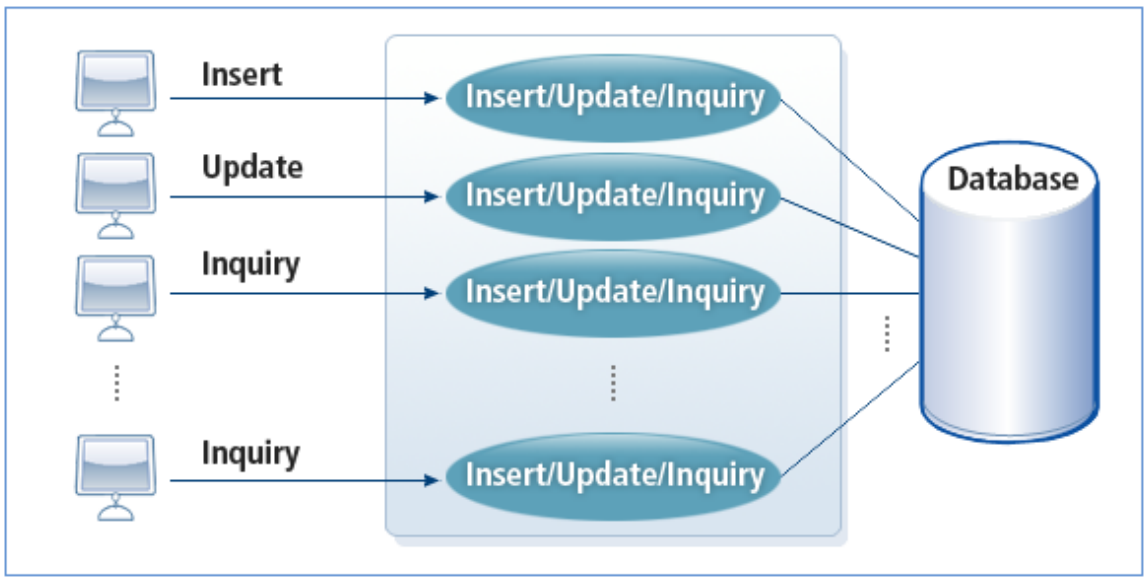
    Systems developed with other middleware, such as Tuxedo, TopEnd, and Entera, can be easily integrated with Tmax without any changes to the source code. This easy integration can improve performance, provide high level technical support, and reduce costs.

## 2.3.1. Process Management

The existing 2-tier client/server environment is the most general environment for developing applications. A single server process is created for a single client in this environment. As the number of clients increases, the number of server processes also increases. For this reason, it takes a long time to create processes and to open/close files and databases. Furthermore, since a server can be used only by a client who is connected to the server, usage of a server process is very low and the maintenance cost is very high.

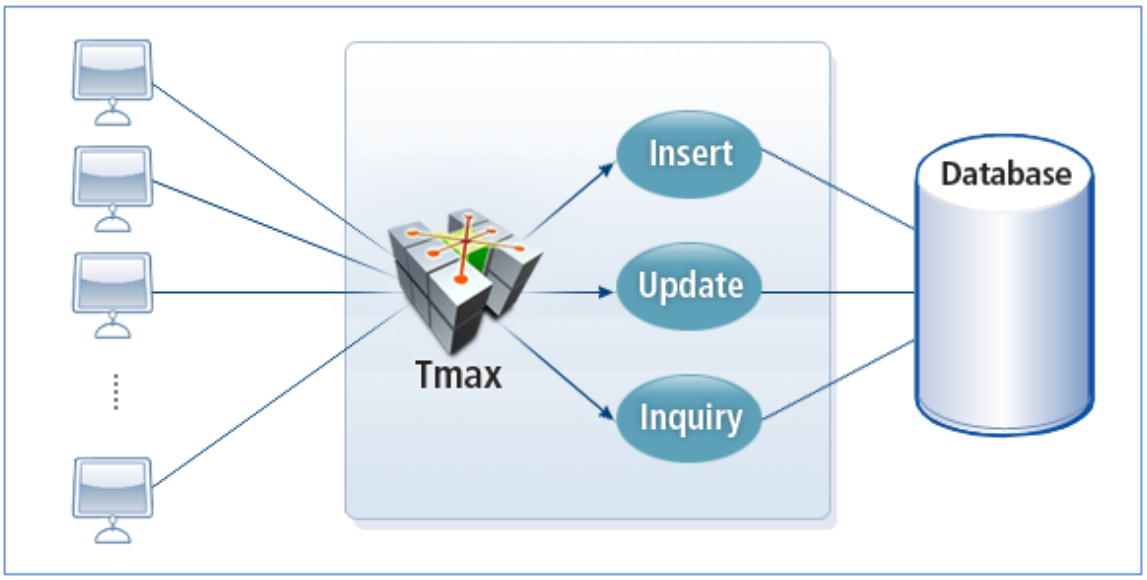
The 3-tier client/server structure using TP-Monitor is adopted to solve the problems. Tmax, the TP-Monitor product, enables a system to be optimized and operated as it adjusts the number of created processes, schedules idle server processes, and tunes server processes.

The following figure shows the 2-tier client/server structure:




2-tier Client/Server Structure

The following figure shows the 3-tier client/server structure:




3-tier Client/Server Structure

Better systems can be built in the 3-tier environment compared to the 2-tier environment in terms of performance, extensibility, management, and failover.

The following are comparisons of 2-tier and 3-tier client/server environments.

- Application

| 2-tier Environment | 3-tier Environment |
| --- | --- |
| • Unit business of a small scale department<br>• A single server<br>• Small number of users (less than 50)<br>• Batch applications | • Enterprise business<br>• Multiple servers<br>• Large number of users (50 or more)<br>• OLTP applications and large number of transactions |

- Advantages

| 2-tier Environment | 3-tier Environment |
| --- | --- |
| • Short program development period (easy to test)<br>• Low initial adoption cost<br>• Easy to develop simple programs | • Modularization for application development<br>• Integration is possible in heterogeneous hardware and in a database environment.<br>• Improved performance by making the best use of system resources<br>• Easy expansion<br>• Can implement security features including system management, load balancing, and failover |

- Disadvantages

| 2-tier Environment | 3-tier Environment |
| --- | --- |
| • Dramatically lower performance as the number of transactions increases<br>• Integration is impossible in heterogeneous platforms and in a database environment<br>• Difficult to be expand<br>• Impossible to implement the security feature including system management, load balancing, and failover | • Long application development periods (client/server integration test)<br>• High initial adoption cost<br>• A program must be separately developed for a client and a server even if the program is simple |

## 2.3.2. Distributed Transaction

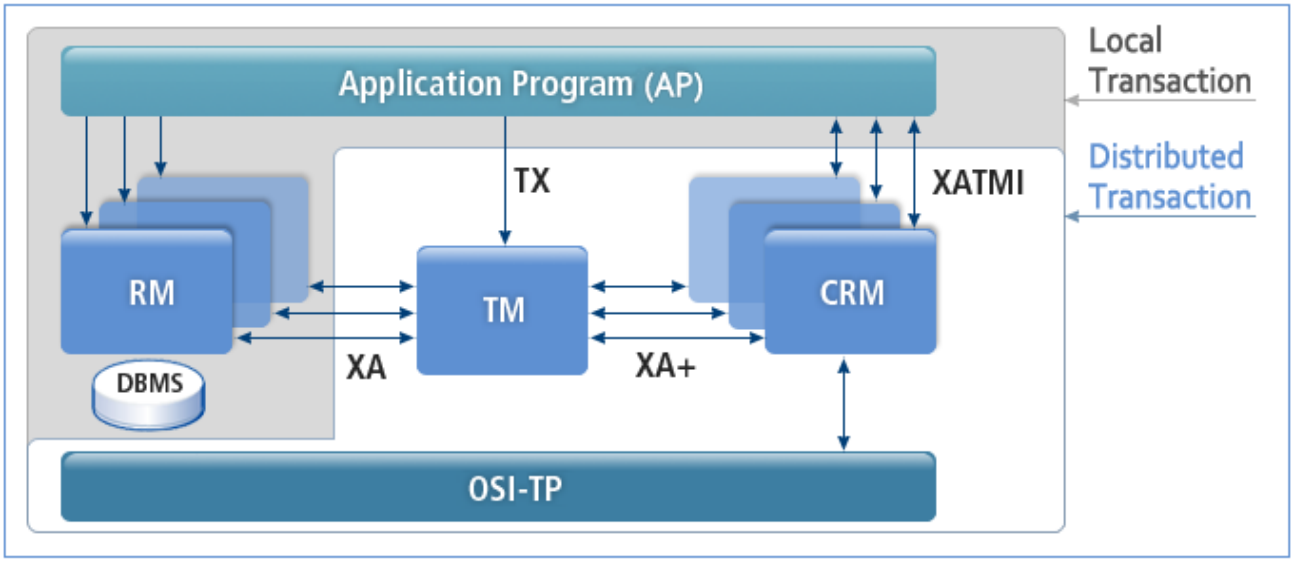
A transaction utilizes various resources by handling a single logical unit and maintains data integrity

among distributed resources. A distributed transaction is a transaction among distributed systems in the network, and it must be meet the ACID (Atomicity, Consistency, Isolation, Durability) transaction properties. The Tmax system guarantees ACID for distributed transactions in heterogeneous DBMSs and multiple homogeneous DBMSs.

| Classification | Description |
|---|---|
| **A**tomicity | All-or-nothing proposition. All work in a transaction is performed, or nothing is performed. |
| **C**onsistency | The successful result of a transaction must be maintained in a consistent status in shared resources. |
| **I**solation | While a transaction is performed, another operation cannot be performed for the transaction. The result cannot be shared before it is committed. |
| **D**urability | The result of a transaction is always maintained after it is committed. |

Tmax manages distributed transactions and complies with the X/Open DTP model that consists of Application Program (AP), Transaction Manager (TM), Resource Manager (RM), and Communication Resource Manager (CRM). Tmax supports the ATMI function, which is the set of standard functions that comply with the X/Open DTP model. Tmax binds and handles transactions that occur in a heterogeneous DBMS that complies with the X/Open DTP model.

The following figure shows the X/Open DTP structure.




X/Open DTP Structure

- Application Program (AP)

  Provides the DT boundary (Distributed Transaction).

- Resource Manager (RM)

  Provides a feature to access resources such as a database.

- Transaction Manager (TM)

Creates an ID for each DT, manages the progress, and provides a recovery feature for both completion and failure.

- Communication Resource Manager (CRM)

  Controls communication between distributed APs.

- Open System Interconnection-Transaction Processing (OSI-TP)

  Handles communication with a separate TM section.

When distributed transactions are handled, 2 step (two-phase) commit is supported for data integrity and APIs are provided for global transactions. Distributed transactions are managed using multiple heterogeneous hardware platforms and databases in a physically distributed environment.

- Two-phase commit (2PC) protocol

  2PC is used to guarantee transaction properties for global transactions related to more than one homogeneous or heterogeneous database. 2PC handles a transaction with 2 steps to guarantee ACID properties when more than one database is integrated.

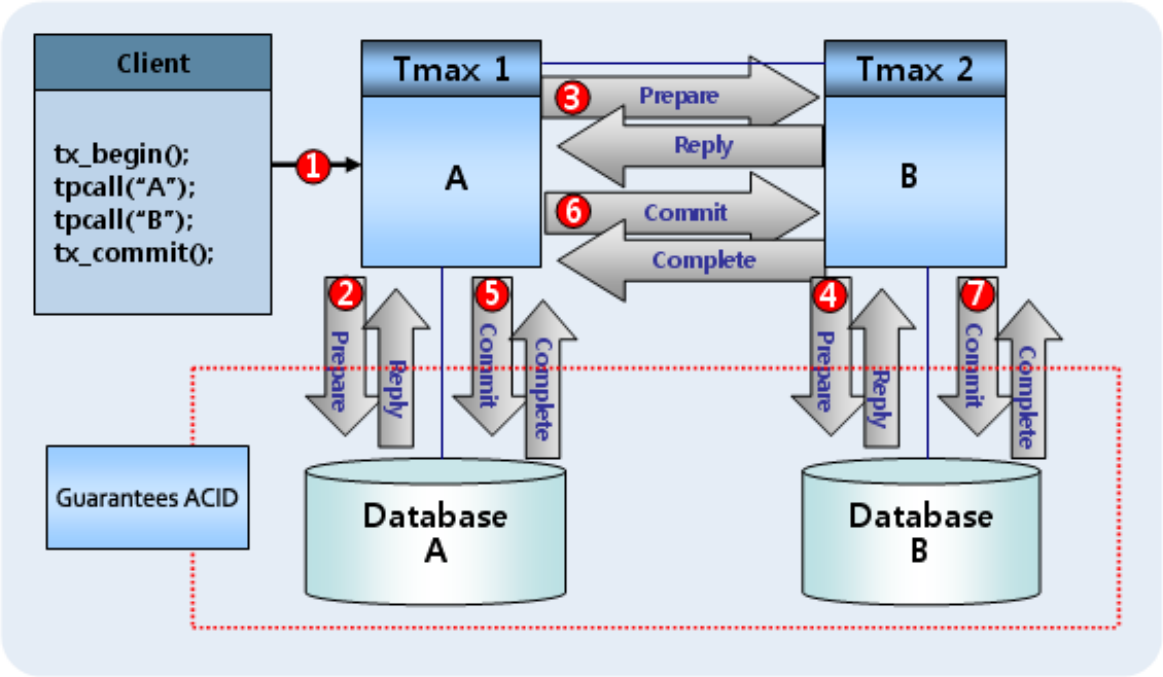
  - Step 1: Prepare Phase

    Checks that all databases related to a transaction are prepared to handle the transaction. If all databases are prepared, a signal is delivered. In this step, whether each database, network, or server combined with a single global transaction can commit or rollback is checked and databases are prepared.

  - Step 2: Commit Phase

    If all databases send a normal signal, the transaction is committed. If one or more databases send an abnormal signal, the global transaction is completed by performing rollback. It is the step that sends a commit message to all nodes and performs the commit request with RM. Notice that a data change job is complete when all nodes that participated in Commit Phase finish, and notify of the successful commit to the node that requests tx_commit().

    The following figure shows how a 2PC (Two-phase commit) is performed.




2PC (Two-phase commit)

- Global transaction

  Multiple heterogeneous hardware and databases are handled as a single logical unit (transaction). A global transaction related to a DBMS located in more than one homogeneous or heterogeneous system is handled by supporting two-phase commit to guarantee data integrity. The Tmax system supports global transactions by providing simple functions such as tx_begin, tx_commit, and tx_rollback. When a global transaction is handled, communication between nodes is handled by a client handler.

  The following describes how a global transaction is executed.

  - Step 1: Prepare Phase

    A node that starts a distributed transaction (global coordinator) checks whether it can perform commit or rollback for other nodes that are participating in the distributed transaction.

  - Step 2: Commit Phase

    The global coordinator receives replies from other nodes and performs commit. If one or more nodes send a message indicating the node is not prepared, the transaction is rolled back.

- Recovery / Rollback

  If a transaction fails, the previous transaction is recovered even if RM is changed.

- Transaction managed from a central location

  A transaction is centrally managed and controlled even if nodes are physically separated.

- Transaction scheduling

  A transaction is controlled with priority and concurrency.

## 2.3.3. Load Balancing

Tmax provides the load balancing feature to increase throughput and reduce handling time by using the following 3 methods: System Load Management (SLM), Data Dependent Routing (DDR), and Dynamic Load Management (DLM).

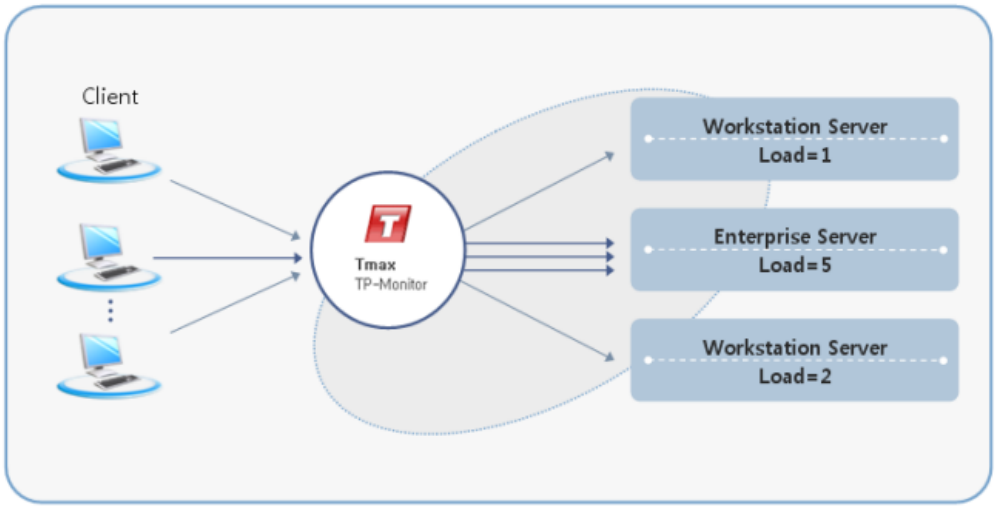
### Load Balancing by SLM

System Load Management (SLM) uses a defined load ratio to distribute loads. The load value is set according to hardware performance. If the number of service requests of a node exceeds the load value, the service connection is switched to another node. The load value can be set for each node.

The following describes how SLM is handled.

1. CLH receives a client request.
2. CLH determines whether it is a SLM service using TIM and checks the throughput of each server.
3. CLH performs scheduling for a proper server group.

The following figure is an example of load balancing by SLM. If the load values of Node 1, Node 2, and Node 3 are 1, 5, and 2, respectively, Node 1 handles 1 job, the next job is handled by Node 2, and the next job by Node 3. The next 3 consecutive jobs are handled by Node 2.




Load Balancing by SLM

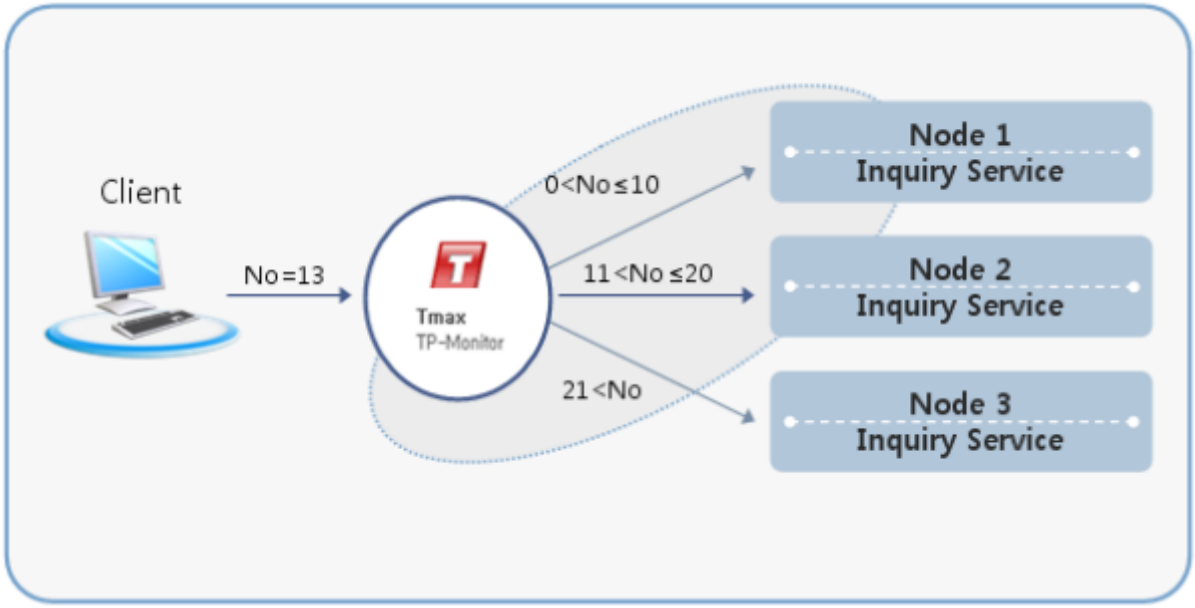
### Load Balancing by DDR

Data Dependent Routing (DDR) uses data values to distribute loads. If multiple nodes provide the same service, routing is possible among the nodes within the data range. Any entered field value is checked, and a service is requested from the proper server group.

The following describes how DDR is handled.

1. CLH receives a client request.
2. CLH determines whether it is a DDR service using TIM and checks the classified field value.
3. CLH performs scheduling for the defined server group.

The following figure is an example of load balancing by DDR. In the following figure, customers aged between 0 and 9, between 10 and 19, and 20 or over are handled in Node 1, Node 2, and Node 3, respectively.




Load Balancing by DDR

### Load Balancing by DLM

Dynamic Load Management (DLM) dynamically selects a handling group according to load ratio. If loads are concentrated at a certain node, Tmax distributes the loads with this method, which dynamically adjusts the load sizes. System loads can be checked with the queuing status of a running process.

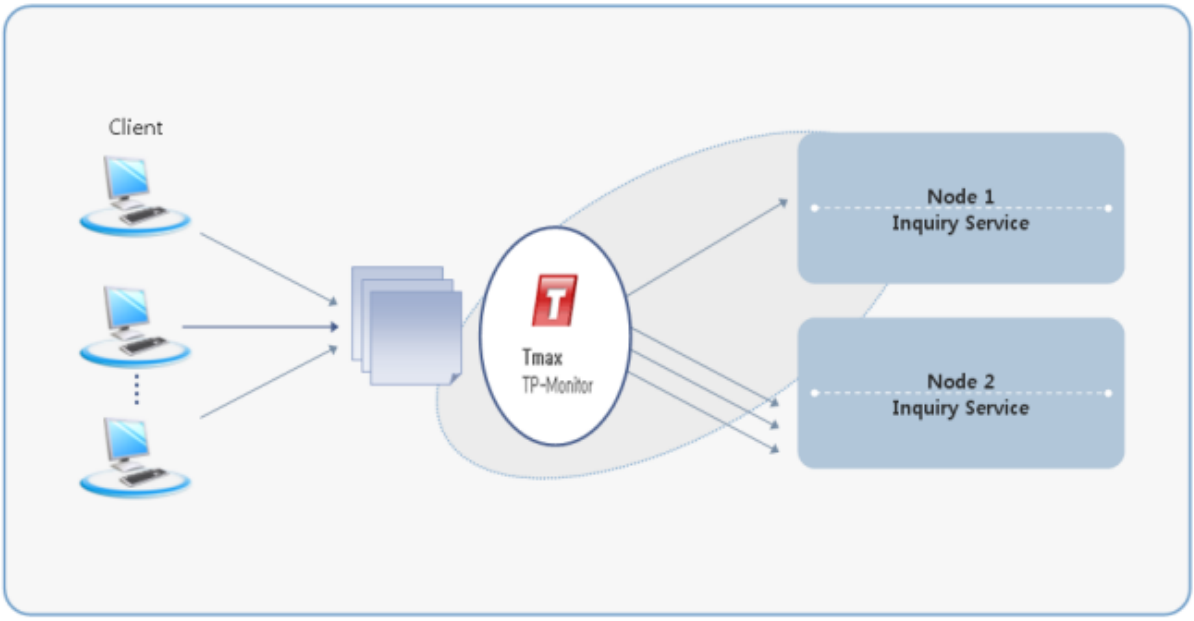
The Tmax system manages a memory queue for each process and saves a request service to this queue if there currently is no process to be mapped. The number of transactions in the memory queue is the system load.

The following describes how DLM is handled.

1. CLH receives a client request.
2. CLH determines whether it is a DLM service using TIM and checks the number of queuing requests by server.
3. If the threshold is reached, CLH performs scheduling for the next server group.

The following figure is an example of load balancing by DLM. In the following figure, it is assumed that Node 1 and Node 2 have the same services. If service requests are concentrated at Node 1,

Tmax distributes the loads by using the dynamic distribution algorithm.




Load Balancing by DLM

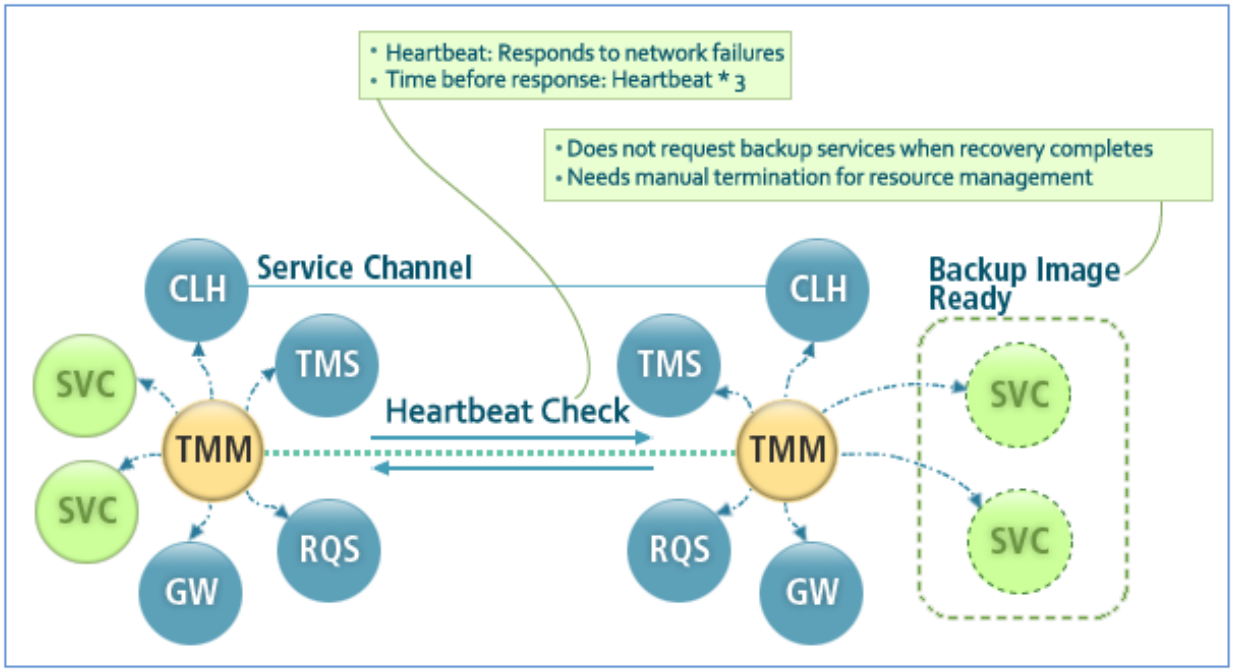
## 2.3.4. Failure Handling

Tmax guarantees high availability of system resources by providing continuous services even if a failure occurs. Failures can be hardware or software failures.

### Hardware Failure

Tmax can normally operate with load balancing or a service backup even if a hardware failure occurs.

Since Tmax is a peer-to-peer system in which nodes monitor each other, a failure can be handled in the same condition regardless of the number of nodes.

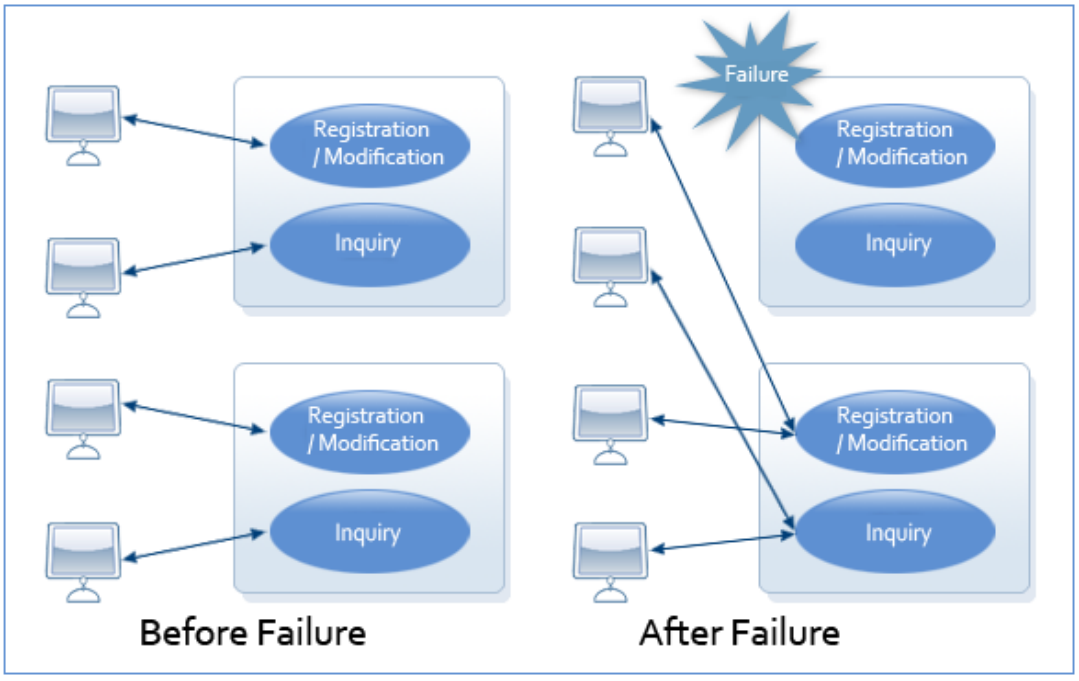



Hardware Failure

A hardware failure can be handled in 2 methods:

- Load balancing

  In an environment in which a certain service is provided by multiple nodes, if a failure occurs in a node, another node provides the service without a break. A client connects to a backup node and requests the service from the node.




Failover by Load Balancing

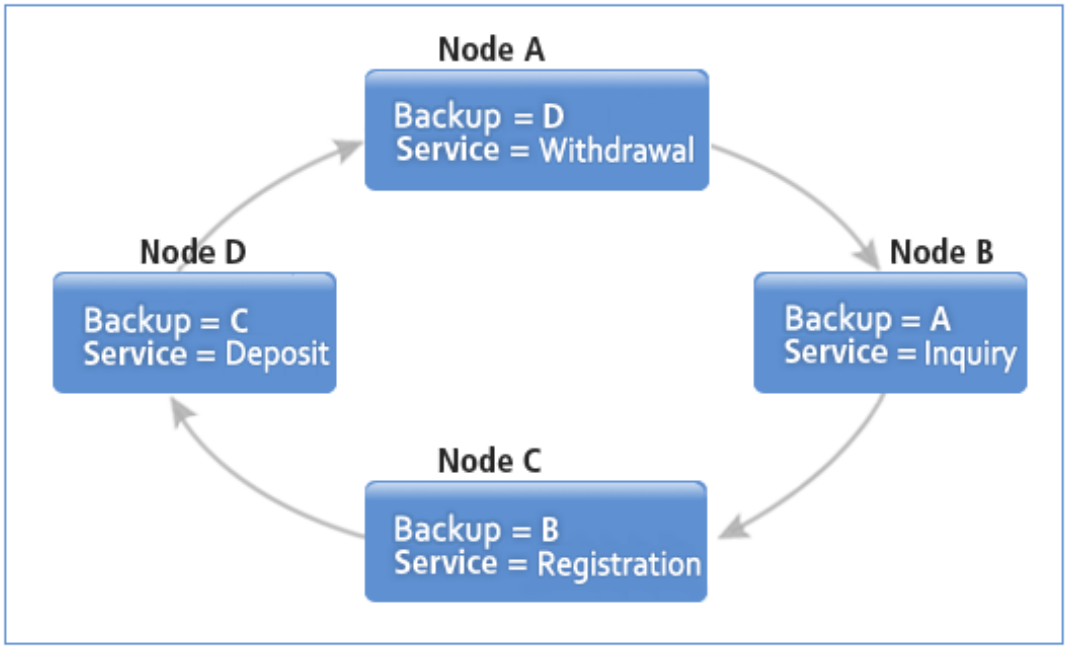
- Service backup

  If a failure occurs in a node, another node runs a prepared backup process and handles the

service.

- Before a Failure (Normal operation)

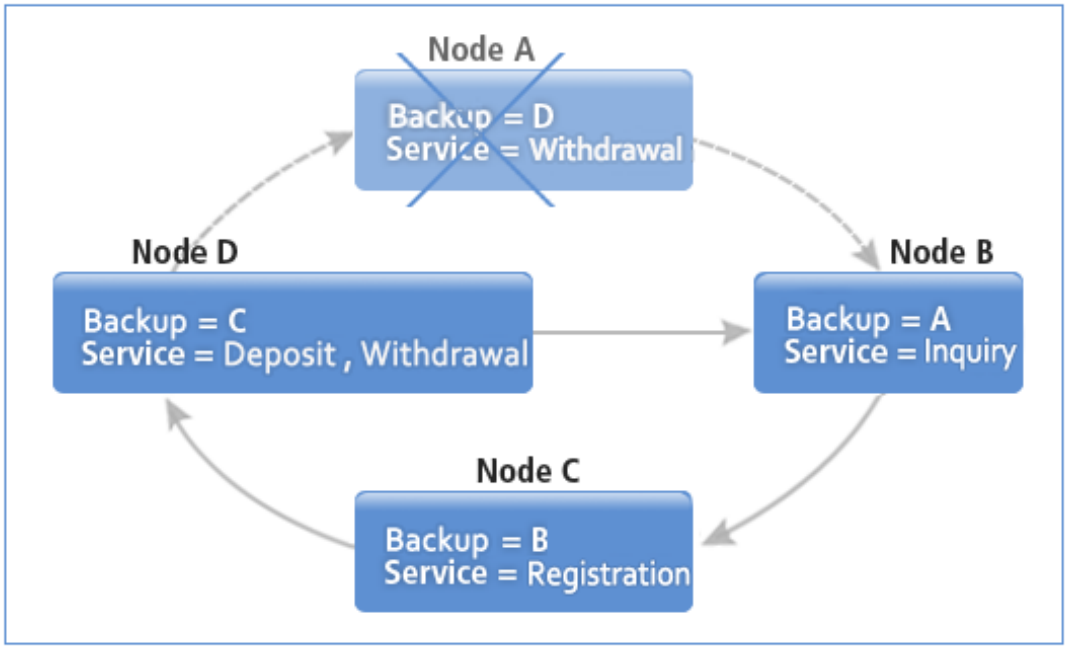
  A failure that can occur in any node can be handled because it is a peer-to-peer system.




Failover by Service Backup - Before a Failure

- After a Failure (Normal operation)

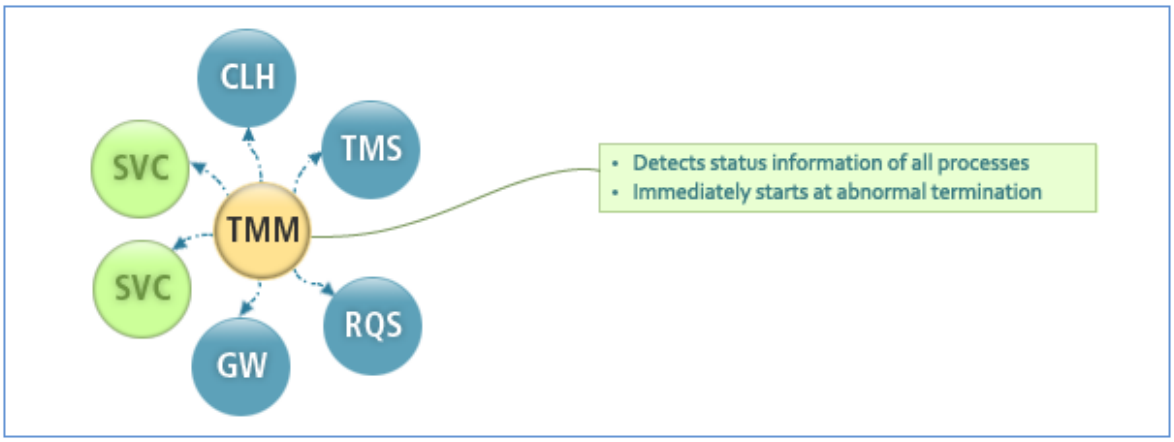
  The specified backup node provides the service without a break.




Failover by Service Backup - After a Failure

### Software Failure

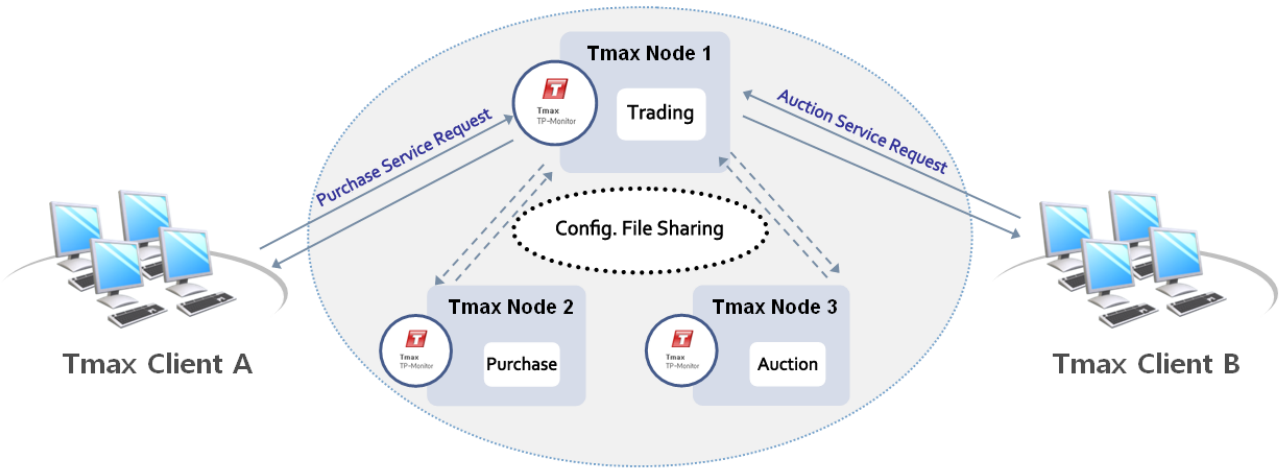
If a server process terminates abnormally due to an internal software bug or a user error, it can automatically restart. Notice that if a system process, such as TMS, CAS, and CLH, restarts endlessly without any conditions and it is abnormally terminated, a running server process can be terminated together.




Software Failure

## 2.3.5. Naming Service

Tmax guarantees location transparency with a naming service, which enables easy service calls by providing service location information in distributed systems. Although a client does not know the server address, the user can get server information with a service name. The naming service makes programming easy because a service can be easily and clearly called and the desired service can be provided with only the service name.




Naming Service

## 2.3.6. Process Control

Tmax supports the following 3 server processes:

- Tmax Control Server (TCS)

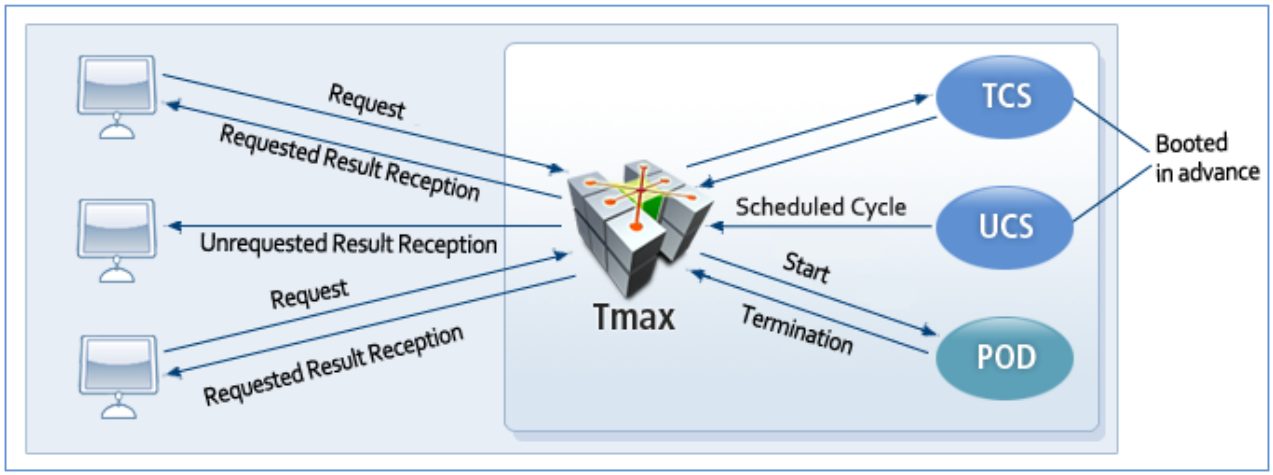
  Passively executed by a client request. It must be booted in advance to handle a request. TCS is the most typical method for handling a client request, receiving a caller request from a Tmax handler, handling the job, returning the result to Tmax, and waiting for another request.

- User Control Server (UCS)

  Actively transfers data without a caller request. It is a unique feature of Tmax. It must be booted in advance to handle a request. It can periodically transfer data to a client without a request as well as handle a client request like TCS. That is, UCS can handle client requests like TCS with added functions that can process applications actively and voluntarily.

- Processing On Demand (POD)

  Executed only when there is a client request and then terminated after handling the job. It is appropriate for seldom performed tasks.

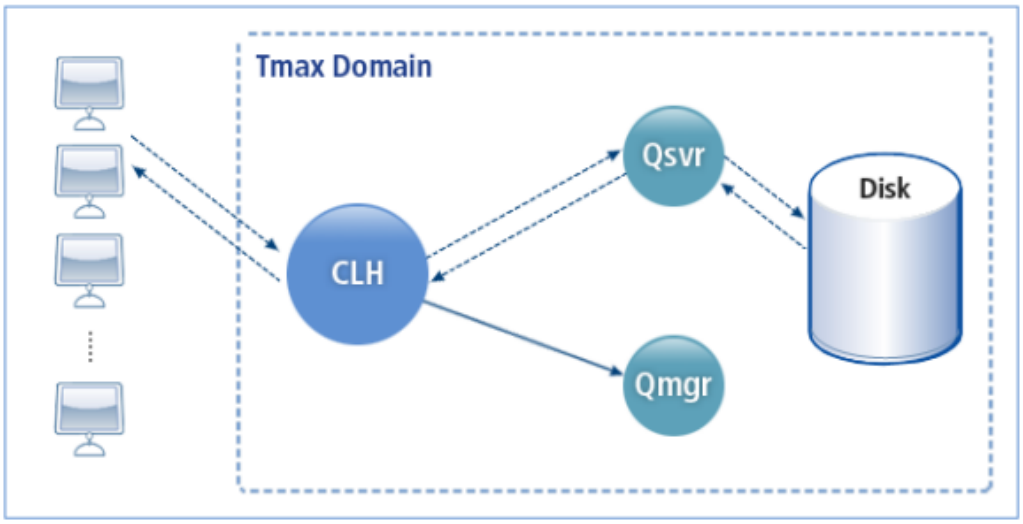



Server Processes

For more information about how to control server processes, refer to "Server Programs" in *Tmax Application Development Guide*.

## 2.3.7. RQ Feature

Reliable Queue (RQ) enables a service to be handled reliably by preventing a request from being deleted due to a failure. A requested job is saved to disk and then handled, if the job requires a long period of time or needs reliability. Even if there is a system failure or other critical error, the job can be normally handled after system recovery.




RQ Process

Whether a requested service was saved correctly to disk can be checked with the return value of the tpenq() function. A queuing job does not affect other jobs because it is independently handled by Queue manager (Qmgr). Whether Qmgr successfully completes the job can be checked with the tpdeq() function.

## 2.3.8. Security Feature

Tmax provides a data protection feature based on the Diffie-Hellman algorithm and supports a 5-step security feature, which includes the UNIX security feature.

## 2.3.9. System and Resource Management

### System Management

The following statuses of the entire system can be monitored: process status, service queuing status, the number of handled services, and the average time of handling a service. System status and queue management system statistics can be analyzed and a report can be created.

The following features are supported:

- Static system management

  The general system environment is set according to the user environment for the Tmax system components such as a domain, node, server group, server, service, etc.

- Dynamic system management

  The following components can be changed while Tmax is running:

| Component | Description |
|---|---|
| Domain | Service timeout, transaction timeout, node (machine) live check time, etc. can be changed. |
| Node | Message queue timeout can be set. |
| Server group | The load value by node, the load balancing method, etc. can be changed. |
| Server | Max queue count, server start count during queuing, server restart count, the number of servers, server priority, etc. can be changed. |
| Service | Service priority, service timeout, etc. can be changed. |

- Monitoring and administration
  - The dynamic environment setting can be changed.
  - Various reports can be displayed and various statistical information is provided such as transaction throughput of a server, the number of handled jobs by service, average processing time, etc.

#### Resource Management

Resources are efficiently managed because applications and databases are integrated and managed.

In existing systems, resources are wasted because the whole system cannot be managed. Tmax manages applications using the centralized monitoring feature for the entire distributed system.

If homogeneous or heterogeneous databases are used together for a single application, Tmax integrates and manages them in the dimension of applications.

## 2.3.10. Multiple Domains and Various Gateway Services

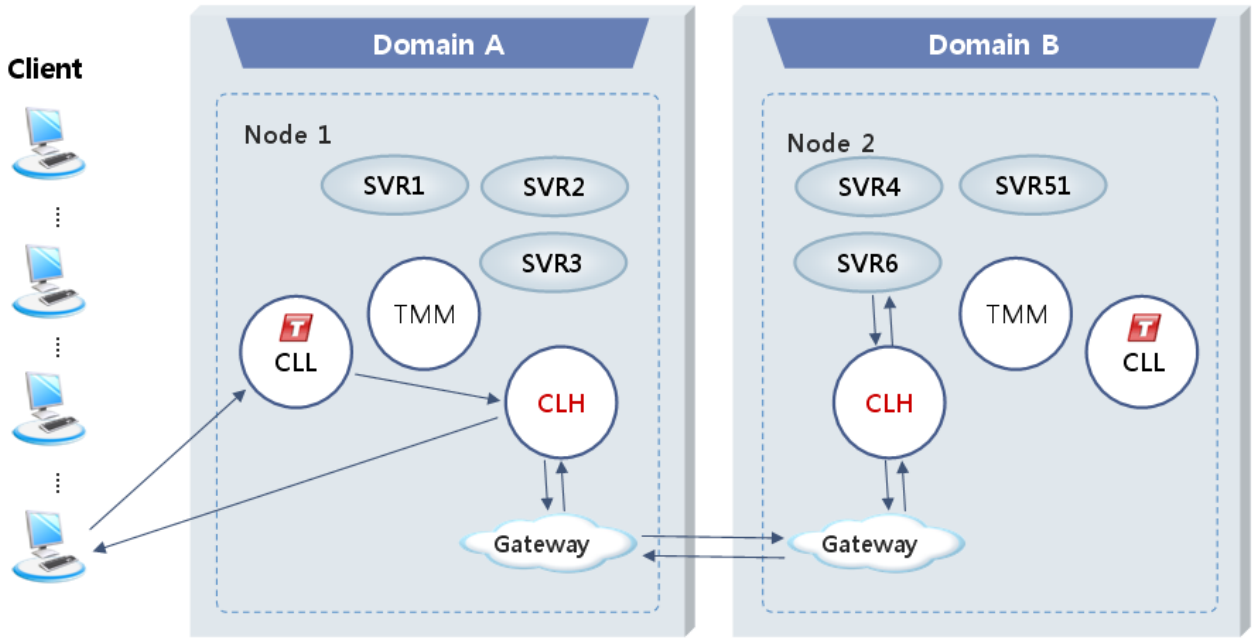
The Tmax domain is the top level unit that is independently managed (started / terminated) by Tmax. Even if a system is distributed by region or application and it is managed in multiple domains, the domains can be integrated. Additionally, the multi-domain 2PC function is supported through a gateway.

Remote distributed systems can exchange data, heterogeneous platform systems can be easily integrated, and various gateway modules, such as SNA CICS, SNA IMS, TCP CICS, TCP IMS, and OSI TP, are supported. It is possible to handle a transaction service and route it to multiple domains.

Multiple domains solve problems, such as difficulty in managing all nodes and the rapid increase of communication traffic between nodes, that may occur when multiple nodes are managed from a single domain. Furthermore, a requested service can be handled in any system. Service methods in a multiple domain environment are different according to service handling, routing, and transactions between multiple domains.

The following figure shows the flow of calling a service in a multi-domain environment. Multiple domains are connected via gateways. A domain gateway acts as a server or client when connecting

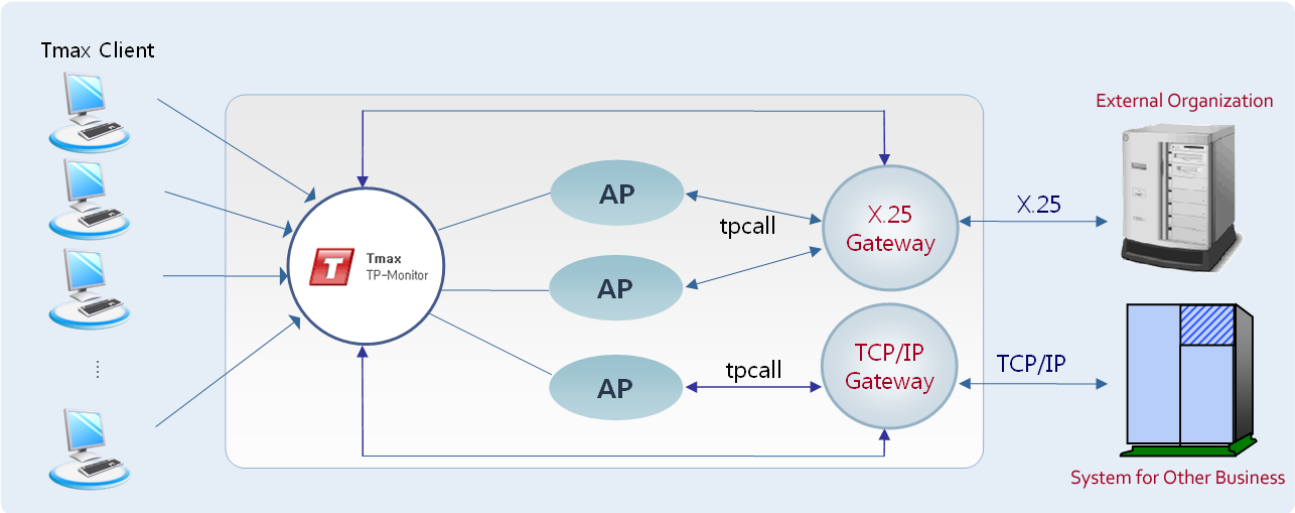
to another gateway. There is no start order for gateways.




Service Flow in a Multiple Domain Environment

Tmax provides various gateways, such as TCP/IP, X.25, and SNA, for easily integration with systems for other business and external organizations. A gateway enables efficient communication and provides convenient management by separating business logics and related modules. Since gateways are managed by Tmax, they can be automatically recovered after a failure and do not need to be managed by an administrator.

The following figure illustrates the role of a gateway. If a client requests a service, it receives the result from a domain, which provides the proper service. At this time, inter-domain routing is possible via a gateway according to transaction services or service values.




Tmax Gateway Service

## 2.3.11. Various Client Agents

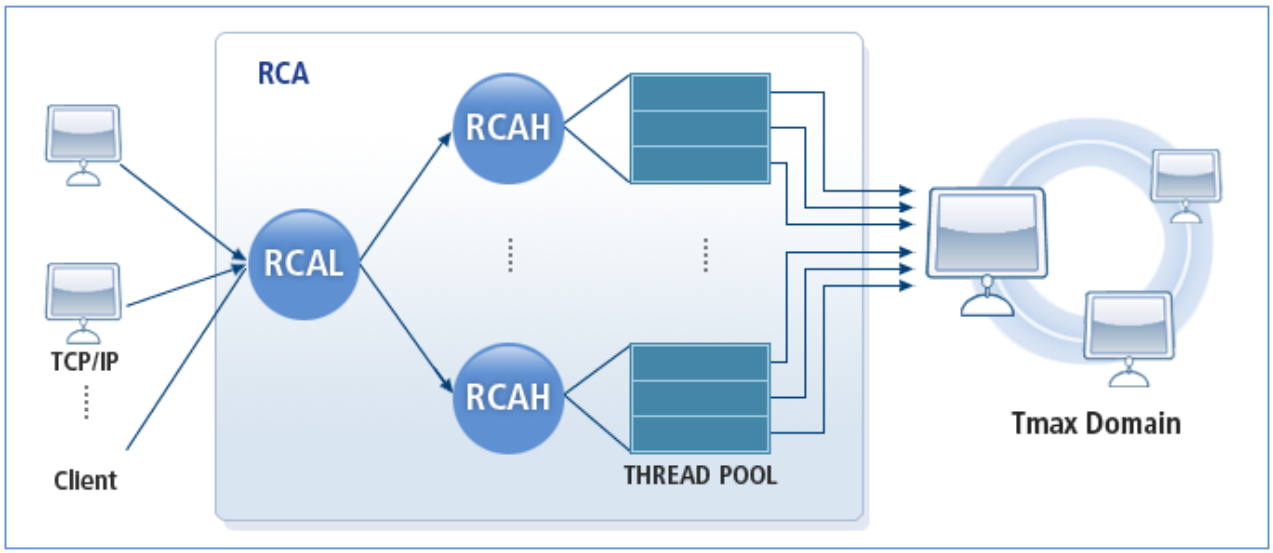
Tmax provides various agents to facilitate easy conversion from a 2-tier system to a 3-tier system.

### Raw Client Agent (RCS)

RCA connects to an existing communication program, which cannot use the Tmax client library, by using the TCP/IP socket and enables the program to use services provided by the Tmax system. It supports services in REMOTE or LOCAL mode. A single client library like the existing Tmax client library can be used by a single client program. However, a single thread acts as a single Tmax client because RCA is developed using the multi-thread method. Up to 32 ports can be specified to support various clients.

RCA provides the reliable failover feature with RCAL, which controls user connections, and RCAH, which was created with user logic. It also supports administration tools such as rcastat and rcakill.

The following figure shows the RCA structure:




RCA Structure

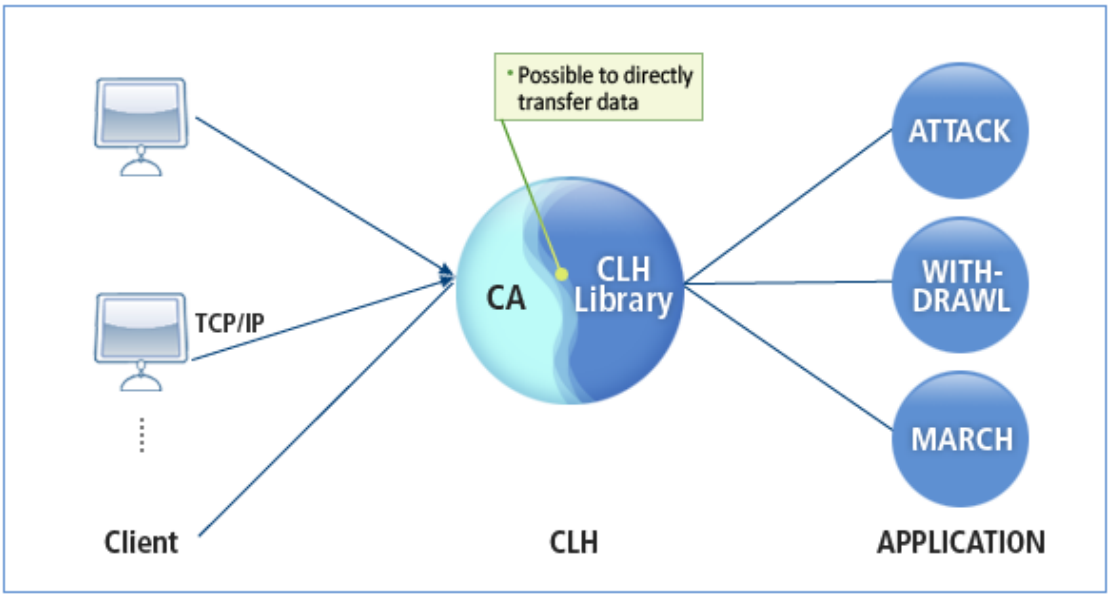
### Simple Client Agent (SCA)

SCA connects to an existing communication program, which cannot use the Tmax client library, via TCP/IP and enables the program to use services provided by the Tmax system. It consists of a customizing routine and the CLH library, which is linked with CLH.

Jobs on the network, such as connect/disconnect to a client and data transmission/reception, are internally handled in the system. A developer can connect to a client by setting the client and the predefined port number in the Tmax configuration file; and change and complement the received data and data to be transferred.

Up to 8 ports can be specified to support various clients. The SCA module can directly transfer data to the CLH module and simultaneously handle both non-Tmax clients and Tmax clients. SCA provides services using a TCP/IP raw socket or the Tmax client library.

The following figure illustrates how to call a service using Client Agent (CA), a type of SCA.



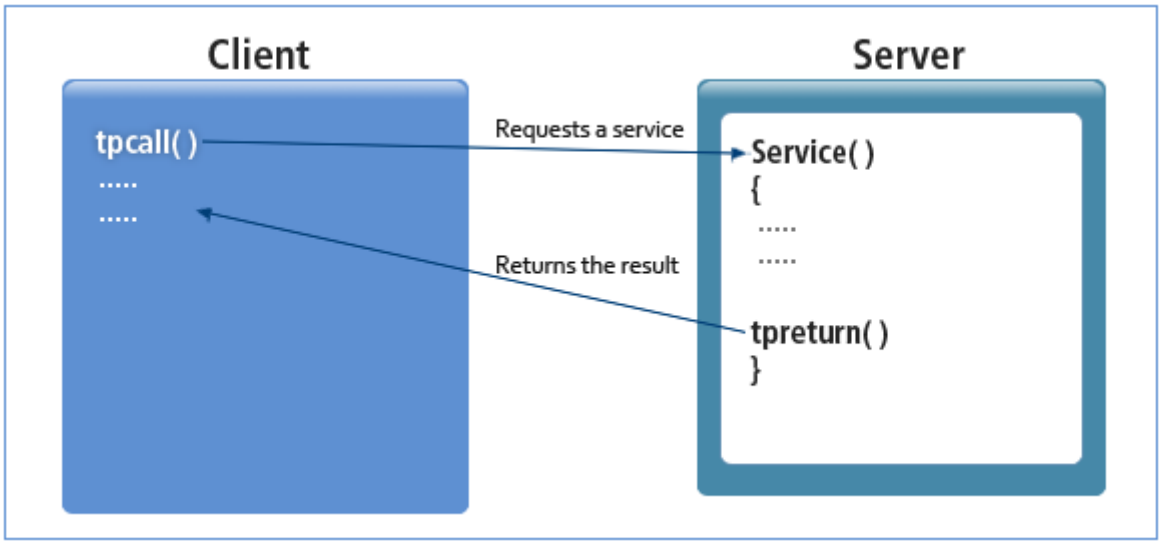

Service Call using CA

## 2.3.12. Various Communication Methods

A client requests communication with a server using one of the following 4 methods:

- Synchronous communication

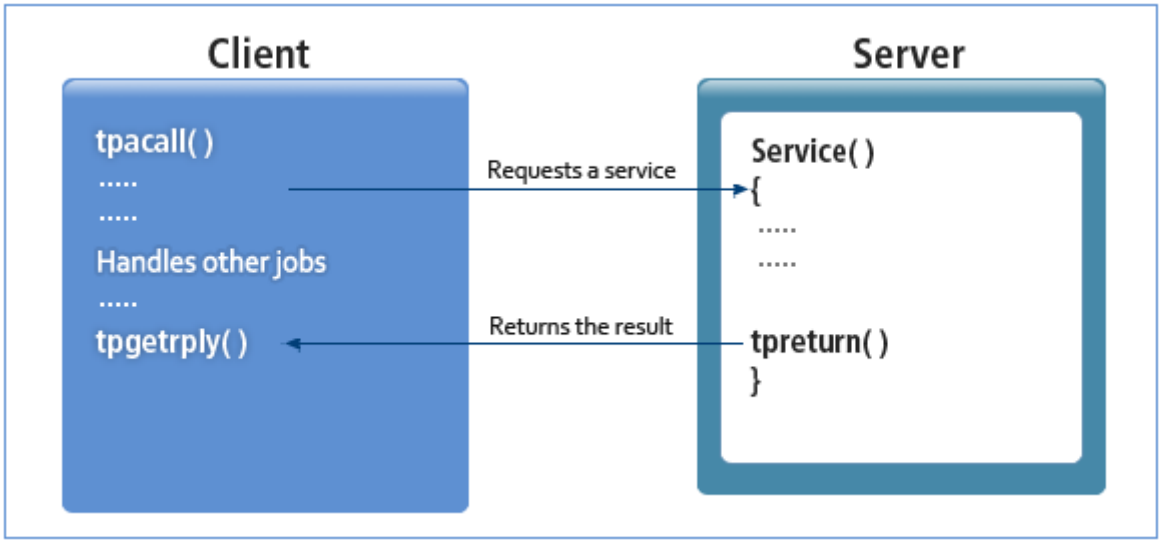
  A client sends a request to a server and waits while blocked until the reply is received.




Synchronous Communication

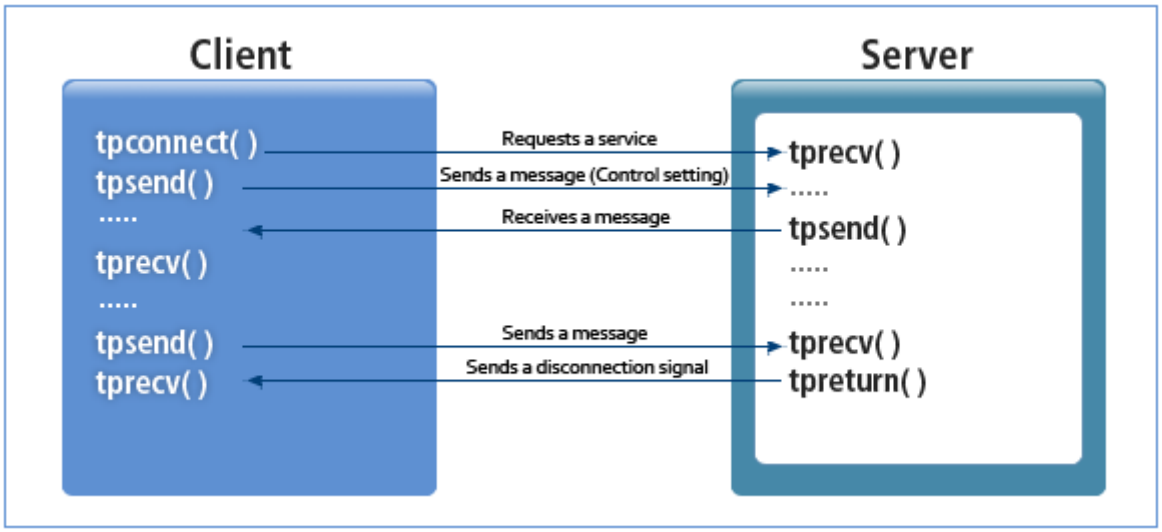
- Asynchronous communication

  A client can perform other tasks after sending a request to a server, while waiting for the reply. The client can receive the reply using a function.




- Conversational communication

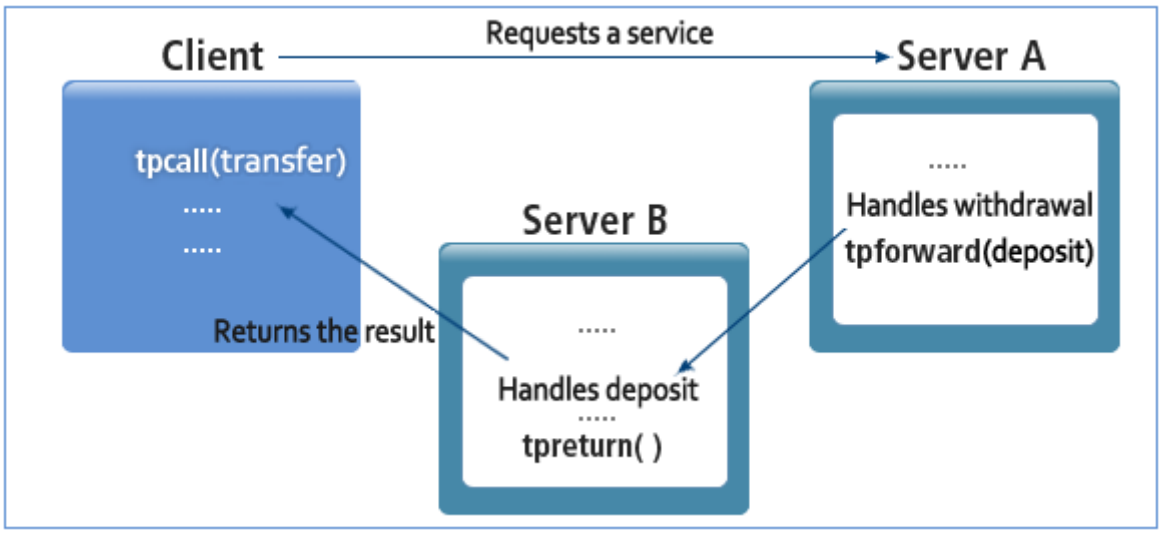
  When a client wants to send a request to a server, a connection is made and the service request is sent. When the client receives a message from the server, a conversational communication function is used. The client and the server send and receive messages by sending and receiving control through local communication. A control owner can send a message. When communication is made, a connection descriptor is returned. The returned connection descriptor is used to verify that the message was transferred.




Conversational Communication

- Forwarding Type
  - Type A

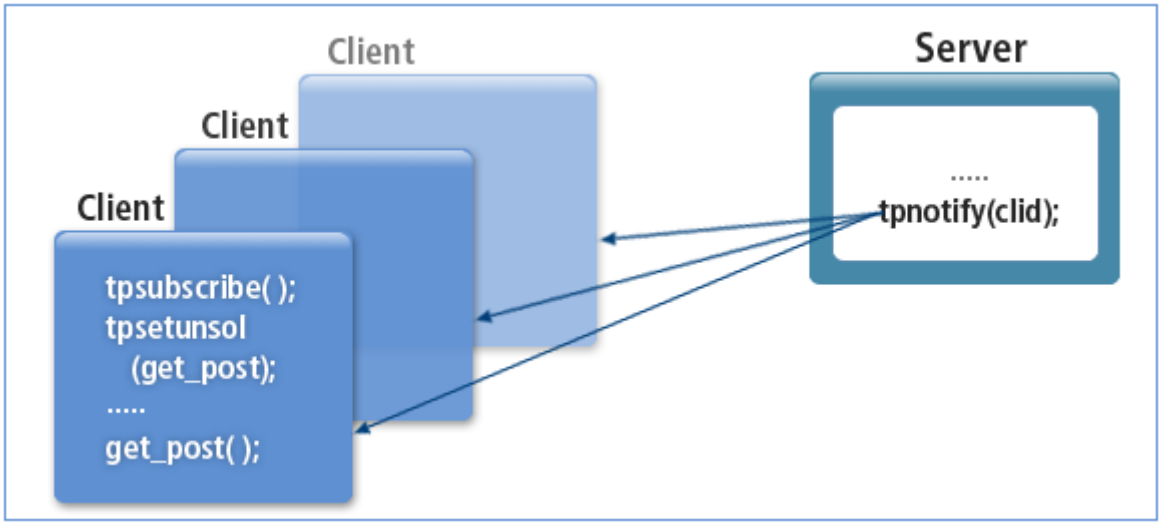
    Type A improves the efficiency of each module by modularizing business logic and handling services step by step. Problems can be systematically analyzed and corrected. Synchronous and asynchronous communication can be used together.




Forwarding Type - Type A

- Type B

  Type B integrates existing legacy systems like external organizations. A server process is not in blocking status in order to continuously handle any service requests received. This type supports a service that transfers a reply from a legacy system to a caller.

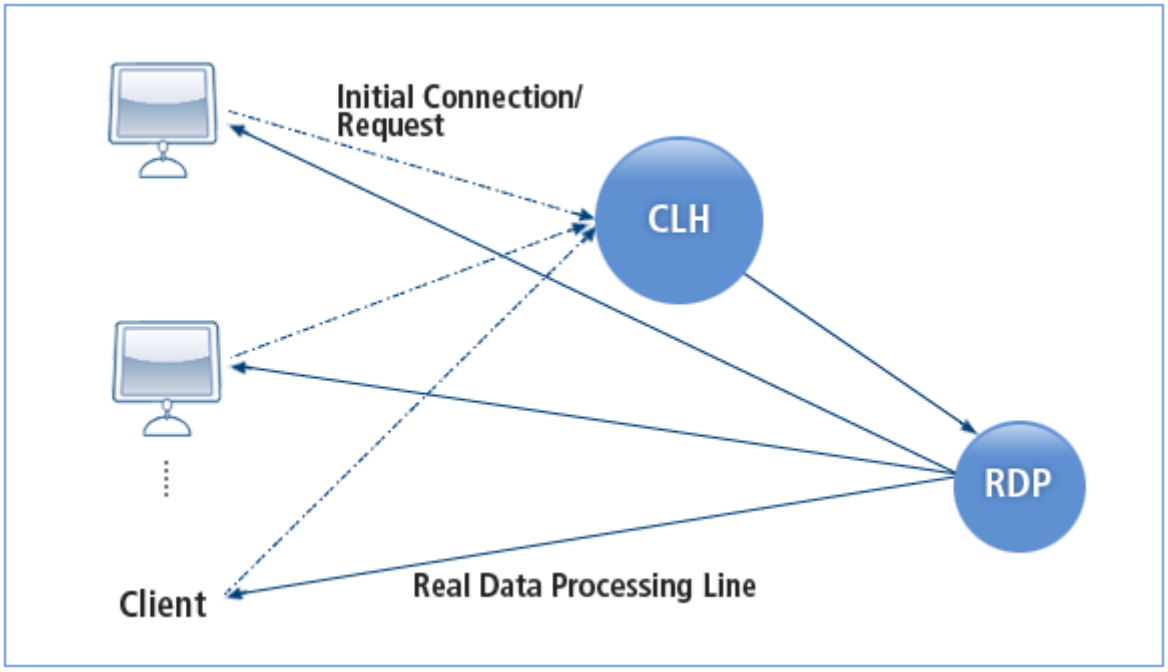



Forwarding Type - Type B

## 2.3.13. Various Development Methods

### Real Data Processor (RDP)

Data can be directly transferred from RDP to a client instead of via Client Handler (CLH) to rapidly handle data that is continuously changing in real time. Therefore, RDP improves performance of the Tmax system by reducing the loads of CLH. It is supported only for the UDP data type, and it is configured as a form of UCS server process.




Direct Data Transfer between a Client and RDP

### Window Control

Libraries are provided that can be conveniently used in a Windows client program. The two libraries provided, which operate based on threads, are WinTmax and tmaxmt.

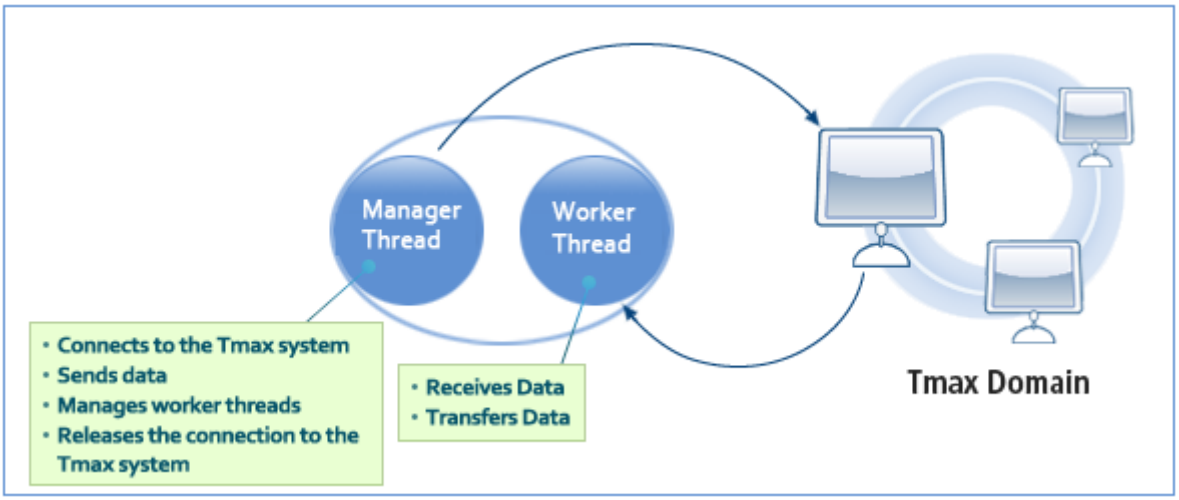
- WinTmax

  WinTmax is the Windows client library and it can set multiple windows.

  It consists of manager and worker threads. Since up to 256 windows can be set, it is useful to handle data that arrives simultaneously.

| Classification | Description |
|---|---|
| Manager thread | Connects to the Tmax system, receives data, manages worker threads, and releases the connection to the Tmax system. |
| Worker thread | Receives data and transfers data to a specific window. |

  The following figure shows the structure of the WinTmax library.




Structure and Features of the WinTmax Library

- tmaxmt

  It enables a client program to act as threads. A developer must develop a program using threads. Each thread sends and receives data using specified functions such as WinTmaxAcall() and WinTmaxAcall2(). Each function internally creates threads, calls a service, and transfers the result to a specified window or function.

  WinTmaxAcall sends data to a specified window using SendMessage while WinTmaxAcall2 sends data to a specified callback function.

## 2.3.14. Reliable Message Transfer

Tmax reliably transfers messages using Hybrid Messaging System (HMS). Tmax HMS has the following characteristics:

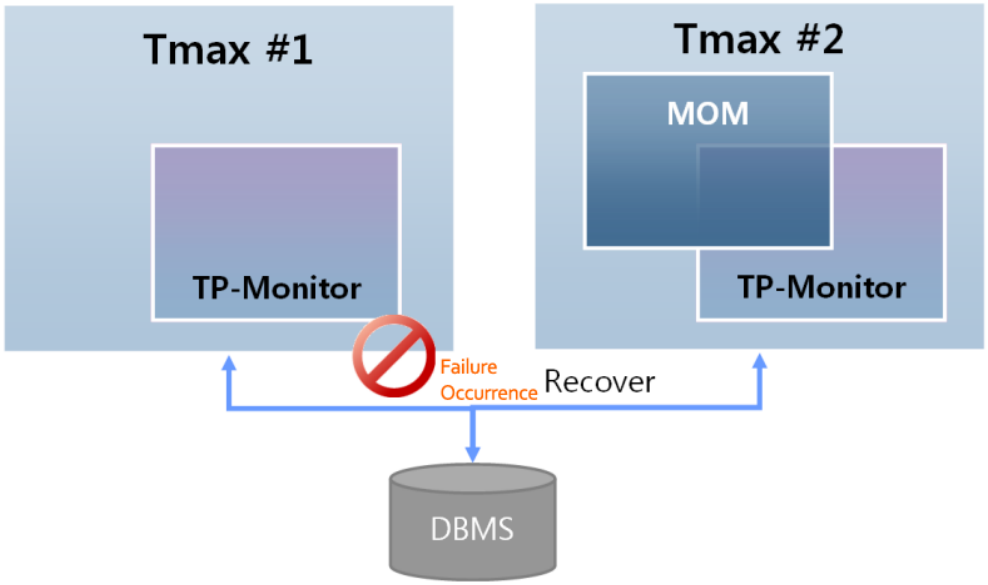
- Hybrid architecture

  TP-Monitor (Tmax) and Messaging Oriented Middleware (MOM) exist together. 2PC is supported between MOM and RM. Tmax HMS enables each module to be flexibly integrated with each other in order to provide MOM features. It has the structure in which TMS, TP-Monitor (basic feature of Tmax), and MOM exist together.

- Reliable failover

  Uses DBMS to save persistent messages.

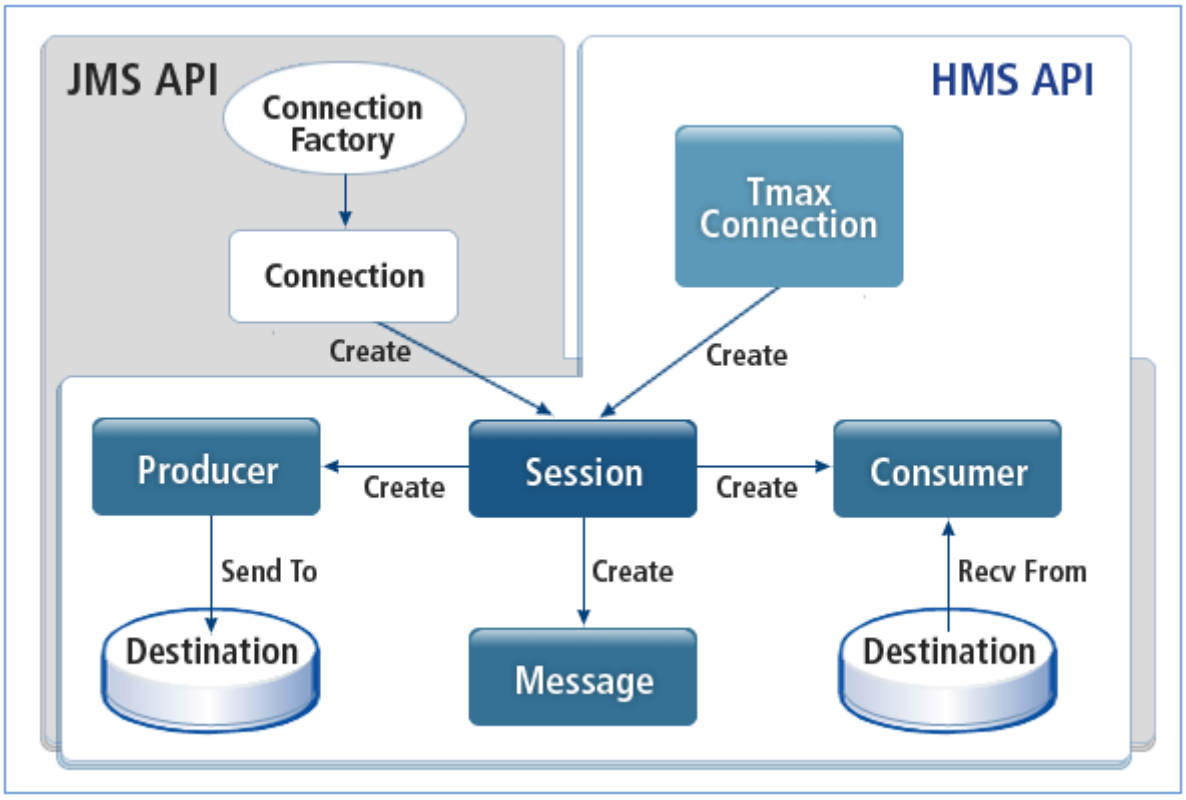
- High availability
  - Responds to a failure by specifying a backup node. It guarantees reliable failover and message transfers by using storage. It uses DBMS to guarantee failover reliability and recovers a persistent message from the DBMS if a failure occurs.
  - Responds to a system failure by configuring High Availability (HA) of Active-Standby.




High Availability of HMS

- JMS Messaging Model Adoption

  Adopts the JMS messaging model to support P2P and Publish/Subscribe, and provides C language APIs similar to the JMS specification for convenience.

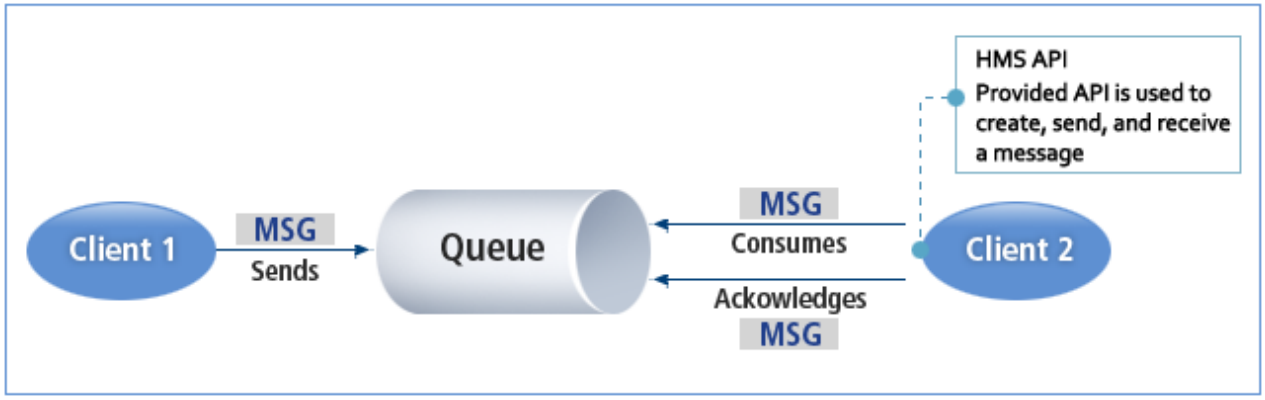



API Compatibility of JMS

HMS, a Tmax feature, is the communication medium for a loosely coupled sender and receiver. It supports the Queue and Topic methods.

- Queue method (Point-to-Point)

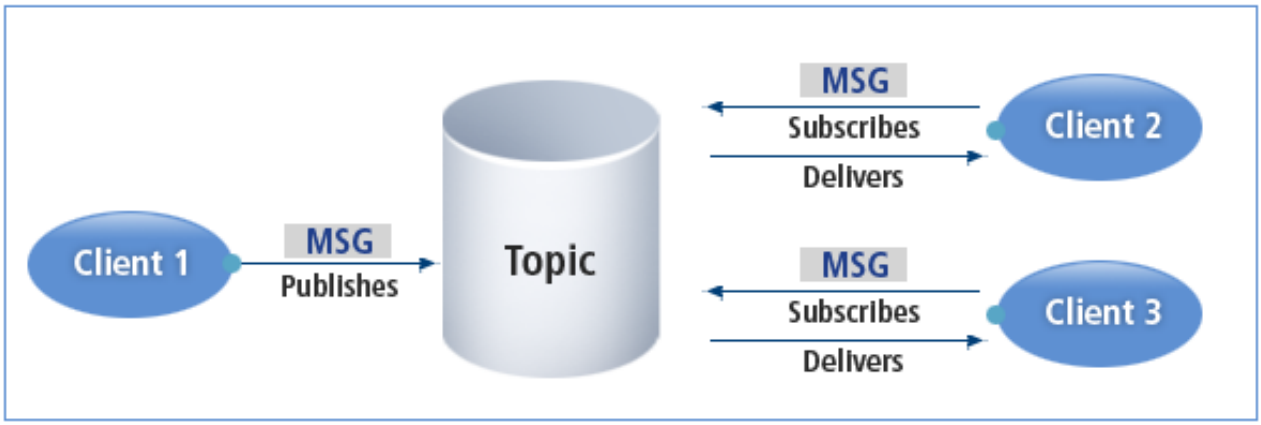
Each message is transferred to a single consumer. There is no timing dependency between transmission and reception.




HMS-Queue Method

- Topic method (Publish/Subscribe)

  Each message is transferred to multiple consumers. Reliability is guaranteed by sending the message through a durable subscription.




HMS-Topic Method

For more information about HMS, refer to *Tmax HMS User Guide*.

## 2.4. Characteristics of Tmax

Tmax adopts the peer-to-peer structure instead of the existing master/slave method. RACD (Remote Access Control Daemon) exists in each node. Tmax prevents Queue Full by using the stream pipe communication method. By blocking heavy transactions in network processes, the stream pipe communication method provides more reliable communication between processes than the message queue method. With these characteristics, Tmax minimizes the memory resource waste, supports rapid failover, and enables an administrator to actively respond to a failure.

The following are the characteristics of Tmax.

- Complies 100% with various DTP international standards such as X/Open and ISO-TP

    - Complies with the X/Open Distributed Transaction Processing (DTP) model, the international standard for processing distributed transactions, and specifies a compatible API and system structure based on the application program (AP), transaction manager (TM), resource manager (RM), communication resource manager (CRM), etc.
    - Specifies the framework for the following: features of a DTP service specified by Open Systems Interconnection (OSI) group, the international standard organization; API for the features; the feature for supporting distributed transactions; and the feature for adjusting multiple transactions that exist in a distributed open system.
    - Supports transparent business handling between heterogeneous systems in a distributed environment and On-Line Transaction Processing (OLTP).
- Enhances the efficiency of protection and multiplexing

  Enhances the efficiency of protection and multiplexing by implementing Inter Process Communication (IPC) of the stream pipe method.

- Provides various message types and communication types
    - Supports various message types such as integer, long, and character
    - Supports various communication types such as synchronous communication, asynchronous communication, conversational communication, and forwarding type communication
    - Supports Field Definition Language (FDL) and structure array
- Fault tolerance and failover
    - TP-Monitor with the peer-to-peer method
    - Handles H/W and S/W failures
    - Supports various features for preventing failure
- Scalability
    - Ensures stable system performance even if the number of clients increases
    - Efficiently changes from a 2-tier model to a 3-tier model using Client Agent (CA)
    - Provides various protocols for legacy systems
    - Provides services in the web environment using WebT
    - Provides various protocols for legacy systems such as TCP/IP, SNA, and X.25
    - Supports various process control methods
- Flexibility
    - Supports various process control methods
    - Supports customized features if necessary
- High Performance
    - Efficiently utilizes system resources
    - Provides an API to enhance productivity for simple and clear development
    - Provides the system monitoring feature that is convenient and easy to use.
- Supports various H/W platforms

    - Supports for IBM OS/390, most UNIX systems, Linux systems, Windows NT system, etc.
- Supports almost all 4GL such as PowerBuilder, Delphi, Visual C/C++, Visual Basic, and .NET(C#, VB)

# 2.5. Issues on the Tmax Adoption

## 2.5.1. System Environment

### Supported Environment

The following shows the basic environment of the Tmax system.

| Classification | Description |
|---|---|
| Protocol | Application API: XATMI and TX |
| | Integrating API: XA |
| | Network: TCP/IP, X.25, and SNA (LU 0/6.2) |
| OS | Server: All UNIX, NT, and Linux |
| | Client: All UNIX, Windows, MS-DOS, etc. |
| Platform | All H/W that support IBM OS/390, UNIX, Linux, and Windows NT |
| Development language for servers | C, C++, and COBOL |
| Development language for clients | Supports interfaces for C, C++, and various 4GL (Power Builder, Delphi, Visual C/C++, Visual Basic, .NET (C#, VB), etc.) |
| DBMS | Oracle, Informix, Sybase, and DB2 (UDB) |

### Server Requirements

The following table shows requirements for a server when adopting Tmax:

| Classification | Description |
|---|---|
| Hardware | Memory: 0.50 MB Disk (Tmax client): 0.277 MB<br><br>(Include - 83 KB, DLL - 86 KB, and Type Compiler - 108 KB) |
| Software | IBM OS 390, UNIX, Linux, and Windows NT |
| | C, C++, or COBOL Compiler |
| Network protocol | TCP/IP |

### Client Requirements

The following table shows requirements for a client when adopting Tmax:

| Classification | Description |
|---|---|
| Hardware | Memory: 0.537 MB + 0.2 - 0.5 MB / application<br>Disk (Tmax client): 0.277 MB<br>(Include - 83 KB, DLL - 86 KB, and Type Compiler - 108 KB)<br>An additional 203 KB (DLL, PBD) for Power Builder is required. |
| Software | Linux, Windows NT, Windows (2000, XP), MS-DOS, and UNIX |
| | Power Builder, Delphi, Visual Basic, Visual C++, C, and .NET (C#, VB) |
| Network protocol | TCP/IP |

## 2.5.2. Issues

There are issues in terms of features, performance, reliability, etc. when adopting Tmax.

The following are the details of the issues of the features.

| Issues | Details |
|---|---|
| Basic features of TP-Monitor | • Process management<br>• Distributed transaction support<br>• Load balancing<br>• Various communication methods between clients and servers<br>• Failure handling (handling and preventing all server failures)<br>• Heterogeneous DBMS support |
| Additional features | • Security feature<br>• System management<br>• Naming service<br>• Multiplexing feature of the BP clients<br>• Appropriate security features for specific systems<br>• Structure array communication support<br>• Ability for integration with a host |

The following are the details of the issues besides features.

| Issues | Details |
| --- | --- |
| Performance | • The average handling time and the maximum number of jobs handled per hour<br>• Resource usage |
| Reliability (failure handling) | • Failure frequency of a customer<br>• Ability and time for handling a failure |
| Education and technical support | • Technology level of engineers<br>• Education and consulting support (professional education for application development (OS, network, and TP-Monitor) and consulting for the system design step) |
| Risk management | • Trial and error minimization when adopting a new system<br>• System building using new technology<br>• Understanding of tasks for building |
| Convenience for development and operation | • Client/server development tool support<br>• Drivers provided for development<br>• System statistical information monitoring<br>• Dynamic change of the TP-Monitor environment<br>• Convenience for system operation<br>• The report feature |
| User satisfaction | • Satisfaction for features<br>• Satisfaction for technical support and education<br>• Satisfaction for unexpected product capabilities |
| Compatibility | • Complies with international standards such as X/Open DTP and OSI-TP<br>• Independence from specific H/W and DBMS |
| Business Size | • Capital scale<br>• The number of employees<br>• Sales<br>• Growth potential |
| Other | • Technology transfer<br>• Version up planning |

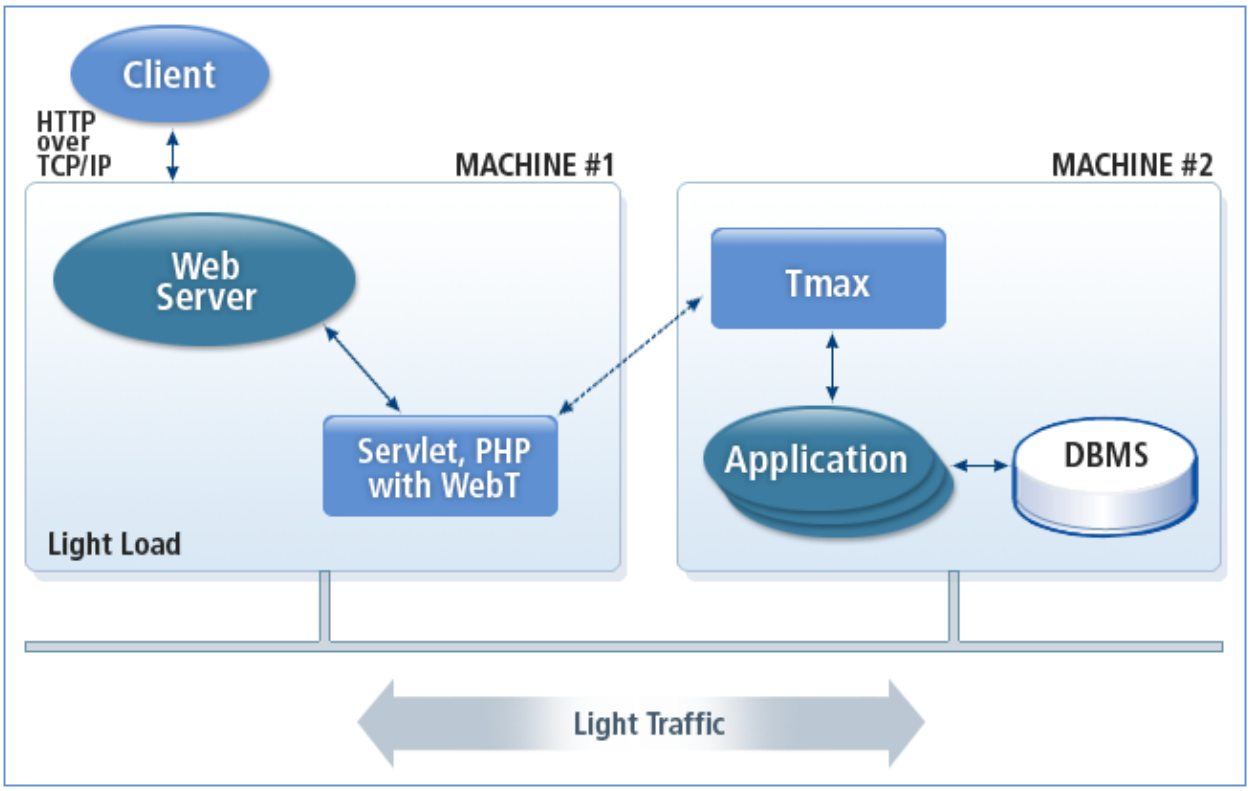
# 3. Introduction to WebT

This chapter describes the basic features and roles of WebT which supports transaction services for Tmax and Java application programs.

## 3.1. Overview

Tmax is a middleware product that operates in a client/server environment.Web Transaction (WebT) supports transaction services between Tmax and Java application programs. WebT is distributed as an API library. It is used by Web Application Server (WAS) products that operate in a web environment such as JEUS. It is designed to provide dynamic data services using the transaction handling and load balancing features of Tmax in a web environment.

The following figure shows the service flow between WebT and Tmax. If a client sends a request, the server program of Tmax is executed through the WebT module and then the service will run.




Service Flow between WebT and Tmax

## 3.2. WebTConnectionPool

In order to efficiently manage Tmax connections, WebT provides a class called WebTConnectionPool. WebTConnectionPool does not create a new connection object whenever a Tmax service is requested, but reuses an old object that was previously used. By doing this, resources and time that are spent to set and close network connections for a Tmax server can be saved.

A WebTConnectionPool consists of one WebTConnectionGroup or more, and each

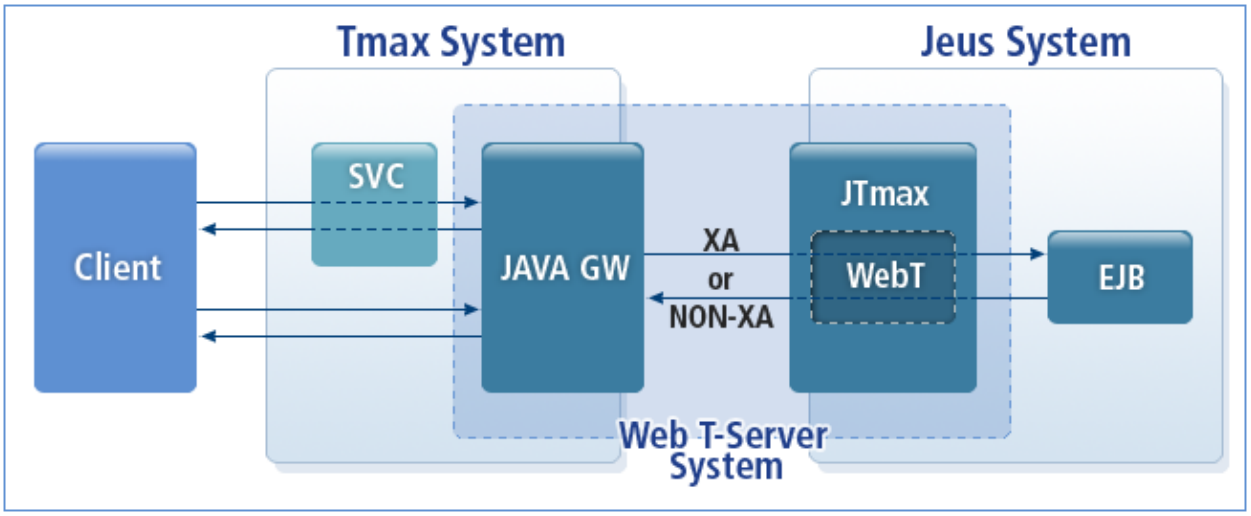
WebTConnectionGroup connects to one Tmax server. A client program can be connected to a Tmax server using the name of a WebTConnectionGroup.

If a WebT module links with JEUS, a JEUS container manages a WebTConnectionPool. The JEUS container creates a connection pool by using the WebT properties configuration file or JEUS configuration file and automatically returns a non-returned connection to the connection pool.

## 3.3. WebT-Server System

A WebT-Server system exists between a Tmax system and JEUS. It enables a Tmax client to call a EJB service of JEUS.

The following figure shows how an EJB service is called through a WebT-Server system.




WebT-Server System

A WebT-Server system consists of the following modules.

- JAVA GW

  Processes service requests sent from Tmax to JEUS.

- JTmax

  A daemon in JEUS receives a Tmax service request.

- WebT Library

  Processes data that is exchanged between Tmax and JEUS.

- Other Utilities

  The webtutil.jar is a file that packages classes to use in jeus.jar and jeusutil.jar when using WebT. It is a library used to start WebT without JEUS.

# 4. Tmax Applications

This chapter describes the basic concepts, client/server programs, APIs, and errors for developing Tmax applications.

## 4.1. Application Configuration

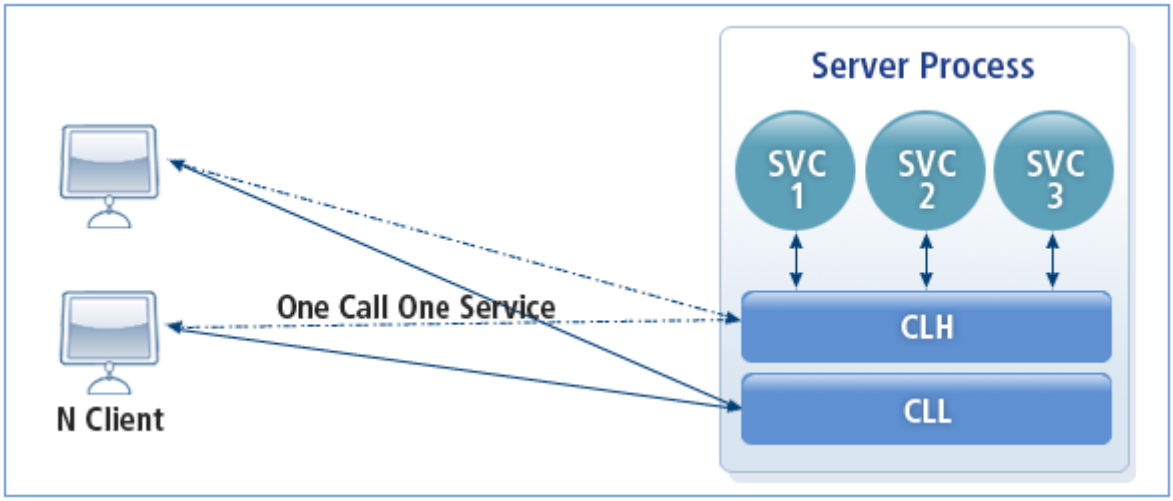
A Tmax application consists of a client program and a server program. The client program handles user interfaces (presentation logic), while the service routine for servers handles tasks and implements access logic.

If a client has a request, a server handles the request. Although there are N clients, a single call starts a single service. When a client has a request, the following occurs: the client connects to CLL, a tpcall is made for the request, CLL handles the request, and the request is returned if it is successfully handled. When a client program connects to Tmax, it needs information from Tmax such as the host address and the port number. The information is saved in the .profile or the tmax.env file and set in the TMAX_HOST_ADDR and TMAX_HOST_PORT properties, respectively.

A server program consists of main() that is provided by Tmax and service routines written by a developer.

Tmax provides various API functions to handle connection data requests from clients and servers. Tmax API complies with the international standard for distributed processing (X/Open DTP model). When an application is developed, only a service routine is developed.

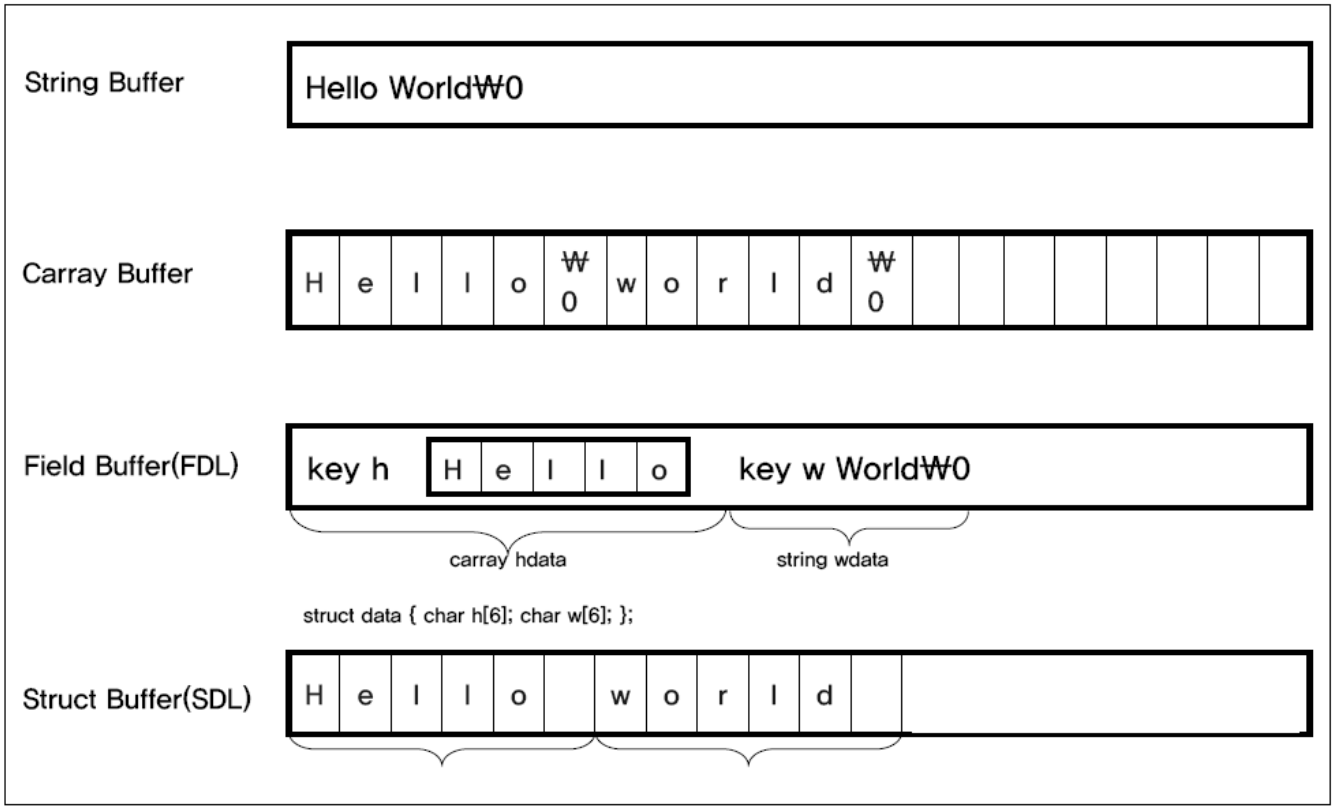
The following figure shows the configuration of a Tmax application.




Tmax Application Configuration

## 4.2. Buffer Types

When a client requests a service from a server, it uses the following communication buffer for communication.




Tmax Communication Buffer Type

- STRING buffer

  Used to save a string that ends with NULL. The buffer length does not need to be specified. There is no problem caused by platform differences.

- CARRAY and X_OCTET buffers

  Used to save binary type data that has the specified number of bytes. The length of a buffer must be specified to exchange data. There is no problem caused by platform differences.

- STRUCT and X_C_TYPE buffers

  Used to use a C language struct. All primitive types can be used as members of the struct. A declared struct and an array of structs can also be used as a member of the struct.

- X_COMMON buffer

  Used to save a C language struct. Only the char, int, and long types can be used as members of the struct.

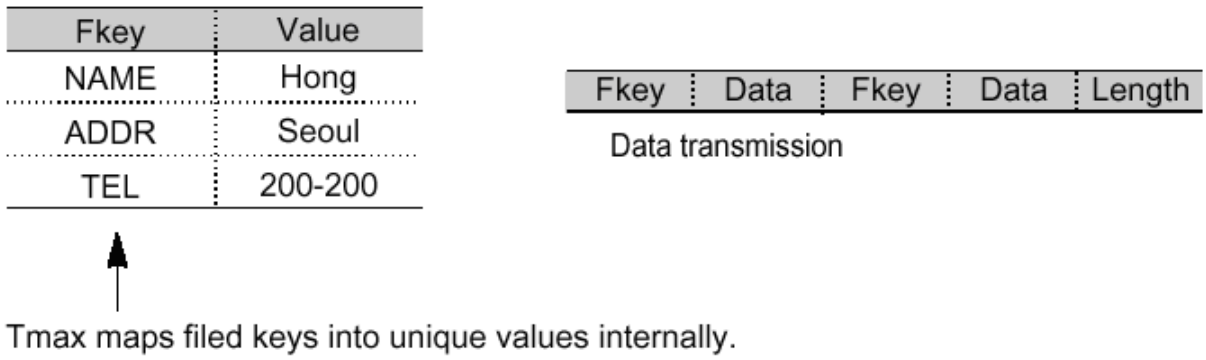
- FIELD buffer

  Used to save field key and data value pairs. All primitive types can be saved in this buffer. It is used when an exchanged data type can be changed. It provides various APIs for accessing and converting data.

### Field Definition Language (FDL)

FDL can operate and handle specific data from the desired information using a field key buffer, unlike a general structure.

FDL specifies names for each field key (NAME, ADDR, and TEL), numbers, and types in a <XXX.f> field buffer file. If <XXX.f> is compiled using FDLC, a file mapped into <XXX_fdl.h> is created. Field keys and values selected by a user can be accessed by referring to the <XXX_fdl.h> included during compilation.




FDL Method

## 4.3. Client/Server Program

The client program handles user interfaces (presentation logic), while the service routine for servers handles tasks and implements access logic.

### 4.3.1. Client Program

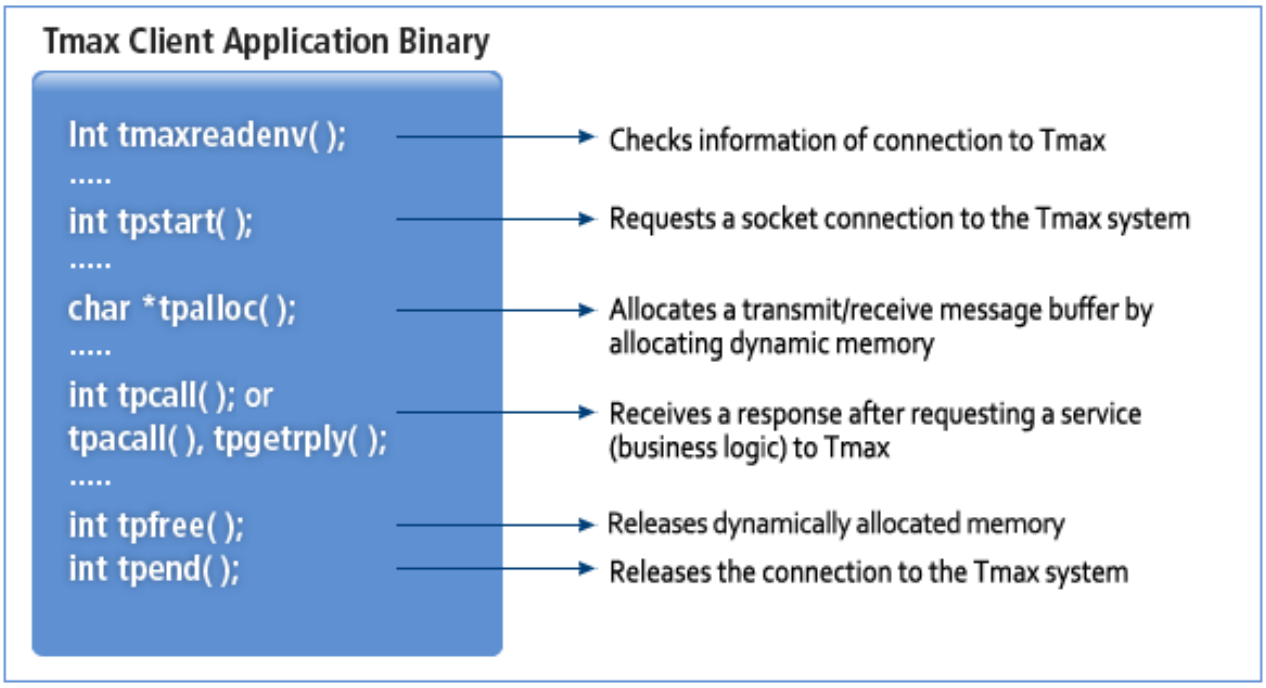
A client program receives input from a user, requests a service from a server, receives a response from the server, and outputs the service response to the user.

#### Client Program Flow

The following is the flow of a client program:

```
main()
{
     Connect to Tmax
     Allocate a buffer for transmission/reception
     Write client business logic (receives a user request and saves it to the transmit message
buffer)
     Request and respond to a service (sends the transmit message buffer to Tmax CLH and
               saves the response data to the reception message buffer through Tmax CLH)
     Write client business logic (displays the response data to the user)
     Release the buffer for transmission/reception
     Release the connection to Tmax
```

```
}
```




Client Program Flow

The following are descriptions for the major functions of a client program.

- Function for Tmax communication environment

| Function | Description |
|---|---|
| tmaxreadenv() | Specifies Tmax environment variables that are referenced when a Tmax client program is executed in a specific file. When writing a program that calls the tmaxreadenv() API, a specific file path and name can be set. The file can then be parsed with a file pointer when the corresponding client program is executed.<br><br>Tmax environment variables, such as TMAX_HOST_ADDR and TMAX_HOST_PORT, are loaded into memory that is allocated when a client program is executed and used to call related API functions. |

- Functions for connecting to a client and a server

| Function | Description |
|---|---|
| tpstart() | Requests a socket connection to CLL and passes the accepted connection to CLH. When this function is called, values set in TMAX_HOST_ADDR and TMAX_HOST_PORT are used. |
| tpend() | Terminates a socket connection. |

- Functions for allocating and releasing a buffer

| Function | Description |
|---|---|
| tpalloc() | Allocates memory used for data transmission/reception between clients and servers via Tmax. The memory allocated is the size required for data transmission/reception.<br><br>Dynamically allocated memory must be explicitly released. |
| tpfree() | Releases memory that was dynamically allocated using tpalloc(). An allocated memory buffer must be returned. If the memory is not returned, garbage (memory leakage) occurs. |

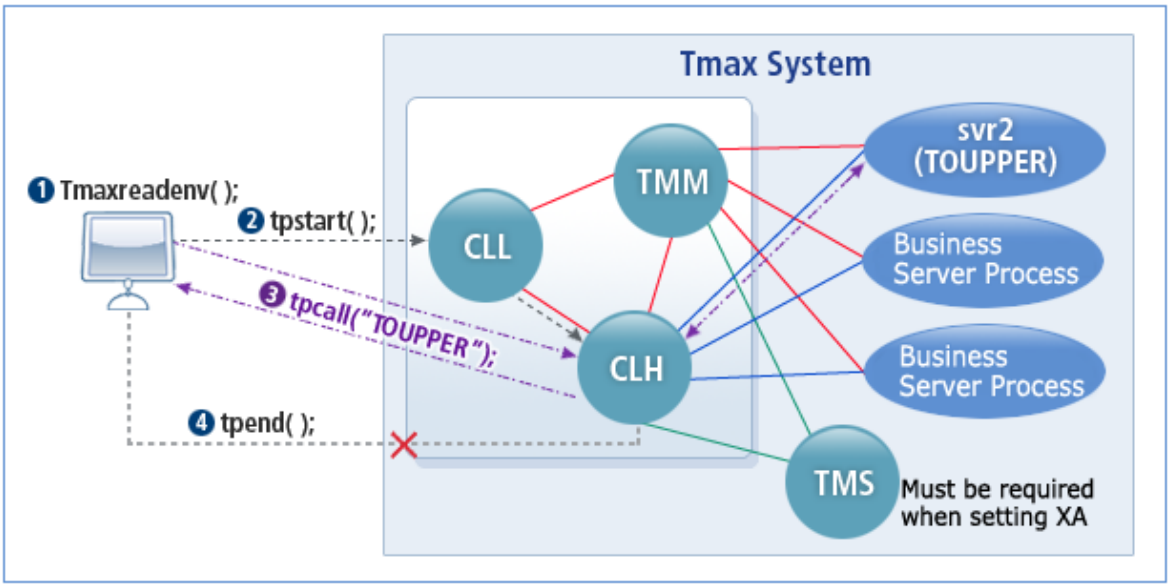
- Function for synchronous communication

| Function | Description |
|---|---|
| tpcall() | Requests a service from CLH and transfers transmission data. CLH checks a service requested by a client and then transfers it to a server application process.<br><br>A client waits until receiving a reply for the service request. |

- Functions for asynchronous communication

| Function | Description |
|---|---|
| tpacall() | Requests a service from CLH and transfers transmission data. CLH checks a service requested by a client and then transfers it to a server application process.<br><br>A client performs logic after calling tpacall() regardless of whether it receives a reply to the service request. |
| tpgetrply() | Requests reply data from CLH through the parameter (cd) value for tpacall().<br><br>If there is reply data to be sent to a corresponding client in CLH, the data is immediately transferred.<br><br>It waits until the reply data is received, or sends a reception error to the client according to flags. |

The following figure shows the process of major functions in a client program.




Process of Functions in a Client Program

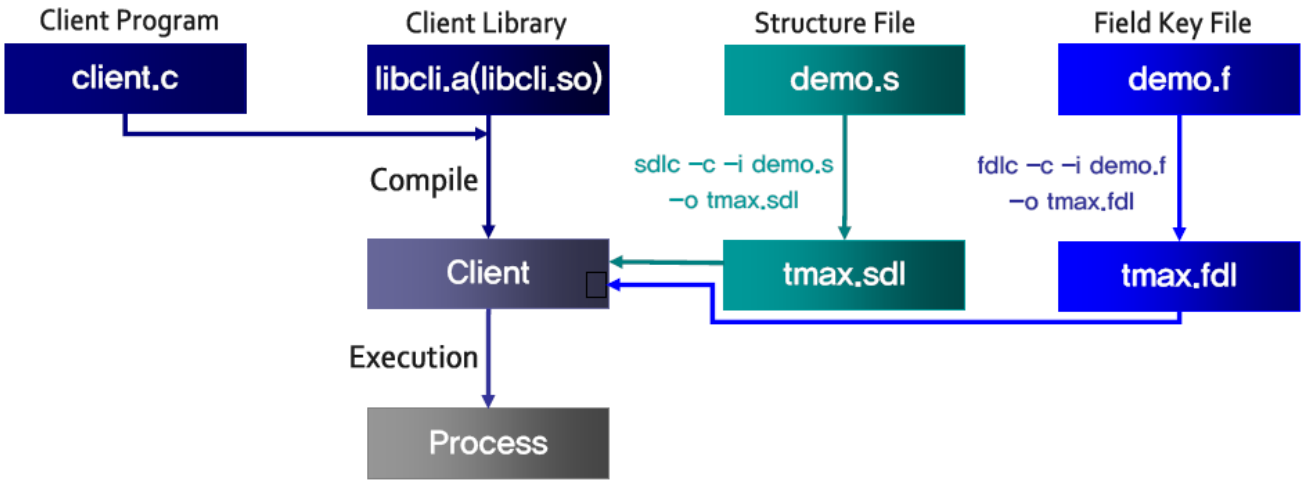
For more information about each function, refer to *Tmax Application Development Guide* and *Tmax Reference Guide*.

### Client Program Configuration

To create an executable file, compile a client program after coding is completed.

To compile a client program, the following must be defined: a client program written by a developer, the Tmax client library, a structure file if a structure buffer is used, and a field key buffer file in which field tables are defined.

The following figure shows the configuration of a client program:




Tmax Client Program Configuration

- Client program

A client program written by a developer.

- Tmax client library (libcli.a / libcli.so)

  The library provided by Tmax. It is the object code of functions used to develop a client program.

- Structure file

  If a client program uses a structure such as STRUCT, X_C_TYPE, or X_COMMON, a structure file that ends with <file_name.s> is needed. The structure file must be compiled with the **sdlc** command in advance. After compilation, the data of a structure is converted to standard communication type data and a binary type file is created. The binary file is used to send and receive data with the standard communication method when a client program is executed.

- Field key file

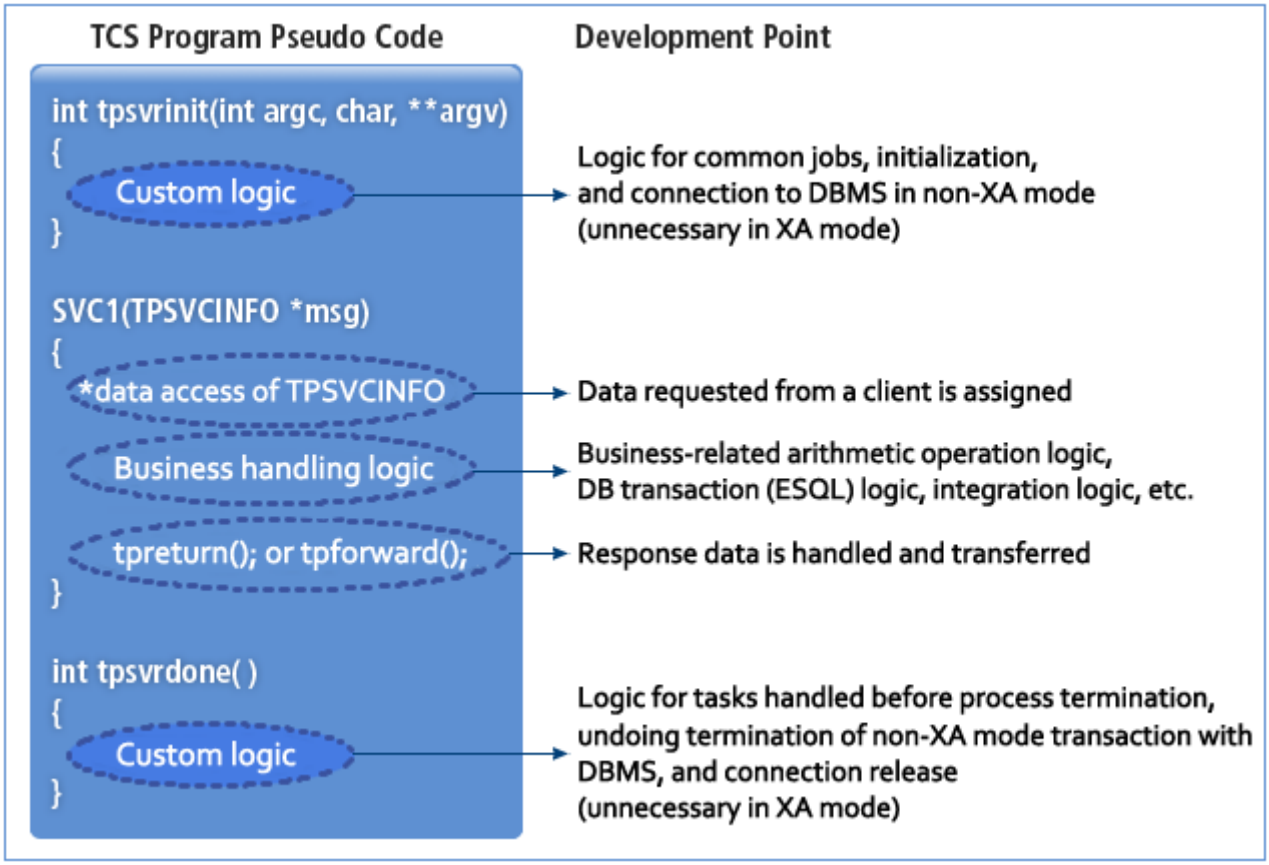
  If a field key structure is used, a field definition file that ends with <.f> is needed. If the file is compiled using the **fdlc** command, a field key buffer file creates <field_key_buffer_name_fdl.h> using the key mapping method and uses it when a program is executed. Unlike an existing structure file, it can reduce resource waste because some user desired fields can be operated and transferred. However, overhead can occur when mapping keys.

## 4.3.2. Server Program

A server program receives a user request, handles it, and returns a reply to the client.

### Server Program Flow

The following is the flow of a server program.




Server Program Flow

```
int tpsvrinit(int argc, char **argv)
{
     Initialize a server (connect to DBMS in the non-XA mode)
}

SVC_NAME(TPSVCINFO *msg)
{
     Handle DB transactions (ESQL)
     Business handling logic
     Send response result to a client
}

int tpsvrdone()
{
     Tasks handled before server termination (release connection to DBMS in non-XA mode)
}

void tpsvctimeout()
{
     Related to SVCTIME properties
     Called from a service timeout handler
}
```

The following are descriptions of major functions of a server program.

- Function for initializing a server

| Function | Description |
| --- | --- |
| tpsvrinit() | Called when a server application process starts.<br><br>The standard input of tpsvrinit() is made through the configuration for the Tmax configuration file. A developer redefines and then uses tpsvrinit(). If it is not redefined, tpsvrinit(), which basic logic of the Tmax server library is applied to, is called. |

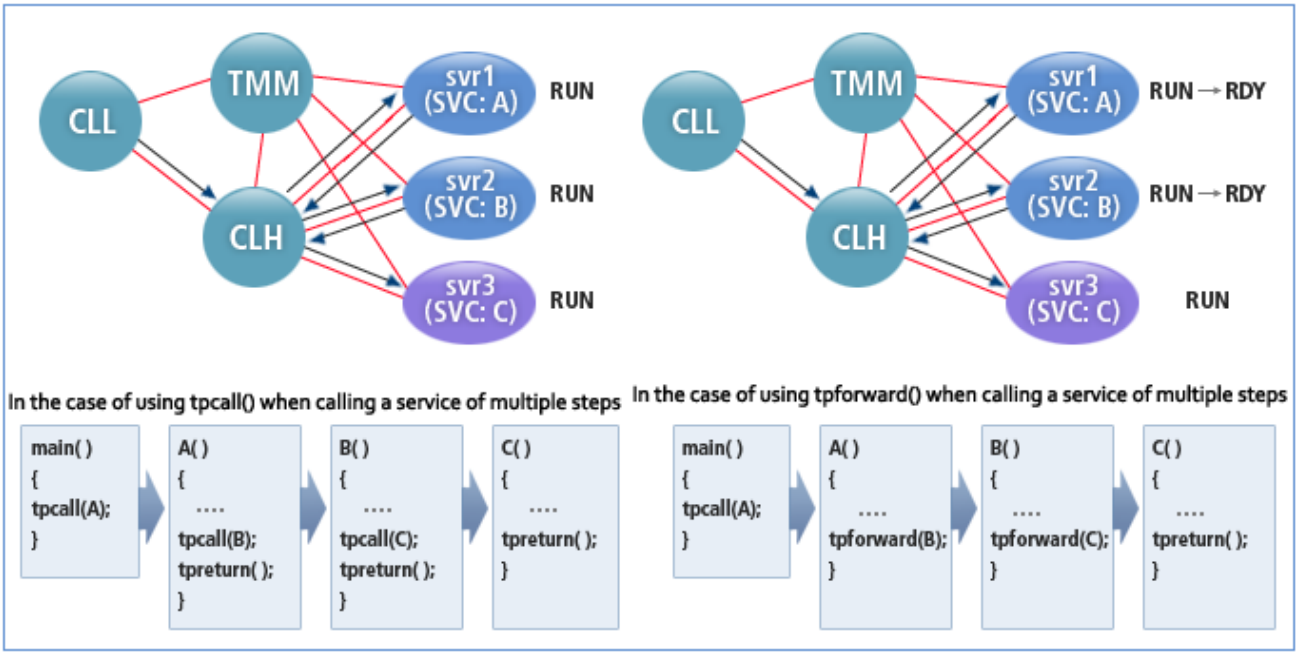
- Function for terminating a server

| Function | Description |
| --- | --- |
| tpsvrdone() | Called just before a server application process is normally terminated.<br><br>A developer redefines and then uses tpsvrdone(). If it is not redefined, tpsvrdone(), which basic logic of the Tmax server library is applied to, is called. |

- Tmax response functions

| Function | Description |
| --- | --- |
| tpreturn() | Notifies CLH of the termination of service logic and transfers reply data. CLH checks the information of a service caller and immediately sends reply data.<br><br>If the dynamically allocated memory pointed to by the reply data buffer is freed, the server application stops any running processes and readies to receive the next reply. |
| tpforward() | Notifies CLH of the termination of service logic and transfers data to SVC. CLH checks whether a server process that includes a service to be transferred is available and immediately sends data.<br><br>If the dynamically allocated memory pointed to by the reply data buffer is freed, the server application stops any running processes and readies to receive the next reply. |

The following figure shows the process of major functions in a server program.

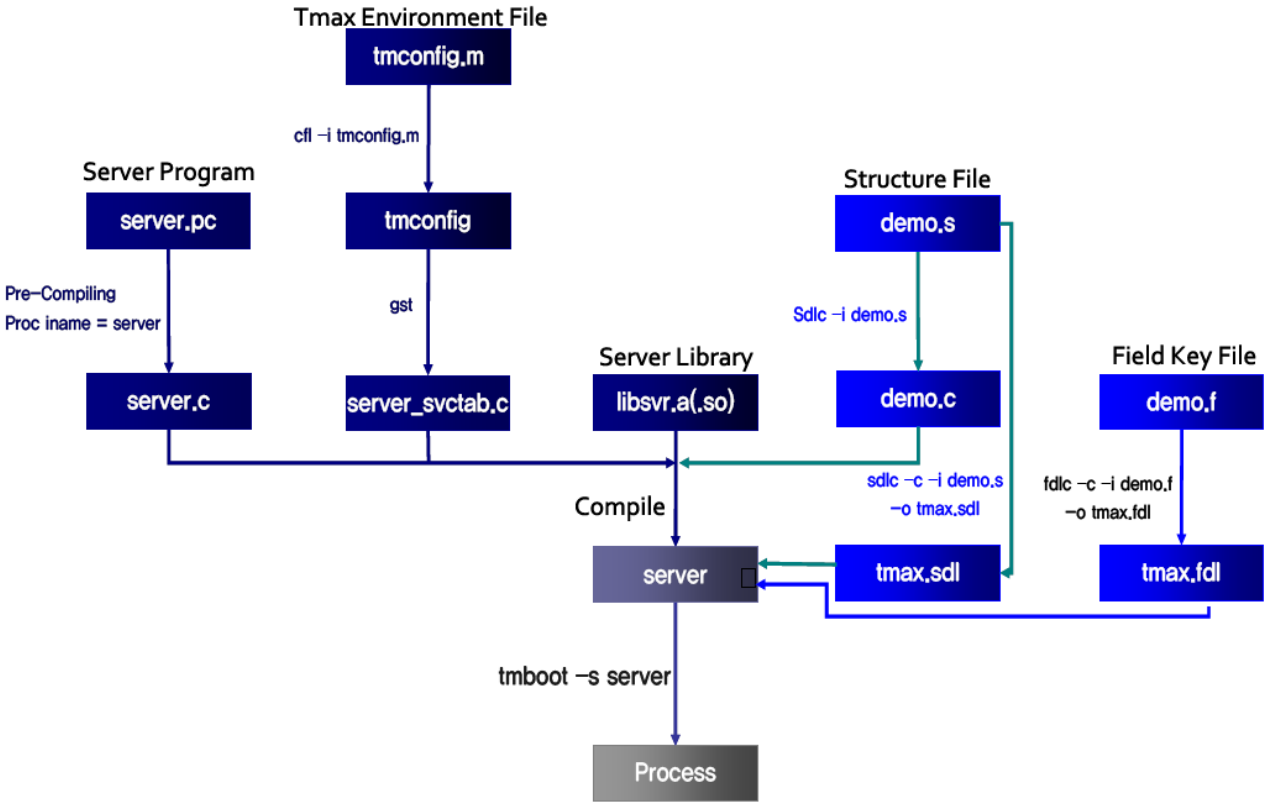



Comparison of tpcall and tpforward

### Server Program Configuration

A server program consists of main(), which is provided by Tmax, and service routines written by a developer.

The following figure shows the configuration of a server program.




Tmax Server Program Configuration

- Server program

  A developer written service routine that handles a client request.

  If a SQL statement is used, it must be precompiled using a tool provided by the corresponding database vendor.

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

  The server library provided by Tmax. It has server main(), tpsvrinit(), tpsvrdone(), and various Tmax functions.

- Service table

  A file that lists the names of services provided by each server. It is created by referring to the Tmax configuration file.

  The service table is used to search for the location of a corresponding service routine in a server when a service is performed. It is provided by a system administrator. For more information, refer to *Tmax Administrator's Guide*.

- Structure binary table (SDLFILE)

  If a structural buffer (STRUCT, X_C_TYPE, and X_COMMON) is used, a structure defined in the <xxx.s> format is needed.

  There are two types of structure files: standard communication type (structure_file_name_sdl.c) and structure header file (structure_file_name_sdl.h). If a structure file is compiled using the **sdlc –c** command, a binary table used to convert the types of structure members to those of standard communication is created.

  If a structure is not used, $TMAXDIR/lib/sdl.o must be compiled with an application server.

- Field buffer binary table (FDLFILE)

  If a field buffer is used, a field buffer file defined in the format <xxxx.f> is needed. If the file is compiled using the **fdlc** command, a binary table, which is used to match field keys and data, and a header file (xxx_fdl.h), which is used to match field keys and field key names, are created. Unlike an existing structure file, the field buffer binary table can reduce resource waste because only the user desired fields can be operated and transferred. However, it can cause an opposite effect if there are too few fields because data values and field key values are managed together.

- Structure and standard buffer conversion/reversion program

  To use a structure buffer in a server program, a structure and a standard buffer conversion/reversion program (xxxx_sdl.c and xxxx_sdl.h) created with the **sdlc** command must be compiled and linked together. If a structure is not used, TMAXDIR/lib/sdl.o must be linked.

# 4.4. System Configuration File

A system configuration file includes information necessary for the Tmax system. The Tmax configuration file sets the configuration of the Tmax system and is written by a Tmax administrator. This file is used to create a service table and start the Tmax system.

The configuration file consists of the following 8 sections.

| Section | Description | Required / Optional |
|---|---|---|
| DOMAIN section | Defines a single independent Tmax system environment. | Required |
| NODE section | Defines an environment related to each node configuring domains. | Required |
| SVRGROUP section | Defines properties related to server groups and databases. | Required |
| SERVER section | Defines properties related to servers. | Required |
| SERVICE section | Defines properties related to services. | Required |
| GATEWAY section | Defines properties related to gateways across domains. | Optional |
| ROUTING section | Defines properties related to data dependent routing. | Optional |
| RQ section | Defines properties related to reliable queues. | Optional |

- Names of sections start with an asterisk (*) (for example, *DOMAIN, *NODE, etc.)
- Names of a section and its child elements must start in the first space of a line.
- Definitions of child elements are separated by a comma (,).

The configuration file is a general text file and compiled with the **cfl** command.

```
cfl -i Tmax Environment File Name
```

The following is an example of the Tmax configuration file. Values inside "< >" must be modified.

```
*DOMAIN
<resrc_name> SHMKEY = <UNIQUE IPCKEY>,
             MAXUSER = <256>,
             TPORTNO = <8999>

*NODE
<uname>       TMAXDIR = <TMAX installed directory>
             APPDIR = <APPLICATION directory>
             PATHDIR = <PATH directory>

*SVRGROUP
<svg_name>    NODENAME = <uname>,
             DBNAME = <ORACLE>,
             OPENINFO = "ORACLE_XA+Acc=P/tmaxsoft/tmaxsoft+SesTm=60"
```

```
*SERVER
<svr_name>   SVGNAME = <svg_name>,
             MIN = <5>,
             MAX = <10>

*SERVICE
<svc_name>   SVRNAME = <svr_name>
```

For more information about each clause, refer to *Tmax Administrator's Guide*.

# 4.5. API

To ease program development, a header file with defined API prototypes must be used. The APIs are implemented in the client/server library. For more information about APIs, refer to *Tmax Application Development Guide* and *Tmax Reference Guide*.

## 4.5.1. Tmax Standard API

### X/Open ATMI

The X/Open Application Transaction Monitor Interface (ATMI) API is provided as a standard of the X/Open DTP model. It can be used as a method of communication between application programs and TP-Monitor.

The functions defined in atmi.h are divided into functions related to a buffer, service request and response, conversational mode, and service termination.

- Functions related to buffer allocation and release

| Function | Description |
|---|---|
| tpalloc() | Allocates a buffer to send and receive data. |
| tprealloc() | Changes the size of a buffer. |
| tpfree() | Frees an allocated buffer. |
| tptypes() | Provides information about the buffer size and type. |

- Functions related to service request and response

| Function | Description |
|---|---|
| tpcall() | Requests a service and waits for the reply. |

| Function | Description |
|---|---|
| tpacall() | Requests a service and then performs other tasks while waiting for the result when tpgetrply() is called. |
| tpcancel() | Cancels the response to a service request. |
| tpgetrply() | Receives the response to tpacall(). |

- Functions related to the conversational mode

| Function | Description |
|---|---|
| tpconnect() | Establishes a connection for sending and receiving messages in the conversational mode. |
| tpdiscon() | Forcefully terminates a connection to a service in the conversational mode. |
| tprecv() | Receives a message in the conversational mode. |
| tpsend() | Sends a message in the conversational mode. |

- Function related to service termination

| Function | Description |
|---|---|
| tpreturn() | Sends a response to a service request to a client and terminates the service routine. |

### X/Open TX API

The X/Open TX API provides communication methods for transactions between application programs and TP-Monitor. The functions defined in tx.h are transaction management functions.

The following is a list of TX APIs.

- Functions related to transactions

| Function | Description |
|---|---|
| tx_begin() | Starts a transaction. |
| tx_commit() | Commits a transaction and saves the result. |
| tx_rollback() | Rolls back a transaction. |
| tx_open() | An internal function that starts a resource manager. |
| tx_close() | An internal function that terminates a resource manager. |
| tx_set_transaction_timeout() | Sets timeout for terminating a transaction. |
| tx_info() | Returns information about a global transaction. |
| tx_set_commit_return() | Sets when a global transaction is permitted. |

| Function | Description |
|---|---|
| tx_set_transaction_control() | Automatically starts the next transaction after the current transaction completes. |

## 4.5.2. Non-standard API

### Tmax ATMI

Tmax ATMI functions defined in tmaxapi.h are divided into functions related to unrequested messages, RQ, error settings, timeout settings, etc. Defined APIs are non-standard interfaces and developed to improve the developer's productivity. It can be used as a method of communication between application programs written by a developer and TP-Monitor.

The following is a list of non-standard APIs defined in tmaxapi.h.

- Functions related to unrequested data

| Function | Description |
|---|---|
| tpbroadcast() | Sends unrequested data to clients registered in a system. |
| tpsetunsol() | Specifies a function for handling unrequested data. |
| tpgetunsol() | Receives unrequested data. |
| tpsetunsol_flag() | Sets a flag for receiving unrequested data. |
| tpchkunsol() | Checks whether unrequested data arrives. |

- Functions related to errors

| Function | Description |
|---|---|
| tpstrerror() | Outputs an error as a string. |
| Userlog() | Logs an error in a buffer. |
| ulogsync() | Saves the contents of 'ulog' in a memory buffer on a disk. |
| UserLog() | Has the features of userlog() and ulogsync(). |
| gettperrno() | Returns an error number that occurs when the Tmax system is called. |
| gettpurcode() | Returns the urcode set by a developer. |
| tperrordetail() | Returns information about an error that occurs when the Tmax system is called. |

- Functions related to socket information

| Function | Description |
|---|---|
| tpgetpeer_ipaddr() | Returns the IP address of a connected client. |

| Function | Description |
|---|---|
| tpgetpeername() | Returns the name of a connected client. |
| tpgetsockname() | Returns the socket name of a connected client. |

- Function related to block timeout

| Function | Description |
|---|---|
| tpset_timeout() | Sets block timeout. |

- Function related to failover

| Function | Description |
|---|---|
| tptobackup() | Establishes a connection to a backup machine. |

- Functions related to connection

| Function | Description |
|---|---|
| tpstart() | Starts a connection to the Tmax system. |
| tpend() | Terminates a connection to the Tmax system. |

- Functions related to RQ

| Function | Description |
|---|---|
| tpenq() | Saves a request of a client in RQ. |
| tpdeq() | Fetches data in RQ. |
| tpqstat() | Requests statistics about data saved in RQ. |
| tpextsvcname() | Requests a service name from data saved in RQ. |

- Functions related to environment variables

| Function | Description |
|---|---|
| tmaxreadenv() | Fetches the environment variables from the file. |
| tpputenv() | Sets the environment variables. |
| tpgetenv() | Returns the values of the environment variables. |

- Functions related to a window operation

| Function | Description |
|---|---|
| WinTmaxStart() | Connects to the Tmax system. |
| WinTmaxEnd() | Disconnects from the Tmax system. |

| Function | Description |
|---|---|
| WinTmaxSetContext() | Specifies a Window handle. |
| WinTmaxSend() | Sends data. |
| WinTmaxAcall() | Asynchronous function for Windows. |
| WinTmaxAcall2() | Asynchronous function that handles data reception with a callback function. |

- Other functions

| Function | Description |
|---|---|
| tpscmt() | Invalidate settings related to transaction control in the configuration file. |
| tpgetlev() | Checks the transaction mode. |
| tpchkauth() | Checks whether certification is required. |
| tpgprio() | Checks the priority of a service request. |
| tpsprio() | Sets the priority of a service request. |
| tpsleep() | Waits for a message for a specified amount of time. |
| tp_sleep() | Waits for data to arrive (in seconds). |
| tp_usleep() | Waits for data to arrive (in microseconds). |
| tpschedule() | Allocates tasks in a queue to UCS so the tasks can be processed. |
| tpuschedule() | Waits a specified amount of time for data in a UCS server process. |
| tpsvrinit() | Initializes a Tmax server process. |
| tpsvrdone() | Calls a termination routine for a Tmax server process. |
| tpsvctimeout() | Shuts downs a UCS server process. |
| tmadmin() | Manages a system as a type of a service call. |

The following is a list of non-standard APIs defined in atmi.h.

- Function related to service termination

| Function | Description |
|---|---|
| tpforward() | Terminates its own service handling and transfers a client request to another service routine. |

- Functions related to client connection

| Function | Description |
|---|---|
| tpstart() | Connects a client application to Tmax. |
| tpend() | Disconnects a client application from Tmax. |

### FDL API

FDL (Field Definition Language) is non-standard API developed to improve developer productivity. FDL is associative-typed data that manages indexes called field keys and data together. APIs related to a FIELD buffer are defined in fbuf.h.

Data is saved in a field key buffer, which is provided by the Tmax system. To operate the buffer, the following functions are provided:

- Functions for mapping field keys

| Function | Description |
|---|---|
| fbget_fldkey() | Returns the field key value of a file name. |
| fbget_fldname() | Returns a field key name. |
| fbget_fldno() | Fetches the field number from a field key. |
| fbget_fldtype() | Fetches the type (integer) from a field key. |
| fbget_strfldtype() | Fetches the pointer value for the type from a field key. |

- Functions related to buffer allocation

| Function | Description |
|---|---|
| fbisfbuf() | Checks whether the specified buffer is a field key buffer. |
| fbinit() | Initializes the memory space allocated to a field key buffer. |
| fbcalcsize() | Calculates the size of a field buffer. |
| fballoc() | Dynamically allocates a field key buffer. |
| fbfree() | Frees a field buffer. |
| fbget_fbsize() | Returns the size of a field key buffer in bytes. |
| fbget_unused() | Checks unused field buffer space. |
| fbget_used() | Returns used field key buffer space in bytes. |
| fbrealloc() | Adjusts the buffer size. |

- Functions for accessing and modifying fields

| Function | Description |
|---|---|
| fbput() | Adds a field key to a field buffer. |
| fbinsert() | Specifies a field key and its location and saves the field value in a field buffer. |
| fbchg_tu() | Moves a specified field buffer before sending data. |
| fbdelete() | Deletes the field data of a buffer. |
| fbdelall() | Deletes all field values. |

| Function | Description |
|---|---|
| fbdelall_tu() | Deletes all field data enumerated in a field key array (fieldkey[]). |
| fbget() | Gets a field value in a buffer. |
| fbgetf() | Gets the field value of a specified field key in a field buffer. |
| fbget_tu() | Gets the value of a specific field key in a specified location. |
| fbnext_tu() | Gets the field values of a specific field key in a field buffer in order. |
| fbgetalloc_tu() | Internally allocates another buffer to save returned data and returns the pointer of the allocated buffer. |
| fbgetval_last_tu() | Gets the occurrence of a specific field key of a field buffer and recent data. |
| fbgetlast_tu() | Gets the most recent entered data of a field specified in a field buffer. |
| fbgetnth() | Searches a specific field value. |
| fbfldcount() | Returns the number of fields included in a specific buffer. |
| fbkeyoccur() | Returns the field number specified in a field key. |
| fbispres() | Checks whether the requested data exists in a field buffer. |
| fbgetval() | Returns the length of the requested data and a pointer to its location. |
| fbgetvall_tu() | Returns the actual value of a field in the long type. |
| fbupdate() | Updates the field value of a field key in a field buffer at a specified location. |
| fbgetlen() | Returns the first occurrence value of a specified field key in a field buffer. |

- Functions for conversion

| Function | Description |
|---|---|
| fbtypecvt() | Converts a data type. |
| fbputt() | Attaches new data and its type into a field buffer. |
| fbget_tut() | Gets the field data of a specified location and specifies the field key type. |
| fbgetalloc_tut() | Converts the returned data type into a defined data type and internally allocates another buffer to save it. |
| fbgetvalt() | Returns a pointer to the returned value. |
| fbgetvali() | Returns the field data of the integer type. |
| fbgetvals() | Returns the field data of the string type. |
| fbgetvals_tu() | Returns the field data of the string type at a specified location. |
| fbgetntht() | Returns a converted value. |

| Function | Description |
|---|---|
| fbchg_tut() | Modifies the field key value from a specific starting point of a field buffer. |

- Function related to buffer operation

| Function | Description |
|---|---|
| fbbufop() | Compares, copies, moves, and modifies the data of two field buffers. |

- I/O-related function

| Function | Description |
|---|---|
| fbbufop_proj() | Modifies a buffer corresponding to a field key. |
| fbread() | Used together with the standard input/output library. Reads a field buffer from a file. |
| fbwrite() | Used together with the standard input/output library. Writes to a file. |
| fbprint() | Standard input/output. Outputs buffer data. |
| fbfprint() | Outputs the available data of a field buffer in a file string. |

- Error-related functions

| Function | Description |
|---|---|
| fbstrerror() | Gets the error message that occurs while operating a field buffer in a string. |
| getfberrno() | Returns an error number. |

- Other functions

| Function | Description |
|---|---|
| fbmake_fldkey() | Automatically creates a new field key without recording in FDLFILE. |
| fbftos() | Moves data saved in a field buffer to a C structure (stname). |
| fbstof() | Moves data saved in a C structure to a field buffer mapped to a structure file. |
| fbsnull() | Checks whether a member variable of a C structure mapped to the occurrence of the field key specified by a field buffer is NULL. |
| fbstelinit() | Initializes the field buffer and member variables of a C structure as NULL. |
| fbstinit() | Initializes the C structure mapped to a field buffer as NULL. |

# 4.6. Error Message

## 4.6.1. X/Open DTP related Error

If an error occurs while using the standard interfaces provided by X/Open DTP and non-standard interfaces provided by the Tmax system, an error value is saved to a global variable called tperrno. By checking tperrno, a developer can handle the error.

The following is a list of tperrno error messages.

| Error Message (tperrno) | Description |
|---|---|
| TPEBADDESC(2) | Occurs when an invalid descriptor is used for an asynchronous or conversational type. |
| TPEBLOCK(3) | Occurs due to a network error. |
| TPEINVAL(4) | Occurs when an invalid argument is entered. |
| TPELIMIT(5) | Occurs when one or more various limit values provided by a system are exceeded. |
| TPENOENT(6) | Occurs when a service is not provided. |
| TPEOS(7) | Occurs when a connection cannot be established due to a system error. |
| TPEPROTO(9) | Occurs due to a protocol error. |
| TPESVCERR(10) | Occurs when a buffer of the Tmax system is damaged because an application program failed. |
| TPESVCFAIL(11) | Occurs due to a level service error of an application program. |
| TPESYSTEM(12) | Occurs due to a Tmax internal error (log message check). |
| TPETIME(13) | Occurs due to transaction timeout (BLOCKTIME). |
| TPETRAN(14) | Occurs when a transaction is cancelled due to a failure. |
| TPGOTSIG(15) | Occurs when a signal occurs. |
| TPEITYPE(17) | Occurs when an unregistered structure type or field key is used. |
| TPEOTYPE(18) | Occurs due to buffer usage or a type error. |
| TPEEVENT(22) | Occurs when an event occurs in the conversational mode. |
| TPEMATCH(23) | Occurs when a proper service does not exist for the tpdeq() function of RQ. |
| TPENOREADY(24) | Occurs when a server process is not ready. |
| TPESECURITY(25) | Occurs due to a security error. |
| TPEQFULL(26) | Occurs when the queue wait time of a server process exceeds timeout. |
| TPEQPURGE(27) | Occurs when an item in a queue is deleted due to a queue purge. |
| TPECLOSE(28) | Occurs when a connection to the Tmax system is released. |

| Error Message (tperrno) | Description |
|---|---|
| TPESVRDOWN(29) | Occurs when a server process is terminated due to an application program error. |
| TPEPRESVC(30) | Occurs when an error occurs while a previous service is processing. |
| TPEMAXNO(31) | Occurs when the number of concurrent users reaches the limit value. |

For more information about errors and handling methods, refer to *Tmax Application Development Guide* and *Tmax Error Message Reference Guide*.

## 4.6.2. FDL-related Error

A FDL interface related error value is saved to a global variable called fberrno. By checking fberrno, a developer can handle the error.

The following is the list of fberror error messages:

| Error Message (fberror) | Description |
|---|---|
| FBEBADFB(3) | Occurs when an improper buffer (not a field key buffer) is used. |
| FBEINVAL(4) | Occurs when an improper argument is used. |
| FBELIMIT(5) | Occurs when one or more various limit values provided by a system are exceeded. |
| FBENOENT(6) | Occurs when a corresponding field key does not exist in a buffer. |
| FBEOS(7) | Occurs when an error occurs in the OS. |
| FBEBADFLD(8) | Occurs when an improper field key is used. |
| FBEPROTO(9) | Occurs due to a protocol error. |
| FBENOSPACE(10) | Occurs when there is insufficient buffer space. |
| FBEMALLOC(11) | Occurs when an error occurs while allocating memory. |
| FBESYSTEM(12) | Occurs when an error occurs in a system. |
| FBETYPE(13) | Occurs when an error occurs due to a type. |
| FBEMATCH(14) | Occurs when there is no matched value. |
| FBEBADSTRUCT(15) | Occurs when an unregistered structure is used. |
| FBEMAXNO(19) | Occurs when an error number that does not exist is used. |

For more information about errors and handling methods, refer to *Tmax FDL Reference Guide* and *Tmax Error Message Reference Guide*.

# 5. Examples

This chapter describes examples of Tmax programs developed in various environments.

## 5.1. Programs for Each Communication Type

This section describes examples of synchronous, asynchronous, and interactive programs.

### 5.1.1. 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);
```

```
}
```

## 5.1.2. 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 TOLOWER service through synchronous communication. The client receives the TOLOWER 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 |
|---|---|
| asyncsvc.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, TOLOWER. |

| 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
asyncsvc      SVGNAME = svg1, MIN = 1, MAX = 5

*SERVICE
TOUPPER        SVRNAME = asyncsvc
TOLOWER        SVRNAME = asyncsvc
```

### 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.

<asyncsvc.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);
}
```

## 5.1.3. 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

TPEVSVCSUCC.

### 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 SUCESS \"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\n", rcvbuf->guarat, rcvbuf->guamon) ;
}
        tpfree(sndbuf);
        tpfree((char *)rcvbuf);
        tpend();
        printf("FINISH\n");
}
```

### Server program

The following is an example.

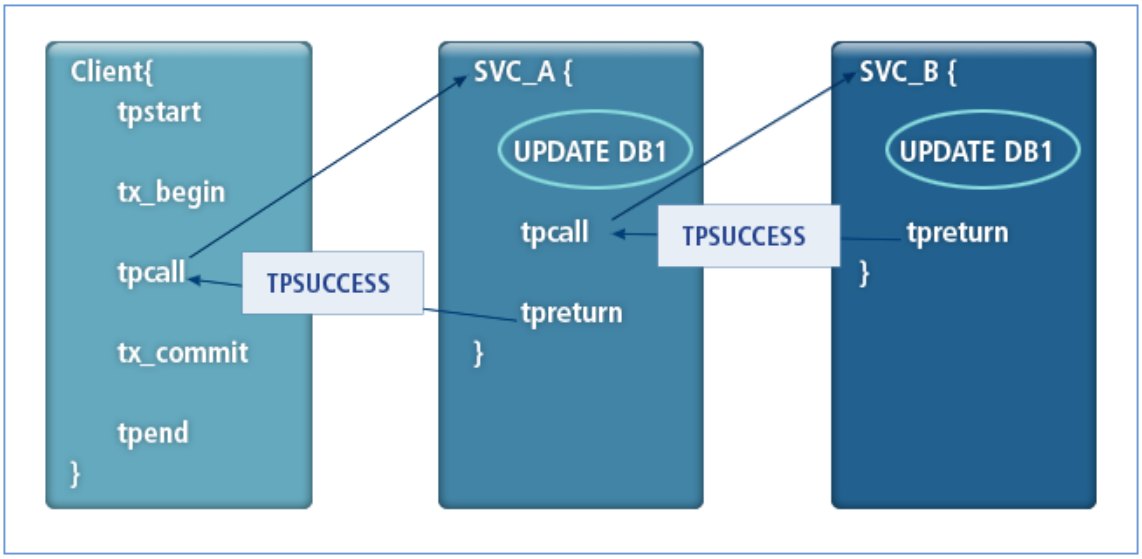
<convsvc.pc>

```
#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 *sndbuf;
EXEC SQL end declare section;
EXEC SQL include sqlca;

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

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

       if ((sndbuf = (struct sel_o *) tpalloc ("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 :sndbuf;

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

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

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

# 5.2. 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
```

```
EOF
```

### 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));
        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);
}
```

# 5.3. Database Programs

This section describes several examples that illustrate the use of Oracle and Informix databases.

## 5.3.1. 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>

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

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

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

        if(tpcall("ORAINS",(char *)sndbuf,0,(char **)&rcvbuf,&rcvlen,TPNOFLAGS)==-1)
        {
             printf("tpcall failed:ORA service, tperrno=%d", tperrno);
             printf("sql code=%d\n", tpurcode);
             tx_rollback();
             tpfree ((char *)sndbuf);
             tpfree ((char *)rcvbuf);
             tpend();
             exit(1);
        }

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

        tpfree ((char *)sndbuf);
        tpfree ((char *)rcvbuf);
        tpend();
}
```

### 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);
}
```

## 5.3.2. 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.<br><br>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 ACOUNT_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 );
}
```

## 5.3.3. 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=tmax
   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.

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

   CFLAGS =-O -I$(INFORMIXDIR)/incl/esql -I$(TMAXDIR)
   # Solaris 32bit, Compaq, Linux: CFLAGS = -O -I$(INFORMIXDIR)/incl/esql
                                          –I$(TMAXDIR)
   # Solaris 64bit: CFLAGS = -xarch=v9 -O -I$(INFORMIXDIR)/incl/esql –I$(TMAXDIR)
   # HP 32bit: CFLAGS = -Ae -O -I$(INFORMIXDIR)/incl/esql –I$(TMAXDIR)
   # HP 64bit: CFLAGS = -Ae +DA2.0W +DD64 +DS2.0 -O -I$(INFORMIXDIR)/incl/esql
                       –I$(TMAXDIR)
   # IBM 32bit: CFLAGS = -q32 –brtl -O -I$(INFORMIXDIR)/incl/esql –I$(TMAXDIR)
   # IBM 64bit: CFLAGS = -q64 –brtl -O -I$(INFORMIXDIR)/incl/esql –I$(TMAXDIR)
   ```

```
INFLIBD = $(INFORMIXDIR)/lib/esql
INFLIBDD = $(INFORMIXDIR)/lib
INFLIBS = -lifsql -lifasf -lifgen -lifos -lifgls -lm -ldl -lcrypt
          $(INFORMIXDIR)/lib/esql/
checkapi.o -lifglx -lifxa

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

#
.SUFFIXES : .ec .s .c .o

.ec.c :
        esql -e $*.ec

#
# server compile
#
all: $(TARGET)

$(TARGET):$(OBJS)
        $(CC) $(CFLAGS) -L$(TMAXLIBDIR) -L$(INFLIBD) -L$(INFLIBDD) -o $(TARGET)
        $(OBJS) $(LIBS) $(INFLIBS)
        mv $(TARGET) $(APPDIR)/.
        rm -f $(OBJS)

$(APOBJS): $(TARGET).ec
        esql -e -I$(TMAXDIR)/usrinc $(TARGET).ec
        $(CC) $(CFLAGS) -c $(TARGET).c

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

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

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

<TMS Makefile>

```
#
TARGET  = info_tms

INFOLIBDIR = ${INFORMIXDIR}/lib
INFOELIBDIR = ${INFORMIXDIR}/esql
INFOLIBD = ${INFORMIXDIR}/lib/esql
INFOLIBS = -lifsql -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$(INFOLIBD)
        -L$(INFOLIBDIR) -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;
```

```
EOF
```

### 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);
}
```

## 5.3.4. 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:$%sLOOP (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%s\n", "ACCOUNT_ID", "PHONE","ADDRESS") ;
                   account_id = transf[j].ACCOUNT_ID;
                   strcpy(phone, transf[j].PHONE);
                   strcpy(address, transf[j].ADDRESS);
                   printf("%-10d %-14s %s\n", account_id, phone, address);
            }/* for2 end */

           /* 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 accound 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);
}
```

## 5.3.5. 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 databse
$ 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
name                           schema    name               Length   Scale Nulls
------------------------------ --------- ------------------ -------- ----- ------
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_T *)NULL) == -1){
        printf("tpstart failed.[%s]\n", tpstrerror(tperrno));
        exit(1);
    }

    if ((sndbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL) {
        printf("sendbuf alloc failed! [%s]\n", tpstrerror(tperrno));
        tpend();
        exit(1);
    }

    if ((rcvbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL) {
        printf("recvbuf alloc failed! [%s]\n", tpstrerror(tperrno));
        tpfree(sndbuf);
        tpend();
        exit(1);
    }

    strcpy(sndbuf, argv[1]);

    ret = tx_begin();
    if (ret < 0) {
        printf("tx_begin is failed.[%s]\n", tpstrerror(tperrno));
        resource_free(sndbuf, rcvbuf);
        exit(1);
    } else
        printf("tx_begin success.\n");

    if (tpcall("XASERVICE2", sndbuf, strlen(sndbuf), &rcvbuf, &rcvlen, 0) == -1){
        printf("Can't send request to service XASERVICE2.[%s]\n", tpstrerror(tperrno));
        ret = tx_rollback();
        if (ret < 0)
            printf("tx_rollback is failed. [%s]\n", tpstrerror(tperrno));

        resource_free(&sndbuf, &rcvbuf);
```

```
        exit(1);
    } else
        printf("XASERVCE2 success.\n");

    ret = tx_commit();
    if (ret < 0) {
        printf("tx_commit is failed. [%s]\n", tpstrerror(tperrno));
        ret = tx_rollback();
        if (ret < 0)
            printf("tx_rollback is failed. [%s]\n", tpstrerror(tperrno));
    } else
        printf("tx_commit success.\n");
    resource_free(sndbuf, rcvbuf);

    return 0;
}

resource_free(char* sndbuf, char *rcvbuf)
{
    if (rcvbuf != NULL)
        tpfree((char*)rcvbuf);

    if (sndbuf != NULL)
       tpfree((char*)sndbuf);

   tpend();
}
```

### 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

#OBJS    = $(APOBJS) $(APOBJS2) $(SVCTOBJ)
OBJS    = $(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) $(TMAXAPPDIR)
         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 $(TMAXAPPDIR)/$(TARGET) $(TARGET).bnd
```

### TMS Makefile

The following is an example.

```
# TMS Makefile for DB2
# Linux 64bit

TARGET  = tms_db2
APOBJ   = dumy.o

APPDIR  = $(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 =
```

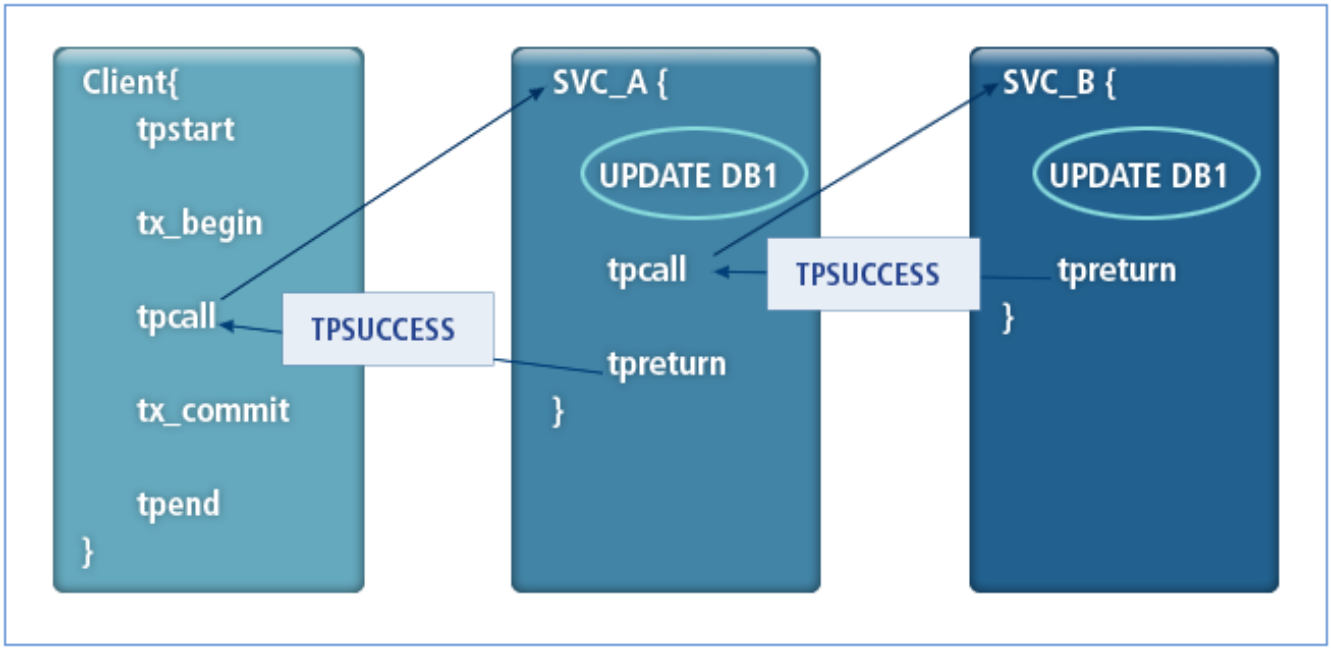
```
all: $(TARGET)

$(TARGET): $(APOBJ)
	$(CC) $(CFLAGS) $(LDFLAGS) -o $(TARGET) -L$(TMAXLIBD) $(TMAXLIBS) $(APOBJ) -L$(DB2LIBDIR)
$(DB2LIB) $(SYSLIBS)
	mv $(TARGET) $(APPDIR)/.

$(APOBJ):
	$(CC) $(CFLAGS) -c dumy.c
#
clean:
	-rm -f *.o core $(APPDIR)/$(TARGET)
```

## 5.4. 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.

### 5.4.1. 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 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 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.

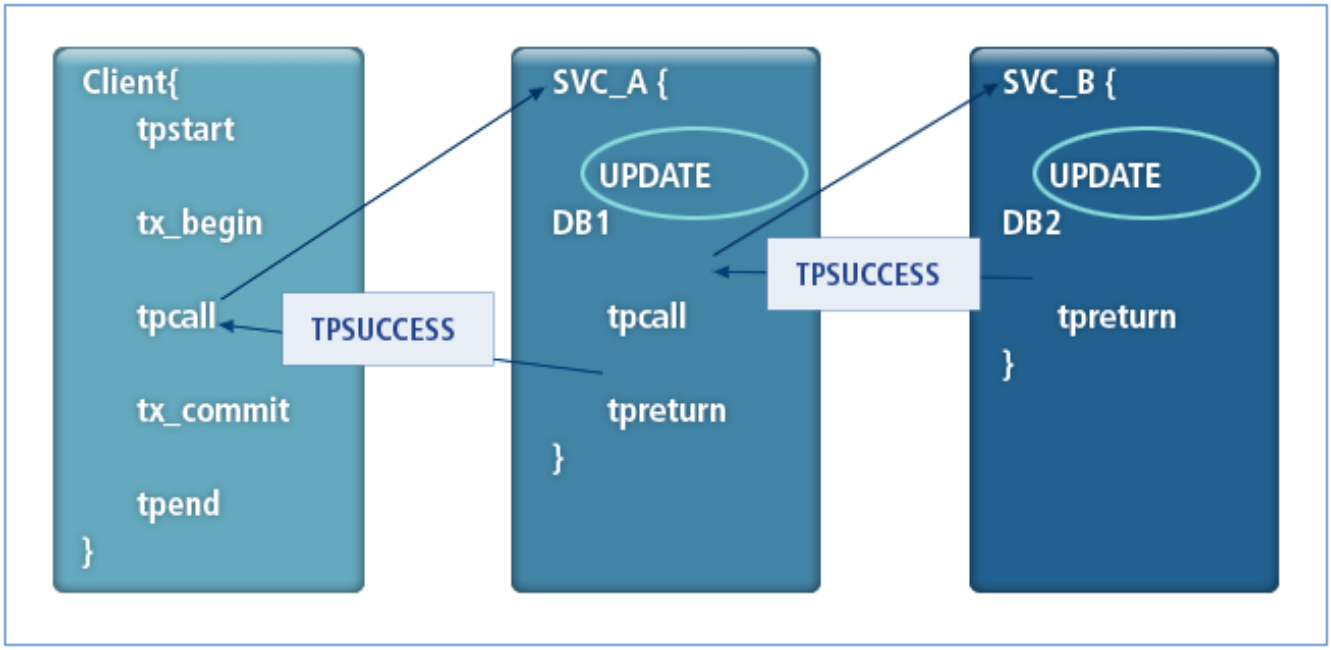
< insert.pc>

```
#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 *)tpalloc("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);
}
```

## 5.4.2. 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

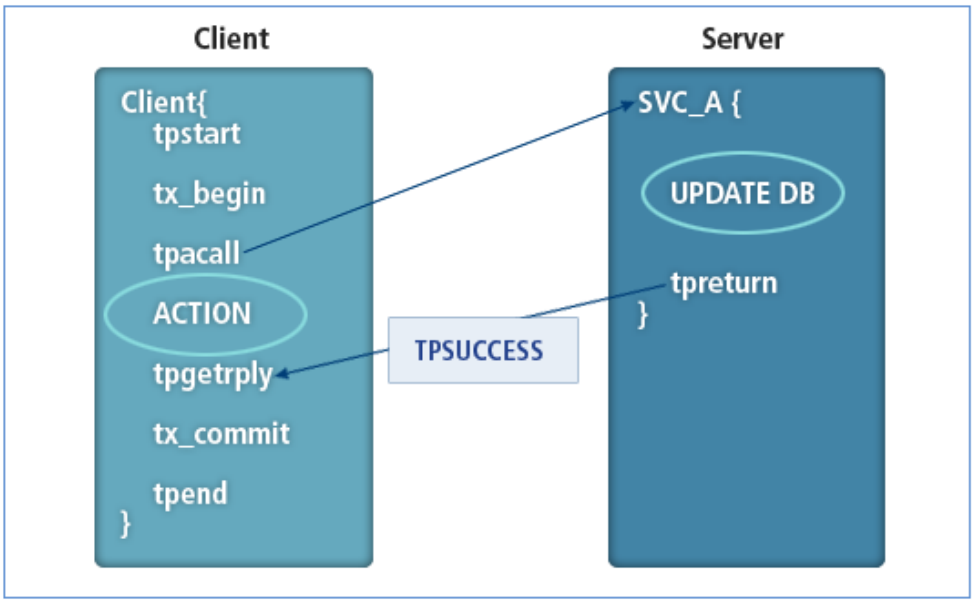
The following is an example.

<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
```

### 5.4.3. 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 acnt_id, n, cd, 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);

        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.

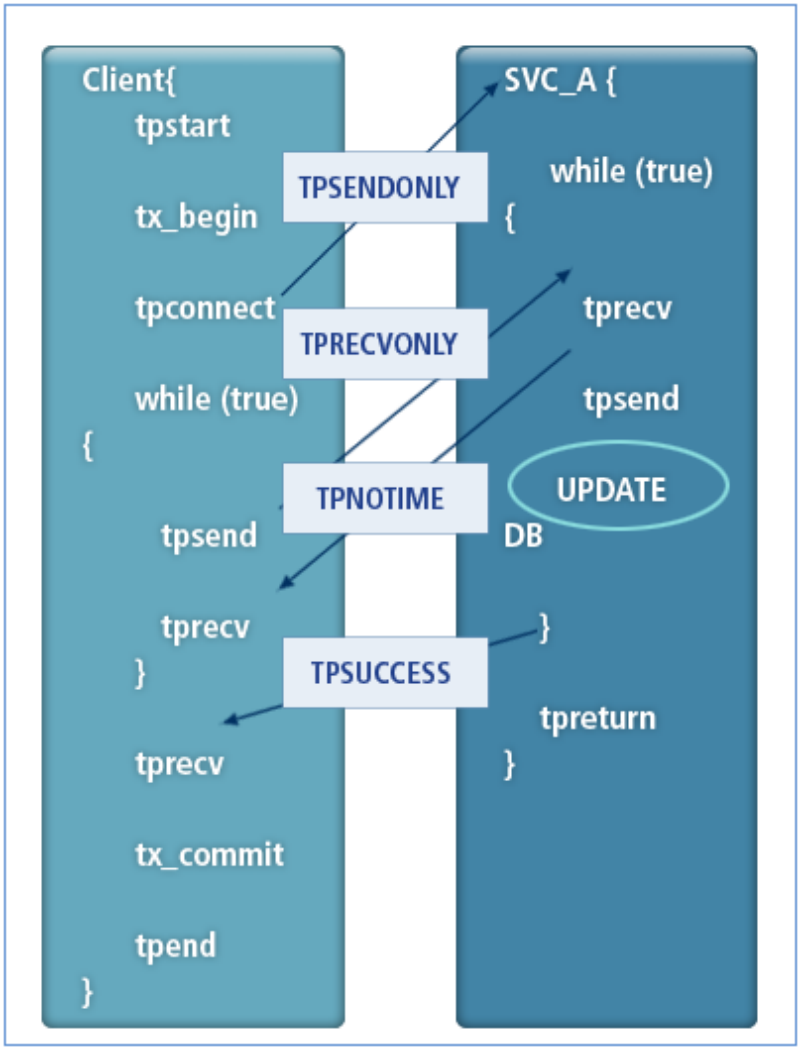
<update.pc>

```
#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);
}
```

### 5.4.4. 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_SENDONLY) {
            fprintf(stderr, "tprecv fail revent = 0x%08x\n", revent);
            tx_rollback();
            tpend();
            exit(1);
        }
        printf("tprecv ok\n");
        sndbuf->account_id++;

        if (revent != TPEV_SENDONLY)
        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_SENDONLY) {
                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);
        }
}
```

# 5.5. 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.

## 5.5.1. 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 tmadmin.

  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:<br>• TIP_TMM<br>• TIP_CLH<br>• TIP_TMS<br>• TIP_SVR |
| TIP_FLAGS (int) | Flags. Options are:<br>• TIP_VFLAG<br>• TIP_GFLGA<br>• 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:<br>• compact<br>• basic<br>• detail<br>• debug1<br>• debug2<br>• debug3<br>• debug4 |
| TIP_ERROR (int) | Error value. Options are:<br>• TIPESVCFAIL: corresponding service was not handled successfully<br>• TIPEOS: memory allocation failed<br>• 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 tmadmin.

  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:<br>• TIP_PFLAG<br>• TIP_VFLAG<br>• TIP_GFLAG<br>• 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:<br>• TIPESVCFAIL: corresponding service was not successfully handled<br>• TIPEOS: memory allocation is failed<br>• TIPEBADFLD: value of TIP_MODULE is not set |

The following must be included in requests to TIPSVC.

- 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. |

## 5.5.2. 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 usrname property of the TPSTART_T structure when accessing the Tmax system. usrname is set only when TIP_SEGMENT is ADMINISTRATION. If usrname is set incorrectly, the TIPEAUTH error is set in the TIP_ERROR field.

```
strcpy(tpinfo->usrname, ".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. |

## 5.5.3. 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_CLICHKINT  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_CMPRPORT    15      string
TIP_CMPRSIZE    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_LIVECTIME   50      string
TIP_NODE_TMMLOGLVL      51      string
TIP_NODE_CLHLOGLVL      52      string
TIP_NODE_TMSLOGLVL      53      string
TIP_NODE_LOGLVL         54      string
TIP_NODE_MAXTHREAD      55      int
TIP_EXTPORT             56      int
TIP_EXTCLHPORT          57      int
TIP_MSGSIZEWARN 58      int
TIP_MSGSIZEMAX  59      int


#
#       SVRGROUP section fields
#
# name          number  type    flags   comments
*base 16000300
#TIP_NAME       0       string
#TIP_NODENAME   1       string
TIP_SVGTYPE     2       string
#TIP_PRODNAME   3       string
TIP_COUSIN      4       string
TIP_BACKUP      5       string
TIP_LOAD        6       int
#TIP_APPDIR     7       string
#TIP_ULOGDIR    8       string
TIP_DBNAME      9       string
TIP_OPENINFO    10      string
TIP_CLOSEINFO   11      string
TIP_MINTMS      12      int
#TIP_MAXTMS     13      int
TIP_TMSNAME     14      string
#TIP_SECURITY   15      string
#TIP_OWNER      16      string
#TIP_ENVFILE    17      string
#TIP_CPC                18      int
TIP_XAOPTION            19      string
TIP_SVG_TMSTYPE         20      string
TIP_SVG_TMSOPT          21      string
TIP_SVG_TMSTHREADS      22      int
```

```
TIP_SVG_TMSLOGLVL       23      string
TIP_SVG_LOGLVL          24      string
TIP_NODENAME            25      string

#
#       SERVER section fields
#
# name          number  type    flags   comments
*base 16000350
#TIP_NAME       0       string
TIP_SVGNAME     1       string
#TIP_NODENAME   2       string
TIP_CLOPT       3       string
TIP_SEQ         4       int
TIP_MIN         5       int
TIP_MAX         6       int
#TIP_ULOGDIR    7       string
TIP_CONV        8       int
TIP_MAXQCOUNT   9       int
TIP_ASQCOUNT    10      int
TIP_MAXRSTART   11      int
TIP_GPERIOD     12      int
TIP_RESTART     13      int
TIP_SVRTYPE     14      string
#TIP_CPC                15      int
TIP_SCHEDULE    16      int
#TIP_MINTHR     17      int
#TIP_MAXTHR     18      int
TIP_TARGET      19      string
TIP_DEPEND      20      string
TIP_CASCADE     21      int
TIP_PROCNAME    22      string
TIP_LIFESPAN    23      string
TIP_DDRI        24      string
TIP_CURSVR      25      int
TIP_SVGNO       26      int
TIP_SVR_LOGLVL  27      string


#
#       SERVICE section fields
#
# name          number  type    flags   comments
*base 16000400
#TIP_NAME       0       string
TIP_SVRNAME     1       string
TIP_PRI         2       int
TIP_SVCTIME     3       int
TIP_ROUTING     4       string
TIP_EXPORT      5       int
TIP_AUTOTRAN    6       int

#
#       ROUTING section fields
#
# name          number  type    flags   comments
*base 16000425
#TIP_NAME       0       string
TIP_FLDTYPE     1       string
```

```
TIP_RANGES      2       string
TIP_SUBTYPE     3       string
TIP_ELEMENT     4       string
TIP_BUFTYPE     5       string
TIP_OFFSET      6       int
TIP_FLDLEN      7       int
#TIP_FLDOFFSET  8       int


#
#       RQ section fields
#
# name          number  type    flags   comments
*base 16000450
#TIP_NAME       0       string
#TIP_SVGNAME    1       string
TIP_PRESVC      2       string
TIP_QSIZE       3       int
TIP_FILEPATH    4       string
TIP_BOOT        5       string
TIP_FSYNC       6       int
TIP_BUFFERING   7       int
#TIP_ENQSVC     8       int
#TIP_FAILINTERVAL 9     int
#TIP_FAILRETRY  10      int
#TIP_FAILSVC    11      string
#TIP_AFTERSVC   12      string


#
#       GATEWAY section fields
#
# name                  number  type    flags   comments
*base 16000500
#TIP_NAME               0       string
TIP_GWTYPE              1       string
TIP_PORTNO              2       int
#TIP_CPC                        3       int
TIP_RGWADDR             4       string
TIP_RGWPORTNO           5       int
#TIP_BACKUP             6       string
#TIP_NODENAME           7       string
TIP_KEY                 8       string
TIP_BACKUP_RGWADDR      9       string
TIP_BACKUP_RGWPORTNO    10      int
TIP_TIMEOUT             11      int
TIP_DIRECTION           12      string
TIP_MAXINRGW            13      int
TIP_GWOWNER             15      string
TIP_RGWOWNER            16      string
TIP_RGWPASSWD           17      string
TIP_PTIMEOUT            18      int
TIP_PTIMEINT            19      int

#
#       FUNCTION section fields
#
# name          number  type    flags   comments
*base 16000550
```

```
#TIP_NAME       0       string
#TIP_SVRNAME    1       string
TIP_FQSTART     2       int
TIP_FQEND       3       int
TIP_ENTRY       4       string


#
#       STATISTICS segment fields
#
# name          number  type    flags   comments
*base 16000600
#TIP_NAME       0       string
TIP_STATUS      1       string
TIP_STIME       2       string
TIP_TTIME       3       int
TIP_SVC_STIME   4       int
TIP_COUNT       5       int
#TIP_NO         6       int
TIP_NUM_FREE    7       int
TIP_NUM_REPLY   8       int
TIP_NUM_FAIL    9       int
TIP_NUM_REQ     10      int
TIP_ENQ_REQS    11      int
TIP_DEQ_REQS    12      int
TIP_ENQ_REPLYS  13      int
TIP_DEQ_REPLYS  14      int
TIP_CLHNO       15      int
TIP_SVR_NAME    16      string
TIP_SVC_NAME    17      string
TIP_AVERAGE     18      float
TIP_QCOUNT      19      int
TIP_CQCOUNT     20      int
TIP_QAVERAGE    21      float
TIP_MINTIME     22      float
TIP_MAXTIME     23      float
TIP_FAIL_COUNT  24      int
TIP_ERROR_COUNT 25      int
TIP_PID         26      int
TIP_TOTAL_COUNT 27      int
TIP_TOTAL_SVCFAIL_COUNT 28      int
TIP_TOTAL_ERROR_COUNT   29      int
TIP_TOTAL_AVG   30      float
TIP_TOTAL_RUNNING_COUNT 31      int
TIP_TMS_NAME    32      string
TIP_SVG_NAME    33      string
TIP_SPRI        34      int
TIP_TI_THRI     35      int
TIP_TI_AVG      36      float
TIP_TI_XID      37      string
TIP_TI_XA_STATUS        38      string
TIP_GW_NAME     39      string
TIP_GW_NO       40      int
TIP_GW_HOSTN    41      string
TIP_GW_CTYPE    42      string
TIP_GW_CTYPE2   43      string
TIP_GW_IPADDR   44      string
TIP_GW_PORT     45      int
TIP_GW_STATUS   46      string
```

```
#
#       ADMIN segment fields
#
# name          number  type    flags   comments
*base 16000650
TIP_IPADDR      0       string
TIP_USRNAME     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
```

## 5.5.4. 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->usrname, ".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("TIPSVC", (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
```

## 5.5.5. 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("TIPSVC", (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
```

## 5.5.6. 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("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();
```

### 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, forkcnt;

      if (argc < 6) {
            printf("Usage: %s svgname svrname nodename  [chlogmodule] [flags]
                                                       [loglvl]\n", argv[0]);
            printf("chlogmodule 1: TIP_TMM, 2: TIP_CLH, 4: TIP_TMS, 8: TIP_SVR\n");
            printf("flags 1: NFLAGS, 2: GFLAGS, 3: VFLAGS\n");
            printf("loglvl : 1: compact, 2: basic, 3: detail, 4: debug1,
                    5: debug2, 6: debug3, 7: debug4\n");
            exit(1);
      }
```

```
        n = tmaxreadenv("tmax.env", "TMAX");
        if (n < 0) {
           fprintf(stderr, "can't read env\n");
           exit(1);
        }

        tpinfo = (TPSTART_T *)tpalloc("TPSTART", NULL, 0);
        if (tpinfo == NULL) {
            printf("tpalloc fail tperrno = %d\n", tperrno);
            exit(1);
        }
        strcpy(tpinfo->usrname, ".tpadmin");
        strcpy(svgname, argv[1]);
        strcpy(svrname, argv[2]);
        strncpy(nodename, argv[3], NODE_NAME_SIZE - 1);

        if (tpstart((TPSTART_T *)tpinfo) == -1){
            printf("tpstart fail tperrno = %d\n", tperrno);
            exit(1);
        }

        if ((sndbuf = (FBUF *)tpalloc("FIELD", NULL, 0)) == NULL) {
            printf("tpalloc failed! errno = %d\n", tperrno);
            tpend();
            exit(1);
        }

        if ((rcvbuf = (FBUF *)tpalloc("FIELD", NULL, 0)) == NULL) {
            printf("tpalloc failed! errno = %d\n", tperrno);
            tpend();
            exit(1);
        }

        ret = case_chlog(argc, argv, sndbuf);
        n = fbput(sndbuf, TIP_OPERATION, "GET", 0);
        n = fbput(sndbuf, TIP_SEGMENT, "ADMINISTRATION", 0);
        n = fbput(sndbuf, TIP_CMD, "CHLOG", 0);
        n = fbput(sndbuf, TIP_NODENAME, nodename, 0);
        n = fbput(sndbuf, TIP_SVGNAME, svgname , 0);
        n = fbput(sndbuf, TIP_SVRNAME, svrname, 0);

        n=tpcall("TIPSVC", (char *)sndbuf, 0, (char **)&rcvbuf, &rcvlen, TPNOFLAGS);
        if (n < 0) {
             printf("tpcall failed! errno = %d[%s]\n", tperrno,
                     tpstrerror(tperrno));
             fbprint(rcvbuf);
             tpfree((char *)sndbuf);
             tpfree((char *)rcvbuf);
             tpend();
             exit(1);
        }

        fbprint(rcvbuf);
        tpfree((char *)sndbuf);
        tpfree((char *)rcvbuf);
        tpend();
}
```

```
int case_chlog(int argc2, char *argv2[], FBUF *sndbuf)
{
        int  chlogmdl, loglvl, flags, n=0;
        char  cloglvl[TMAX_NAME_SIZE];
        const int true = 1, false = 0;

        chlogmdl = atoi(argv2[4]);
        if( (chlogmdl != 1) && (chlogmdl != 2) && (chlogmdl != 4) &&
            (chlogmdl != 8)
        {
                printf("couldn't support such a chlogmdl\n");
                exit(1);
        }

        flags = atoi(argv2[5]);
        if( (flags != NFLAG) && (flags != GFLAG) && (flags != VFLAG) )
        {
                printf("couldn't support such a flags\n");
                exit(1);
        }

        loglvl = atoi(argv2[6]);
        if( (loglvl < 1) || (loglvl > 7) )
        {
                printf("couldn't support such a loglvl\n");
                exit(1);
        }

        switch (loglvl)
        {
                case 1 :
                        strcpy(cloglvl, "compact");
                        break;
                case 2 :
                        strcpy(cloglvl, "basic");
                        break;
                case 3 :
                        strcpy(cloglvl, "detail");
                        break;
                case 4 :
                        strcpy(cloglvl, "debug1");
                        break;
                case 5 :
                        strcpy(cloglvl, "debug2");
                        break;
                case 6 :
                        strcpy(cloglvl, "debug3");
                        break;
                case 7 :
                        strcpy(cloglvl, "debug4");
                        break;
        }
        n = fbput(sndbuf, TIP_MODULE, (char *)&chlogmdl, 0);
        n = fbput(sndbuf, TIP_FLAGS, (char *)&flags , 0);
        n = fbput(sndbuf, TIP_LOGLVL, cloglvl , 0);
        return 1;
}
```

**[Result] (TIP_SVR, => DEBUG4)**

```
$ client  xa1 svr23_stat_ins $HOSTNAME 8 1024 7
fkey = 217326601, fname = TIP_ERROR, type = int, value = 0
>>> tmadmin (cfg -v)

         loglvl = DEBUG4
```

# 5.6. 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);
            tpreturn(TPFAIL, 0, 0, 0, 0);
        }
        strcat(rcvbuf, "_Success");
        tpreturn(TPSUCCESS,0,(char *)rcvbuf, 0,0);
}

SVC15004_3(TPSVCINFO *msg)
{
        int     i;
        char    *rcvbuf;
        long    rcvlen;

        if ((rcvbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL)
             printf("rcvbuf tpalloc fail \n");

        if (tpcall("SVC15004_4", msg->data, 0, &rcvbuf, &rcvlen, 0) == -1)
        {
            printf("tpcall fail [%s]\n", tpstrerror(tperrno));
            tpfree((char *)rcvbuf);
            tpreturn(TPFAIL, 0, 0, 0, 0);
         }

        strcat(rcvbuf, "_Success");
        tpreturn(TPSUCCESS,0,(char *)rcvbuf, 0,0);
}
```

### 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
OBJS    = $(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)
```

# 6. Guide Organization

This chapter presents a list of all Tmax guides and describes each guide.

## 6.1. Overview

A user who is unfamiliar with Tmax can find it difficult to find a desired guide due the sheer number of available product guides.

For example, to develop a program using a 4GL language, such as Delphi, in the environment where Tmax is used, a developer must know the flow of the corresponding program and the APIs to be used. But, it is difficult to determine a guide to refer to. For the user's convenience, this chapter shows a list of all Tmax guides and describes each of them and their relationship.

It is recommended for a user who uses Tmax for the first time to carefully read this chapter. This chapter is helpful in understanding the overall organization of guides even though it is not directly related to actual product usage.

## 6.2. Guide Organization and Description

The list of all Tmax guides is as follows:

| No | Guide Name |
|---|---|
| 1 | Tmax Getting Started Guide |
| 2 | Tmax Installation Guide |
| 3 | Tmax Administrator's Guide |
| 4 | Tmax Application Development Guide |
| 5 | Tmax Error Message Reference Guide |
| 6 | Tmax Reference Guide |
| 7 | Tmax JTmaxServer Guide |
| 8 | Tmax FDL Reference Guide |
| 9 | Tmax Host-link Guide (SNA LU 0, SNA LU 6.2) |
| 10 | Tmax WebtAsync User Guide |
| 11 | Tmax WebtJCA User Guide |
| 12 | Tmax JTC User Guide |
| 13 | Tmax HMS User Guide |
| 14 | Tmax WebT User Guide |
| 15 | Tmax COBOL User Guide |
| 16 | Tmax Gateway Guide (SERIAL) |

| No | Guide Name |
|---|---|
| 17 | Tmax Gateway Guide (TCP/IP) |
| 18 | Tmax Gateway Guide (TCP/IP Thread) |
| 19 | Tmax Gateway Guide (TCP/IP Service) |
| 20 | Tmax Gateway Guide (WebService) |
| 21 | Tmax Gateway Guide (X.25) |
| 22 | Tmax Programming Guide (4GL) |
| 23 | Tmax Programming Guide (Dynamic Library) |
| 24 | Tmax Programming Guide (RCA) |
| 25 | Tmax Programming Guide (RPC) |
| 26 | Tmax Programming Guide (RQ) |
| 27 | Tmax Programming Guide (SQ) |
| 28 | Tmax Programming Guide (UCS) |
| 29 | Tmax Programming Guide (MultipleRM) |
| 30 | Tmax XA Library & XA Gateway Guide |
| 31 | Tmax WebAdmin User Guide |
| 32 | TmaxGrid User Guide |
| 33 | Tmax TChache Guide |
| 34 | Tmax Deployment User Guide |

Tmax provides 34 guides. Refer to the following to search specific contents:

- Tmax Getting Started Guide

  This guide. It describes the basic concepts of Tmax and its configuration.

- Tmax Installation Guide

  Describes how to install Tmax.

- Tmax Administrator's Guide

  Describes the environment setting file for using Tmax and how to operate the system.

  - Descriptions of environment variables and how to set them
  - How to set an configuration file and the sections for each environment
  - How to start and terminate a system
  - How to use the administration tool, search for information, and operate and manage the Tmax system

- Tmax Application Development Guide

Describes basic information about program development for users who want to develop programs using Tmax.

  - Concepts, configuration, and features of applications that use Tmax
  - Flows, characteristics, development environments, and compilation methods of client programs
  - Flows, characteristics, development environments, and compilation methods of server programs
  - Communication types of clients and servers
  - Buffer types and management methods used for the communication types of clients and servers
  - Characteristics and management methods of transactions
  - Flows, implementation methods, and examples of programs in multi-thread environments
  - The security system provided by Tmax and its characteristics for each step
  - APIs that can be used in a client and their usage
  - APIs that can be used in a server and their usage
  - Examples of applications in various environments
  - How to handle and debug errors that occur in Tmax
  - Examples of a Tmax environment setting file and Makefile

- Tmax Error Message Reference Guide

  Describes errors that may occur while using Tmax and how to handle those errors.

  - The structure of a Tmax error message
  - Descriptions of common error messages and how to handle them
  - Descriptions of error messages by module and how to handle them

- Tmax Reference Guide

  Describes the concepts of commands, their usage, and examples for users who want to develop an application program using Tmax. It also explains functions used for connection and communication between clients and servers, their usage, and examples.

  - Concepts, usage, and examples of commands
  - Descriptions, usage, and examples of functions
  - Descriptions of errors and how to handle them

- Tmax JTmaxServer Guide

  Describes methods for configuring and setting an environment, APIs, and examples for developers who want to use JTmaxServer.

- Tmax FDL Reference Guide

Describes how to utilize all features of FDL with definitions of Tmax FDL functions and examples.

  - Concepts of FDL, a description of a field buffer, and FDL table creation
  - Usage and examples of FDL functions

- Tmax Host-link Guide (SNA LU0, SNA LU6.2)

  Describes how to develop Host-link, which links Tmax and hosts.

  - Concepts, architecture, and features of Host-link and how to start and terminate it
  - Management methods for INBOUND and OUTBOUND sessions
  - INBOUND and OUTBOUND services
  - Problems that may occur while using Host-link and how to solve them
  - Environment setting methods, status monitoring APIs, error codes, user functions, and a list of files used in Host-link

- Tmax WebtAsync User Guide

  Describes WebTAsync the Java library for asynchronous inbound/outbound communication with the Tmax Async Java Gateway. It also explains the concept of WebTAsyn, inbound/outbound, and provides examples.

- Tmax JCA User Guide

  Describes how to develop a program using WebTJCA for developers who use Tmax WebTJCA.

  - Concepts of JCA, WebTJCA, and JCA's architecture
  - WebTJCA APIs used for CCI, transactions, logging, and asynchronous and conversational communication
  - WebTJCA inbound communication
  - Examples of using WebTJCA in WebLogic and JEUS6

- Tmax JTC User Guide

  Describes how to develop a program using JTC (JEUS-Tuxedo Connector), which can be used by accessing JEUS and Tuxedo, and is included in the WebT library.

  - Environment settings for accessing JEUS and Tuxedo
  - Usage and examples of JTC classes and functions
  - Methods for setting Tuxedo and WebT logs

- Tmax HMS User Guide

  Describes the following for users and administrators who want to use HMS in Tmax.

  - Concepts and architecture of HMS and the concepts necessary for programming
  - How to set HMS in a configuration file and examples
  - The usage and examples of HMS API functions and how to build, start, terminate, and

manage HMS

- Environment settings for using HMS, program examples, and the compilation method

- Tmax WebT User Guide

  Describes how to develop client programs using the JEUS WebT API to use services of Tmax through the web.

  - The features of WebT and the concepts and components of WebTConnectionPool and the WebT-Server system
  - How to set the environment for WebT and JMax
  - Usage and examples of WebT API for various situations

- Tmax COBOL User Guide

  Describes how to develop a Tmax server program using COBOL.

  - How to create and compile a Tmax configuration file, create a service table, and start an engine
  - How to develop, start, and test a server program, and examples of programs and Makefiles
  - How to use FDL in a server program, how to develop a global transaction program, and how to use TPSVRINIT/TPSVRDONE
  - Usage and examples of COBOL commands and functions

- Tmax Gateway Guide (SERIAL)

  Describes the SERIAL (RS232) Gateway, which acts as an interface when a Tmax server and a non-Tmax server use serial communication.

  - Concepts and operation of the SERIAL Gateway
  - Service types of the SERIAL Gateway
  - How to set an environment to use the SERIAL Gateway
  - Examples of programs using the SERIAL Gateway

- Tmax Gateway Guide (TCP-IP)

  Describes the TCP/IP Gateway, which acts as an interface for a Tmax server and a non-Tmax server.

  - Concepts and operation of the TCP/IP Gateway
  - Service types of the TCP/IP Gateway
  - How to set an environment to use the TCP/IP Gateway
  - Examples of programs using the TCP/IP Gateway

- Tmax Gateway Guide (TCP-IP Service)

  Describes the TCP/IP Service Gateway, which acts as an interface when a Tmax server and a non-Tmax server use TCP/IP communication.

- Concepts and operation of the TCP/IP Service Gateway
- Service types of the TCP/IP Service Gateway
- How to set an environment to use the TCP/IP Service Gateway
- APIs for using the TCP/IP Service Gateway
- Examples of programs using the TCP/IP Service Gateway

- Tmax Gateway Guide (TCP-IP Thread)

  Describes the TCP/IP Thread Gateway, which acts as an interface between non-Tmax clients and Tmax.

  - Concepts and processes of the TCP/IP Thread Gateway
  - Types of the TCP/IP Thread Gateway
  - How to set an environment to use the TCP/IP Thread Gateway
  - User API functions for using the TCP/IP Thread Gateway
  - Examples of programs using the TCP/IP Thread Gateway

- Tmax Gateway Guide (WebService)

  Describes the WebService Gateway, which is the gateway server provided for using a Tmax service as a web service without any modifications.

  - Concepts of a web service and an introduction to the WebService Gateway
  - How to set an environment to use the WebService Gateway
  - How to use and manage the WebService Gateway

- Tmax Gateway Guide (X.25)

  Describes the X.25 Gateway, which acts as an interface for a Tmax server and a non-Tmax server.

  - Concepts and operation of the X.25 Gateway
  - Service types of the X.25 Gateway
  - How to set an environment to use the X.25 Gateway
  - Examples of programs using the X.25 Gateway

- Tmax Programming Guide (4GL)

  Describes how to develop Tmax applications using 4GL languages.

  - How to develop a program using Power Builder, usage and examples of functions, and examples developed applications
  - How to develop a program using Visual Basic, usage and examples of functions, and examples of developed applications
  - How to develop a program using Delphi, usage and examples of functions, and examples of developed applications
  - How to develop a program using Visual Basic .net, usage and examples of functions, and

examples of developed applications

  - How to develop a program using C# .net, usage and examples of functions, and examples of developed applications
  - How to develop a program using ASP, usage and examples of functions, and examples of developed applications

- Tmax Programming Guide (Dynamic Library)

  Describes how to develop a program using Tmax TDL (Tmax Dynamic Library).

  - Concepts, characteristics, and components of TDL
  - Environment settings for using TCDL and Tmax
  - TDL commands and usage and examples of TDL API

- Tmax Programming Guide (RCA)

  Describes how to install and set an environment for the RCA module. The RCA module enables existing communication programs and PDAs that cannot use the Tmax client library to connect to the Tmax system.

  - Basic concepts, structure, and characteristics of RCA
  - How to integrate RCA with the Tmax system
  - Environment variables for using RCA
  - Files provided by each platform and how to set them in the Tmax configuration file
  - Notices, examples, and error messages needed to program RCAH

- Tmax Programming Guide (RPC)

  Describes the basic functions, features, and components of RPC (Remote Procedure Call), a Tmax program model.

  - Basic concepts, types, and constraints of RPC
  - RPC components

- Tmax Programming Guide (RQ)

  Describes the concepts and usage of RQ (Reliable Queue) in Tmax.

  - Definitions, concepts, and characteristics of RQ
  - System configuration for RQ and related APIs
  - Environment settings for using RQ, and how to manage status information

- Tmax Programming Guide (SQ)

  Describes the concepts and usage of SQ (Session Queue) in Tmax.

  - Definitions, concepts, and characteristics of SQ
  - System configuration for SQ and related APIs

    - Environment settings for using SQ, and how to manage status information
    - Examples that illustrate the use of SQ
- Tmax Programming Guide (UCS)

  Describes the concepts and usage of Tmax UCS, a server process.

    - A basic description and a process flow
    - UCS program configuration, environment settings, the compilation method, usage of functions, and program examples
    - RDP program configuration, environment settings, the compilation method, and program examples
- Tmax Programming Guide (MultipleRM)

  Describes the concepts, features, and characteristics of Tmax Multiple RM, a Tmax server process.

    - A basic description of Multiple RM
    - How to set an environment, how to use a server, and program examples
- Tmax XA Library and XA Gateway Guide

  Describes the usage of the XA library and the XA gateway.

- Tmax WebAdmin User Guide

  Describes the concepts of WebAdmin, the tool for managing Tmax, and how to install and use it.

    - The basic concepts and constraints of WebAdmin
    - How to install, start, and operate WebAdmin
    - Features and usage of each menu
    - Descriptions of messages in WebAdmin
- TmaxGrid User Guide

  Introduces TmaxGrid that synchronizes in-memory data in a multi-node environment, and describes how to use it.

    - Introduction to TmaxGrid
    - Environment configuration, APIs, and program examples
- Tmax TCache Guide

  Describes the basic concepts and usage of Tmax TCache.

    - Introduction to TCache
    - TCache tool usage
    - TCache API usage
    - TCache synchronization method

- Tmax Deployment User Guide

  Describes Tmax Deployment that deploys server applications, resource files, and configuration files.

  - Introduction to Tmax Deployment
  - Deployment configuration
  - Deployment operation
  - Usage example