Reference Guide

Tmax 6 Fix#1

TMAXSOFT

저작권 공지

제한된 권리

상표 공지

오픈소스 소프트웨어 공지

본 제품의 일부 파일 또는 모듈은 다음의 라이선스를 준수합니다. : openssl-0.9.7.m, zlib-1.1.4, expat-2.0.0, netsnmp, DCE1.0, pthread, google-diff-match-patch, libevent, getopt

관련 상세한 정보는 제품의 다음의 디렉터리에 기재된 사항을 참고하시기 바랍니다. : ${INSTALL_PATH}/license/oss_licenses

안내서 이력

제품 버전	안내서 버전	발행일	비고
Tmax 6 Fix#1	3.1.1	2024-09-24	-
Tmax 6 Fix#1	2.1.2	2019-09-11	-
Tmax 6	2.1.1	2015-07-31	-

목차

1. 개요

Tmax 시스템을 사용하기 위해서 제공하는 명령어와 함수를 사용해야 한다.

1.1. 명령어

다음은 Tmax에서 제공되는 명령어의 목록이다.

명령어	설명
cfl	텍스트 형태의 Tmax 환경 파일을 컴파일하여 tmconfig(이진 Tmax 환경 파일)을 생성한다.
fdlc	필드 키 테이블을 컴파일한다.
gst	이진 Tmax 환경 파일을 참조하여 서비스 테이블을 생성한다.
hkcfl	텍스트 형태의 Host-link 환경설정 파일을 컴파일하여 이진 Host-link 환경설정 파일(hlinkcfg)을 생성한다.
idlc	RPC 모듈의 정의 파일을 실제 프로그래밍 가능한 소스로 변환해 준다.
mkcli	Tmax 클라이언트 모듈을 생성한다.
mkacl	ACL(Access Control List)를 생성한다.
mkgrp	사용자 그룹을 생성한다.
mkpw	암호를 관리한다.
mksvr	Tmax 서버 모듈을 생성한다.
racd	멀티 노드로 분산된 환경에서 중앙 집중 관리한다.
racdr	멀티 노드, 멀티 도메인으로 분산된 환경에서 파일 복사, 기동을 수행한다.
rcakill	RCA를 종료하려는 경우 RCA가 사용하는 자원을 제거하기 위하여 사용한다.
rcastat	RCA의 설정 내용을 확인하거나 RCA에 접속한 클라이언트의 수 등을 모니터링 할 때 사용한다.
sdlc	구조체를 정의한 파일을 컴파일한다.
svcrpt	Tmax 시스템을 운용할 때 서비스 수행에 관련된 로그 기록을 분석하여 출력한다.
tdlclean	run 디렉터리의 구버전 라이브러리 파일이나 불필요한 파일을 정리한다.
tdlinit	TDL 공유 메모리 및 동적 모듈 초기화를 수행한다.
tdlnm	지정한 라이브러리에 대한 자동 export될 함수 목록을 조회한다.
tdlrm	TDL을 더 이상 사용하지 않을 경우에 공유 메모리를 완전히 제거한다.
tdlseqno	TDL의 시퀀스 번호를 조회한다.
tdlshm	TDL 공유 메모리 정보를 조회하거나, 통계 모니터링 활성화 여부 및 모듈 활성화 여부를 설정한다.
tdlsync	TDL 공유 메모리와 백업 파일 동기화를 수행한다.
tdltrace	TDL의 환경 정보와 통계 정보를 조회한다.

명령어	설명
tdlupdate	지정한 동적 모듈을 업데이트한다.
tencrypt	환경설정의 OPENINFO절을 암호화한다.
tmadmin	Tmax 시스템 관리를 한다.
tmapm	시그알람을 사용할 수 없거나, 용이하지 않은 경우 서비스 타임아웃을 설정하고 사용할 수 있는 별도의 서버이다.
tmaxlibver	Tmax 라이브러리의 버전 정보를 조회한다.
tmaxtrace	애플리케이션에 대하여 Run time tracing을 한다.
tmboot	Tmax 시스템의 전체나 또는 일부분을 실행한다.
tmd	서버 프로그램을 테스트하기 위한 클라이언트 시뮬레이션을 한다.
tmdown	Tmax 시스템 전체 또는 일부분을 종료한다.
tmmbfgen	서비스 정보 파일(text)을 서비스 정보 바이너리 파일로 만든다.
tmsnmpd	표준 SNMP 프로토콜에 의해서 Tmax 구성 및 성능 정보를 조회한다.
tperr	Tmax 에러 번호와 에러 타입을 이용하여 에러에 관한 자세한 정보를 조회한다.
twagent	Tmax 웹 Agent와 연결하는 데몬 프로세스 기동을 위한 명령어이다.
uncfl	텍스트 형태의 Tmax 환경 파일을 컴파일하여 생성된 tmconfig를 텍스트 형태의 환경 파일로 만든다.
untmmbfgen	서비스 정보 바이너리 파일을 서비스 정보 파일(text)로 변환한다.
xwsdlgen	웹 서비스 스펙 중 명세서 역할을 하는 WSDL 문서를 생성한다.

1.2. 함수

다음은 각 용도별 사용되는 함수에 대한 설명이다.

1.2.1. 서버/클라이언트 함수

함수	설명
gettperrno	Tmax 시스템을 호출할 때 설정된 에러 코드(errno)를 반환한다.
tgstrerror	TmaxGrid API 사용할 때 발생하는 tgerrno에 해당하는 번호의 메시지를 출력한다.
tmax_chk_conn	클라이언트의 연결 상태를 체크하는 함수로 tpstart 수행 여부 체크, 소켓 상태 점검, 메시지 전달로서 연결 상태를 점검하는 역할을 한다.
tmax_get_sessionid	현 세션의 ID를 반환한다.
tmax_gq_count	GQ에 저장된 데이터의 개수를 반환한다.
tmax_gq_get	GQ에서 데이터를 가져오는 함수로 키를 지정하면 해당 키의 데이터를 가져온다.
tmax_gq_getkeylist	GQ의 키 리스트를 가져온다.
tmax_gq_keygen	시스템 키를 생성하고 가져온다.

함수	설명
tmax_gq_purge	GQ의 데이터를 삭제한다.
tmax_gq_put	데이터를 GQ에 저장하는 함수로 키와 데이터 값을 전달한다.
tmax_grid_count	전체 Key 개수를 참조한다.
tmax_grid_create	Key를 생성한다.
tmax_grid_create2	Key를 생성하고 Key에 value를 설정한다.
tmax_grid_dequeue	tmax_grid_enqueue()에 의해서 입력한 Value 중 가장 처음에 입력한 Value를 참조한다.
tmax_grid_destroy	Key를 삭제한다.
tmax_grid_enqueue	Key 이름으로 데이터를 입력한다.
tmax_grid_get	value를 가져오고 해당 Key의 Value는 삭제한다.
tmax_grid_get_child_with_index	자식 Key의 정보를 담고 있는 grid_KEYLIST_T 핸들러에서 nth번째의 Key에 대한 정보를 grid_KEYINFO_T 구조체에 저장한다.
tmax_grid_get_children	Key 이름으로 자식 Key 이름의 리스트를 참조한다.
tmax_grid_is_exist	Key가 존재하는지 검사한다.
tmax_grid_keylist_free	자식 Key의 정보를 담고 있는 grid_KEYLIST_T 핸들러의 자원을 해제한다.
tmax_grid_lock	Key 이름으로 Lock을 수행한다.
tmax_grid_set	Key에 value를 설정한다.
tmax_grid_unlock	Key 이름으로 Lock을 해제한다.
tmax_grid_set_watcher	해당 Key의 이벤트 발생할 때 호출하는 함수를 등록한다.
tmax_grid_wait_watcher	이벤트 발생할 때 까지 timeout으로 지정한 시간 동안 대기한다.
tmax_keylist_count	keylist 핸들로부터 키 리스트의 개수를 반환한다.
tmax_keylist_free	keylist 핸들의 메모리나 기타 자원들을 해제한다.
tmax_keylist_getakey	keylist 핸들로부터 n번째 키 정보를 가져온다.
tmax_sq_count	현재 SQ에 저장된 데이터 개수를 반환한다.
tmax_sq_get	데이터를 세션 큐에 저장한다.
tmax_sq_getkeylist	현 세션 SQ의 키 리스트를 가져온다.
tmax_sq_keygen	시스템 키를 생성하고 가져온다.
tmax_sq_purge	SQ의 데이터를 삭제한다.
tmax_sq_put	서버 데이터를 세션 큐에 저장한다.
tmaxlastsvc	가장 마지막에 수행된 조회하는 함수로 에러가 발생한 서비스명 또는 최후로 루틴을 수행한 서비스명을 반환한다.
tmaxreadenv	환경변수를 파일에서 접속할 시스템의 정보를 읽어서 환경변수에 새로운 값을 설정한다.
tp_sleep	데이터의 도착을 초 단위로 기다린다.
tp_usleep	데이터의 도착을 1000000(백만)분의 1초 단위로 기다린다.

함수	설명
tpabort	전역 트랜잭션을 rollback한다.
tpacall	비동기 서비스 요청을 송신한다.
tpacallsvg	COUSIN으로 묶인 멀티 서버 그룹 환경에서 특정 서버 그룹에 속하는 서비스에 비동기형 통신으로 서비스 요청을 송신한다.
tpalivechk	클라이언트의 연결 상태를 체크하는 함수로 소켓의 상태를 점검하는 역할을 한다.
tpalloc	유형 버퍼(typed buffer)를 할당한다.
tpbegin	트랜잭션 시간 설정 및 전역 트랜잭션을 시작한다.
tpbroadcast	Tmax 시스템에 등록된 클라이언트들에게 요청하지 않은 메시지를 송신한다.
tpcall	동기형 서비스 요청을 송수신한다.
tpcallsvg	특정 서버 그룹에 속하는 서비스를 서버와 클라이언트에서 호출한다.
tpcancel	응답을 취소한다.
tpcommit	전역 트랜잭션을 commit한다.
tpconnect	프로그램이 대화형 서비스 svc와 통신을 연결한다.
tpdeq	RQ로부터 데이터를 로드한다.
tpdeq_ctl	트랜잭션을 지원하며 RQ로부터 데이터를 로드한다.
tpdiscon	대화형 통신의 연결을 종료한다.
tpenq	RQ에 데이터를 저장한다.
tpenq_ctl	트랜잭션을 지원하며 RQ 데이터를 저장한다.
tperrordetail	서버와 클라이언트에서 Tmax 시스템 호출할 때 발생한 오류의 자세한 정보를 얻을 때 사용한다.
tpextsvcinfo	**tpdeq()**로 RQ에서 데이터를 읽은 경우 해당 데이터에 대한 상세한 정보를 제공한다.
tpextsvcname	RQ에 저장된 데이터 중 서비스명을 출력한다.
tpfree	유형 버퍼(typed buffer)에 할당된 메모리를 해제한다.
tpget_timeout	블록 타임아웃 시간을 반환한다.
tpgetactivesvr	현재 활성화되어 있는 서버들의 목록을 조회한다.
tpgetcliaddr	Tmax 시스템에 접속된 클라이언트 중 clid에 해당하는 클라이언트의 IP와 포트 번호를 얻는 함수이다.
tpgetcliaddr_ipv6	Tmax 시스템에 접속된 클라이언트 중 clid에 해당하는 클라이언트의 IP와 포트 번호를 얻는 함수로 IPv6 환경에서 사용한다.
tpgetclid	Tmax 시스템에 접속된 클라이언트의 번호를 알 수 있는 함수이다.
tpgetcliinfo	Tmax 시스템에 접속된 클라이언트 중 clid에 해당하는 클라이언트의 정보를 얻어 온다.
tpgetctxt	현재 컨텍스트를 반환한다.
tpgetenv	name이라는 이름으로 등록된 환경변수의 값을 반환한다.
tpgetlev	트랜잭션 모드에 있는지의 여부를 확인한다.

함수	설명
tpgetnodelist	현재 접속된 Tmax 시스템의 노드의 목록을 가져온다.
tpgetpeername	연결된 상대방의 소켓 주소를 얻어온다.
tpgetrcahseqno	tpgetsvrseqno API와 동일하게 RCAH 프로세스 번호를 반환한다.
tpgetrcainfo	RCAH의 Thread 정보를 알려준다.
tpgetrply	비동기적으로 요청한 서비스에 대한 응답을 수신한다.
tpgetsockname	Tmax 시스템 내부적으로 사용되는 소켓 주소를 얻는다.
tpgetsprlist	서버 프로세스 단위로 호출을 하기 위하여 해당 서비스가 속한 서버 프로세스의 인덱스를 가져오는 함수이다.
tpgetsvglist	해당 서비스가 속하는 서버 그룹과 이 서버 그룹의 COUSIN으로 설정된 서버 그룹들에 대한 정보를 제공한다.
tpgetsvglist_bynode	지정한 서비스를 제공하는 서버그룹 중에 지정한 노드명에 해당하는 서버그룹 목록을 가져온다.
tpgprio	요청받은 서비스의 우선순위를 보여주는 함수이다.
tpmcall	COUSIN으로 묶인 모든 서버 그룹 서버의 서비스를 호출한다.
tpmcallx	기존 tpmcall()의 확장 기능 제공을 목적으로 하는 함수로 기존과 달리 COUSIN 서버 그룹의 서비스들로부터 모두 응답을 받을 때까지 기다리는 함수이다.
tpnotify	서버에서 지정된 클라이언트에 비요청 메시지를 송신한다.
tppost	특정 사건을 발생시키고 메시지를 전달한다.
tpputenv	환경변수 값을 재설정한다.
tpqstat	RQ에 저장된 데이터의 통계를 요청한다.
tpqsvcstat	RQ에 저장된 데이터 중 지정한 서비스에 대한 통계를 요청하는 함수로 현재 RQ에 쌓여 있는 데이터의 통계를 구한다.
tprealloc	RQ에 저장된 데이터 건수를 조회한다.
tprecv	대화형 통신을 하는 경우 메시지를 수신한다.
tprecvfromcli	서버에서 클라이언트로 메시지를 요청하고 받는 함수이다.
tpreissue	해당 RQ의 Fail 큐에 쌓인 요청 데이터를 다시 Request 큐에 넣어주는 함수이다.
tpremoteconnect	리모트와 TCP로 연결하는 함수이다.
tpscmt	commit 방법을 재설정한다.
tpsend	대화형 통신에서 메시지를 송신한다.
tpset_timeout	블록 타임아웃 시간을 설정한다.
tpsetctxt	현재 컨텍스트를 설정한다.
tpsetfd	소켓 FD를 UCS 프로세스의 스케줄러에 등록한다.
tpsetsvctimeout	서버에서 사용되는 함수로 서버에 설정되어 있는 서비스 타임아웃 시간을 설정한다.
tpsleep	데이터가 도착할 때까지 대기한다.

함수	설명
tpspracall	**tpgetsprlist()**를 통해 얻어온 서버 프로세스의 인덱스 중 특정 프로세스에게 서비스를 호출한다.
tpspracall2	DYN 서버 타입용 함수로 **tpgetsprlist()** 를 통해 얻어온 starti로부터 spri 순번을 지정하여 해당 서버 프로세스에게 서비스를 호출한다.
tpsprio	서비스 요청의 우선순위를 설정한다.
tpstrerror	에러 번호에 해당하는 메시지를 출력한다.
tpsubqname	서브 큐 번호에 해당하는 큐의 이름을 반환한다.
tpsubscribe	특정 사건의 메시지에 대한 요청을 등록한다.
tptypes	버퍼의 유형 및 하위 유형에 대한 정보를 조회한다.
tpunsubscribe	특정 사건의 메시지에 대한 등록을 해제한다.
tuxgetenv	환경변수 값을 반환한다.
tuxputenv	환경변수를 적용한다.
tuxreadenv	파일의 환경변수를 읽어오는 함수이다.
tx_begin	전역 트랜잭션을 시작한다.
tx_commit	전역 트랜잭션을 Commit한다.
tx_info	전역 트랜잭션 정보를 반환한다.
tx_rollback	전역 트랜잭션을 rollback한다.
tx_set_commit_return	commit_return 특성을 설정한다.
tx_set_transaction_control	transaction_control 특성을 control 값으로 설정한다.
tx_set_transaction_timeout	transaction_timeout 특성을 타임아웃 값으로 설정한다.
ulogsync	메모리 버퍼의 ulog를 파일로 저장한다.
userlog	메모리 버퍼에 ulog를 저장한다.
UserLog	ulog를 즉시 파일에 저장한다.

1.2.2. 서버 함수

이름	설명
_tmax_check_license	AnyLink 및 OpenFrame에서 해당 라이선스가 발급되었는지 체크한다.
_tmax_event_handler	SVRTYPE이 EVT_SVR인 경우 SLOG가 발생하는 경우 호출되는 콜백 함수이다.
_tmax_main	사용자 프로그램에 main()이 포함되어 있는 경우 사용한다.
CliWatcherCallback	UCS 프로세스에서 사용되며 tpsetcliwatcher의 client 종료를 통지받을 콜백함수이다.
tmadmin	Tmax 시스템 관리 툴인 tmadmin에서 조회할 수 있는 통계정보를 출력한다.

이름	설명
tmax_get_db_passwd	현재 Tmax가 접속하고 있는 데이터베이스의 username에 대한 비밀번호를 조회한다.
tmax_get_db_tnsname	현재 Tmax가 접속하고 있는 데이터베이스의 tnsname을 조회한다.
tmax_get_db_usrname	현재 Tmax가 접속하고 있는 데이터베이스의 username을 조회한다.
tmax_get_svccnt	자신이 속한 서버의 서비스 개수를 반환한다.
tmax_get_svclist	자신이 속한 서버의 서비스 목록을 가져오기 위한 함수이다.
tmax_is_restarted	Tmax AP 서버 루틴 내에서 자신이 속한 서버 프로세스가 비정상적으로 종료된 후 재기동되었는지의 여부를 판단할 수 있도록 한다.
tmax_is_xa	현재 자신이 속한 서버가 XA인지 NON-XA인지 체크해 주는 함수이다.
tmax_my_rminfo	자기 자신이 속한 서버 그룹의 OPENINFO 정보를 가져온다.
tmax_my_svrinfo	서버 프로세스의 시스템 설정 정보를 획득한다.
tmgetsmgid	SysMaster trace 기능을 지원하기 위한 함수로 현재 자신의 GID를 얻어온다.
TPADMNTOI	tmadmin으로부터 admntoi 명령으로 들어오는 요청에 대해서 처리한다.
tmget_smtrclog	로깅 데이터를 구조체 버퍼에 저장한다.
tmget_smtrclog_count	데이터의 개수를 반환한다.
tpadvertise	서버 프로세스가 제공하는 서비스를 서버에 advertise한다.
tpcancelctx	서버 라이브러리 내의 CTX_T 구조체의 내용을 삭제한다.
tpchkclid	UCS 프로세스내에서 데이터를 송부하기에 앞서 해당 클라이언트가 정상적으로 접속되어 비요청 데이터를 수신할 수 있는지 판별한다.
tpclrcliwatcher	UCS 프로세스에서 사용되며 tpsetcliwatcher()으로 등록한 clid에 대해서 종료 이벤트 감지를 취소한다.
tpclrfd	UCS 방식 프로세스 내부의 fdset의 소켓 FD를 off시키는 데 사용된다.
tpclrfd_w	UCS 방식 프로세스 내부의 writable fdset의 소켓 FD를 off시키는 데 사용된다.
tpforward	서비스 요청을 또 다른 서비스 루틴으로 전달한다.
tpgetctx	서버 라이브러리 내의 CTX_T 구조체의 값을 사용자 변수에 저장한다.
tpgetdbsessionid	RM 세션 정보를 얻어오는 함수이다.
tpgetfclid	Tmax 시스템에 최소 서비스를 시작시킨 클라이언트 번호를 가져오는 함수이다.
tpgetmaxsvr	서버 프로세스의 최대 실행 개수를 출력한다.
tpgetmaxuser	서버 프로세스가 속한 노드의 최대 동시 접속자 수를 출력한다.
tpgetminsvr	서버 프로세스의 최소 실행 개수를 출력한다.
tpgetmynode	서버에서 특정 노드명과 노드 번호를 얻는 함수이다.
tpgetmysvgno	현재 자신이 속해 있는 서버 그룹의 번호를 알려주는 함수이다.
tpgetmysvrid	서버 프로세스 ID를 출력한다.
tpgetnodename	서버에서 지정된 노드명을 얻는 함수이다.
tpgetnodeno	서버에서 nodename을 가지는 노드의 번호를 얻는 함수이다.

이름	설명
tpgetorgclh	해당 클라이언트가 현재 접속되어 있는 CLH 번호를 알아내는 함수이다.
tpgetorgnode	해당 클라이언트가 접속된 노드 번호(node number)를 반환한다.
tpgetpeer_ipaddr	연결된 상대방의 IP 주소를 출력한다.
tpgetsvcname	서비스 인덱스로부터 서비스명을 가져오는 함수이다.
tpgetsvrseqno	같은 서버 프로세스들 간의 서버 프로세스에 대한 일련번호를 반환한다.
tpissetfd	서버에서 UCS 프로세스에서 소켓 FD로 데이터가 도착했는지를 검사한다.
tpissetfd_w	UCS 방식 서버 프로세스의 FDSET을 검사하여 파라미터 값으로 주어진 소켓 FD에 보낼 데이터가 있는지 확인한다.
tpprechk	RM의 상태 점검을 위한 사용자 콜백 함수로 tpprechk()는 Tmax 시스템 접속 전에 호출된다.
tpregancb	tmadmin 으로부터 admntoi 명령으로 들어오는 요청에 대해서 처리한다.
tpregcb	서버에서 UCS 상에서 비동기형 요청에 대한 응답을 받는 루틴을 설정한다.
tprelay	UCS 서버 프로세스에서 서비스를 요청한 클라이언트의 정보를 담아서 또 다른 서비스를 요청한다.
tpresumetx	현재 중지된 전역 트랜잭션을 재개하는 함수로 중지된 전역 트랜잭션을 **tpresumetx(), tpsuspendxt()**를 통하여 재개시킬 수 있다.
tpreturn	서버의 서비스를 종료한다.
tpsavectx	UCS 프로세스에서 사용되며 클라이언트의 정보를 내부적으로 관리하도록 한다.
tpschedule	UCS 서버 프로세스에서 데이터의 도착을 기다리는 함수이다.
tpsendtocli	지정된 클라이언트에 비요청 메시지를 송신한다.
tpsetcliwatcher	UCS 프로세스에서 사용되며 clid에 대응하는 클라이언트가 종료될 때 감지한다.
tpsetdbsessionid	RM 세션 정보를 얻기 위한 함수이다.
tpsuspendtx	기존 전역 트랜잭션을 중지하기 위한 함수이다.
tpsvctimeout	서비스 타임아웃이 발생했을 경우 호출한다.
tpsvrdone	Tmax 서버 프로세스 종료 루틴을 호출한다.
tpsvrdown	서버 프로세스를 down한다.
tpsvrinit	Tmax 서버 프로세스를 초기화한다.
tpsvrthrdone	Multithread 및 Multicontext 서버는 서버 프로 세스가 종료될 경우 tpsvrdone 함수를 수행하기에 앞서 서비스 Thread들을 종료시킨다.
tpsvrthrinit	Multithread 및 Multicontext 서버에서 tpsvrinit 함수가 호출된 이후 Thread들을 초기화한다.
tptsleep	TMM으로부터 서버 프로세스 종료 이벤트를 대기한다.
tpunadvertise	서버 프로세스가 제공하는 서비스를 서버에서 등록해제(unadvertise)한다.
tpunregancb	tmadmin으로부터 admntoi 명령으로 들어오는 요청에 대해서 처리하는 콜백 함수를 해제한다.

이름	설명
tpunregcb	서버에서 UCS에서 비동기형 요청에 대한 응답을 받는 루틴을 재설정한다.
tpuschedule	UCS 서버 프로세스에서 데이터 도착을 기다리는 함수이다.
tx_close	리소스 관리자들과의 연결을 종료한다.
tx_open	관련된 리소스 관리자와 연결한다.

1.2.3. 클라이언트 함수

이름	설명
gettpurcode	urcode 서비스에 설정된 urcode를 클라이언트에 반환한다.
tpchkunsol	함수를 호출할 당시에 서버로부터 비요청 메시지가 있는 경우 해당 메시지를 처리하기 위한 함수를 호출하기 위한 함수이다.
tpend	클라이언트에서 Tmax 시스템과의 연결을 해제한다.
tpgetclid	Tmax에 연결되어 있을 경우 Tmax에서 관리하는 자신의 clid 값을 얻어온다.
tpgethostaddr	Tmax 클라이언트와 Tmax 시스템에 접속여부를 확인하거나, 접속된 Tmax 시스템의 IP와 포트 번호에 대한 정보를 얻고자 할 때 사용한다.
tpgetunsol	클라이언트의 요청없이 일방적으로 전달받은 메시지를 처리한다.
tpinit	Tmax 시스템에 연결한다.
tpreset	현재 접속된 클라이언트 연결을 해제한다.
tpsetunsol	비요청 수신 메시지를 처리하는 루틴을 설정한다.
tpsetunsol_flag	비요청 메시지 수신 flags를 변경한다.
tpstart	Tmax 시스템에 연결한다.
tpterm	Tmax 시스템과의 연결을 해제한다.
tptobackup	Tmax 백업 시스템으로 연결한다.

1.2.4. TCP/IP 게이트웨이 함수

함수	설명
allow_connection	리모트 노드와 새로운 연결이 맺어지기 전에 호출되는 함수이다. 해당 연결 요청의 허용 여부를 반환값으로 결정할 수 있다.
allow_connection_ipv6	IPv6 프로토콜 환경에서 리모트 노드와 새로운 연결이 맺어지기 전에 호출되는 함수이다. 해당 연결 요청의 허용 여부를 반환값으로 결정할 수 있다.
chk_end_msg	리모트 노드로부터 데이터를 수신할 때 데이터의 끝을 나타내는 특정 문자 또는 비트 스트림이 존재할 경우 사용자가 호출할 수 있는 함수이다.
chk_extpong_msg	채널 장애를 감지할 때 서버로부터 받은 메시지를 확인하는 함수이다.
chk_pong_msg	채널 장애 감시 응답 메시지 여부를 체크하는 사용자 함수이다.

함수	설명
get_channel_num	Tmax 서비스나 클라이언트로부터 요청한 데이터를 리모트 노드에 전송할 때 사용자가 채널을 선택할 수 있도록 하는 함수이다.
get_extmsg_info	get_msg_info와 일부 기능을 제외하고 기본적으로 동일한 기능을 갖는다.
get_msg_info	리모트 노드으로부터 요청이나 응답이 도착하여 데이터를 읽은 후 TCPGW 라이브러리와 custom.c와의 인터페이스 역할을 하는 info를 참조 또는 가공하는 함수이다.
get_msg_length	리모트 노드로부터 요청이나 응답이 도착하는 경우 호출하는 함수로 반환된 값만큼, 실 데이터를 다시 read하는 함수이다.
get_msg_security	get_msg_info 호출된 이후에 데이터를 가공할 수 있는 함수이다.
get_service_name	tpreply()나 tpacall()을 할 서비스명의 오류 코드에 따라서 설정하는 함수이다.
init_remote_info	리모트 노드와 연결을 맺기에 전에 호출되는 함수이다.
inmsg_recovery	리모트 노드로부터의 요청을 처리했을 때 오류가 발생하는 경우 호출하는 함수이다.
outmsg_recovery	리모트 노드로 요청을 보낼 때 오류가 발생할 경우 호출된다.
prepare_shutdown	TCPGW가 종료하기 직전에 call되는 함수이다.
put_extmsg_info	put_msg_info와 일부 기능을 제외하고 기본적으로 동일한 기능을 갖는다.
put_msg_info	리모트 노드로 메시지를 전송하고자 할 때 호출되는 함수이다.
put_msg_complete	리모트 노드로 메시지를 전송한 후에 호출되는 함수이다.
put_msg_security	put_msg_info 호출되기 전에 데이터를 가공할 수 있는 함수이다.
remote_closed	리모트 노드와 연결을 종료한 후 호출되는 함수이다.
remote_connected	리모트 노드와 연결을 맺은 후에 호출되는 함수이다.
remote_connected_ipv6	IPv6 프로토콜 환경에서 리모트 노드와 연결을 맺은 후에 호출되는 함수이다.
reset_extping_msg	reset_ping_msg와 일부 기능을 제외하고 기본적으로 동일한 기능을 갖는다.
reset_ping_msg	TCP/IP Ping(채널 장애 감지) 메시지의 전송 여부와 전송 메시지를 재설정하기 위해 주기적으로 호출되는 함수이다.
set_error_msg	리모트 노드와의 거래 중 에러가 발생한 경우 자동으로 호출되는 함수이다.
set_extping_msg	서버에 채널 장애를 감지할 때 보낼 메시지를 설정하는 함수이다.
set_ping_msg	채널 장애 감시를 위해서 보낼 메시지 설정 및 주기, 타임아웃 등을 설정할 수 있는 사용자 함수이다.
set_service_timeout	서비스 타임아웃이 발생할 경우 사용자가 호출할 수 있는 함수이다.

1.2.5. TDL 함수

이름	설명
tdlcall	최신 버전의 동적 모듈 함수를 호출한다.
tdlcall2	최신 버전의 동적 모듈 함수를 호출한다.

이름	설명
tdlcall2s	최신 버전의 동적 모듈 함수를 호출한다.
tdlcall2v	최신 버전의 동적 모듈 함수를 호출한다.
tdlcallva	최신 버전의 동적 모듈 함수를 호출한다.
tdlcallva2	최신 버전의 동적 모듈 함수를 호출한다.
tdlclose	해당 모듈의 레퍼런스 카운트를 0으로 초기화하거나, 모듈을 직접 메모리에서 해제한다.
tdlcreate	최신 버전의 동적 모듈에서 Class Factory를 사용하여 클래스 인스턴스를 생성하는 함수로 TDL 환경 파일(tdl.cfg)에 VERSION=4로 설정된 경우 사용 가능하다.
tdldestroy	최신 버전의 동적 모듈에서 Class Factory를 사용하여 클래스 인스턴스를 파괴하는 함수로 TDL 환경 파일(tdl.cfg)에 VERSION=4로 설정된 경우 사용 가능하다.
tdldone	공유 메모리를 초기화한다.
tdlend	명시적 버전 정합성(Explicit Version Consistency) 유지를 종료한다.
tdlerror	tdlcall()에 대한 에러가 발생했을 때 문자열 형태로 변환한다.
tdlfind	모듈의 인덱스를 찾는다.
tdlfind2	모듈의 인덱스를 찾는다.
tdlgetseqno	글로벌 시퀀스 번호를 가져온다.
tdlinit	공유 메모리를 초기 설정한다.
tdlload	Hashtable 검색 및 라이브러리 적재를 tdlcall()을 하기 전에 미리 수행하여 로컬 캐시에 해당 모듈의 정보를 저장한다.
tdlload2	Hashtable 검색 및 라이브러리 적재를 tdlcall()을 하기 전에 미리 수행하여 로컬 캐시에 해당 모듈의 정보를 저장한다.
tdlresume	일시적으로 중지된 버전 정합성 유지를 재개한다.
tdlstart	명시적 버전 정합성(Explicit Version Consistency) 유지를 시작한다.
tdlstat	TDL 통계 정보를 출력한다.
tdlstat2	TDL 통계 정보를 출력한다.
tdlsuspend	일시적으로 버전 정합성 유지를 중지한다.

1.2.6. 보안 함수

이름	설명
tmax_accept_crypt	tmax_init_crypt와 마찬가지로 tpstart() 내부에서 클라이언트와 CAS 둘 다 호출된다.
tmax_auth_plugin_init	Tmax 환경설정의 CASOPT에 설정된 값들을 전달한다.
tmax_chk_authen	tpstart() 내부에서 클라이언트의 tmax_init_auth() 호출에 대한 CAS의 인증 검사한다.
tmax_chk_author	CAS에서 인가 검사할 때 호출한다.

이름	설명
tmax_crypt_plugin_init	Tmax 환경설정의 CASOPT에 설정된 값들을 전달한다.
tmax_crypt_plugin_fini	Tmax 환경설정의 CASOPT에 설정된 값들을 전달한다.
tmax_fini_auth	tmax_init_auth 함수를 통해 생성된 해당 클라이언트의 보안 토큰 정보를 삭제한다.
tmax_fini_crypt	tmax_init_crypt 함수를 통해 생성된 own_ctoken을 정리한다.
tmax_init_auth	클라이언트가 tpstart() 내부에서 호출한다.
tmax_init_crypt	암호화 작업을 위한 초기화 과정을 수행한다.
tmax_req_auth	클라이언트의 tpcall 호출 내부에서 요청하는 서비스에 대한 인가를 위해 호출한다.
tmax_tencrypt	Tmax 암/복호화 함수이다.
tmax_unwrap	클라이언트와 Tmax 간의 암호화된 메시지를 복호화한다.
tmax_wrap	클라이언트 와 Tmax간의 메시지를 암호화한다.

1.2.7. Windows 관련 함수

이름	설명
WinTmaxAcall	MuitiThread 환경에서의 비동기 서비스의 송신 요청한다.
WinTmaxAcall2	MuitiThread 환경에서의 비동기 서비스의 송신 요청한다.
WinTmaxEnd	Tmax 시스템과 연결을 종료한다.
WinTmaxSend	데이터를 송신한다.
WinTmaxSetContext	Windows 핸들과 메시지 타입을 설정한다.
WinTmaxStart	Multi Windows 환경에서 Tmax 시스템과 연결하는 데 사용되는 함수이다.

1.2.8. 기타 함수

이름	설명
tlog_close	트랜잭션 로그를 분석하기 위한 함수 중 하나로, 해당 로그 파일을 닫는 함수이다.
tlog_find	해당 트랜잭션 로그 파일에서 entry에 지정된 정보와 매칭(matching)되는 entry를 찾는 함수이다.
tlog_nodeno	XID를 이용하여 트랜잭션이 시작된 노드 번호를 찾는 함수이다.
tlog_open	트랜잭션 로그를 분석하기 위한 함수로 해당 로그 파일을 불러(open)온다.
tmaxoserrno	시스템 호출 도중 에러가 발생할 경우 통합된 에러 번호가 설정되는 변수이다.
Usiginit	Tmax 시그널 핸들러를 초기화한다.
Usignal	사용자 시그널 핸들링에 필요한 매크로 설정에 사용되는 함수로 Windows 시스템 환경에는 사용되지 않는다.
Ustrerror	시스템 에러코드(errno)에 대한 통합 에러 메시지를 반환한다.

이름	설명
Uunixerr	시스템 호출 도중 에러가 발생할 경우 통합된 에러 번호가 설정되는 변수이다.
Uunix_err	ATMI API 호출이 실패하고 tperrno가 TPEOS로 설정된 경우 시스템 에러의 종류를 stderr로 출력한다.

2. 명령어

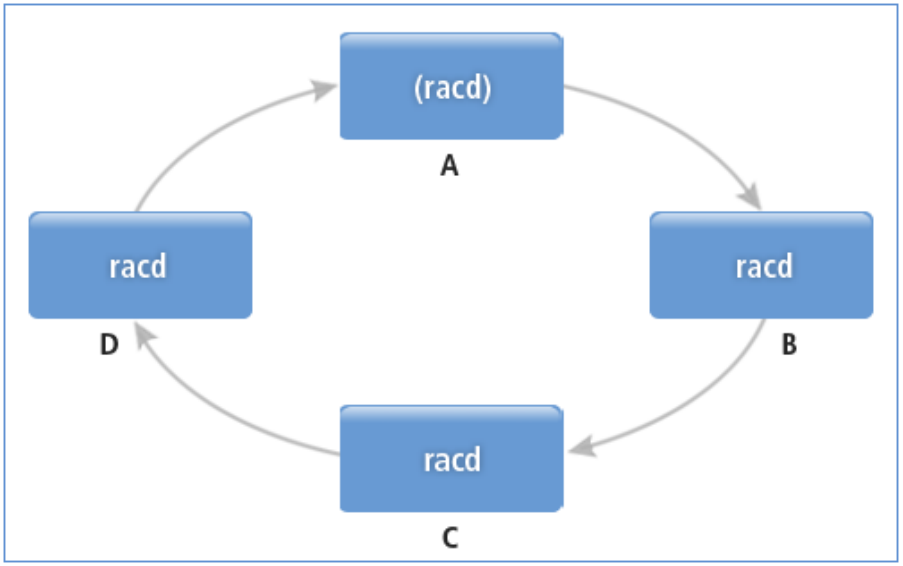
본 장에서는 Tmax에서 사용할 수 있는 명령어를 설명한다.

2.1. cfl

텍스트 형태의 Tmax 환경 파일을 컴파일하여 **tmconfig**(이진 Tmax 환경 파일)을 생성하는 명령어이다.
프로그램을 동작시키기 위해서는 프로그램에 맞는 환경 파일 정의가 필요하고 환경 파일의 정의가 완료된 후엔 환경 파일이 올바르게 생성되었는지 검증이 이루어져야 한다. cfl은 텍스트 형태로 작성된 Tmax 환경 파일을 이진 파일로 컴파일하는 명령어이다.

컴파일하는 중에 에러가 발견되면 이진 환경 파일을 만들지 않고 컴파일을 중단된다. 에러가 발견되지 않으면 이진 파일로 변환된 Tmax 환경 파일이 생성된다. 컴파일이 완료된 이진 Tmax 환경 파일은 **gst**, **tmboot**, **tmdown** 등에서 사용한다. 멀티 노드 환경에서 특정 노드의 환경 파일을 컴파일하고 싶은 경우에 [-n *node_name*] 옵션을 사용한다.

다음은 멀티 노드 감시 환경을 설명한다.



멀티 노드 감시 환경

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ cfl [-i 텍스트 Tmax 환경 파일명] [-o 이진 Tmax 환경파일명] [-h] [-V]
      [-n node_name] [-A] [-v num] [-I] [-r] [-x] [-P] [-g]
```

항목	설명
[-i 텍스트 *Tmax* 환경 파일명]	컴파일 대상이 되는 원본 설정 파일인 텍스트 형태의 Tmax 환경 파일명을 설정한다. 디렉터리는 사용자가 지정할 수 있으며 지정하지 않았을 경우에 기본적으로 환경설정 디렉터리는 **${TMAXDIR}/config**이다. 원본 설정 파일을 찾지 못한 경우에는 경고 메시지를 출력한다.
[-o 이진 *Tmax* 환경파일명]	컴파일 결과물인 이진 Tmax 환경 파일명을 설정한다. 경로와 함께 지정할 수 있으며, 경로가 지정되지 않은 경우 **${TMAXDIR}/config**에 이진 Tmax 환경 파일이 생성된다. 옵션이 생략되면 파일명은 **tmconfig**로 생성된다.
[-h]	명령어 도움말 옵션이다.
[-V]	실행 파일의 버전을 확인할 수 있다.
[-n *node_name*]	멀티 노드 환경에서 특정 노드의 환경 파일을 컴파일하고 싶은 경우에 사용하는 옵션이다. 멀티 노드 환경에서 다른 노드를 관리하기 위해서는 racd를 설정해야 한다.
[-A]	서비스 접근 제어(제3단계 보안)를 사용할 때만 유효한 옵션이다. 하나의 도메인 내에 있는 ACL 서비스에 대한 접근 권한은 모든 노드에서 동일하게 적용되어야 하므로 현재 노드의 ${TMAXDIR}/config/group, acl, user 파일을 같은 도메인에 속해 있는 다른 노드들의 ${TMAXDIR}/config에 배포해 주기 위한 옵션이다. 옵션을 사용하여 환경 파일을 컴파일할 경우에는 미리 **mkgrp**, **mkpw**, **mkacl**을 이용하여 **group**, **acl**, **user** 파일이 생성되어 있어야 한다. 생성된 파일은 하나의 도메인 내에 있는 모든 노드들이 공용으로 사용하기 때문에 파일을 생성할 때 도메인에 속하는 노드의 모든 사용자를 고려해야 한다. group_name, group_id, user_name, user_id는 한 도메인을 통틀어서 유일하게 생성해야 한다. passwd 파일은 복사되지 않으므로 노드별로 생성하거나, 별도로 복사해야 한다.
[-v *num*]	num에는 0과 1을 지정할 수 있다. • 0 : 멀티 노드를 구성하고 있는 노드가 모두 같은 타입의 머신이며 관리자가 노드별로 환경 파일을 관리하는 경우에 0으로 설정한다. 컴파일된 이진 파일은 각각의 머신으로 복사해서 관리한다. • 1 : racd를 통해서 환경 파일이 자동적으로 머신에서 컴파일된다. (기본값)
[-I]	기본적으로 cfl을 수행할 때 환경 파일 DOMAIN 절의 SHMKEY 항목에 설정한 값이 현재 사용 중이라면 해당 값의 UID를 비교하며, 다를 경우 에러 메시지가 발생한다. 해당 옵션은 공유 메모리 값의 현재 사용 여부 및 UID 동일 여부를 체크하지 않을 때 사용한다.

항목	설명
[-r]	cfl 수행 단계에서 ulimit –n으로 조회할 때 출력되는 현재 시스템의 사용 가능한 FD 최대 수를 미리 체크하여 사용자에게 알려준다. CLH 1개당 열 수 있는 최대 FD 개수를 미리 계산하여 체크한다. Tmax 시스템에서 사용하는 FD 값이 시스템에서 사용 가능한 FD보다 더 크게 설정되어 있다면 다음과 같은 에러가 발생한다. `(E) CFL9990 Current Tmax configuration contains more servers or nodes than current system can support[CFL5056]`
[-a *Tmax* 환경파일명]	동적 추가할 수정된 Tmax 환경 파일명을 지정한다. 동적 추가의 경우 cfl을 이용하여 이진(binary) 환경 파일을 만들 때에는 [-a] 옵션을 반드시 사용해야 한다. 만약 이 옵션을 사용하지 않고 동적 추가한 경우에는 (E) TMAX00157 에러가 발생한다. 자세한 내용은 Tmax Administration Guide의 "cfgadd(ca)"를 참고한다.
[-Z]	현재 cfl에서 MAXSACALL/MAXCACALL 1024 이하 제한되어 있으나, 이 제한을 해제하는 옵션이다. 예) `- cfl -Z -i sample.m`

[-x]	SERVER 절 설정의 TARGET 및 CLOPT의 -x 옵션을 사용할 때 존재하는 서버, 서비스명인지 체크한다.
	
	• SERVER 절의 TARGET 설정에 환경 파일에서 선언되지 않은 서버명이 설정되었을 때 다음과 같은 에러 메시지를 출력한다.
	
	`line %d(서버가 선언된 라인): invalid target server name %s(TARGET에 설정한 서버명)`
	
	• SERVICE 절의 CLOPT 설정에 환경 파일에서 선언되지 않은 서비스명으로 -x 옵션이 설정되었을 때 다음과 같은 에러 메시지를 출력한다.
	
	`line %d(서버가 선언된 라인): invalid service name %s(-x로 설정한 서비스명)`
	
	라인 번호는 TARGET이나 CLOPT가 설정된 라인이 아닌, 서버가 선언된 라인이 출력된다.
	
	예)
	
	`- cfl -x -i sample.m`

항목	설명

[-P]	전체 공유 메모리의 크기를 조회하는 옵션이다.
	
	`shmsize info:`
	`shmkey = 80001, shmsize = 552652`
	`shmkey = 80002, shmsize = 111968`
	`shmkey = 80003, shmsize = 120224`
	`shmkey = 80004, shmsize = 8192`
	`shmkey = 80005, shmsize = 0`

[-g]	서로 다른 게이트웨이에 동일한 RGWADDR, RGWPORTNO을 지정할 수 있도록 하는 옵션이다. 모든 게이트웨이 타입에 적용된다.

- 적용 환경

  Tmax 시스템이 설치된 운영 시스템 환경에서 사용할 수 있다.

- 예제
  - 다음은 /user1/tmax/temp 디렉터리에 있는 <basic> Tmax 환경 파일(텍스트 형태)을 컴파일하여 현재 디렉터리에 기본값으로 <tmconfig>라는 이진 Tmax 환경 파일을 생성하는 예제이다.

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

  - 다음은 디렉터리의 <ex_config>라는 텍스트 형태의 Tmax 환경 파일을 컴파일하여, /user1/tmax/bin 디렉터리에 <tmconfig>라는 이름으로 이진 Tmax 환경 파일을 생성하는 예제이다.

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

gst, tmboot, tmdown 명령어에 대한 자세한 내용은 각각 gst, tmboot, tmdown을 참고한다.

2.2. fdlc

필드 키 테이블을 컴파일하는 명령어이다. 필드 키 방식은 서버와 클라이언트 사이의 데이터 통신에서 구조체를 전송하는 방법처럼 전체 구조체의 항목을 전달하지 않고 필요한 항목만 전달한다. 항목을 전달하기 위해서는 각각의 항목을 구별할 수 있는 유일한 키가 있어야 한다. fdlc는 텍스트 형태로 정의된 필드키 테이블을 컴파일하여 필드 키를 생성하는 명령어이다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ fdlc {-a|c|d|u} [-f] [-h 헤더 파일명] {-i 필드 키 테이블 파일명}
```

```
[-jc|ji] [-o 결과물 이름] [-p 패키지명] [-x] [-r 출력파일명][-V]
```

항목	설명
{-a}	생성된 이진 형태의 파일에, 텍스트로 작성된 필드 키 테이블을 컴파일하여 필요한 부분을 추가하는 옵션이다. [-f] 옵션을 사용하여 대상 파일을 지정하며 기본 파일은 **<tmax.fdl>**이다. 중복되는 필드의 경우에 대해서는 새로운 값으로 대체된다.
{-c}	텍스트로 작성된 필드 키 테이블을 컴파일하여 이진 형태의 파일을 생성한다. 이미 이진 파일이 존재하면 새로운 내용을 대체한다. (기본값)
{-d}	생성된 이진 형태의 파일에 텍스트로 작성된 필드 키 테이블을 컴파일하여 필요한 부분을 삭제한다. [-f] 옵션을 사용하여 대상 파일을 지정하며 기본 파일은 **<tmax.fdl>**이다.
{-u}	생성된 이진 형태의 파일에 텍스트로 작성된 필드 키 테이블을 컴파일하여 필요한 부분을 수정하거나 추가하는 옵션이다. [-f] 옵션을 사용하여 대상 파일을 지정하며 기본 파일은 **<tmax.fdl>**이다.
[-f]	텍스트로 작성된 필드 키 테이블을 컴파일하여 필요한 부분을 추가, 수정, 삭제하는 경우 대상 파일을 지정하는 옵션이다.
[-h 헤더 파일명]	헤더 파일명을 다른 이름으로 변경할 때 사용하는 옵션이다. [-h] 옵션 없이 컴파일하면 FDL 헤더 파일명은 **<필드 키 테이블명_fdl.h>**이다.
{-i 필드 키 테이블 파일명 }	클라이언트 프로그램과 서버 프로그램에서 사용될 필드 키 테이블을 정의한 파일을 지정한다. 필수 옵션이며 경로와 함께 지정할 수 있다.
[-jc|ji]	WebT에서 사용되는 것으로서 생성될 필드 정의 클래스의 형태를 지정한다. • ji : 인터페이스 형태의 필드 정의 클래스 파일을 생성한다. • jc : 클래스 형태의 필드 키 정의 Java 파일을 생성한다.
[-o 결과물 이름]	컴파일 결과물을 다른 이름으로 변경하고자 할 때 사용하는 옵션이다. [-o] 옵션 없이 컴파일하면 fdl 파일명은 **<tmax.fdl>**이다.
[-p 패키지명]	WebT에서 사용되는 옵션으로, 생성된 필드 정의 클래스의 패키지명을 주어진 값으로 설정한다.
[-x]	반드시 [-a] 옵션과 함께 사용되어야 하며 이 옵션이 사용되는 경우에는 중복되는 필드에 대해서 원본값을 유지한다.

항목	설명
[-r 출력파일명]	FDL 파일을 텍스트로 출력파일명 파일로 환원하는 옵션이다. 반드시 [-f *fdl*명] 옵션과 함께 사용되어야 한다. 다음 명령으로 fdl 파일을 텍스트 파일로 디코딩할 수 있다. `fdlc -f fdl명 -r 출력파일명`
[-V]	실행 파일의 버전을 확인할 수 있다.

- 적용 환경

  UNIX 운영 시스템과 MS-DOS 운영 시스템에서 지원된다.

  서버와 클라이언트가 UNIX와 Windows, Intel x86 계열과 같이 다를 경우 바이트 오더링 차이로 문제가 발생할 수 있기 때문에 각각의 플렛폼에서 fdlc를 실행해야 한다.

- 예제
  - 다음은 [-jc|ji] 옵션을 설정해서 생성한 Java 파일의 예이다. **'number'** 값으로 사용할 수 있는 값의 범위는 0 ~ 2^24-1(0~16777215) 이다.

    <demo.f>

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

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

  - 다음은 생성된 <demo_fdl.java>의 내용이다.

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

  - 다음은 현재 디렉터리에 있는 'demo.f' 필드 키 테이블 파일을 fdlc로 컴파일하는 명령이다. 컴파일 후 <tmax.fdl>과 <demo_fdl.h> 파일이 생성된다. <demo_fdl.h>는 필드 키가 정의되어있는 헤더 파일이다.

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

필드 키

서버 / 클라이언트 데이터 통신에서 필드 키 버퍼를 사용하여 데이터를 보내는 경우 FDL 방식을 사용한다. FDL(Field Definition Language)은 필드 키 버퍼에 2개의 쌍으로 되어 있는 식별자와 식별자에 대응하는 데이터 값을 저장하여 서로 다른 프로세스 간에 데이터를 교환하는 방식이다. 필드 키 버퍼에서 각각의 식별자는 데이터 타입과 중복되지 않는 식별 번호의 조합으로 만들어지는데 이 식별자가 **필드 키**이다.

애플리케이션에서는 필드 키 대신에 프로그램에 대한 판독성을 높이기 위한 필드명을 사용한다. 각각의 필드명은 실행 중에 필드 키 이진 파일을 참조하여 필드 키로 전환되어 버퍼에 저장된다.

FDL 방식은 해당 필드를 사용하는 프로그램의 변경없이도 필드의 타입(type)과 길이를 변경할 수 있고 각 필드의 길이를 가변적으로 사용할 수 있다. 데이터 타입은 일반 C 언어에서 제공되는 Char, Short, Integer, Long, Float, Double, String, Carray 등의 타입을 지원하며 필드명은 16자까지 사용 가능하다.

fdlc 컴파일러는 텍스트 형태로 정의된 필드 키 정의 파일을 컴파일하여 사용자 프로그램에서 참조되는 필드 키를 이진 파일을 생성한다. 사용되는 필드 키 정의 파일은 텍스트 파일이며 테이블 파일명은 <.f> 형식의 파일이어야 한다. 기본 결과물은 <tmax.fdl>과 <필드 키 정의 파일명_fdl.h>이다. 서버 프로그램은 생성된 헤더 파일을 참조하여 컴파일되어 필드의 데이터를 저장하거나 읽어올 수 있다. 클라이언트는 fdlc 명령어로 생성된 파일을 **FDLFILE**이라는 환경변수에 반드시 등록해야 한다.

필드 키 버퍼의 장점은 데이터의 독립성이다. 구조체 버퍼의 경우에는 사용되지 않는 필드가 존재하더라도 구조체 전체를 전송할 수 밖에 없지만 필드 키 버퍼의 경우에는 필요한 필드만을 선택하여 전송할 수 있다. 또한 개발의 생산성 향상을 위해 다양한 필드 키 조작 함수를 제공하고 있다.

1. fdl number 값이 16777216 이상일 경우에는 허용하지 않는다.
2. 필드 키 조작 함수의 자세한 내용은 "Tmax FDL Reference Guide"를 참고한다.

2.3. gst

이진 Tmax 환경 파일을 참조하여 서비스 테이블을 생성하는 데 사용되는 명령어이다. gst는 cfl 명령어에 의해서 생성된 이진 Tmax 환경 파일의 SERVER 절과 SERVICE 절을 참조하여 각 서버별로 서비스 테이블을 생성한다. 이 테이블은 서버 프로세스에서 제공하는 서비스 목록으로, 서버 프로그램과 함께 작성하여 프로그램이 컴파일될 때 함께 컴파일된다. 서버 프로세스가 동작할 때 서비스 위치를 찾는 데 사용된다.

gst 명령의 결과 파일은 지정된 TMAXDIR 디렉터리 하위에 svct 디렉터리에 **<서버명_svctab.c>**로 생성된다. TMAXDIR은 source config 파일의 NODE 절을 참고한다. 서버명은 SERVER 절에 등록된 서버명이며, 각 파일의 내용은 SERVICE 절에 등록된 해당 서버가 제공하는 서비스명이다.

Tmax 환경 파일에 등록된 서비스는 반드시 해당 서버의 서비스 테이블에도 등록되어야 한다. Tmax 환경 파일의 SERVER 절이나 SERVICE 절에 서버명, 서비스명, 서비스의 SVRNAME 등이 변경된 경우는 gst 명령을 사용해서 서비스 테이블과 서버 프로그램을 컴파일해야 한다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ gst [-f 이진 Tmax 환경파일명| -v server_name | -h | -n node_name | -V]
```

항목	설명
[-f 이진 *Tmax* 환경파일명]	참조할 이진 Tmax 환경 파일명을 설정한다. **cfl** 명령의 결과물로 **tmboot**와 **tmdown**에서도 참조되는 파일이다. 경로와 함께 지정될 수 있으며, 생략되면 기본값으로 **${TMAXDIR}/config**로 지정된 디렉터리 하위의 config 디렉터리에서 **tmconfig**를 참고한다.
[-v *server_name*]	서버명에 해당하는 서비스 테이블을 설정한다.
[-h]	명령어 도움말 옵션이다.
[-n *node_name*]	멀티 노드 환경에서 타 노드의 서버 프로세스의 서비스 테이블을 현재 노드의 ${TMAXDIR}/svct에 생성하도록 한다.
[-V]	실행 파일의 버전을 확인할 수 있다.

- 적용 환경

  Tmax 시스템이 설치된 운영 시스템 환경에서 사용할 수 있다.

- 예제

  다음은 /user1/park/tmax/bin 디렉터리의 <exconfig> 환경 파일을 참조하여 서버별로 서비스 테이블을 생성하는 예제이다.

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

  예를 들어 환경 파일을 다음 페이지와 같이 등록한 경우 /user1/park/tmax/svct 디렉터리에 <svr1_svctab.c>와 <svr2_svctab.c>라는 서비스 테이블이 생성된다.

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

cfl, tmboot, tmdown 명령어에 대한 자세한 내용은 cfl, tmboot, tmdown을 참고한다.

2.4. hkcfl

hkcfl은 텍스트 형태의 Host-link 환경설정 파일을 컴파일하여 이진 Host-link 환경설정 파일(hlinkcfg)을 만드는 명령어이다. 컴파일하는 중 에러가 발생하면 컴파일을 중단하고 정상적으로 컴파일이 완료되면 이진 환경설정

파일이 생성된다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
hkcfl [-h] [-p] [-o 이진 Host-link 환경설정 파일명] -i 텍스트 Host-link 환경설정 파일명
      -c 주석처리 문자
```

항목	설명
[-h]	온라인 도움말을 보여준다.
[-p]	Host-link 환경설정 파일의 컴파일 실행 내용을 화면에 출력한다.
[이진 *Host-link* 환경설정 파일명]	컴파일 결과물인 이진 Host-link 환경설정 파일명을 명시하는 데 사용된다. 경로와 함께 지정할 수 있으며, 경로가 지정되지 않은 경우 ${TMAXDIR}/config에 결과물이 생성된다. (기본값: 'hlinkcfg')
-i 텍스트 *Host-link* 환경설정 파일명	텍스트 형태의 Host-Link 환경 파일명을 명시하는 데 사용된다. 필수 옵션으로, 디렉터리가 지정되지 않았을 경우 ${TMAXDIR}/config로 설정된다. 경로와 함께 지정될 수 있으며, 소스 파일을 찾지 못한 경우에는 경고 메시지가 출력된다.
-c 주석처리 문자	텍스트 형태의 환경설정 파일의 주석 문자를 변경한다. 기본적으로는 '#' 문자가 주석 문자로 사용된다.

2.5. idlc

Entera의 conversion할 때 정의 파일을 실제 프로그래밍 가능한 소스(stub)로 변환해 주는 유틸리티이다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
idlc -inf  인터페이스명
     -sql  SQL 인터페이스명
     -dbtype  ora | tbr | syb
     -clang  c | vbasic | pbuilder | java | delphi | delphi6 | delphixe
     -xa
     -header  파일명
     -sstub  파일명
     -cstub  파일명
```

항목	설명
-inf 인터페이스명	DA 서버를 사용할 경우의 인터페이스명을 지정한다.
-sql *SQL* 인터페이스명	DA 서버와 TP 서버를 사용할 경우의 인터페이스 파일명이다.

항목	설명
-dbtype	DA 서버를 사용할 경우의 DB 타입을 선택한다. RPC가 지원하는 OS와 DB 종류는 Tmax Programming Guide (RPC)의 "제약사항"을 참고한다. • ora : Oracle • syb : Sybase • tbr : Tibero
-clang	생성할 클라이언트의 소스 Stub 파일의 언어 타입을 선택한다. • c • vbasic • pbuilder • java • delphi • delphi6 • delphixe
-xa	XA 서버로 생성할 경우 지정한다.
-header 파일명	생성할 header 파일명을 지정한다.
-sstub 파일명	생성할 서버 Stub 파일명을 지정한다.
-cstub 파일명	생성할 클라이언트 Stub 파일명을 지정한다.

- 예제

  다음은 서버 종류에 따른 사용 예제이다.

  ◦ DA 서버

  Oracle을 사용하고 dstest.sql을 업무로 사용하며 클라이언트는 C 언어를 사용할 경우 다음과 같이 옵션을 설정한다.

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

  ◦ TS 서버

  ▪ '.def' 파일을 사용할 경우

  TP 서버는 '.def' 파일을 사용할 경우 다음과 같이 옵션을 설정한다.

```
idlc -xa
```

```
     -clang c tstest.def
```

- DA 서버 사용할 경우

  TP 서버에서 호출할 DA 서버를 사용할 경우 다음과 같이 옵션을 설정한다.

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

◦ Function 서버

Function 서버는 TP 서버와 동일하게 옵션을 설정하여 수행한다.

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

2.6. mkcli

Tmax 클라이언트 모듈을 생성하는 명령어이다. 파라미터는 최대 1024자까지 사용할 수 있으며 이 명령어가 실행되면 cc 컴파일러에 의해 클라이언트 모듈이 생성된다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

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

항목	설명
{-o *outfile*}	클라이언트 모듈명을 outfile에 지정한다.
{-f *firstfiles*}	사용자 정의 파일을 지정하면 Tmax에서 제공하는 클라이언트 라이브러리보다 먼저 링크되어 컴파일된다. 사용자 정의 파일으로 보통 클라이언트 프로그램을 지정한다.
[-v]	verbose 모드로 컴파일되어 컴파일하는 과정이 모두 콘솔창에 출력된다.
[-l *lastfiles*]	사용자 정의 파일을 지정하면 Tmax에서 제공하는 클라이언트 라이브러리보다 나중에 링크되어 컴파일된다.

항목	설명
[(-32)|-64]	생성하려는 클라이언트 모듈이 32Bit 또는 64Bit인지를 지정한다. • [-32] : Tmax 라이브러리 경로가 **${TMAXDIR}/lib**로 지정된다. (기본값) • [-64] : Tmax 라이브러리 경로가 **${TMAXDIR}/lib64**로 설정된다.
[-V]	실행 파일의 버전을 확인할 수 있다.
[-u *UserCompileOption*]	mkcli 명령어가 자체적으로 설정한 플랫폼에 맞는 기본 컴파일 옵션을 제거하고 사용자가 지정한 컴파일 옵션으로 대체한다. 라이브러리 및 include 관련 옵션은 적용되지 않으므로 주의한다. 둘 이상의 옵션을 지정해야 할 경우 다음과 같이 설정한다. `-u opt1 opt2`

- 적용 환경

  Tmax가 설치된 운영 시스템 환경에서 사용할 수 있다.

2.7. mkacl

ACL(access control list)을 생성하는 명령어이다. 서비스 접근 권한 제어 (제 3단계 보안)는 사용자 그룹별로 이루어진다. 서비스는 접근 권한이 허용된 그룹에 해당되는 user만 접근이 가능하다. 해당 기능을 사용하기 위해서는 **group** 파일을 생성해야 하며, 해당 그룹에 속하는 user 파일이 있어야 한다. 그리고 서비스별 접근 가능한 사용자 그룹을 지정하는 **acl** 파일이 있어야 한다. acl 파일을 생성하기 위해서는 mkacl 명령어를 사용해야 한다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

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

항목	설명
[-a]	기존의 ACL 파일에 새로운 하나의 ACL 서비스를 추가하기 위한 옵션이다. ACL 파일이 존재할 경우 파일의 가장 끝에 새로운 서비스가 추가되며, 존재하지 않을 경우 새로운 ACL 파일이 생성된 후 추가된다.
[-d]	생성된 ACL 서비스를 삭제하기 위한 옵션으로 [-G], [-s], [-t] 옵션과 함께 지정해야 한다. ACL 파일에서 [-s] 옵션으로 지정한 ACL 서비스를 삭제하며, [-G] 옵션은 해당 서비스와 반드시 일치해야 한다.
[-G *group_name*]	[-s]로 지정한 서비스에 대해 접근이 허용된 그룹명을 지정한다. 반드시 **${TMAXDIR}/config/group**에 지정된 이름을 사용해야 한다.

항목	설명
[-t *type*]	ACL의 타입을 지정한다. 현재 'SERVICE'만을 지원한다.
[-s *service_name*]	ACL을 적용할 서비스명을 지정한다. [-G] 옵션으로 지정된 하나의 서버 그룹에 속한 사용자만 접근 가능하다.
[-h]	명령어 도움말 옵션이다.
[-V]	명령어를 지원하는 Tmax 버전 정보를 조회한다.

- 결과

  ${TMAXDIR}/config/acl 파일이 생성되고 형식과 내용은 다음과 같다.

```
service_name:type:group_id
```

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

멀티 노드에서 사용하는 경우에는 cfl의 [-A] 옵션 설명을 참고한다.

2.8. mkgrp

사용자 그룹을 생성하는 명령어이다. 서비스 접근 권한 제어(제 3단계 보안)는 사용자 그룹별로 이루어진다. 하나의 서비스는 서비스에 대해 접근 권한이 허용된 그룹에 해당되는 사용자만 접근이 가능하다. 따라서 이 기능을 사용하기 위해서는 그룹 파일을 생성해야 하며, 해당 그룹에 속하는 사용자 파일이 있어야 한다. 그리고 서비스별 접근 가능한 사용자 그룹을 지정하는 **acl** 파일이 있어야 한다. 그룹 파일을 생성하려면 mkgrp 명령어를 사용해야 한다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

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

항목	설명
{-a}	기존의 그룹 파일에 새로운 하나의 사용자 그룹을 추가하기 위한 옵션이다. 기존의 그룹 파일이 존재할 경우 파일의 맨 끝에 새로운 그룹이 추가되며, 존재하지 않을 경우 새로운 그룹 파일이 생성된 후 추가된다.
{-G *group_name*}	생성할 그룹의 이름을 지정한다. 다른 그룹과 중복되지 않도록 지정해야 한다.

항목	설명
{-g *group_id*}	생성할 그룹의 ID를 지정한다. 이 옵션도 다른 그룹과 중복되지 않도록 지정해야 한다.
[-d]	기존에 생성된 그룹을 삭제하기 위한 옵션이다. [-g], [-G] 옵션과 함께 지정해 주어야 하며, [-g] 옵션으로 지정한 ID의 사용자 그룹을 삭제한다.
[-h]	명령어 도움말 옵션이다.
[-V]	명령어를 지원하는 Tmax 버전 정보를 조회한다.

- 결과

  ${TMAXDIR}/config/group 파일이 생성이 되며 형식과 내용은 다음과 같다.

```
Group_name:x:Group_id
```

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

2.9. mkpw

암호 관리 명령어로 Tmax 사용자가 선택할 수 있는 보안 확인 메커니즘은 4가지가 있다.

1. 보안 사용 안함
2. 도메인 보안 확인
3. 사용자 보안 확인
4. 서비스 접근 제어

보안 확인 메커니즘을 사용하려면 사용자명과 암호를 **$(TMAXDIR)/config** 디렉터리의 **passwd** 파일에 등록해야 한다. mkpw는 passwd 파일을 관리하는 데 사용된다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

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

항목	설명
[-f *filename*]	passwd 파일명을 지정한다. 기본 passwd 파일은 **$(TMAXDIR)/config/passwd**이다.
[-a]	사용자를 추가하는 옵션이다.
[-d]	사용자를 삭제하는 옵션이다.
[-i]	사용자의 정보를 변경하는 옵션이다.
[-n]	새로운 패스워드 파일을 생성하는 옵션이다.
[-p]	사용자의 암호를 변경하는 옵션이다. 파라미터가 지정되지 않을 때 다음을 기본값으로 실행한다. `$ mkpw -f $(TMAXDIR)/config/passwd -p`
[-h]	사용법을 보여주는 옵션이다.
[-V]	실행 파일의 버전을 확인할 수 있다.
[-G *group_name*]	해당 사용자가 속하는 그룹명을 지정한다. 서비스 접근 제어 보안 기능을 사용할 경우 group 파일을 생성해야 하며, 해당 그룹에 속하는 사용자가 있어야 하고 user 파일을 생성해야 한다. user 파일은 **mkpw** 명령어를 이용하여 passwd 파일과 함께 생성된다. passwd 파일을 생성할 경우 [-G] 옵션을 사용하여 이전 버전과는 다른 방식으로 생성해야 한다. ACL로 지정된 서비스를 호출하기 위해서는 하나의 그룹에 속해야 하기 때문이다. [-G] 옵션을 사용하여 그룹명을 지정할 경우 반드시 group 파일에 지정된 그룹 중 하나를 선택해서 지정해야 하며, [-u] 옵션과 함께 지정한다.
[-u *uid*]	해당 사용자의 유일한 ID를 지정한다. [-G] 옵션과 함께 지정하며, 제 3단계 보안인 서비스 접근 제어를 사용할 경우에만 지정한다.

- 결과

  다음은 [-G], [-u] 옵션을 사용했을 경우에 대한 passwd 파일과 user 파일의 형식 및 예제이다.

  - passwd 파일

    형식은 다음과 같다.

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

    <passwd 파일 예제>

    ```
    starbj1:UnQGcdDkNqXNc:1:1:starbj1:x:x
    starbj2:mPLY7VZtNvRXs:2:2:starbj2:x:x
    starbj3:aiu6Mt36rqwe6:3:3:starbj3:x:x
    ```

```
starbj4:vVdS9naV02jA.:4:4:starbj4:x:x
starbj5:568kCzyzYXriQ:5:5:starbj5:x:x
starbj6:ouKrHf/89QMW6:6:6:starbj6:x:x
starbj7:Mx8PaESrqWR4I:7:7:starbj7:x:x
starbj8:LL59popHJp59U:8:8:starbj8:x:x
starbj9:RG/S5BetAPeFs:9:9:starbj9:x:x
starbj10:Ebbzv1Ecx0abE:10:10:starbj10:x:x
```

◦ user 파일

형식은 다음과 같다.

```
user 파일:username:user_id:group_id
```

<user 파일 예제>

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

- 적용환경

Tmax가 설치된 운영 시스템 환경에서 사용할 수 있다.

2.10. mksvr

Tmax 서버 모듈을 생성하는 명령어이다. mksvr를 사용하여 서버 모듈을 생성하는 경우 Tmax의 환경 파일에 서비스를 등록하지 않아도 동적으로 서비스를 등록할 수 있다. 파라미터는 최대 1024자까지 사용할 수 있으며 이 명령어가 실행되면 cc 컴파일러에 의해 서버 모듈이 생성된다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

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

항목	설명

[-s {*@filename*|
service[,service...]
[:func]}]
	서비스명을 service에 지정한다.
	
	filename인 파일에 서비스명을 한 줄에 하나씩 기록하여 그 파일명을 '@' 뒤에 추가하여 서버 모듈을 생성할 수 있다. 서비스에 대한 func을 지정하면 서비스에 대한 요청이 올 때 그 처리를 func에서 할 수 있다. 서비스명을 세미콜론(;) 뒤에 계속해서 넣으려면 세미콜론 뒤에 스페이스를 사용해서는 안 된다. 옵션은 여러 번 지정할 수 있다.
	
	예)
	
	`-s TOUPPER -s TOLOWER`
	
	AUTOTRAN과 SVCTIME에 관한 내용을 함께 설정할 수도 있다. 항상 큰따옴표(" ")를 사용해야 한다.' -s TOUPPER -a 1 -T 10'으로 설정한 것과 동일한 것으로 간주된다. Function List 파일 안에서도 위와 동일하게 설정할 수 있다.
	
	예)
	
	`-s "TOUPPER 1 10"`

[-o *outfile*]	서버 모듈명을 outfile에 지정한다.
[-f *firstfiles*]	사용자 정의 파일을 지정하면 Tmax에서 제공하는 서버 라이브러리보다 먼저 링크되어 컴파일된다. 사용자 정의 파일로 보통 서버 프로그램을 지정한다.
[-v]	verbose 모드로 컴파일되어 컴파일하는 과정이 모두 콘솔창에 출력된다.
[-r *rmname*]	생성하려고 하는 서버 모듈과 연결되는 리소스 매니저를 지정한다. rmname은 **${TMAXDIR}/config/RM** 파일 안에 다음과 같은 형식으로 지정한다. `rmname:stub:XA libraries` Stub으로는 Tmax에서 제공하는 DBMS stub(liboras.so, libsybs.so, libinfs.so, libdb2s.so)이 설정된다. mksvr 명령어를 사용할 때 [-r *rmname*]을 지정하면 Stub들과 XA libraries들이 cc 컴파일할 때 링크되어 컴파일된다. [-r] 옵션이 지정되지 않은 경우 기본적으로 **libnodb.so**가 링크된다.
[-S *sdlfilename*]	SDL 버퍼를 사용하는 서버 모듈인 경우 sdl 파일(.s)의 object file을 sdlfilename으로 설정한다. [-S] 옵션이 지정되지 않은 경우 기본적으로 **${TMAXDIR}/lib/sdl.o**가 링크된다.
[-l *lastfiles*]	사용자 정의 파일을 지정하면 Tmax에서 제공하는 서버 라이브러리보다 나중에 링크되어 컴파일된다.

항목	설명
[-t *servertype*]	Tmax 환경 파일의 SERVER 절에 'SVRTYPE'으로 지정되는 값을 mksvr에서 설정한다. • [-t] 옵션이 지정되지 않은 경우 : 서버 타입을 STD라고 인식을 하고 cc 컴파일할 때 **<libsvr.so>**가 링크된다. • [-t] 옵션이 지정된 경우 : 설정된 Tmax 라이브러리가 cc 컴파일할 때 링크된다.
[(-32)|-64]	생성하려는 서버 모듈이 32Bit 또는 64Bit인지를 설정한다. • [-32] : Tmax 라이브러리 경로가 **${TMAXDIR}/lib**로 지정된다. (기본값) • [-64] : Tmax 라이브러리 경로가 **${TMAXDIR}/lib64**로 설정된다.
[-V]	실행 파일의 버전을 확인할 수 있다.
[-a *autotran*]	동적으로 등록한 서비스의 AUTOTRAN 유무를 설정한다. • 0 : NO • 1 : YES
[-T *svctime*]	동적으로 등록한 서비스의 SVCTIME를 설정한다. • 0 : 무한대로 SVCTIME을 설정한다. • 0보다 큰 수 : 해당 값만큼 SVCTIME을 설정한다.
[-u *UserCompileOption*]	mksvr 명령어가 자체적으로 설정한 플랫폼에 맞는 기본 컴파일 옵션을 제거하고 사용자가 지정한 컴파일 옵션으로 대체한다. 라이브러리 및 include 관련 옵션은 적용되지 않으므로 주의한다. 둘 이상의 옵션을 지정해야 할 경우 다음과 같이 설정한다. `-u opt1 opt2`
[-d]	mksvr 명령으로 빌드시에 기존의 svctab.c을 참조하지 않고 서비스를 빌드할 수 있도록 한다. 기존 환경 설정의 영향을 받지 않고 독립적으로 서비스를 동적 등록할 때에 사용한다.

- 적용 환경

  Tmax가 설치된 운영 시스템 환경에서 사용할 수 있다. Tmax 3.8.16 버전부터 Windows NT와 Windows 2000 환경에서 사용 가능하다.

- 주의사항

  mksvr는 COUSIN이나 BACKUP 서버 그룹 형태의 서버 모듈에 서비스를 동적으로 등록하는 경우 사용에 제약이 따른다. COUSIN이나 BACKUP 서버 그룹의 경우 여러 노드에 등록된 서비스의 정보를 관리해야 하기 때문에 mksvr로 특정 서비스를 동적으로 등록한 경우 이 서비스를 해제할 수 없다.

  일반 서버 그룹에서는 해당 서버를 **tmdown** 후 **mksvr**로 재등록하면 서비스의 이동과 삭제가 가능하다.

그러나 COUSIN/BACKUP 서버 그룹에서는 등록된 서비스 해제를 허용하지 않기 때문에 서비스의 이동과 삭제가 불가능하다.

mksvr를 통한 서비스 동적 등록을 할 경우에는 다음의 사항을 고려하여 사용해야 한다.

- 해당 서버의 서버 그룹 유형
- 해당 서비스의 기 등록 여부

2.11. racd

멀티 노드로 분산된 환경에서 각 노드를 중앙 관리하기 위한 명령어이다. racd는 여러 노드를 하나의 도메인으로 Tmax 시스템을 구축한 경우 한 노드에서 Tmax 시스템을 집중 관리하기 위해 각각의 노드에서 미리 기동되는 데몬 프로세스이다. Tmax 시스템을 관리하는 노드에서는 racd를 실행하지 않아도 된다.

racd는 도메인 내의 한 노드에서 **tmadmin**을 통하여 전체 노드를 관리하거나 또는 **cfl**로 환경 파일을 도메인 내의 모든 노드에 동일한 내용으로 적용 가능하도록 처리한다.

아래는 racd의 특징에 대한 추가 설명이다.

- racd를 통해 tmdown, tmboot 명령어를 실행한 뒤 종료할 때까지 기본 60초를 대기하며, clh와 tdl 요청은 완료될 때까지 무제한 대기한다.
- racd에서 명령 실행 중에 타임아웃이나 에러가 발생하면 cfl, tmboot, tmdown, racdr 유틸리티에서 이와 관련된 내용을 출력한다.
- racd를 통해 실행한 cfl 에서 SIGPIPE가 발생하더라도 이를 무시한다.

IP 버전이 IPv6이거나 InfiniBand의 SDP(Socket Direct Protocol)일 경우에는 환경 파일에 다음을 반드시 지정해야 한다.

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

만약 racd를 -k 옵션과 함께 사용한다면 환경변수에 다음을 지정해야 한다.

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

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ racd [-d] [-f 이진 Tmax 환경파일명] [-h] [-k] [-i filename] [-l label]
       [-P umask] [-V] [-t wait_time]
```

항목	설명
[-d]	디버그 모드로 racd를 수행할 때 설정한다.
[-f 이진 *Tmax* 환경파일명]	참조할 이진 Tmax 환경 파일명을 설정한다. cfl의 결과로 **tmboot**와 **tmdown**에서도 참조되는 파일이다. 경로와 함께 지정될 수 있으며, 이 옵션이 생략되면 기본적으로 TMAXDIR로 지정된 디렉터리 하위의 config 디렉터리에서 **tmconfig**를 참고한다. tmboot, tmdown 명령어의 자세한 내용은 tmboot, tmdown의 설명을 참고한다.
[-h]	명령어 도움말 옵션이다.
[-k]	이진 Tmax 환경 파일의 참조 여부를 설정한다. 옵션을 지정하면 이진 Tmax 환경 파일을 참조하지 않는다. 보통 racd는 이 옵션을 이용하여 실행한다. (passive listen mode) 만약 IP 버전이 IPv6이거나 InfiniBand의 SDP(Socket Direct Protocol)일 경우에는 이 옵션을 사용할 경우 환경 파일을 참조하지 않기 때문에 환경변수에 'TMAX_RAC_IPV6=IPV6' 또는 'TMAX_RAC_IPV6=SDP'를 반드시 지정해야 한다.
[-i *filename*]	하나의 물리적 머신에서 여러 논리적 노드를 정의할 때 사용한다. 논리 노드의 NODETYPE이 SHM_RACD일 경우 각 논리 노드당 하나씩의 racd를 실행해야 하며 RACPORT는 모두 달라야 한다. racd를 실행하기 전에 환경변수의 TMAX_RAC_PORT 변수를 설정해야 하는데, 논리 노드의 수가 많을 경우 이를 일일이 모두 변경할 수 없으므로 TMAXHOME, TMAXDIR, TMAX_RAC_PORT가 정의된 파일명을 지정할 수 있다. 자세한 내용은 예제를 참고한다. 논리 노드 설정에 대한 자세한 내용은 Tmax Administration Guide의 "NODE 절" 에서 HOSTNAME 항목을 참고한다.
[-l *label*]	Label은 파일 내에 등록된 환경 정보의 구분자이다. 2개 이상의 시스템 정보를 하나의 파일에 등록할 경우에 각각의 시스템을 구별할 수 있는 값이다.
[-P *umask*]	racd를 통해 기동되는 프로세스의 경우 umask를 사용자가 설정하여 원하는 권한의 파일을 생성할 수 있도록 한다.
[-V]	실행 파일의 버전을 확인할 수 있다.
[-t *wait_time*]	racd를 통해 tmdown, tmboot 명령어를 실행한 뒤 종료시까지의 대기시간을 조절하는 옵션이다. 음수를 설정하면 무제한 대기하며, 0은 지정이 불가하다. 1부터 지정한 값만큼 대기하게 된다.

- 적용 환경

  Tmax가 설치된 운영 시스템 환경에서 사용할 수 있다.

- 예제
  - 다음은 racd에 [-P] 옵션을 지정하여 원하는 umask를 설정하는 예제이다.

    <tmax.racd>

```
[tmaxs1]
TMAXHOME = /user2/starbj81/tmax32
TMAXDIR = /user2/starbj81/tmax32
TMAX_RAC_PORT = 3333

[NODE1]
TMAXHOME = /user2/starbj81/tmax32
TMAXDIR = /user2/starbj81/proj1
TMAX_RAC_PORT = 4335

[NODE2]
TMAXHOME = /user2/starbj81/tmax32
TMAXDIR = /user2/starbj81/proj2
TMAX_RAC_PORT = 4337
```

    1. 환경 파일을 작성한다.

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

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

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

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

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

*SERVICE
TOUPPER    SVRNAME = svr2
```

    2. 리모트 노드에 racd를 기동한다.

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

    3. HOST 노드에서 전체 Tmax를 기동한다. (tmboot)

4. RMT 노드의 ULOGDIR에 svr2.out의 파일 권한을 확인한다.

◦ 다음은 NODE1의 RACD를 실행시키는 예제이다.

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

◦ 다음은 환경 파일을 참조하지 않고 다른 명령어(tmboot)에서 사용한 정보만 이용하는 예제이다.

```
$ racd -k
```

2.12. racdr

멀티 노드, 멀티 도메인으로 분산된 환경에서 각 노드에 명령 수행, 파일 전송을 수행하기 위한 명령어이다. 관리를 원하는 노드에는 racd 데몬이 기동이 되어 있어야 한다. 환경설정 파일을 일정한 형식에 맞춰서 미리 작성을 해야 한다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ racdr [-f racdr 환경파일명] [-n 노드명]
        [[[[-W 셸프로그램 | -w] -e] -c 명령어] | [-s 파일명]
        [-d 저장경로] [-r]] [-h] [-V]
```

항목	설명
[-f *racdr* 환경파일명]	다른 도메인의 노드 정보를 저장한 환경 파일을 지정한다. 별도로 지정하지 않으면 현재 디렉터리의 racdr.cfg 또는 ${TMAXDIR}/config/racdr.cfg 파일을 사용한다.
[-n 노드명]	환경 파일에서 정의한 노드이름을 지정한다. 모든 요청들은 해당 노드의 racd 로 전달되어 수행된다.
[-W 셸프로그램]	[-e], [-c] 옵션을 통해 명령을 실행할 때 지정한 셸 프로그램을 통해서 실행되도록 한다. ksh, tsh, bash 등의 사용하고자 하는 셸 프로그램 이름을 인자로 지정한다.
[-w]	[-e], [-c] 옵션을 통해 명령을 실행할 때 셸 프로그램을 통해서 실행되도록 한다. 기본적으로 사용되는 셸은 "sh" 프로그램이다. 다른 셸 프로그램을 사용하려면 이 옵션 대신 [-W] 옵션을 사용한다. 명령의 파라미터에 와일드카드 문자를 사용하는 경우에는 와일드카드 문자 그대로 racd로 전달하기 위해서 반드시 명령 또는 명령행의 파라미터를 작은따옴표('')로 감싸줘야 한다.

항목	설명
[-e]	[-c] 옵션에서 허용하지 않는 명령들을 실행할 수 있도록 한다. [-c] 옵션 앞에 지정해야 한다. 명령이 실행될 때 해당 노드에서 셸 프로그램에 의해 명령이 수행될 필요가 있을 경우에는 [-w] 옵션을 함께 사용한다.
[-c 명령어]	racd에서 명령을 실행한다. [-c] 옵션 이후의 파라미터들은 모두 명령행의 파라미터로 간주한다. 따라서 racdr의 다른 옵션들을 지정할 때 반드시 [-c] 옵션 앞에 지정해야 한다. 명령은 tmboot, tmdown, cfl, gst 만 허용된다. 그 외의 명령이나 셸 스크립트를 수행하려면 [-e] 옵션을 지정해야 한다. **[주의]** tmboot, tmdown의 경우 racd에서 명령을 실행할 때 명령행 파라미터의 끝에 내부적으로 사용하는 옵션을 추가한다. 만약 반드시 인자를 지정해야 하는 옵션을 마지막 파라미터로 사용하면서 인자를 지정하지 않는 경우에 의도하지 않는 동작을 할 수 있으므로 주의해야 한다. 예를 들어 "racdr -c tmboot -d" 명령을 수행하면 원래는 tmboot의 잘못된 파라미터를 사용한 것([-d] 옵션은 인자를 반드시 입력해야 한다)이므로 에러가 출력되는 것이 정상인데, racd에 의해 수행될 때는 [-d] 뒤에 추가된 옵션에 의해서 이상한 값을 처리하여 명령이 실행될 수 있다. 따라서 명령어를 입력할 때 파라미터 입력에 세심한 주의가 필요하다.
[-s 파일명]	파일을 해당 노드로 전송한다. 반드시 [-d] 옵션과 함께 사용해야 한다. 저장될 위치에 이미 파일이 존재하는 경우, 해당 파일의 이름을 변경하고 전송된 파일을 기존 파일 이름으로 저장한다. 기존 파일의 이름은 뒷부분에 현재의 "_년월일시분초"가 추가되며, 같은 이름의 파일이 또 존재하면 그 뒤에 "_번호"를 추가한다. 파일 전송 도중 실패하면 전송중이던 파일은 삭제하고 기존 파일을 복원한다. 만약 기존 파일의 백업을 원하지 않는다면 [-r] 옵션을 함께 지정한다.
[-d 저장경로]	파일이 저장될 경로를 지정한다. 경로는 최대 3개까지 지정 가능하다. 경로는 환경 파일에서 설정하며, dest_no는 설정한 디렉터리의 번호를 지정한다. 첫 번째 경로(dest1)는 1, 두 번째 경로(dest2)는 2, 세 번째 경로(dest3)는 3을 지정하면 된다.
[-r]	파일이 저장될 노드에 같은 이름의 파일이 이미 존재하더라도 기존 파일의 백업을 생성하지 않는다.
[-h]	Usage 화면을 출력한다.
[-V]	실행 파일의 버전을 확인할 수 있다.

- 적용 환경

  racdr과 racdr 환경 파일이 있으면 실행 가능하다.

- 주의사항

◦ 명령을 실행할 때 명령 인자의 파라미터에 반드시 -n 옵션을 설정해서 해당 노드만 수행하도록 해야 한다.

◦ 입력을 요구하는 명령은 처리할 수 없다. 입력을 요구하지 않는 방식으로 수행되도록 명령 인자를 설정해야 한다.

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

◦ 실행 화면 결과는 해당 명령이 모든 수행을 마치고 종료될 때 클라이언트로 전달됨을 유의한다.

◦ 실행 화면 결과 중 STDOUT과 STDERR 출력은 실제 화면과 다른 순서로 출력될 수 있음을 유의한다.

◦ racdr 수행 결과에서 tmboot의 STDERR 내용은 출력되지 않음을 유의한다.

◦ 명령이 실행되는 중에는 racd는 다른 요청을 처리할 수 없음을 유의한다.

- 예제

◦ 다음은 3개의 노드를 관리하는 racdr 환경설정 파일 예제이다.

<racdr.cfg>

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

◦ /app/svr2 파일을 tmnode1 노드의 /tmax/appbin 디렉터리로 파일을 전송한다.

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

◦ tmnode1 노드에 cfl 명령을 수행한다.

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

◦ tmnode1 노드에 tmboot 명령을 수행한다.

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

◦ tmnode1 노드에서 특정 서버를 종료한다.

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

2.13. rcakill

RCA를 종료하고자 하는 경우 RCA가 사용하는 자원을 제거하기 위하여 사용한다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

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

항목	설명
[-h]	도움말을 조회한다.
[-p *pid*]	pid를 통한 특정 RCAH에 대한 상태 정보를 지운다.
[-n *rcah_no*]	RCAH 번호를 통한 특정 RCAH에 대한 상태 정보를 지운다.
[-k *shmkey*]	RCA가 사용하는 공유 메모리 키값의 정보를 지운다.

2.14. rcastat

RCA는 Tmax 시스템 입장에서는 클라이언트 프로세스로 관리되기 때문에 Tmax 시스템 관리 툴인 tmadmin으로 모니터링이 가능하다. Tmax에서는 RCA를 모니터링하기 위한 **rcastat**이라는 별도의 툴을 제공한다. 툴을 사용해서 관리자는 현재 RCA의 설정 내용을 확인할 수 있으며 현재 RCA에 접속한 클라이언트의 수 등을 모니터링 할 수 있다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

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

항목	설명
[-h]	도움말을 조회한다.
[-p *pid*]	pid를 통한 특정 RCAH에 대한 상태 정보를 출력한다. pid가 지정되지 않는다면 RCA에서 동작하고 있는 모든 RCAH에 대한 정보를 출력한다.
[-n *rcah_no*]	RCAH 번호를 통한 특정 RCAH에 대한 상태 정보를 출력한다. rcah_no가 지정되지 않는다면 RCA에서 동작하고 있는 모든 RCAH에 대한 정보를 출력한다.

항목	설명
[-k *shmkey*]	RCA가 사용하는 공유 메모리 키값을 설정한다. 공유 메모리 키값이 지정되지 않는 경우에는 사용자 시스템 환경변수에서 설정된 RCA_SHMKEY 값을 읽어 들이며. 만약 시스템 환경변수에도 설정되어 있지 않다면 기본값인 74565가 사용된다. 잘못된 키값을 입력하는 경우에는 올바른 정보를 출력할 수 없다.

- 예제

```
-----------------------------------------------
rca_dir: /user/tmax/server/
rca_mode: Local
rca_port: 8123
rcal_name: rca
rcal_pid: 1722
rcah_name: rcah
shmkey: 74565,  shmsize: 4832
#rcah: 2,  #thread per rcah: 60
-----------------------------------------------
rcah_no     pid    #client
-----------------------------------------------
    0       1724         0
    1       1723         0
-----------------------------------------------
```

출력 항목명	설명
rca_dir	RCAH 실행 파일과 통신용 파이프가 생성될 디렉터리이다.
rca_mode	현재 사용되고 있는 모드 정보이다.
rca_port	사용 가능한 포트 번호이다.
rcal_name	RCAL의 이름이다.
rcal_pid	RCAL의 pid이다.
rcah_name	RCAH의 이름이다.
shmkey	RCA에서 사용하고 있는 공유 메모리 키값이다.
shmsize	공유 메모리 크기이다.
#rcah	사용되고 있는 RCAH의 개수이다.
#thread per rcah	하나의 RCAH당 기동되어 있는 스레드 개수이다.
rcah_no	RCA 내부적으로 관리되는 번호이며 각각의 RCAH별로 별도의 값이다.
pid	RCA에 기동되어 있는 RCAH별 pid이다.
#client	클라이언트의 수이다.

명령어를 사용해서 조회되는 시스템 설정 정보는 다음과 같다.

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

2.15. sdlc

서버 및 클라이언트 사이의 데이터 통신에 사용하는 방식은 여러 가지가 있지만 그 중에서 구조체(Struct)를 사용할 경우에는 해당 구조체를 Tmax 시스템에서 인식할 수 있어야 한다. sdlc는 이런 구조체를 정의한 파일을 컴파일하는 명령어이다.

SDL(Structure Data Language)은 Tmax에서 정한 표준 구조체 데이터 형식이다. 이기종 노드 간 통신할 때 Integer, Float, Double 등의 데이터 타입과 관련해서 다음의 문제를 갖는다.

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

SDL은 이러한 문제를 해결하기 위해서 하나의 표준형을 정한 것으로 Integer, Float, Double의 통신을 지원하여 자유로운 통신 타입을 지원한다.

클라이언트 프로그램과 서버 프로그램에서 사용되는 구조체는 동일해야 한다. 구조체가 변경되었을 경우 클라이언트와 서버별로 구조체가 sdlc에 의하여 재컴파일되어야 하며, 새롭게 생성된 <구조체 파일명_sdl.c>는 서버 프로그램과 함께 반드시 재컴파일해야 한다. 사용되는 구조체 파일명은 <.s> 형식의 파일이고 결과물은 <구조체 파일명_sdl.c> 형식이다. 구조체 파일에는 서버 및 클라이언트 사이의 통신에 사용되는 버퍼 타입 구조체가 정의되고 이 구조체 이름은 통신 버퍼로 **tpalloc()**, **tpcall()**, **tpacall()** 등에 사용된다. sdlc 명령어로 생성된 파일은 클라이언트에 **SDLFILE**이라는 환경변수에 반드시 등록해야 한다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ sdlc [-c] [-h 헤더 파일명] {-i 구조체 파일명} [-o 결과물 이름]
       [-s] [-v VIEW 정의 파일] [-32] [-4vb|-4dp] [-f] [-V]
```

항목	설명
[-c]	클라이언트를 위해 사용되는 옵션이다. 클라이언트 프로그램에서 사용하는 구조체 파일을 sdlc로 컴파일하면 구조체의 각 데이터가 표준 데이터형으로 변환되어 필요한 정보의 이진 형태의 파일이 생성된다. <구조체 파일명.sdl>은 클라이언트 프로그램을 실행할 때 표준 통신형으로 데이터가 송수신되는 데 사용된다. [-c] 옵션을 사용하는 경우 여러 개의 구조체 파일들을 함께 컴파일할 수 있다(예: sdlc -c -i *.s 또는 sdlc -c -i demo.s sam.s abc.s). 생성되는 sdl 파일명은 <demo.sdl>이며 다른 이름을 사용하려면 [-o] 옵션을 이용하여 원하는 파일명을 적는다. [-c] 옵션을 사용하지 않는 경우 sdlc는 구조체 파일을 컴파일하여 구조체의 각 필드를 표준 통신형인 SDL 형식에 맞춰 암호화(encoding:노드 데이터형을 표준 데이터형으로 변환), 복호화 (decoding:표준 데이터형을 노드 데이터 형으로 변환)하는 프로그램이 생성된다. 서버 프로그램과 함께 컴파일되어 표준 SDL 형식의 데이터형으로 통신이 이루어진 <구조체 파일명_sdl.c> 파일이 생성된다.
[-h 헤더 파일명]	[-h] 옵션 없이 컴파일하면 sdl 헤더 파일명은 <구조체 파일명_sdl.h>이다. 헤더 파일명을 변경할 때 사용하는 옵션이다.
{-i 구조체 파일명}	서버/클라이언트 프로그램에서 사용하는 구조체를 정의한 파일을 설정한다. 필수 옵션이며 경로와 함께 지정할 수 있다. 구조체 이름은 16자까지 설정할 수 있다.
[-o 결과물 이름]	구조체 파일이 하나인 경우 [-o] 옵션 없이 컴파일하면 sdl 파일명은 <구조체 파일명.sdl>이다. 파일명을 다른 이름으로 변경할 때 사용하는 옵션이다.
[-s]	서버 프로그램에서 사용되는 옵션으로 구조체 정의 이진 파일을 생성한다. 구조체 정의 이진 파일은 [-c] 옵션을 사용한 경우에 생성되는 파일과 같다. 구조체 파일을 컴파일하여 구조체의 각 필드를 표준 통신형인 SDL 형식에 맞춰 암호화, 복호화하는 프로그램이 생성된다. 생성되는 파일명은 <구조체 파일명_sdl.c>이며 이 파일은 서버 프로그램과 함께 컴파일해야 한다. [-c] 옵션과는 배타적으로 사용되고 [-c] 옵션이 사용되지 않는 경우 기본값으로 사용된다.
[-v *VIEW* 정의 파일]	구조체 버퍼와 필드 키 버퍼 사이에서 형태 변환을 원하는 경우에 이 옵션을 사용하여 구조체 정의 이진 파일을 생성해야 한다. 구조체 버퍼의 데이터를 필드 키 버퍼에 저장하여 서비스를 요청하거나 그 반대로 필드 키 버퍼의 내용을 구조체 버퍼에 저장하여 서비스를 요청할 수 있다. 사용되는 함수는 **fbftos()**와 **fbstof()**가 있다. 자세한 설명은 예제를 참고한다.
[-32]	64Bit 환경에서 32Bit 라이브러리를 사용하기 위해서 구조체 파일을 32Bit로 컴파일한다.
[-4vb]	Visual Basic 인터페이스 파일을 생성한다.
[-4dp]	Delphi 인터페이스 파일을 생성한다.

항목	설명
[-f]	[-4vb] 또는 [-4dp] 옵션과 함께 사용해야 하며 Visual Basic 또는 Delphi 인터페이스 파일명을 지정한다.
[-V]	실행 파일의 버전을 확인할 수 있다.

- 적용 환경

  UNIX 운영 시스템과 MS-DOS 운영 시스템에서 지원된다.

- 예제
  - VIEW 정의 파일의 구조는 다음과 같다.

    <demo.v>

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

항목	설명
type	구조체 안의 멤버 변수의 데이터 타입이다.
Cname	멤버 변수명이다.
fldkey	지정한 멤버 변수와 매핑되는 필드 키이다.
count	구조체에서 저장할 수 있는 최대의 필드 순번이다.
flag	현재 사용하지 않는다.
size	string 타입일 경우 배열의 크기이다.
null	초기화할 때의 값이다.

    구조체 정의 이진 파일은 다음과 같은 명령어로 생성될 수 있다.

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

  - 다음은 서버를 위하여 현재 디렉터리에 있는 <demo.s> 구조체 파일을 sdlc로 컴파일하는 명령에 대한 예제이다. 결과로 <demo.sdl>과 <demo_sdl.c>라는 파일이 생성된다. <demo_sdl.c>는 서버 프로그램을 생성하는 데 사용된다.

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

  - 다음은 클라이언트를 위하여 현재 디렉터리에 있는 <demo.s> 구조체 파일을 sdlc로 컴파일하는 명령에 대한 예제이다. 결과로 이진 데이터 파일인 <demo.sdl>라는 파일이 생성된다.

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

2.16. svcrpt

Tmax 시스템을 운용할 때 서비스 수행에 관련된 로그 기록을 분석하여 출력하는 명령어이다. 출력 내용은 수행된 서버 및 서비스의 이름과 횟수, 평균 서비스 수행시간이며 원하는 시간 범위 내에서 1시간 단위로 출력이 가능하다. 서비스 로그는 서버별로 CLOPT 절에 [-l] 옵션을 설정하면 5분 간격으로 $ULOGDIR에 <svclog.mmddyyyy>라는 이름의 파일이 저장된다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

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

항목	설명
[-T|(-N)|-A]	• [-T] : 서비스의 총 수행시간 순서로 결과를 정렬한다. • [-N] : 서비스 수행 횟수를 바탕으로 정렬한다. (기본값) • [-A] : 평균 처리시간으로 서비스 로그를 정렬한다.
[-s *svcname*]	출력을 원하는 서비스명을 설정한다.
[-v *svrname*]	출력을 원하는 서버명을 설정한다.
[-x]	서버 프로그램명, 최대 및 최소 처리시간 등 더 자세한 내용을 조회한다.
[-S]	서비스 요약을 조회한다.
[-d *mm:dd*]	서비스 로그 분석할 날짜를 입력한다.
[-c *column*]	출력하는 기본 컬럼을 입력한 숫자로 변경한다. (기본값: 5)
[-f *hh:mm:ss*]	분석을 시작하는 시간대를 입력한다.
[-t *hh:mm:ss*]	분석을 종료하는 시간대를 입력한다.
[-V]	실행 파일의 버전을 조회한다.
-i *logfile*	분석할 로그 파일명을 입력한다.

- 적용 환경

  Tmax가 설치된 운영 시스템 환경에서 사용할 수 있다.

- 예제
  - 다음은 'svclog.04152002'라는 로그 파일을 분석하여 1시간 단위로 결과를 표시하는 예제이다.

```
$ svcrpt -i svclog.04152002
```

◦ 다음은 'svclog.04152002'라는 로그 파일의 4월15일 오전에 수행된 account라는 서비스의 수행 횟수 및 평균 수행시간을 1시간 단위로 출력하는 예제이다.

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

2.17. tdlclean

run 디렉터리의 구버전 라이브러리 파일이나 불필요한 파일을 정리하는 명령어이다. 특히, [-m] 옵션 또는 [-M] 옵션을 사용하면 공유 메모리에서 지정한 동적 모듈이 완전히 삭제된다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ tdlclean [-p TDL 루트 디렉터리 경로] [-m 라이브러리명] [-M 함수명] [-b]
           [-d yyyymmddhhmi] [-D "n hour|day" [-N 개수] [-v | -V] [-h] [-f]
```

항목	설명
[-p *TDL* 루트 디렉터리 경로]	TDL 루트 디렉터리는 **${TDLDIR}** 또는 **${TMAXDIR}**을 사용한다. 루트 경로를 별도로 지정해서 실행하는 경우 [-p] 옵션을 이용하여 경로를 지정할 수 있다.
[-m 라이브러리명]	TDL 공유 메모리에서 지정한 라이브러리와 run 디렉터리에 파일을 삭제한다.
[-M 함수명]	공유 메모리에서 지정한 함수만 삭제하며, run 디렉터리의 관련 파일은 삭제하지 않는다. 단, VERSION=1로 설정된 경우는 [-m] 옵션과 동일하게 동작한다.
[-b]	TDL 환경 파일(tdl.cfg)에 BACKUP 파라미터가 지정되어 있더라도, 공유 메모리 파일 백업을 수행하지 않는다. [-m] 또는 [-M] 옵션과 함께 사용한다.
[-d *yyyymmddhhmi*]	지정한 시각(yyyymmddhhmi) 이전의 구버전 라이브러리 파일을 모두 삭제한다. [-m] 또는 [-M] 옵션과 함께 사용할 수 없다.
[-D "n hour|day"]	n은 시간(hour) 또는 일(day)을 설정한다. 지정한 시간 또는 일 이전의 구버전 라이브러리 파일을 모두 삭제한다. [-m] 또는 [-M] 옵션과 함께 사용할 수 없다.
[-N 개수]	지정한 개수(number)만큼의 구버전 라이브러리 파일을 남기고 나머지 구버전 파일을 모두 삭제한다. [-m] 또는 [-M] 옵션과 함께 사용할 수 없다.
[-v|-V]	버전 정보를 출력한다.
[-h]	사용법을 출력한다.
[-f]	run에 있는 라이브러리 정보를 힙 메모리에 모두 등록한 후 한번에 TDL 공유 메모리에 있는 최신 라이브러리의 함수들과 비교한다. [-f] 옵션 사용으로 성능향상을 기대할 수 있다. [-N], [-m], [-M] 옵션과 함께 사용할 수 없다.

- 예제
  - 다음은 구버전 라이브러리 파일을 삭제하는 예제이다.

```
$ tdlclean
```

  - 다음은 2009년 2월 1일 00시 00분 이전의 구버전 라이브러리 파일을 삭제하는 예제이다.

```
$ tdlclean –d 200902010000
```

  - 다음은 5일 이전의 구버전 라이브러리 파일을 삭제하는 예제이다.

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

  - 다음은 mylibrary를 TDL 공유 메모리에서 완전히 제거하고 구버전 파일들도 삭제하는 예제이다.

```
$ tdlclean -m mylibrary
```

2.18. tdlinit

TDL 공유 메모리 및 동적 모듈을 초기화하는 명령어이다. tdlinit은 설치될 때 단 한 번만 수행하며 Tmax를 부팅하기 전에 수행해야 한다. 멀티 노드 환경인 경우 마스터 노드에서만 수행 가능하다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ tdlinit [-p TDL 루트 디렉터리 경로] [-x export 함수 추출 스크립트 파일 경로] [-f]
          [-b] [-B 백업 파일 경로] [-i] [-v | -V] [-h]
```

항목	설명
[-p *TDL* 루트 디렉터리 경로]	TDL 루트 디렉터리는 **${TDLDIR}** 또는 **${TMAXDIR}**을 사용한다. 루트 경로를 별도로 지정해서 실행하는 경우 [-p] 옵션을 이용하여 경로를 지정할 수 있다.
[-x *export* 함수 추출 스크립트 파일 경로]	export 함수 추출을 위한 스크립트 파일 경로를 지정한다.
[-f]	공유 메모리가 이미 존재할 경우 강제로 초기화한다.
[-b]	백업 파일로부터 공유 메모리를 복구한다.
[-B 백업 파일 경로]	지정된 백업 파일로부터 공유 메모리를 복구한다.

항목	설명
[-i]	백업 복구 후 run 디렉터리를 체크한다. [-b] 또는 [-B] 옵션과 함께 사용한다.
[-v|-V]	버전 정보를 출력한다.
[-h]	사용법을 출력한다.

- 예제
  - 다음은 TDL 환경 파일(tdl.cfg)을 참조하여 공유 메모리 및 모듈을 초기화하는 예제이다.

```
$ tdlinit
```

  - 다음은 장비 장애 또는 재시작할 경우 백업 파일로부터 공유 메모리를 복구하는 예제이다.

```
$ tdlinit -b
```

- 참고

  VERSION=4로 설정한 경우에는 export 함수 추출을 위한 스크립트 파일이 반드시 필요하다.

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

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

2.19. tdlnm

VERSION=2 이상에서 지정한 라이브러리에 대한 자동 export될 함수 목록을 조회하는 명령어이다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ tdlnm [-p TDL 루트 디렉터리 경로] [-x export 함수 추출 스크립트 파일 경로]
        [-m 라이브러리명] [-v | -V] [-h]
```

항목	설명
[-p *TDL* 루트 디렉터리 경로]	TDL 루트 디렉터리는 **${TDLDIR}** 또는 **${TMAXDIR}**을 사용한다. 루트 경로를 별도로 지정해서 실행하는 경우 [-p] 옵션을 이용하여 경로를 지정할 수 있다.

항목	설명
[-x *export* 함수 추출 스크립트 파일 경로]	export 함수 추출을 위한 스크립트 파일 경로를 지정한다.
[-m 라이브러리명]	자동 export 함수 목록 조회를 위한 라이브러리명을 지정한다.
[-v|-V]	버전 정보를 출력한다.
[-h]	사용법을 출력한다.

- 예제

  다음은 mylibrary 파일의 export 함수 목록을 조회하는 예제이다.

```
$ tdlnm -m mylibrary
```

2.20. tdlrm

TDL을 더 이상 사용하지 않을 경우에 공유 메모리를 완전히 제거하기 위한 명령어이다. tdlrm을 할 경우 **tdlcall()**을 사용할 수 없다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ tdlrm [-p TDL 루트 디렉터리 경로] [-v | -V] [-h]
```

항목	설명
[-p *TDL* 루트 디렉터리 경로]	TDL 루트 디렉터리는 **${TDLDIR}** 또는 **${TMAXDIR}**을 사용한다. 루트 경로를 별도로 지정해서 실행하는 경우 [-p] 옵션을 이용하여 경로를 지정할 수 있다.
[-v|-V]	버전 정보를 출력한다.
[-h]	사용법을 출력한다.

- 예제

  다음은 TDL 공유 메모리를 삭제하는 예제이다.

```
$ tdlrm
```

2.21. tdlseqno

특정 모듈과 함수의 시퀀스 번호를 조회한다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ tdlseqno [-m module|-M library [-p TDL 루트 디렉터리 경로]] [-V] [-h]
```

항목	설명
[-m *module*]	해당 함수 이름을 지정한다.
[-M *library*]	해당 라이브러리 이름을 지정한다.
[-p *TDL* 루트 디렉터리 경로]	TDL 루트 디렉터리는 **${TDLDIR}** 또는 **${TMAXDIR}**을 사용한다. 루트 경로를 별도로 지정해서 실행하는 경우 [-p] 옵션을 이용하여 경로를 지정할 수 있다.
[-V]	버전 정보를 출력한다.
[-h]	사용법을 출력한다.

- 예제
  - ${TDLDIR} 또는 ${TMAXDIR}의 해당 모듈과 함수의 시퀀스 번호를 조회하는 예제이다.

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

  - 특정 디렉터리 아래의 해당 모듈과 함수의 시퀀스 번호를 조회하는 예제이다.

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

2.22. tdlshm

TDL 공유 메모리 정보를 조회하거나, 통계 모니터링 활성화 여부 및 모듈 활성화 여부를 설정하는 명령어이다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ tdlshm [-p TDL 루트 디렉터리 경로] [-r] [-S] [-n 노드명] [-m 함수명]
         [-M 라이브러리명] [-C] [-c start_index end_index] [-s e|d|r]
         [-u e|d] [-I 최소충돌횟수] [-v | -V] [-h]
```

항목	설명
[-p *TDL* 루트 디렉터리 경로]	TDL 루트 디렉터리는 **${TDLDIR}** 또는 **${TMAXDIR}**을 사용한다. 루트 경로를 별도로 지정해서 실행하는 경우 [-p] 옵션을 이용하여 경로를 지정할 수 있다.
[-r]	멀티 노드 환경에서 업데이트 동기화 중인 모듈 정보를 조회한다.
[-S]	동적 모듈 통계 정보를 조회한다. 통계 정보는 각 모듈별 수행시간 AVG/MIN/MAX와 CPU AVG/MIN/MAX가 조회된다.
[-n 노드명]	멀티 노드 환경에서 조회할 노드를 지정한다.
[-m 함수명]	조회할 함수명을 지정한다.
[-M 라이브러리명]	조회할 라이브러리명을 지정한다.
[-C]	전체 모듈에 대해 dlopen, dlsym을 수행한다.
[-c *start_index end_index*]	지정한 인덱스 범위의 모듈에 대해 dlopen, dlsym을 수행한다.
[-s e|d|r]	동적 모듈 통계 수집을 설정한다. • e : 활성화 • d : 비활성화 • r : 초기화
[-u e|d]	동적 모듈을 설정한다. • e : 활성화 • d : 비활성화 한 번에 하나의 모듈만 설정 가능하므로, 반드시 [-m] 또는 [-M] 옵션을 함께 사용해야 한다. • VERSION=1, VERSION=2인 경우 모듈은 전체 라이브러리에서 유일함으로 보장해야 하기 때문에 [-m] 옵션만 지정한다. 함수명이 전체적으로 유일해야 한다. • VERSION=3이고 라이브러리가 다를 경우 함수명은 중복되어도 상관없기 때문에 [-m], [-M] 옵션은 반드시 입력해야 한다. [-m] 옵션 없이 [-M] 옵션만 사용해서는 안 된다.
[-I 최소충돌횟수] (대문자 i)	각 모듈마다 hash collision에 의한 버킷을 검색할 때 비교횟수를 출력한다. 0 이상의 값을 설정할 수 있으며, 지정한 횟수 이상의 모듈에 대해서만 출력한다. [-p], [-m], [-M], [-r] 옵션과 함께 사용할 수 있다.
[-v|-V]	버전 정보를 출력한다.
[-h]	사용법을 출력한다.

- 예제
  - 다음은 기본적인 tdlshm 사용 예이다.

```
$ tdlshm -S TDLDIR = /home/jeffry/tmax LOGDIR = /home/jeffry/tmax/log/dlog
- BACKUP = /home/jeffry/tmax/log/dlog/tdl.bak VERSION = 2, SHMKEY = 0x90000,
- IPCPERM = 0750 MAXMODULES = 256, CURMODULES = 3, Global SEQNO = 45e27d28,
- MONITOR = Y MODE = SINGLE, DOMAINID = 1 Index = 125, Funcname = myfunction1,
- Libname = myfunction1, Seqno = 45e27d28, Active = Y Count = 0 SVC: Avg = 0.000,
- MinTime = 0.000, Maxtime = 0.000 CPU: Avg = 0.000, MinTime = 0.000,
- Maxtime = 0.000 Index = 126, Funcname = myfunction2, Libname = myfunction2,
- Seqno = 45e27d28, Active = Y Count = 0 SVC: Avg = 0.000, MinTime = 0.000,
- Maxtime = 0.000 CPU: Avg = 0.000, MinTime = 0.000, Maxtime = 0.000 Index = 127,
- Funcname = myfunction3, Libname = myfunction3, Seqno = 45e27d28, Active = Y
- Count = 0 SVC: Avg = 0.000, MinTime = 0.000, Maxtime = 0.000 CPU: Avg = 0.000,
- MinTime = 0.000, Maxtime = 0.000 $ tdlshm TDLDIR = /home/jeffry/tmax LOGDIR =
/home/jeffry/tmax/log/dlog
- BACKUP = /home/jeffry/tmax/log/dlog/tdl.bak VERSION = 2, SHMKEY = 0x90000,
- IPCPERM = 0750 MAXMODULES = 256, CURMODULES = 3, Global SEQNO = 45e27d28,
- MONITOR = Y MODE = SINGLE, DOMAINID = 1 Index = 125, Funcname = myfunction1,
- Libname = myfunction1, Seqno = 45e27d28, Active = Y Index = 126,
- Funcname = myfunction2, Libname = myfunction2, Seqno = 45e27d28, Active = Y,
- Index = 127, Funcname = myfunction3, Libname = myfunction3, Seqno = 45e27d28,
 Active = Y
```

◦ 다음은 TDL 통계 정보 조회를 위한 사용 예이다.

```
$ tdlshm -S TDLDIR = /home/jeffry/tmax LOGDIR = /home/jeffry/tmax/log/dlog
- BACKUP = /home/jeffry/tmax/log/dlog/tdl.bak VERSION = 2, SHMKEY = 0x90000,
- IPCPERM = 0750 MAXMODULES = 256, CURMODULES = 3, Global SEQNO = 45e27d28,
- MONITOR = Y MODE = SINGLE, DOMAINID = 1 Index = 125, Funcname = myfunction1,
- Libname = myfunction1, Seqno = 45e27d28, Active = Y Count = 0 SVC: Avg = 0.000,
- MinTime = 0.000, Maxtime = 0.000 CPU: Avg = 0.000, MinTime = 0.000,
- Maxtime = 0.000 Index = 126, Funcname = myfunction2, Libname = myfunction2,
- Seqno = 45e27d28, Active = Y Count = 0 SVC: Avg = 0.000, MinTime = 0.000,
- Maxtime = 0.000 CPU: Avg = 0.000, MinTime = 0.000, Maxtime = 0.000 Index = 127,
- Funcname = myfunction3, Libname = myfunction3, Seqno = 45e27d28, Active = Y
- Count = 0 SVC: Avg = 0.000, MinTime = 0.000, Maxtime = 0.000 CPU: Avg = 0.000,
- MinTime = 0.000, Maxtime = 0.000
```

◦ 다음은 [-s] 또는 [-u] 옵션 사용 예이다.

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

◦ 다음은 [-C] 또는 [-c] 옵션 사용 예이다.

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

2.23. tdlsync

TDL 공유 메모리와 백업 파일 동기화를 수행하는 명령어이다. 자동 백업을 사용하지 않을 경우 필요한 관리 툴이다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ tdlsync [-p TDL 루트 디렉터리 경로] [-B 백업 파일 경로] [-v | -V] [-h]
```

항목	설명
[-p *TDL* 루트 디렉터리 경로]	TDL 루트 디렉터리는 **${TDLDIR}** 또는 **${TMAXDIR}**을 사용한다. 루트 경로를 별도로 지정해서 실행하는 경우 [-p] 옵션을 이용하여 경로를 지정할 수 있다.
[-B 백업 파일 경로]	백업 파일 경로를 지정한다.
[-v|-V]	버전 정보를 출력한다.
[-h]	사용법을 출력한다.

- 예제

  다음은 TDL 환경 파일(tdl.cfg)의 'BACKUP' 파라미터에 지정된 파일로 공유 메모리를 백업하는 예제이다.

```
$ tdlsync
```

2.24. tdltrace

TDL 환경 설정 정보와 통계 정보를 조회한다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ tdltrace [-p 디렉터리 경로] [-P PID] [-V] [-h] [-c]
```

항목	설명
[-p 디렉터리 경로]	TDL 루트 디렉터리는 **${TDLDIR}** 또는 **${TMAXDIR}**을 사용한다. 루트 경로를 별도로 지정해서 실행하는 경우 [-p] 옵션을 이용하여 경로를 지정할 수 있다.
[-P *PID*]	특정 서버 프로세스의 정보를 조회한다.

항목	설명
[-V]	버전 정보를 출력한다.
[-h]	사용법을 출력한다.
[-c]	좀비 프로세스를 찾고 이와 관련된 정보를 지운다.

- 예제

```
$ tdltrace
```

```
$ tdltrace -P 20113
```

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

```
$ tdltrace -c
```

2.25. tdlupdate

지정한 동적 모듈을 업데이트하는 명령어이다. [-m] 옵션으로 라이브러리명을 반드시 지정해야 한다. 지정한 라이브러리가 이미 등록되어 있을 경우는 지정한 라이브러리를 업데이트하며 아직 등록되어 있지 않을 경우 새로 추가한다. 멀티 노드 환경인 경우 마스터, 슬레이브 노드에서 모두 수행 가능하다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ tdlupdate [-p TDL 루트 디렉터리 경로] [-r 디렉터리 경로]
            [-x export 함수 추출 스크립트 파일 경로] [-f] [-m 라이브러리명] [-b]
            [-c] [-l 파일명] [-i] [-Q seqno] [-C [cp|mmap|self]] [-v | -V] [-h]
```

항목	설명
[-p *TDL* 루트 디렉터리 경로]	TDL 루트 디렉터리는 **${TDLDIR}** 또는 **${TMAXDIR}**을 사용한다. 루트 경로를 별도로 지정해서 실행하는 경우 [-p] 옵션을 사용하여 경로를 지정할 수 있다.
[-r 디렉터리 경로]	tdlupdate를 실행 할 원격 노드의 TDLDIR을 지정한다. [-p] 옵션을 사용할 경우 두 개 이상의 TDL 루트 디렉터리를 가지게 되는데 이를 지정하는 용도로 사용한다.

항목	설명
[-x *export* 함수 추출 스크립트 파일 경로]	export 함수 추출을 위한 스크립트 파일 경로를 지정한다.
[-f]	VERSION=2에서 다른 라이브러리에 함수명이 이미 존재하더라도 강제로 업데이트 한다.
[-m 라이브러리명]	업데이트할 라이브러리명을 지정한다.
[-b]	TDL 환경 파일(tdl.cfg)에 BACKUP 파라미터가 지정되어 있더라도 공유 메모리 파일 백업을 수행하지 않는다.
[-c]	멀티 노드 환경에서 노드 사이 버전 동기화를 수행한다.
[-l 파일명]	업데이트할 목록을 파일로 작성하여 업데이트를 수행한다. 이때 전체 파일을 검사 후 이상이 없을 경우에만 업데이트를 수행한다. 이상이 있을 경우에는 rollback을 원칙으로 한다. 모듈의 구분자는 콤마(,) 또는 <Enter>이다.
[-i]	반영해야 하는 대상 파일이 존재하지 않더라도 중단하지 않고 계속 다음 파일을 update한다.
[-Q *seqno*]	현재 시간이 아닌 강제로 부여한 seqno를 사용한다.
[-C [cp|mmap|self]]	TDL 환경 파일(tdl.cfg)에 'COPY' 파라미터 설정보다 우선하여 파일 복사 방법을 변경한다. • cp • mmap • self
[-v|-V]	버전 정보를 출력한다.
[-h]	사용법을 출력한다.

- 예제
  - 다음은 <mylibrary.so> 파일을 업데이트하는 예제이다.

```
$ tdlupdate -m mylibrary
```

  - 다음은 여러 개의 모듈을 업데이트하는 예제이다. 구분자는 콤마(,)이며, 공백 없이 작성해야 한다. 개수 제한은 없으며, 멀티 노드일 경우는 1024개로 제한한다.

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

  - 다음은 업데이트할 목록을 파일로 작성하여 업데이트를 수행하는 예제이다. 파일 내에 명시하는 모듈 수 제한은 없으며, 멀티 노드일 경우는 1024개로 제한한다.

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

◦ update.list 작성 방법이다. 다음과 같이 콤마(,)나 <Enter>를 구분자로 사용하여 작성한다.

```
mylibrary,mylibrary2
mylibrary3
```

◦ 다음은 멀티 노드 환경에서 공유 메모리 정합성에 문제가 있을 경우 동기화를 수행하는 예제이다.

```
$ tdlupdate -c
```

- 참고

  VERSION=4로 설정한 경우에는 export 함수 추출을 위한 스크립트 파일이 반드시 필요하다.

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

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

2.26. tencrypt

SVRGROUP 절의 OPENINFO에 들어가 문장에 대해서 암호화하는 유틸리티로 OPENINFO에는 DB에 접속 아이디 비밀번호가 조회된다.

DB 관리자와 TP 관리자가 분리되어 보안이 되어야 하는 상황에서는 cfl을 실행하면 DB 관리자가 있어야 하는 일이 발생하기 때문에 불편한 사항이 발생할 수 있다. tencrypt를 사용하면 DB 관리지가 OPENINFO에 들어가는 값을 암호화해서 TP 관리자에게 문장을 넘겨주면 DB 정보 노출없이 자연스럽게 TP관리자와 DB관리자가 보안에 문제 없이 운영될 수 있다.

tencrypt는 cfl에서 해석 가능한 암호화한 문장을 출력하는 기능으로 사용법은 아래에서 설명한다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ tencrypt [-e 문장 [-d]] [-g]
```

항목	설명

[-e 문장 [-d]]	암호화할 문장을 입력한다.
	
	`tencrypt -e ORACLE_XA+Acc=P/scott/tiger+SesTm=6`
	
	사용하면 다음과 같이 화면에 출력된다.
	
	`Please insert the following text in the configuration file "OPENIFNO" field.`
	`[@@QEAzMWU4Y0lOYXd2Y2lkSzdpTW1lL0F4U0d1NCswekkxSFlaMjJweld OTERaM2ptVT0jQEBvNVJPVUE9PQ==@@]`
	
	그러면 OPEINFO 필드에 [] 안의 내용물을 copy한다.
	
	`OPEINFO = "@@QEAzMWU4Y0lOYXd2Y2lkSzdpTW1lL0F4U0d1NCswekkx SFlaMjJweldOTERaM2ptVT0jQEBvNVJPVUE9PQ==@@"`
	
	-d 옵션을 같이 사용할 경우 복호화된 결과도 함께 출력한다.

[-g]	SEED 사용을 위해서 모든 모듈 및 노드에서 공통적으로 사용하는 비밀키가 저장된 파일이 생성된다. 비밀키 파일을 생성해야만 암복호화가 가능하며, 비밀키를 변경하면 암호화 과정을 다시 수행해야 한다. 멀티 노드에 적용하는 경우에는 똑같은 비밀키가 적용되도록 하기 위해서, 생성된 비밀키 파일의 내용을 복사해서 다른 노드에도 적용해야한다. 암호화 방식이 SEED가 아닌 기존 방식의 경우 비밀키 파일을 생성하지 않는다.

- 적용 환경

  tencrpyt만 따로 copy하여 사용 가능하다.

cfl 명령어에 대한 자세한 내용은 cfl을 참고한다.

2.27. tmadmin

Tmax 시스템 관리 명령어이다. Tmax 시스템의 동적인 관리를 위해 Command 인터프리터 형태로 제공되는 모니터 프로그램이다. Tmax 시스템이 사용하는 공유 메모리의 정보를 읽어서 동작 중인 시스템의 환경설정 내용이나 서버 프로세스의 동작 상태, 또는 서비스 상태 등을 파악할 수 있다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ tmadmin [-l] [-s|m] [-h] [-f [config_file]] [-n [node_name]] [-v] [-V]
          [-p] [-t] [-i lic_info]
```

항목	설명
[-l]	racd를 통하여 여러 노드로 구성된 시스템이 중앙에서 집중 관리될 경우에 로컬 노드만을 관리하기 위해서 사용하는 옵션이다. 각각의 노드들은 이 옵션으로 자신의 시스템만을 관리할 수 있다.
[-s]	읽기 전용 모드로 각각의 tmadmin 툴을 10개까지 사용할 수 있는 옵션이다. 이 옵션을 사용하면 환경을 동적으로 변경시킬 수 없다. (기본값)
[-m]	동적으로 환경을 변경할 수 있는 마스터 모드이다. 1명의 사용자만이 마스터 모드를 사용할 것을 권장한다. 1명 이상의 사용자가 환경을 변경할 경우 심각한 시스템 문제가 발생할 수 있다. 환경 파일은 변경하지 않고 정해진 매뉴얼을 따른다.
[-h]	명령어 도움말 옵션이다.
[-f [*config_file*]]	지정해 준 이진 파일을 관리한다. 환경 파일을 기본값인 tmconfig가 아닌 다른 이름으로 이진 파일을 생성한 경우 tmadmin을 구동할 때 이진 파일명을 지정한다.
[-n [*node_name*]]	특정 노드에 대한 모니터링이 가능하도록 한다. 멀티 노드 환경에서 tmadmin을 구동하는 경우 이 옵션을 설정하면 지정된 노드에 대한 정보만을 확인하므로 보다 편리하게 시스템을 관리할 수 있다.
[-v]	Tmax의 버전을 확인한다. 이 옵션을 설정하면 Tmax 기동과 상관없이 어느 위치에서나 버전 확인이 가능하다.
[-V]	실행 파일의 버전을 확인할 수 있다.
[-p]	st –p, st –s, si, ci, cfg의 명령어가 수행될 때 결과가 화면 단위로 출력되도록 하는 옵션이다(more 기능).

항목	설명
[-t]	명령어 수행에 대한 시작 및 종료시간을 출력한다. [-t] 옵션을 주고 tmadmin을 실행하면 다음과 같은 형식으로 시간을 출력한다. • 형식 `[TIME][타입 ] : hh:MM:ss:millisec` 다음은 타입 항목에 대한 설명이다. ◦ START : 콘솔에서 명령어를 실행할 때 시작시간을 출력한다. ◦ END : 콘솔에서 명령어의 실행이 종료되면 종료시간을 출력한다(리모트 노드의 명령까지 수행한 이후의 시간이다). ◦ R_END : 콘솔에서 명령어를 실행할 때 리모트 노드의 수행이 종료되면 종료한 시간(tmadmin을 실행한 로컬의 시간으로 리모트 노드의 시간이 아니다)을 출력한다. ◦ RP_START : tmadmin의 repeat 명령어를 사용하여 수행할 경우 한 건의 명령을 시작하기 전 시간을 출력한다. ◦ RP_END : tmadmin의 repeat 명령어를 사용하여 수행할 경우 한 건의 명령이 종료되면 종료시간을 출력한다(리모트 노드의 명령까지 수행한 이후의 시간이다).
[-i *lic_info*]	입력한 license.dat 파일의 라이선스 정보를 출력한다.

- 적용 환경

  Tmax 시스템이 설치된 운영 시스템 환경에서 사용할 수 있다.

- 예제

  다음은 tmadmin 프로그램이 실행 상태임을 나타내는 예제이다.

```
$ tmadmin
```

  tmadmin을 실행하면 다음의 메시지와 함께 프롬프트(prompt)가 나타낸다.

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

cfl, tmboot, tmdown 명령어에 대한 자세한 내용은 각각 cfl, tmboot, tmdown을 참고한다.

tmadmin 툴에서 사용할 수 있는 명령어

다음은 tmadmin을 실행한 후 사용할 수 있는 명령어이다.

- 환경 정보 조회

명령어	설명
tmaxinfo(ti)	Tmax 시스템 정보를 확인한다.
history(hist)	이전에 저장된 명령어를 조회한다.
config(cfg)	환경설정 내용을 조회한다.

- 동작상태 정보 조회

명령어	설명
stat(st)	프로세스 및 서비스 상태의 통계를 조회한다. 와일드카드(Wild card)를 사용할 수 있다.
gwinfo	GATEWAY 절에 정의한 게이트웨이들의 채널 상태를 확인한다.
txgwinfo(txgwi)	Tmax 게이트웨이의 정보를 확인한다.
nontxgwinfo	Tmax Non 트랜잭션 게이트웨이의 정보를 확인한다.
jgwinfo	JEUS 게이트웨이의 정보를 확인한다.
ajgwinfo	JEUS Async 게이트웨이의 정보를 확인한다.
wsgwinfo	웹 서비스 게이트웨이의 정보를 확인한다.
smtrc	GID를 이용하여 해당 서비스의 수행 상태를 조회한다.
clhsinfo	멀티 노드 환경에서 CLH 사이의 연결 상태 정보를 확인한다.
tmmsinfo	멀티 노드 환경에서 TMM 사이의 연결 상태 정보를 확인한다.
repeat(r)	명령어를 반복한다.
clientinfo(ci)	접속 클라이언트를 확인한다.
svrinfo(si)	서버 정보를 확인한다. 와일드카드(Wild card)를 사용할 수 있다.
txquery(txq)	트랜잭션 처리 정보를 확인한다.
rqstat(rqs)	RQ의 상태를 확인하거나 디스크 큐에 쌓여 있는 서비스를 처리한다.

- 운용 관리

명령어	설명
suspend(sp)	동작 중인 서버 프로세스를 중지한다.
resume(rs)	중지된 서버 프로세스를 재개한다.
advertise / unadvertise	특정 서비스명을 advertise/unadvertise한다.
restat	특정 서버 프로세스 또는 모든 서버 프로세스의 통계 정보를 초기화한다.
rebootsvr(rbs)	서버 프로그램을 교체한다.

명령어	설명
cfgadd(ca)	동적으로 서비스를 추가한다.
set	현재 설정된 환경설정 값을 동적으로 변경한다.
qpurge(qp)	큐에 쌓인 업무를 삭제한다.
discon(ds)	접속 중인 클라이언트를 강제로 해제한다.
logstart / logend	로깅을 시작 및 종료한다.
chtrc	Trace 관리를 한다.
chlog	TMM, CLH, 특정 서버의 로그 레벨을 런타임 도중 동적으로 변경한다.
txcommit / txrollback	트랜잭션 처리 중에 장애가 발생하는 경우 commit 및 rollback을 재이슈하여 해당 트랜잭션을 종료하는 기능이다.
wsgwreload	웹 서비스 게이트웨이의 서비스 정보, 설정 변경을 적용한다.
restart	서버 프로세스를 재기동한다.

- 기타

명령어	설명
!	바로 앞에서 사용한 명령어를 반복한다.
quit(q)	tmadmin을 종료한다.
help(h)	사용 가능한 옵션 리스트를 조회한다.
nodeset(ns)	멀티 노드 환경에서 특정 노드에 대한 정보만을 얻을 경우에 사용한다.
nodeunset(nus)	멀티 노드 환경에서 특정 노드에 대한 정보만을 얻을 때 설정한 것을 해제한다.
tmd	가상의 클라이언트 애뮬레이터로서 서비스 프로그램의 유효성을 검토하는 경우 클라이언트 프로그램을 별도로 작성하지 않고 이 유틸리티를 이용한다.

각 명령어의 사용방법 및 처리 결과에 대한 자세한 내용은 Tmax Administration Guide의 "tmadmin"을 참고한다.

2.28. tmapm

Tmax 시스템에서 서비스 타임아웃은 시그널 알람을 사용하여 동작한다. tmapm은 서비스에서 시그알람을 재정의하거나 시그널 알람을 사용하지 못하는 서비스에 대해서 서비스 타입 아웃을 제공하는 서버이다.

tmapm은 UCS 서버로 환경설정 파일의 SERVER 절에 다음과 같이 설정한다.

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

항목	설명
[-i *sec*]	서비스 타임을 체크하는 주기(초)이다. 짧을수록 시스템 부하가 증가한다. 소수점도 설정 가능하다.
[-r *sec*]	서비스 타임 이후 커맨드를 실행하기까지의 제한 시간(초)이다. 정수형으로 설정해야 한다.
[-c *command*]	해당 서비스가 수행시간(SVCTIME + 커맨드 실행 제한시간)까지 RUNNING 상태일 경우 실행할 커맨드이다. 셸 명령어나 스크립트가 될 수 있다. [-c] 옵션으로 설정한 커맨드를 실행할 경우 다음과 같이 파라미터를 전달한다. ◦ User Command : kill.sh ◦ PID : 9659 ◦ SVRNAME : svr2 ◦ SVCNAME[Elapse Time] : TOUPPER[5]
[-l [0|1|2]]	사용자 로그 레벨이다. 기본값은 0이며, 높을수록 좀 더 자세한 정보 출력한다.

다음은 tmapm 설정의 예이다.

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

2.29. tmaxlibver

Tmax 라이브러리의 버전 정보를 조회하는 명령어이다. Tmax 6 버전부터 컴파일하지 않고도 라이브러리의 버전 정보를 볼 수 있도록 tmaxlibver의 기능이 변경되었다.

CFLAGS 환경변수를 설정한 후 tmaxlibver를 실행하면 CFLAGS에 지정된 컴파일 옵션을 사용한다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

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

항목	설명
{-l *filename*}	조회하기 위한 라이브러리명을 지정한다.
{-d|-s}	라이브러리의 dynamic (-d) 또는 static (-s)의 여부를 설정한다.
[-6]	라이브러리가 64Bit일 경우 설정하는 옵션으로 설정하지 않으면 32Bit로 동작한다. -6이 설정되고 [–L] 옵션이 설정되지 않은 경우 ${TMAXDIR}/lib64 디렉터리에 위치한 라이브러리를 자동 참조한다.

항목	설명
[-L *directory*]	조회하기 위한 라이브러리가 위치하는 절대 경로 또는 상대 경로를 입력한다. 지정하지 않은 경우 [-6] 옵션에 따라 디폴트 경로가 달라지게 된다.
[-h]	명령어 도움말 옵션이다.

- 예제

```
$ tmaxlibver -l libsvr.so -d -6
libsvr.so for TMAX Version 6.0 Sp #0 Fix #1 r11518 fd 32768 64bit x86 Linux2 6 Glibc2 3 patch
```

DB stub 등 일부 라이브러리에서는 버전 정보 조회 기능을 사용할 수 없다.

2.30. tmaxtrace

Tmax 애플리케이션 관리자나 개발자가 각 애플리케이션에 대하여 Runtime Tracing을 할 수 있도록 하는 명령어이다. Runtime Tracing은 애플리케이션이 실행하는 동안의 수행에 대하여 기록을 남기는 Trace point 개념에 근거하고 있다. Trace point는 tpcall()과 같은 atmi 함수의 시작이나 끝, 트랜잭션의 시작 등을 예로 들 수 있다.

Trace point가 발생했을 때 나타나는 현상은 다음과 같다.

1. Trace point가 유효한 것인지를 확인하기 위해 필터가 적용된다. 유효하다면 Trace record가 receiver 파일에 기록되고 마지막으로 Action이 일어난다. Action은 프로세스를 종료(abort)시키거나, 시스템 call을 호출하는 등 여러 가지가 있을 수 있으며 이는 선택 옵션이다.
2. **Filter, Receiver, Trigger** 항목은 아래에서 설명하는 스펙에 따라 정의된다. 이 항목은 **TMAX_TRACE**의 환경변수를 선언하면 정의할 수 있으며 클라이언트와 서버 모두 설정할 수 있다.

   실행 중인 프로세스의 spec을 변경하기 위해서는 **tmadmin** 툴의 **chtrc** 명령어로 변경할 수 있다.
3. TMAX_TRACE 스펙 중 Trigger 옵션을 'dye'로 설정하였을 경우에는 해당 프로세스의 요청을 받는 서버의 프로세스들도 Runtime Tracing이 수행된다. 클라이언트에서 dye를 설정하였다면 해당 서비스를 수행하는 프로세스가 atmi category가 되며, Runtime Tracing이 수행된다.
4. 클라이언트에서 TMAX_TRACE를 사용할 경우 클라이언트 환경설정 파일(tmax.env)이나 환경변수에 ULOGPFX를 반드시 지정한다. 아래와 같이 지정할 경우 해당 디렉터리에 <clilog.날짜> 파일이 생성되며, 이 파일에 해당 로그가 쌓인다.

   예)

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

다음은 TMAX_TRACE의 사용법에 대한 설명이다.

- 사용 방법

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

◦ filter-spec : category

특정 category에 대하여 Runtime Tracing을 수행할지를 결정한다. 모든 category를 표현하기 위해서 애스터리스크(*) 문자를 설정할 수 있으며 filter-spec에 값을 설정하지 않을 경우 아무런 category가 선택되지 않았다는 것을 의미한다.

다음은 category의 목록이다.

category	**설명**
atmi	사용자가 호출한 ATMI 및 TX 인터페이스(tp 또는 tx로 시작하는 함수)를 호출하는 경우 Runtime Tracing이 수행된다.
iatmi	사용자가 호출한 ATMI 및 TX 인터페이스 외에도 인터페이스에서 내부적으로 호출한 인터페이스들까지도 Runtime Tracing이 수행된다.
xa	모든 XA 인터페이스를 호출하는 경우 Runtime Tracing이 수행된다.
trace	Tmax Trace가 설정된 정보를 출력한다.
gqs	사용자가 호출한 GQ 인터페이스(gq로 시작하는 함수)를 호출하는 경우 Runtime Tracing이 수행된다.
tdl	사용자가 호출한 TDL 인터페이스(tdl로 시작하는 함수)를 호출하는 경우 Runtime Tracing이 수행된다.

category를 설정할 때 여러 항목을 파이프라인(|) 구분자로 지정할 수 있다.

```
atmi|xa|tdl
```

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

receiver-spec를 설정하지 않을 경우 아무런 trace record도 기록이 되지 않는다.

항목	설명
receiver	receiver는 trace record가 보내지는 파일이다. receiver는 ulog, stdout, stderr가 사용된다. ulog로 설정된 경우 user log로 Runtime Tracing이 수행된다. stdout, stderr의 경우 각각 stdout, stderr로 Runtime Tracing이 수행된다. receiver에 다음과 같은 형태로 기록된다. `process-name.pid.hhmmss: TMAXTRACE:cc data` cc는 category에 따라서 달라진다. ◦ atmi인 경우 : at ◦ iatmi인 경우 : dt ◦ xa인 경우 : xa ◦ trace인 경우 : tr
regular-expression	filter-spec에 적용되는 Trace point를 제한할 수 있다.

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

항목	설명
trigger	trace record가 receiver로 보내진 후의 action을 설정하는 항목으로 옵션 항목이다.
action	다음은 action에 설정할 수 있는 동작에 대한 설명이다. ◦ abort : abort()를 호출하여 프로세스를 종료시킨다. ◦ ulog(message) : user log에 메시지를 쓴다. ◦ system(command) : system()을 사용하여 command를 실행시킨다. Windows에서는 지원하지 않는다. ◦ trace(trace-spec) : trace-spec으로 Tmax trace의 스펙을 변경한다. ◦ dye : 메시지를 dyeing한다. ◦ undye : 메시지를 dyeing하지 않는다. ◦ sleep(seconds) : 설정한 seconds만큼 sleep한다. Windows에서는 지원하지 않는다.

- 예제
  - 다음은 atmi 인터페이스를 호출한 경우 모두 Runtime Tracing이 수행되는 예제이다.

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

◦ 다음은 atmi/tx 인터페이스 중에서 tpcall()을 호출한 경우에만 Runtime Tracing이 수행되는 예제이다.

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

◦ 다음은 atmi/tx 인터페이스가 호출될 경우 abort()를 이용하여 해당 서버 프로세스를 종료시키는 예제이다.

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

◦ 다음은 atmi/tx 인터페이스를 호출한 경우 모두 Runtime Tracing이 수행되지만 tpend()를 호출한 경우에는 해당 프로세스는 abort()에 의해 종료되는 예제이다.

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

◦ 다음은 모든 category에 대해서 Runtime Tracing이 수행되는 예제이다.

```
export TMAX_TRACE=*:ulog:dye
```

◦ 다음은 atmi/tx 인터페이스를 호출한 후에 trace spec이 "*:ulog:dye"로 수정되는 예제이다.

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

◦ 다음은 tpalloc() 호출 후에 trace spec이 "*:ulog:dye"로 수정되는 예제이다.

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

◦ 다음은 'TMAX_TRACE=atmi:ulog:dye'와 동일한 예제이다.

```
export TMAX_TRACE=on
```

◦ 다음은 tmaxtrace를 사용하지 않는 예제이다. (기본값)

```
export TMAX_TRACE=off
```

◦ 다음은 tx, tp함수만 로깅하는 예제이다.

```
export TMAX_TRACE='*:/t[px].*/ulog:undye'
```

2.31. tmboot

Tmax 시스템의 전체나 또는 일부분을 실행시키는 명령어로 Tmax 환경 파일을 바탕으로 시스템을 실행한다. 옵션 없이 실행되거나 [-f] 옵션만이 사용되면, 모든 Tmax 관리 프로세스들과 Tmax 환경 파일의 SERVER 절에 등록된 모든 서버 프로세스들을 실행시킨다.

NODE 절에 등록된 모든 노드에서 Tmax 관리 프로세스인 **TMM**, **CLL**, **CLH** 프로세스가 순서대로 실행된다. SVRGROUP 절에 OPENINFO 항목이 등록된 서버 그룹이 존재한다면, 각 서버 그룹별로 **TMSNANE**과 **MINTMS** 항목을 참조하여 TMS 프로세스들이 실행된다. Tmax 관리 프로세스들은 노드별로 정의된 TMAXDIR 디렉터리 하위의 bin 디렉터리에서 실행된다.

Tmax 관리 프로세스들이 생성된 후에는 SERVER 절의 모든 응용 서버 프로세스들이 실행된다. 응용 서버 프로세스들은 SERVER 절에 등록된 순서대로 실행된다. tmboot는 실행된 서버 프로세스의 초기화를 실행한 후에 다음 서버 프로세스를 실행시킨다. 프로세스 초기화는 tpsvrinit()를 이용해서 진행한다. 모든 서버 프로세스들이 초기화를 모두 완료할 때까지 다음 프로세스가 실행되지 않는다. tmboot는 각 응용 서버 프로세스들을 그들의 MIN 항목에 정의된 개수만큼 실행시킨다. MIN 항목이 정의되지 않은 경우 기본값은 1개이다.

tmboot는 SERVER 절의 서버들에 대하여 **CLOPT**, **MIN**, **MAX** 항목의 값을 사용한다. 서버 프로세스가 실행될 때 tmboot에 의해 사용되는 서버의 boot 파라미터이며, 서버의 나머지 항목들은 서버가 실행된 후 시스템에 의해 실행되는 runtime 파라미터이다. 설정 정보는 소스 설정 파일의 SERVER 절을 참고한다.

모든 응용 서버 프로세스들은 동작하는 노드에 정의된 APPDIR 디렉터리에서 실행된다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$tmboot [-A] [-b] [-c] [-f 이진 Tmax 환경파일명]
        [-g servergroup_name [-i] [-s] [-S]} [-h] [-V] [-n node_name]
        [-o clopt_string] [-q rq_svg_name] [-s server_name [-k count] [-a]]
        [-S server_name [-a]] [-t tms_name [-k all]] [-B trb_node] [-T] [-w]
        [-d boot_time] [-W] [-D] [-e clh | cas | tlm] [-R]
```

항목	설명
[-A]	Tmax 환경 파일의 **SERVER** 절에 정의된 모든 응용 서버 프로세스를 실행하는 옵션이다.
[-b]	백업으로 지정된 서버 프로세스를 임의로 기동시킬 때 사용하는 옵션이다.
[-c]	CLH 프로세스를 하나 더 실행시키는 옵션이다. 현재 동작 중인 CLH 프로세스 개수가 Tmax 환경 파일에 정의된 **MAXCLH** 값을 넘지 않는 범위 내에서만 사용 가능하다.
[-f 이진 *Tmax* 환경파일명]	참조할 이진 Tmax 환경 파일(소스 설정 파일을 cfl로 컴파일한 결과물)을 경로와 함께 지정해야 한다. [-f] 옵션이 생략될 경우 기본으로 TMAXDIR 디렉터리 하위의 config 디렉터리에서 **tmconfig** 파일을 참조한다.

항목	설명
[-g *servergroup_name* [-a] [-i] [-s] [-S]]	지정된 서버 그룹에 존재하는 서버 프로세스를 실행하는 옵션이다. 사용되는 서버 그룹명은 Tmax 환경 파일 내의 SVRGROUP절에 등록한다. [-a] 옵션을 함께 사용하면 서버의 기동을 TMBOOT 프로세스가 아닌 TMM 프로세스에 위임한다. [-i] 옵션을 함께 사용하지 않는다면 기본적으로 서버를 부팅시키는 과정에서 기동 과정 중간에 이미 max에 도달한 서버가 있을 때 전체 기동 과정이 중지된다. 그러나 [-i] 옵션을 함께 사용하면 max에 도달한 서버가 있더라도 다음 서버들을 기동할 수 있도록 중지하지 않고 계속 진행하게 된다. 대신 모든 서버들이 max에 도달했을 경우에는 다음과 같은 메시지를 출력한다. `(I) BOOT3022 all servers in group (svgname) reached max [BOOT0039]` [-s] 또는 [-S] 옵션과 함께 사용하면, 지정된 서버 그룹 내에서 지정한 서버명을 가진 프로세스만을 대상으로 기동된다.
[-h]	명령어 도움말 옵션이다.
[-V]	실행 파일의 버전을 확인할 수 있다.
[-n *node_name*]	지정된 노드에 존재하는 서버 프로세스들을 실행시키는 옵션이다. 노드명은 Tmax 환경 파일 내의 NODE 절에 미리 등록되어 있어야 한다.
[-o *clopt_string*]	CLOPT string을 추가하기 위한 옵션이다.
[-q *rq_svg_name*]	RQS를 시동시키기 위한 옵션이다.
[-s *server_name* [-k *count*] [-a]]	지정된 서버 프로세스만을 실행시키는 옵션이다. 사용되는 서버 프로세스명은 Tmax 환경 파일 내의 SVRGROUP 절에 미리 정의되어 있어야만 한다. [-k] 옵션을 함께 사용하여 서버 프로세스 개수를 지정할 수 있다. 서버 프로세스 개수는 현재 실행되어 있는 개수를 포함하여 SERVER 절의 MAX 항목에 정의된 개수를 넘어서는 안 된다. 만약 이미 기동된 서버 프로세스들이 존재한다면, [-k] 옵션으로 지정한 값이 그보다 클지라도 MAX를 넘지 않는 선까지만 추가로 기동한다. [-k] 옵션을 생략하면 해당 서버 프로세스는 하나만 실행된다. 만약 [-a] 옵션을 함께 사용하면 서버의 기동을 TMBOOT 프로세스가 아닌 TMM 프로세스에 위임한다.
[-S *server_name* [-a]]	지정된 서버 프로세스를 MIN 개수만큼 실행시키는 옵션이다. 만약 남은 slot보다 MIN 개수가 더 크다면 MAX를 넘지 않는 선까지만 추가로 기동한다. [-a] 옵션을 함께 쓰면 서버의 기동을 TMBOOT 프로세스가 아닌 TMM 프로세스에 위임한다. [-g] 옵션과 함께 쓰면 지정한 서버 그룹에 대해서만 기동되며, [-g] 옵션을 사용하지 않으면 동일한 서버명을 가진 모든 서버 그룹을 대상으로 하여 기동된다.

항목	설명
[-t *tms_name* [-k all]]	지정된 TMS 프로세스를 하나 더 실행시키기 위한 옵션이다. Tmax 환경 파일에 정의된 **MAXTMS** 값을 넘지 않을 경우에만 가능하다. [-k all]은 트랜잭션 Recovery 기능을 사용할 경우 필요한 옵션이다. 그룹 전체를 종료했다가 기동할 경우 Recovery가 수행되기 때문에 Recovery는 TMS 그룹 단위로 가능하다. 특정 이름을 가진 TMS 전체를 기동/종료할 경우 이 옵션을 사용할 수 있다.
[-B *trb_node*]	TRB 노드를 기동시킨다.
[-T]	Tmax 시스템 프로세스(TMM, CLL, CLH, TMS)만을 실행시키는 옵션이다.
[-w]	옵션이 없는 경우에는 등록된 서버 프로세스를 동시에 기동시킨다. 이 경우 운영체제에 따라 동시에 리소스를 만들어내지 못해 서버 프로세스가 올바르게 기동되지 못하는 경우가 발생한다. 이러한 문제를 해결하기 위해 프로세스를 하나씩 기동시켜 올바르게 기동시키는 옵션이다. • 서버 프로세스들이 TMM에 접속할 때 LOCK 사용 조건 (LOCK|NOLOCK) • 서버 프로세스가 기동될 때 WAIT 조건 (NO-WAIT|FINITE-WAIT) ◦ "-d -1000000 (1sec)"와 동일한 효과를 가진다. ◦ [-d] 옵션이 없는 경우에만 의미가 있다. ◦ [–d] 옵션이 사용되면 [–w] 옵션은 무시된다.
[-d *boot_time*]	한 번에 많은 서버 프로세스를 기동시킬 때 CLH로의 등록요구가 폭주하여 생기는 문제를 해결하기 위해 서버 프로세스들이 기동하는 데 걸리는 총 시간을 지정하여 등록 간격을 조절할 수 있다. (기본값: LOCK, NO-WAIT, 단위: usec) • 서버 프로세스들이 TMM에 접속할 때 LOCK 사용 조건 (LOCK|NOLOCK) • 서버 프로세스 기동될 때 WAIT 조건 (NO-WAIT|FINITE-WAIT) ◦ -d val < 0 : LOCK, |VAL| FINITE-WAIT * ◦ -d val = 0 : NO-LOCK, NO-WAIT ◦ -d val > 0 : NO-LOCK, |VAL| FINITE-WAIT ◦ 기본적으로 [–d] 옵션의 val이 0이 아닌 경우 절대값(|VAL|)을 사용하며 단위는 usec이다. • FINITE-WAIT의 경우 |VAL| 값은 각 프로세스마다 최대의 WAIT시간이다. (전체 프로세스의 총 WAIT 시간이 아님) • 서버 프로세스가 signal을 줄 경우 WAIT은 해제된다. 즉, |VAL| 시간이 되지 않더라도 signal을 받으면 다음 프로세스의 기동을 시도한다. VAL이 음수인 경우 LOCK을 사용한다. 이 옵션이 사용되었을 경우 [-w] 옵션은 무시된다.
[-D]	[-d] 옵션과 거의 유사하지만 finite-wait될 때 signal이 오더라도 |VAL|까지는 무조건 WAIT한다.

항목	설명
[-W]	tmboot를 수행하는 경우 현재 기동 중인 서버 프로세스가 완전히 기동이 될 때까지 다음 프로세스를 기동시키지 않고 대기한다.
[-e clh|cas|tlm|tsm]	Tmax 엔진 프로세스 중 CLH, CAS, TLM을 부팅하기 위한 옵션으로 tmboot할 경우에 문제가 발생했거나, kill로 인하여 잘못하여 엔진 프로세스가 비정상 종료된 경우에 사용한다. tmdown에서는 이 옵션을 제공하지 않는다. • clh : CLH를 부팅한다. • cas : CAS를 부팅한다. • tlm : TLM을 부팅한다. • tsm : TSM을 부팅한다.
[-R]	리모트 셸(rsh 또는 remsh)을 사용하여 리모트 노드의 Tmax 시스템을 부팅시키는 경우 [-R] 옵션을 사용한다. 이 옵션을 사용하게 되면 tmboot로 기동되는 프로세스에게 tmboot의 stdin/stdout이 상속되지 못하도록 처리한다. • 사용 방법 (remote) rsh $HOSTNAME tmboot.sh - tmboot.sh TMAXDIR=/data3/starbj81/tmax64; export TMAXDIR PATH=$PATH:$TMAXDIR/bin; export PATH export LD_LIBRARY_PATH=$LIBPATH:$TMAXDIR/ (BIT) export LIBPATH=$LIBPATH:$TMAXDIR/ (BIT) exec tmboot -R echo "tmboot success" exit 0

- 적용 환경

  Tmax 시스템이 설치된 운영 시스템 환경에서 사용할 수 있다.

- 예제
  - 다음은 TMAXDIR 디렉터리 하위의 config 디렉터리에 있는 tmconfig 파일을 참조하여 Tmax 프로세스와 응용 서버 프로세스들을 모두 실행하는 예제이다.

    ```
    $ tmboot
    ```

  - 다음은 tms_name으로 생성된 모든 TMS가 기동되는 예제이다.

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

  - 다음은 서버 그룹의 설정 옵션을 사용한 예제이다.

    서로 다른 서버 그룹에서 tms_name을 동일하게 사용하는 경우 다음과 같이 [–g] 옵션으로 해당 TMS가

속해있는 서버 그룹명을 지정한다. 그렇지 않으면 해당 TMS명을 가진 가장 첫 번째 서버 그룹의 TMS가 기동된다.

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

- 다음은 tmconfig 환경 파일을 참조하여 SERVER 절에 정의된 모든 응용 서버 프로세스들을 실행하는 예제이다.

```
$ tmboot -A
```

- 다음은 /user1/tmax/con 디렉터리의 exconfig 환경 파일을 참조하여 NODE 절에 등록된 cosmo 노드에 존재하는 응용 서버 프로세스들을 실행하는 예제이다.

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

- 다음은 tmconfig2 환경 파일을 참조하여 svr1프로세스를 5개 실행하는 예제이다.

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

cfl, tmdown 명령어에 대한 자세한 내용은 각각 Tmax Reference Guide의 "cfl" , tmdown을 참고한다.

tmboot를 실행할 때의 tmconfig 경로 참조

tmboot 명령어로 Tmax를 기동할 경우 이진 바이너리 환경 파일 ${TMAXDIR}/config/tmconfig를 ${TMAXDIR}/path/tmconfig에 복사한 후 ${TMAXDIR}/path 디렉터리에 있는 환경 파일을 사용한다.

하지만 이미 Tmax 엔진이 기동되어 있는 상황에서 tmboot –S 또는 –s 를 이용하여 특정 서버만을 기동할 경우에도 ${TMAXDIR}/config/tmconfig를 참조한다면 아래와 같은 환경에서 문제가 될 수 있다.

- 환경 파일

  <기존 운영 환경 파일>

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

  <운영 중 변경된 환경 파일>

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

다음과 같이 운영 중 변경된 환경 파일을 CFL로 재컴파일한다.

```
$ cfl –i node1.m
```

새로 추가된 서버를 기동한다.

```
$ tmboot –S svr3
```

운영 중인 환경에서 환경 파일을 변경한 후 tmboot –S를 시도하려면 다음과 같은 에러가 발생할 수 있다.

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

운영 환경에서 CFL을 수행하였을 경우 ${TMAXDIR}/config/tmconfig는 변경된 내용이 적용되지만 실제 공유 메모리에는 변경되기 이전인 ${TMAXDIR}/path/tmconfig와 동일하게 구성되어 있기 때문에 tmboot –S로 새로 추가한 서버 프로세스가 기동될 때 실제로는 기존에 이미 실행 중인 서버 프로세스를 추가로 기동하는 결과가 발생한다.

Tmax 엔진이 기동되어 있는 상황에서의 CFL은 허용되지 않지만 이 실수로 인한 에러가 발생하면 해당 에러는 디버깅하기가 쉽지 않다. 따라서 Tmax 엔진 동작 중 tmboot [–S], [-s], [–g], [-q], [-t], [-A] 등의 옵션으로 각 서버를 기동시킬 때 참조하는 tmconfig의 경로는 ${TMAXDIR}/config/tmconfig가 아닌 ${TMAXDIR}/path/tmconfig이다. 단, 엔진을 기동할 경우 ${TMAXDIR}/path/tmconfig를 참조한다.

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

tmboot를 수행할 때 [-f] 옵션을 사용하여 특정 이진 바이너리 환경 파일을 지정하면 ${TMAXDIR}/config/tmconfig를 참조하여 서버를 기동한다. 동적 서버를 추가할 때에는 반드시 [–f] 옵션으로 특정 환경 파일을 지정하여 ${TMAXDIR}/config/의 변경된 이진 바이너리 환경 파일을 참조하도록 한다.

2.32. tmd

서버 프로그램을 테스트하기 위해 클라이언트를 시뮬레이션하는 명령어이다. 프로그래머는 직접 클라이언트 프로그램을 작성하지 않아도 쉽게 서버 프로그램의 동작 상태를 확인할 수 있다.

tmd 명령어는 Windows NT와 Windows 2000 환경에서는 작동하지 않는다.

tmd에서 지원하는 버퍼 유형은 STRING, CARRAY, FIELD, 구조체 버퍼이다. CARRAY 버퍼의 경우에는 print 가능한 데이터에 대해서 지원한다. STRING이나 CARRAY 데이터는 공백으로 데이터의 끝을 간주하기 때문에 계속되는 데이터를 표시하기 위해서는 큰따옴표(" ")로 데이터를 처리해야 한다. 구조체 버퍼의 경우 단일 구조체에 대해서 지원하고 있으며 송수신 구조체 타입이 일치해야 한다. 수신한 버퍼 유형과 같은 버퍼를 tpreturn() 함수 내의 파라미터로 사용해야 한다.

그렇지 않으면 다음과 같은 에러 메시지를 출력한다.

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

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

  ```
  $ tmd [-X] [-T 시간(초)] [-t 시간(초)] [-s] [-V] [-i file_name] [-l 크기(byte)] [-q]
  ```

항목	설명
[-X]	트랜잭션 사용 유무를 지정한다. flags가 지정되면 주어진 서비스를 트랜잭션으로 간주하여 처리한다.
[-T 시간(초)]	트랜잭션 타임아웃을 지정한다. tx_set_transaction_timeout(시간)과 같은 기능을 수행하게 된다.
[-t 시간(초)]	블록 타임아웃을 지정한다. tpset_timeout(시간)과 같은 기능을 수행한다.
[-s]	tpstart 정보를 콘솔에서 입력받을 수 있도록 하여 사용자에 대한 정보가 시스템에 전달될 수 있도록 하는 기능을 수행한다. 옵션을 설정한 경우 tmd 수행할 때 domain password와 user name, user password를 입력받게 된다.
[-V]	실행 파일의 버전을 확인할 수 있다.
[-i *file_name*]	호출할 서비스를 정의한 파일이다.
[-l *size*]	명령어를 통해 입력받을 데이터의 최대 크기를 지정한다. (기본값: 1024byte)
[-q]	따옴표(")가 포함된 string 형의 데이터를 처리할 수 있다. string 데이터 중의 \"를 "로 인식하여 처리한다.

- 예제
  - 다음은 tmd 명령어를 사용한 예이다.

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

◦ 다음은 필드 키 버퍼를 사용하는 예제이다. 필드 키 버퍼는 애스터리스크(*)로 시작되며 함수명, 서비스명, 버퍼 타입으로 첫 번째 행이 구성된다. 다음 행에는 필드명과 데이터가 나열된다.

```
Input File  ( input_data ) 작성
*tpcall BR_ADD FIELD
BRANCH_ID     1
LAST_ACCT     9999
LAST_TELLER   99
ADDRESS      “25 Powell St. San Francisco, CA 94188”
PHONE         415-753-9000
```

◦ STRING 버퍼를 사용하는 경우에는 버퍼 타입에 FIELD 대신 STRING을 입력한다.

<tmdTest File 작성 예>

```
*tpcall TOUPPER STRING
abc
```

2.33. tmdown

Tmax 시스템 전체나 또는 일부분을 종료시키는 명령어이다. tmdown은 Tmax 환경 파일을 바탕으로 하여 Tmax 시스템을 종료시키기 때문에 기본적으로 [-f] 옵션을 사용하여 참조할 이진 Tmax 환경 파일(소스 설정 파일)을 cfl로 컴파일한 결과물을 경로와 함께 지정해야 한다.

[-f] 옵션이 생략될 경우 기본값으로 TMAXDIR 디렉터리 하위의 config 디렉터리에 있는 **tmconfig** 파일을 참조한다. [-f] 이외의 옵션이 사용되지 않는다면 tmdown은 Tmax의 모든 관리 프로세스와 Tmax 환경 파일의 SERVER 절에 등록된 모든 서버 프로세스를 종료시키고 Tmax 시스템과 관련된 IPC 자원들을 제거한다. 종료 순서는 다음과 같다.

1. SERVER 절에 등록된 응용 서버 프로세스들이 종료된다.
2. 서버 그룹별로 TMS 프로세스가 동작 중이라면 TMS 프로세스가 종료된다.
3. Tmax 시스템 관리 프로세스들이 종료된다. 프로세스 종료는 **CLH**, **CLL**, **TMM** 순서로 진행된다. 이 순서가 일반적인 시스템 관리 프로세스 종료 순서나 CLH의 MIN 값이 1이 아닌 경우에는 CLL이 CLH보다 먼저 종료될 수도 있다.

백업으로 기동된 서버에 동적으로 등록된 서비스가 있는데 이 서버가 모두 다운되어 동적으로 등록된 서비스가 공유 메모리에서 사라질 경우를 가정해본다. 이 경우 장애 복구 후(백업 서버들이 모두 종료되고 정상 노드의 서버들이 재기동되어 있는 상태), 백업 노드로 접속한 클라이언트에 대해서 naming 서비스를 제공할 수 없게 된다. 따라서 백업 서버의 동적 서비스는 해당 서버가 모두 종료된 후에도 공유 메모리에서 삭제되지 않도록 처리한다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$tmdown [-A] [-f 이진 Tmax 환경파일명] [-g servergroup_name] [-h] [-V][-i] [-b]
        [-n node_name] [-p server_num] [-q rq_svg_name]
        [-s server_name [-k count]] [-S server_name] [-t tms_name[-k all]]
        [-w wait_time] [-B trb_node][-R] [-y]
```

항목	설명
[-A]	모든 응용 서버 프로세스들을 종료시킨다.
[-f 이진 *Tmax* 환경파일명]	시스템이 종료될 때 참조할 이진 Tmax 환경 파일명을 지정하는 항목으로 파일명을 지정하지 않으면 기본값으로 TMAXDIR 디렉터리 하위의 config 디렉터리에 있는 **tmconfig** 파일을 참조한다.
[-g *servergroup_name*]	지정된 서버 그룹에 속한 서버 프로세스들을 종료한다. 만약 [-s] 또는 [-S] 옵션과 함께 쓰면 지정한 서버 그룹 내에서 지정한 서버명에 대해서만 서버 프로세스를 종료한다.
[-h]	명령어 도움말 옵션이다.
[-V]	실행 파일의 버전을 확인할 수 있다.
[-i]	tmdown 명령을 즉시 수행한다. 기본적으로 tmdown 명령은 해당 업무를 모두 종료하고 수행되지만 [-i] (immediately) 옵션에 의한 종료는 현재 수행 중인 업무를 무조건 중단하기 때문에 신중하게 사용해야 한다.
[-b]	POD 서버가 명시적으로 tmboot를 호출하기 전에 ASQCOUNT에 의해서 자동으로 기동하지 않도록 방지한다. 반드시 [-S] 옵션이나 [–g] 옵션, [–A] 옵션과 함께 사용해야 한다.
[-n *node_name*]	지정된 노드에 존재하는 모든 서버 프로세스들을 종료시킨다. 노드명은 Tmax 환경 파일의 **NODE** 절에 등록되어 있어야 한다.
[-p *server_num*]	지정된 서버 프로세스를 종료시킨다. [-s] 옵션에 의한 종료와 달리 tmadmin에서 "st -p" 명령으로 확인할 수 있는 프로세스 번호(spr_no)를 사용하여 특정 프로세스를 종료시킨다.
[-q *rq_svg_name*]	RQS를 종료시킨다.
[-s *server_name* [-k *count*]]	지정된 서버 프로세스 하나만 종료시킨다. 사용되는 서버 프로세스명은 Tmax 환경 파일 내의 SERVER 절에 미리 등록되어 있어야 한다. [-k] 옵션을 함께 사용하여 서버 프로세스 개수를 지정할 수 있다. 서버 프로세스 개수를 현재 실행되어 있는 개수보다 크게 지정하여도 에러는 발생되지 않고 현재 실행되고 있는 모든 프로세스를 종료시킨다. [-k] 옵션을 생략하면 해당 서버 프로세스는 하나만 종료된다.
[-S *server_name*]	지정된 서버 프로세스들을 모두 종료시킨다. 하나 이상의 프로세스가 지정되어 있다면 해당되는 모든 프로세스를 종료시킨다. 만약 -g 옵션과 함께 쓰면 지정한 서버 그룹에 대해서만 종료되며, -g 옵션을 사용하지 않으면 동일한 서버명을 가진 모든 서버 그룹을 대상으로 하여 종료된다.

항목	설명
[-t *tms_name* [-k all]]	지정된 TMS 프로세스를 하나만 종료시킨다. [-k all]은 트랜잭션 Recovery 기능을 사용할 경우 필요한 옵션이다. 그룹 전체를 종료했다가 기동할 경우 Recovery가 수행되기 때문에 Recovery는 TMS 그룹 단위로 가능하다. 특정 이름을 가진 TMS 전체를 기동/종료할 경우 이 옵션을 사용할 수 있다.
[-w *wait_time*]	wait_time에 지정된 시간이 지나면 tmdown을 수행한다.
[-B *trb_node*]	TRB 노드를 종료시킨다.
[-R]	Rolling Down될 때 사용한다.
[-y]	tmdown될 때 종료 여부(y| n)를 묻지 않고 바로 종료시킨다.

- 예제
  - 다음은 TMAXDIR 디렉터리 하위의 config 디렉터리에 있는 tmconfig 파일을 참조하여 전체 Tmax 시스템을 종료하는 예제이다. Tmax 관리 프로세스와 응용 서버 프로세스들을 모두 종료한다.

```
$ tmdown
```

  - 다음은 이름이 tms_name인 모든 TMS가 종료되는 예제이다.

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

  - 다음은 서버 그룹을 설정 옵션을 사용한 예제이다. 서로 다른 서버 그룹에서 tms_name을 동일하게 사용하는 경우 위와 같이 [-g] 옵션으로 해당 TMS가 속해있는 서버 그룹명을 지정해 준다. 그렇지 않으면 해당 TMS 이름을 가진 가장 첫 번째 서버 그룹의 TMS가 종료된다.

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

  - 다음은 tmconfig2 환경 파일을 참조하여 전체 Tmax 시스템을 종료하는 예제이다.

```
$ tmdown -f tmconfig2
```

  - 다음은 tmconfig 환경 파일을 참조하여 svr1이라는 응용 서버 프로세스를 모두 종료하는 예제이다.

```
$ tmdown -S svr1
```

  - 다음은 tmconfig 환경 파일을 참조하여 svr1이라는 응용 서버 프로세스를 강제로 모두 종료하는 예제이다. svr1 중 특정 서버 프로세스의 서비스가 종료되지 않는 경우 [-i] 옵션을 사용하여 강제로 종료하는 예제이다.

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

◦ 다음은 tmconfig 환경 파일을 참조하여 <spr_no>인 서버 프로세스만 강제로 종료하는 예제이다. 해당 서버의 서버 프로세스가 여러 개일 때 특정 서버가 looping인 경우 해당 프로세스만 강제로 종료하는 예제이다.

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

◦ 다음은 /user1/tmax/con 디렉터리의 exconfig 환경 파일을 참조하여, NODE 절에 등록된 cosmo 노드에 존재하는 응용 서버 프로세스를 종료하는 예제이다.

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

◦ 다음은 tmconfig2 환경 파일을 참조하여 동작 중인 svr1프로세스 하나만 종료하는 예제이다.

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

Rolling Down 기능

클라이언트의 요청을 처리하고 있던 Tmax 시스템이 비정상적으로 종료될 경우 기존 버전에서는 현재 처리 중인 요청에 대해서만 응답을 처리하여 전달한 후 큐에 쌓여 있는 요청에 대하여 TPECLOSE 에러를 전달하였다. 하지만 Tmax 5에서는 Tmax 엔진을 다운시키기 이전에 요청된 모든 클라이언트에 대하여 정상적으로 응답을 주는 Rolling Down 기능을 제공한다.

- 사용 방법

```
$ tmdown –R –n node_name
```

항목	설명
–n *node_name*	종료되는 노드명을 지정한다.

- 적용 환경

  Tmax 시스템이 설치된 운영 시스템 환경에서 사용할 수 있다.

- 예제

  다음은 NODEA와 NODEB가 멀티 노드(또는 멀티 도메인)로 구성되어 있으며, 총 100개의 클라이언트가 NODEA에 접속되어 있다고 가정할 경우의 처리 과정을 설명하는 예제이다.

  ◦ NODEA의 Tmax 시스템을 종료시킬 경우

```
$ tmdown –R –n NODEA
```

1. NODEA의 CLL은 클라이언트로부터의 Listen 포트를 막는다.

2. NODEA에서 기존에 처리되고 있던 요청에 대해서는 처리를 완료한 후 클라이언트에게 처리 결과를 전달한다.
3. 큐에 쌓여 있는 요청에 대해서는 TMAX_BACKUP_ADDR로 설정된 NODEB에 요청을 보낸다.
4. NODEA의 Tmax 시스템이 종료된다.
5. NODEB에서는 NODEA로부터 받은 요청을 처리한 후 처음 해당 request를 요청한 클라이언트에게 처리 결과를 직접 전달한다.
6. NODEA에 접속되어 있는 모든 클라이언트는 정상 응답을 받는다.

◦ NODEB의 Tmax 시스템을 종료시킬 경우

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

1. 100개의 클라이언트 요청을 NODEA의 CLH가 약 50:50으로 NODEA와 NODEB에 분배된다.
2. tmdown -R –n NODEB로 NODEB의 Tmax 시스템을 종료시킨다.
3. NODEB의 CLL은 클라이언트부터 Listen 포트를 막는다.
4. NODEB에서 기존에 처리되고 있던 요청에 대해서는 처리를 완료한다.
5. 클라이언트는 NODEA에 접속되어 있는 상황이므로 해당 처리 결과를 NODEA의 CLH에게 전달하고, NODEA의 CLH는 클라이언트에게 처리 결과를 전달한다.
6. NODEB의 큐에 쌓여 있는 요청에 대해서는 TMAX_BACKUP_ADDR로 설정된 NODEA에 요청을 보낸다.
7. NODEB의 Tmax 시스템이 종료된다.
8. NODEA에서는 NODEB로부터 받은 요청을 처리한 후 처음 해당 request를 요청한 클라이언트에게 처리 결과를 전달한다.
9. NODEA에 접속되어 있는 모든 클라이언트는 정상 응답을 받는다. 100개의 클라이언트가 모두 정상 응답을 받아야 한다.

NODEA의 요청을 NODEB가 처리하기 위해서는 NODEA에 접속한 클라이언트의 TMAX_BACKUP_ADDR, TMAX_BACKUP_PORT가 NODEB로 설정되어 있어야 한다. 그렇지 않을 경우 NODEA의 Tmax 시스템이 종료되는 순간 아직 처리되지 않은 클라이언트 요청에 대하여 TPESYSTEM 에러를 전달한다.

2.34. tmmbfgen

텍스트의 서비스 정보 파일을 서비스 정보 바이너리 파일로 만드는 명령어이다. 사용자가 작성한 서비스 정보 파일은 그대로 웹 서비스 게이트웨이와 xwsdlgen에서 사용할 수 없기 때문에 tmmbfgen으로 새로운 파일을 생성해야 한다. tmmbfgen에 의해서 생성된 파일을 서비스 정보 바이너리 파일이라고 한다. tmmbfgen을 하면 문법을 체크하고 파라미터의 타입 체크를 통해서 웹 서비스 게이트웨이에서 참조하기 전에 미리 유효성 검사를 할 수 있으며 텍스트 문서를 여러 개로 분할하여 관리가 가능하다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

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

항목	설명
[-r *text_file*]	서비스 정보 파일(text) 리스트를 입력한다.
[-i *text_file*]	서비스 정보 파일(text)을 입력한다.
[-d *svc*]	삭제할 서비스명 리스트를 입력한다.
-o *binary_file*	생성할(변경할) 서비스 정보 바이너리 파일(default = sample)한다.

- 예제
  - 다음은 새로운 서비스 정보 바이너리 파일을 생성하는 예제이다. sample을 생성하고 sample.txt와 sample2.txt에 정의된 서비스 정보를 sample에 입력한다.

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

  - 다음은 기존 서비스 정보 바이너리 파일에 서비스 정보를 추가하는 예제이다. 기존 sample 파일에 sample.txt에 정의된 서비스 정보를 추가한다. 같은 서비스가 있는 경우 파일이 교체된다.

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

  - 다음은 기존 서비스 정보 바이너리에서 특정 서비스 삭제하는 예제이다. sample 파일에서 서비스 SVC1, SVC2의 정보를 삭제한다.

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

untmmbfen 명령어에 대한 자세한 내용은 untmmbfgen을 참고한다.

2.35. tmsnmpd

표준 SNMP 프로토콜에 의해서 Tmax 구성 및 성능 정보 조회하는 명령어로 조회가 가능하도록 SNMP Agent인 tmsnmpd가 추가되었다. CNMP 프로토콜은 기본적으로 UDP 161번을 사용하므로 이 포트를 사용하려면 root 권한이 있어야 하며, 1024 이후의 포트를 사용할 경우 일반 사용자 계정으로도 사용 가능하다. Tmax SNMP Agent는 tmboot와 무관하게 tmsnmpd를 실행해야 한다. racd와 동일하게 tmboot / tmdown과 무관하게 동작한다. tmsnmpd를 실행하기 전에 환경변수로 TMAXDIR이 지정되어 있어야 한다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

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

항목	설명
[-h]	사용법을 조회한다.
[-d]	디버그 모드로 동작한다.
[-f]	config_file로 해당 위치의 config 파일을 사용하고, 설정하지 않을 경우 ${TMAXDIR}/snmp/tmsnmpd.conf를 사용한다.
[-n *node_name*]	멀티 노드 환경에서 노드명을 지정한다.
[-i]	env_file로 해당 위치의 env_file 파일을 사용한다.
[-l]	환경 파일 내에서 적용할 라벨을 지정한다.
[-t]	TMM과의 연결 체크 주기를 설정한다.
[-p]	성능 정보의 수집 주기를 설정한다.
[-V]	버전 정보를 조회한다.
[프로토콜]	TCP, UDP 중 지정한다. (기본값: UDP)
[*IP*]	모니터링할 IP를 지정한다.
[*PORT*]	SNMP 유틸리티가 접속할 때 사용할 tmsnmpd의 Listen 포트를 지정한다.

- 예제

  tmsnmpd를 실행할 때 root 권한이 없으면 아래와 같이 포트를 지정한다. 또는 tmsnmpd.conf에서 agentAddress로 지정할 수 있다.

```
$ tmsnmpd 9999
```

tmmbfgen 명령어에 대한 자세한 내용은 tmmbfgen을 참고한다.

Tmax SNMP MIB

사용자는 Tmax 서버의 속성 및 통계 정보에 접근할 수 있으며, 이것들은 Tmax MIB에 정의되어 있다. MIB는 표준 SNMP의 한 부분이다. MIB에 대한 내용은 www.ietf.org 사이트를 참고한다.

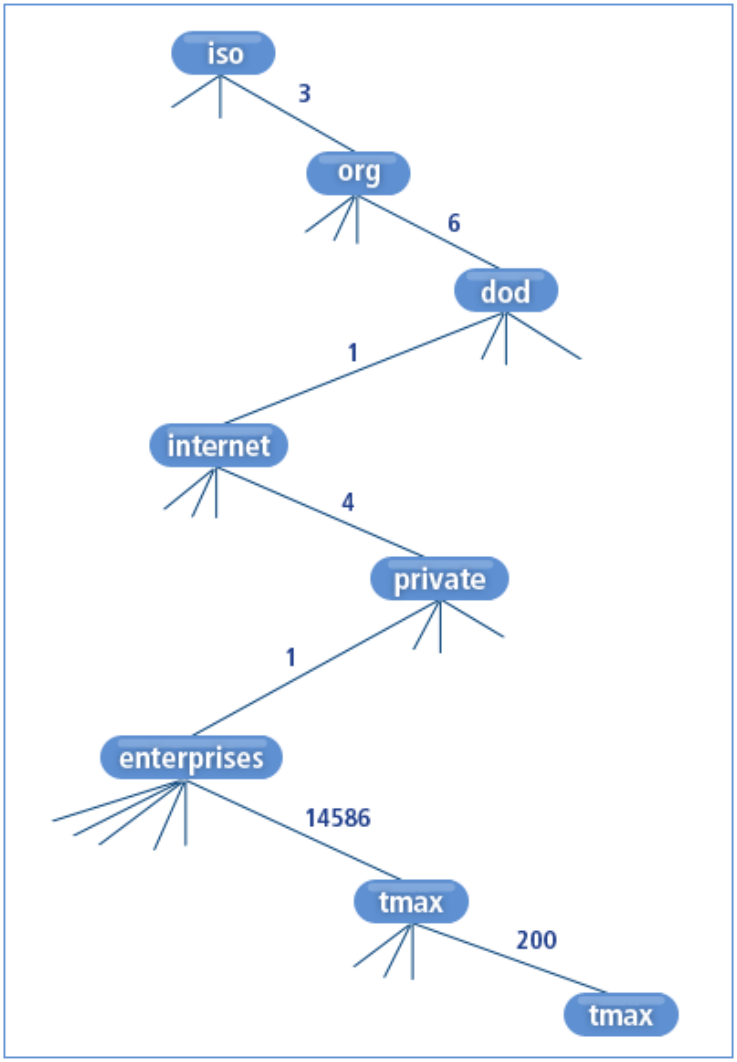
- MIB 브라우징

  Tmax MIB 파일은 ${TMAXDIR}/snmp/mib/TMAXSOFT-TMAX-MIB.mib로 존재해야 한다. 직접 MIB 파일을 확인할 수 있으며, 3rd-party MIB 브라우저를 사용할 수 있다. Tmax에서는 MIB 브라우저를 제공하지는 않지만, SNMP 유틸리티 벤더 대부분에서 MIB 브라우저를 제공한다.

- 객체 식별(OID)

OID는 관리되는 객체를 구별하기 위한 정수를 나열한 것으로, OID 트리 구조를 사용해서 객체의 경로를 정의한다. OID는 MIB 파일에서 정의된다. Tmax SNMP Agent로 특정한 OID 값을 가진 SNMP 패킷을 보내면 OID와 일치하는 정보를 얻을 수 있다.

다음은 Tmax MIB의 OID 트리를 나타낸다. TmaxSoft의 Enterprise OID는 1.3.6.1.4.1.14586이다. 모든 Tmax의 속성값의 접두어는1.3.6.1.4.1.14586.200이 된다.



Tmax SNMP의 OID

- 주의사항

  현재 버전에서는 SNMP v1, v2c만을 지원한다. 구성 및 성능에 대한 조회만 가능하다(READ-ONLY 기능만 지원). SNMP 유틸리티(snmpget, snmpwalk 등)는 해당 OS 벤더에서 지원받거나 NET-SNMP(http://net-snmp.sourceforge.net/) 등을 사용할 수도 있다.

  NET-SNMP의 간단한 사용 예는 아래와 같다.

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

환경설정

- 환경설정 파일의 위치

SNMP Agent(tmsnmpd)를 사용하기 위해서 **${TMAXDIR}/snmp/tmsnmpd.conf** 환경설정 파일이 있어야 한다.

- 환경설정 방법

  SNMP 유틸리티(snmpget, snmpwalk 등)를 사용하여 정보를 조회하거나 값을 설정하는 등의 통신을 위한 community string을 지정한다. Tmax 4 SP1 이상에서는 rocommunity 설정만 지원한다. IPv6 통신 환경이라면 'rocommunity6'로 설정한다.

```
rocommunity <string>
```

- 환경설정 예제

  만약 'tmax'라는 community명으로 UDP로 포트 '9999'로 IPv4와 IPv6로 동시에 열고 싶다면 ${TMAXDIR}/snmp/tmsnmpd.conf 파일에 아래와 같이 설정하고 tmsnmpd를 기동한다.

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

2.36. tperr

tperr은 Tmax 에러 번호와 에러 타입을 이용하여 에러에 관한 자세한 정보를 조회할 수 있도록 하는 명령어이다. Tmax 시스템 운용 중에 발생하는 에러에 대해서 쉽게 원인을 찾아 해결할 수 있도록 해준다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ tperr {-t error_type} {-e errno_number} [-f filename] [-h] [-V]
```

항목	설명
{-t *error_type*}	에러 타입을 설정한다. 모듈명을 지정하는 옵션으로 반드시 필요하다. 다음은 에러 타입에 대한 설명이다. • TPE : Tmax API 에러이다. • ADM : Tmax Administration 툴(tmadmin) 에러이다. • BOOT : Tmax Boot-up 툴(tmboot) 에러이다. • CAS : CAS(Client Authentication Server) 에러이다. • CFL : CFL(Tmax Configuration File Compiler) 에러이다. • CLH : CLH(Client Handler) 에러이다. • CLI : CLI(Client Library) 에러이다. (libcli.a, tmax.dll) • CLL : CLL(Client Listener) 에러이다. • DOWN : Tmax Shutdown 툴(tmdown) 에러이다. • FDLC : FDLC(FDL File Compiler) 에러이다. • GST : 서비스 테이블 생성 툴(gst) 에러이다. • MKPW : 패스워드 관리 Tool(mkpw) 에러이다. • RAC : RACD(Remote Access Control Daemon) 에러이다. • RQS : RQS(Reliable Queue Server) 에러이다. • SDLC : SDLC(SDL File Compiler) 에러이다. • SVR : 서버 라이브러리 에러이다. (libsvr.a) • TCPGW : Custom TCP/IP 게이트웨이 에러이다. (libtcpgw.a) • TMD : Server Application Test Tool이다. • TMGW : Custom TCP/IP Gateway이다. (libtcpgw.a) • TMM : Tmax 관리(tmm) 에러이다. • TMS : TMS 라이브러리 에러이다. (libtms.a)
{-e *errno_number*}	에러 번호를 지정하는 옵션으로 반드시 필요하다.
[-f *filename*]	에러 메시지에 관한 내용이 담겨 있는 텍스트 파일로 지정하지 않을 경우 ${TMAXDIR}/bin/_tmax_errno라는 텍스트 파일이 적용되며 이 파일은 기본적으로 제공된다.
[-h]	명령어 도움말 옵션이다.
[-V]	실행 파일의 버전을 확인할 수 있다.

- 적용 환경

  Tmax 시스템이 설치된 운영 시스템 환경에서 사용할 수 있다.

- 예제

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

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

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

에러 메시지의 자세한 내용은 "Tmax Error Message Reference"를 참고한다.

2.37. twagent

Tmax 웹 Agent와 연결하는 데몬 프로세스 기동을 위한 명령어이다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ twagent -m 환경 파일 [-v|-V] [-d] [-p sec] [-l filename]
```

항목	설명
-m 환경 파일	twagent 기동을 위해 환경 설정 파일을 지정한다.
[-v|-V]	버전 정보를 출력한다.
[-d]	디버그 모드를 활성화한다.
[-p *sec*]	TMM과 연결을 시도하는 주기(초)를 설정한다.
[-l *filename*]	로그 파일을 지정한다.

- 예제

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

2.38. uncfl

텍스트 형태의 Tmax 환경 파일을 컴파일하여 생성된 tmconfig(이진 Tmax 환경 파일)을 다시 역으로 분석하여

텍스트 형태의 환경 파일로 변환하는 명령어이다. 환경 파일을 컴파일하고 기동시킨 후 실수로 텍스트 환경 파일을 삭제한 경우에 유용하게 사용할 수 있다.

또한 여러 개의 텍스트 환경 파일을 작성 후 시스템 운용 중에 현재 어떤 환경으로 동작하고 있는지 확인할 경우에도 유용하게 사용할 수 있으며 tmadmin의 cfgadd 명령어를 이용하여 서비스를 동적으로 추가할 때에도 유용하게 사용될 수 있다. cfgadd 명령어에 대한 자세한 설명은 Tmax Administration Guide의 "cfgadd(ca)"를 참고한다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

```
$ uncfl [-i 이진 Tmax 환경파일명] [-o 텍스트 Tmax 환경파일명] [-h] [-V]
```

항목	설명
[-i 이진 *Tmax* 환경파일명]	역으로 분석하여 텍스트 파일로 변환할 이진 Tmax 환경 파일명을 설정한다. 경로와 함께 지정할 수 있으며, 경로가 지정되지 않은 경우 현재 디렉터리에 위치해 있는 이진 환경 파일이 참조된다. 옵션이 생략되면 기본으로 **tmconfig**라는 이름의 파일을 참조하여 텍스트 환경 파일이 생성된다.
[-o 텍스트 *Tmax* 환경파일명]	이진 Tmax 환경 파일을 역으로 분석하여 변환할 텍스트 형태의 Tmax 환경 파일명을 명시하는 데 사용되는 옵션으로 반드시 필요하다. 경로와 함께 지정될 수 있으며 경로가 지정되지 않았을 경우에 기본적으로 현재 디렉터리에 텍스트 환경 파일이 생성된다.
[-h]	명령어 도움말 옵션이다.
[-V]	실행 파일의 버전을 확인할 수 있다.

- 적용 환경

  Tmax 시스템이 설치된 운영 시스템 환경에서 사용할 수 있다.

- 예제
  - 다음은 현재 디렉터리에 있는 'tmconfig'라는 이진 Tmax 환경 파일을 참조하여 /user1/tmax/temp 디렉터리에 'basic.m'이라는 텍스트 형태의 Tmax 환경 파일을 생성하는 예제이다.

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

  - 다음은 /user1/tmax/bin 디렉터리에 있는 'tmconfig'라는 이진 Tmax 환경 파일을 참조하여 현재 디렉터리에 'basic.m'이라는 텍스트 형태의 Tmax 환경 파일을 생성하는 예제이다.

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

cfl 명령어의 자세한 내용은 cfl을 참고한다.

2.39. untmmbfgen

untmmbfgen은 서비스 정보 바이너리 파일을 서비스 정보 파일(텍스트)로 변환하는 명령어이다. tmmbfgen 명령어의 자세한 내용은 tmmbfgen를 참고한다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

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

항목	설명
-i *binary_meta_file*	서비스 정보 바이너리 파일을 지정한다.
-o *text_file*	생성될 서비스 정보 파일(텍스트)을 지정한다.

- 예제

  다음은 untmmbfgen으로 서비스 정보 바이너리 파일을 사용자들이 알아볼 수 있도록 텍스트 형태의 파일로 변환하는 예제이다.

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

2.40. xwsdlgen

웹 서비스 스펙 중 명세서 역할을 하는 WSDL 문서를 생성하는 명령어이다. 현재 WSDL 문서는 1.1, 2.0이 존재한다. xwsdlgen은 웹 서비스 게이트웨이 환경설정 파일과 서비스 정보 바이너리 파일을 입력받아서 WSDL 문서를 생성한다.

다음은 명령어 사용법에 대한 설명이다.

- 사용 방법

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

  - [options]

항목	설명
-w *version*	◦ 0 : WSDL1.1 (기본값) ◦ 1 : WSDL2.0
-b *binding_style*	◦ 0 : rpc (기본값) ◦ 1 : document

◦ 입력항목

항목	설명
-g *wsgw_config_file*	웹 서비스 게이트웨이 환경설정 파일을 설정한다.
-m *binary_meta_file*	서비스 정보 바이너리 파일을 설정한다.
-o *wsdl_file*	생성할 WSDL 파일을 설정한다.

3. 함수

본 장에서는 Tmax에서 사용할 수 있는 함수를 설명한다. 에러 코드에 대한 자세한 내용은 "Tmax Application Development Guide"를 참고한다.

3.1. 서버/클라이언트 함수

3.1.1. gettperrno

Tmax 시스템을 호출할 때 서버와 클라이언트에서 설정된 에러 코드(errno)를 반환하는 함수이다.

- 프로토타입

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

- 반환값

  에러 코드(errno)를 반환한다. 에러 코드에 대한 자세한 내용는 에러별 대응 방안이나 "Tmax Application Development Guide"를 참고한다.

- 예제

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

- 관련 함수

  gettpurcode()

3.1.2. tgstrerror

TmaxGrid API 사용할 때 발생하는 tgerrno에 해당하는 번호의 메시지를 출력한다.

- 프로토타입

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

- 파라미터

파라미터	설명
tgerrno	출력할 에러 번호이다.

- 반환값

반환값	설명
에러 메시지	에러 코드에 대한 메시지가 있는 경우이다.

3.1.3. tmax_chk_conn

클라이언트의 연결 상태를 체크하는 함수로 tpstart 수행 여부 체크, 소켓 상태 점검, 메시지 전달로써 연결 상태를 점검하는 역할을 한다.

- 프로토타입

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

- 파라미터

파라미터	설명
sec	다음과 같이 3가지 값 중 하나를 지정할 수 있다. • 음수 : **tpstart()** 수행 여부를 체크한다. • 0 : 소켓의 상태를 점검한다. **tpalivechk()**와 동일한 역할을 수행한다. • 양수 : 메시지 전달로써 연결 상태를 점검한다. sec만큼 응답을 대기한다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tmax_chk_conn()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEPROTO]	아직 tpstart()가 수행되지 않았다.
[TPESYSTEM]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

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

        if (argc != 2)
            error processing…

        if ( (ret = tmaxreadenv( “tmax.env”,”TMAX” )) == -1 )
            error processing…

        /* tpstart 수행 여부 체크 */
        /***
        chkno = tmax_chk_conn(-1);
        printf(“chkno = %d\n”, chkno);
        if(chkno < 0)
        {
              printf(“tpstart is not started\n”);
              printf(“errno = %d\tstrerror=%s\n”, tperrno, tpstrerror(tperrno) );
        }
        else if(chkno == 0)
        {
              printf(“tpstart is started\n”);
        }
        ***/

        if (tpstart((TPSTART_T *)NULL) == -1)
              error processing…
        if ((sndbuf = (FBUF *)tpalloc(“FIELD”, NULL, 0)) == NULL)
              error processing…
        if ((rcvbuf = (FBUF *)tpalloc(“FIELD”, NULL, 0)) == NULL)
              error processing…
        /* 소켓의 상태 점검 */
        /***********
```

```
        chkno = tmax_chk_conn(0);
        printf("chkno = %d\n", chkno);
        if(chkno < 0)
        {
              printf("The situation of socket is bad\n");
              printf("errno = %d\tstrerror=%s\n", tperrno, tpstrerror(tperrno) );
        }
        else if(chkno == 0)
        {
              printf("The situation of socket is good\n");
        }
        **********/
        printf("My Count = %d\n", Count++);
        strcpy(input, "INPUT");
        strcpy(sndata, argv[1]);
        fc = fbput(sndbuf, fbget_fldkey(input), sndata, 0);
        if (tpcall("FDLTOUPPER", (char *)sndbuf, 0, (char **)&rcvbuf,
                    (long *)&rcvlen, TPNOFLAGS) == -1)
              error processing…
        /* 메시지 전달로써 연결 상태 점검 */
        chkno = tmax_chk_conn(1);
        printf("chkno = %d\n", chkno);
        if(chkno < 0)
        {
              printf("The situation of connection(Message Transfer) is bad\n");
              printf("errno = %d\tstrerror=%s\n", tperrno, tpstrerror(tperrno) );
        }
        else if(chkno == 0)
        {
              printf("The situation of connection(Message Transfer) is good\n");
        }

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

- 관련 함수

  tpalivechk()

3.1.4. tmax_get_sessionid

현 세션의 ID를 반환하는 함수이다.

- 프로토타입

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

- 반환값

반환값	설명
-1이 아닌 값	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tmax_get_sessionid()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEPROTO]	세션이 이미 종료되었다.

3.1.5. tmax_gq_count

GQ에 저장된 데이터의 개수를 반환한다.

- 프로토타입

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

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tmax_gq_count()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPENOENT]	현 세션을 위한 SQ가 존재하지 않는다.

3.1.6. tmax_gq_get

GQ에서 데이터를 가져오는 함수로 키를 지정하면 해당 키의 데이터를 가져온다. 기본적으로 tmax_gq_get() 을 수행하면 큐에서 데이터가 삭제된다. 데이터를 보존하려면 TPSQ_PEEK flags를 설정해야 한다.

- 프로토타입

```
#include <tmaxapi.h>
```

```
int tmax_gq_get(char *key, long keylenl, char **data, long *lenl, long flagsl)
```

- 파라미터

파라미터	설명
key	GQ에서 가져올 데이터를 위한 키값을 저장할 버퍼이다.
keylenl	키 버퍼의 크기를 지정한다. (단위: Byte)
data	GQ에서 가져올 데이터 버퍼에 대한 포인터이다. 반드시 tpalloc()로 할당된 버퍼에 대한 포인터이어야 한다.
lenl	GQ에서 가져온 데이터 버퍼의 크기가 저장된다. (단위: Byte)
flagsl	TPSQ_PEEK이 지정될 수 있다. 이 flags를 지정할 경우 데이터를 큐에서 삭제하지 않는다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tmax_gq_get()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEMATCH]	지정한 키의 데이터가 존재하지 않는다.

3.1.7. tmax_gq_getkeylist

GQ의 키 리스트를 가져오기 위한 함수이다. 키 리스트는 **SQ_KEYLIST_T** 타입의 핸들이 반환되며, 각 키에 대한 정보는 **tmax_keylist_count()**, **tmax_keylist_count()**, **tmax_keylist_free()** 함수를 통해서 확인할 수 있다. 키 리스트는 Byte 단위로 ASCII 코드값을 비교하여 낮은 값 우선으로 정렬된다.

키 값을 지정할 경우 해당 키 값보다 크거나 같은 키 리스트가 정렬되어 조회된다. 키 값을 NULL로 지정할 경우 전체 키 리스트가 조회된다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	키 값을 저장할 버퍼이다.
keylenl	키 버퍼의 크기를 지정한다. (단위: Byte)

- 반환값

반환값	설명
SQ_KEYLIST_T	함수 호출에 성공한 경우이다.
NULL	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tmax_gq_getkeylist()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEMATCH]	지정한 키보다 크거나 같은 키가 존재하지 않는다.

3.1.8. tmax_gq_keygen

시스템 키 생성 및 가져오기 위한 함수로 시스템 키 사이즈는 SQ_SYSKEY_SIZE(16Byte)이므로 이보다 버퍼 크기를 크거나 같게 할당해야 한다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	생성된 시스템 키값이 저장될 버퍼이다. 버퍼의 크기는 SQ_SYSKEY_SIZE(16Byte)보다 크게 할당해야 한다.
keylenl	키 버퍼의 크기를 지정한다. 반드시 SQ_SYSKEY_SIZE(16Byte)보다 커야 한다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tmax_gq_keygen()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	잘못된 키를 입력한 경우이다.

3.1.9. tmax_gq_purge

GQ의 데이터를 삭제하기 위한 함수로 키를 지정할 경우 해당 키의 데이터를 삭제하고 키를 지정하지 않을 경우 모든 데이터를 삭제한다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	GQ에서 삭제할 데이터를 위한 키값을 저장할 버퍼이다. NULL일 경우 GQ의 모든 데이터를 삭제한다.
keylenl	키 버퍼의 크기를 지정한다. (단위: Byte)

- 반환값

반환값	설명
삭제된 데이터의 개수	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tmax_gq_purge()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	잘못된 키를 입력한 경우이다.
[TPENOENT]	현 세션을 위한 SQ가 존재하지 않는다.

3.1.10. tmax_gq_put

데이터를 GQ에 저장하는 함수로 키와 데이터 값을 전달한다. **tmax_gq_keygen()** 함수를 통해 생성된 시스템 키를 사용한다면 **TPSQ_SYSKEY** flags를 사용해야 한다. 시스템 키를 생성하여 저장하려면 **TPSQ_KEYGEN** flags를 사용해야 한다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	GQ에 저장할 데이터를 위한 키값을 저장할 버퍼이다.
keylenl	키 버퍼의 크기를 지정한다. (단위: Byte)
data	GQ에 저장할 데이터를 담고 있는 버퍼이다. 반드시 **tpalloc()**으로 할당된 버퍼에 대한 포인터이어야 한다.
lenl	GQ에 저장할 데이터 버퍼의 크기를 지정한다. (단위: Byte)
flagsl	TPSQ_UPDATE, TPSQ_SYSKEY, TPSQ_KEYGEN이 지정될 수 있다. 다음은 flagsl에 설정 가능한 값의 설명이다. • TPSQ_UPDATE 키 값이 같을 경우 업데이트한다. 이 flags가 설정되어 있지 않으면 TPEMATCH 에러가 반환된다. • TPSQ_SYSKEY 시스템 키를 사용할 경우 설정한다. • TPSQ_KEYGEN 시스템 키를 자동으로 생성하여 저장한다. flags를 사용할 경우 key는 SQ_SYSKEY_SIZE 크기(keylenl = SQ_SYSKEY_SIZE)로 할당해야 한다. 성공적으로 수행할 경우 key에 생성된 키 값이 저장된다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tmax_gq_put()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEMATCH]	사용된 키값이 이미 존재한다.
[TPELIMIT]	GQ의 최대 키 개수를 초과하였다.

3.1.11. tmax_grid_count

전체 Key 개수를 참조한다.

- 프로토타입

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

- 파라미터

파라미터	설명
flags	사용 않는다.

- 반환값

반환값	설명
>= 0	함수 호출에 성공한 경우이다. 반환값이 전체 Key의 개수이다.
< 0	함수 호출에 실패한 경우이다. tgerrno에 에러 코드가 설정된다.

- 오류

  tmax_grid_count()가 정상적으로 수행되지 않을 경우 tgerrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TGEOS]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TGENOMEM]	메모리가 부족하여 할당에 실패한 경우 발생한다.
[TGECONF]	TmaxGrid가 설정되지 않은 경우이다.
[TGEENABLE]	TmaxGrid가 동작하지 않는 경우이다.

3.1.12. tmax_grid_create

Key를 생성한다. Key를 생성할 때 key의 타입을 지정해야 한다.

만약 생성하려는 Node에 대해서 enqueue, dequeue를 사용하거나 Lock, Unlock을 사용하기 위해서는 반드시 TG_QUEUE 또는 TG_LOCK 타입으로 설정해야 한다. 해당 타입으로 생성된 Node에는 enqueue, dequeue, Lock, Unlock API를 사용하는 것 이외의 방법으로 Child Node를 생성하는 것은 불가능하다. 또한 이들 타입은 항상 TG_PERSISTENT 타입으로 생성된다. TG_TEMPORARY로 생성된 Node의 Child Node를 생성할 때 TG_PERSISTENT로 생성하면 안된다.

TG_TEMPORARY 타입의 Node가 삭제될 때 모든 Child Node가 함께 삭제되므로 주의해야 한다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	생성할 Key 이름이다. 127Byte를 넘을 수 없다.
keylen	Key의 길이이다.
flags	동작 방식을 결정한다. • TG_PERSISTENT 생성한 클라이언트가 종료해도 삭제되지 않는다. • TG_TEMPORARY 생성한 클라이언트가 종료하면 자동으로 삭제되거나 timeout이 지날 경우 자동으로 삭제된다. • TG_QUEUE enqueue, dequeue을 사용할 Key를 생성한다. • TG_LOCK Lock, Unlock을 사용할 Key를 생성한다.

- 반환값

반환값	설명
>= 0	함수 호출에 성공한 경우이다.
< 0	함수 호출에 실패한 경우이다. tgerrno에 에러 코드가 설정된다.

- 오류

tmax_grid_create()가 정상적으로 수행되지 않을 경우 tgerrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TGEINVAL]	Key가 NULL이거나 key_len <=0일 경우 또는 Key의 길이가 128Byte가 넘어갈 경우 또는 flags에 적절하지 않은 값이 지정된 경우이다.
[TGEOS]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TGENOMEM]	메모리가 부족하여 할당에 실패한 경우 발생한다.
[TGEKEY]	Key를 정상적으로 인식하지 못하는 경우이다.

에러 코드	설명
[TGEMAXCHL]	환경설정의 Max Child 제한에 도달해서 더 이상 Child Key를 생성할 수 없는 경우이다.
[TGEPROTO]	TG_QUEUE 또는 TG_LOCK 타입의 Key에 대한 Child Key를 생성한 경우이다.
[TGESHMFULL]	TmaxGrid의 공유 메모리의 공간 부족할 경우 또는 환경설정 TGMAX 최댓값까지 Key가 생성되어서 더 이상 Key를 생성할 수 없는 경우이다.
[TGEDUPKEY]	Key 이름이 중복될 경우이다.
[TGECONF]	TmaxGrid가 설정되지 않은 경우이다.
[TGEENABLE]	TmaxGrid가 동작하지 않는 경우이다.

3.1.13. tmax_grid_create2

Key를 생성한다. Key를 생성하면서 함께 Value를 입력할 수 있고, watcher도 함께 등록할 수 있다. 자세한 정보는 tmax_grid_create, tmax_grid_set, tmax_grid_set_watcher API를 참고한다. data가 NULL이 아닌 경우 value를 입력한다. Callback 함수가 NULL이 아닌 경우 watcher가 설정된다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	생성할 Key 이름이다. 127Byte를 넘을 수 없다.
keylen	Key의 길이이다.
data	Value이다. tpalloc으로 할당받은 포인터이다. NULL일 경우 Value를 저장하지 않고 Key만 생성한다.
len	data의 길이이다.
callback	watcher 등록 후 이벤트가 발생할 때 호출할 Callback 함수 포인터이다.
args	callback를 함수 호출하는 경우 넘겨줄 인자 포인터이다.

파라미터	설명
flags	동작 방식을 결정한다. • TG_PERSISTENT 생성한 클라이언트가 종료해도 삭제되지 않는다. • TG_TEMPORARY 생성한 클라이언트가 종료하면 자동으로 삭제된다. • TG_QUEUE enqueue, dequeue을 사용할 Key를 생성한다. • TG_LOCK Lock, Unlock을 사용할 Key를 생성한다. • TG_WATCHER_PERSISTENT watcher를 등록할 때 영원히 해당 클라이언트에게 공지를 수행한다. Key가 삭제되면 watcher는 자동으로 삭제된다. • TG_WATCHER_ONCE watcher를 등록할 때 한 번만 공지하고 그 이후로는 공지하지 않는다. 만약 더 관심이 있다면 watcher를 재등록해야 한다. • TG_EVENT_DELETE_SELF 자신의 Key가 삭제되었을 때 이벤트를 수신한다. callback 함수에서 해당 이벤트를 수신하게 되면 등록된 watcher가 삭제되므로, watcher를 재등록해야 한다. • TG_EVENT_CREATE_CHILD Child Key가 생성되었을 때 이벤트를 수신한다. • TG_EVENT_SET_SELF 자신의 Key에 데이터가 set되었을 때 이벤트를 수신한다. • TG_EVENT_GET_SELF 자신의 Key에 데이터가 get되었을 때 이벤트를 수신한다.

- 반환값

반환값	설명
>= 0	함수 호출에 성공한 경우이다.
< 0	함수 호출에 실패한 경우이다. tgerrno에 에러 코드가 설정된다.

- 오류

  tmax_grid_create2()가 정상적으로 수행되지 않을 경우 tgerrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TGEINVAL]	Key가 NULL이거나 key_len <=0일 경우 또는 Key의 길이가 128Byte가 넘어갈 경우 또는 flags에 적절하지 않은 값이 지정된 경우이다.
[TGEOS]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TGENOMEM]	메모리가 부족하여 할당에 실패한 경우 발생한다.
[TGEKEY]	Key를 정상적으로 인식하지 못하는 경우이다.
[TGEMAXCHL]	환경설정의 Max Child 제한에 도달해서 더 이상 Child Key를 생성할 수 없는 경우이다.
[TGEPROTO]	TG_QUEUE 또는 TG_LOCK 타입의 Key에 대한 Child Key를 생성한 경우이다.
[TGESHMFULL]	TmaxGrid의 공유 메모리의 공간 부족할 경우 또는 환경설정 TGMAX 최댓값까지 Key가 생성되어서 더 이상 Key를 생성할 수 없는 경우이다.
[TGEDUPKEY]	Key 이름이 중복될 경우이다.
[TGECONF]	TmaxGrid가 설정되지 않은 경우이다.
[TGEENABLE]	TmaxGrid가 동작하지 않는 경우이다.

3.1.14. tmax_grid_dequeue

Key 이름에 해당하는 Child Key의 제일 처음 번호의 Value를 가져오며, 해당 Child Key는 삭제된다. Child Key에 대해서 tmax_grid_get, tmax_grid_set API로 Value에 접근하는 것은 허용하지 않는다. 그러나 Key 이름에 대해서는 가능하다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	Key 이름이다. 127Byte를 넘을 수 없다.
keylen	Key의 길이이다.
type	함수 수행 후 데이터의 type을 반환한다.
data	Key에서 가져올 Value의 버퍼(tpalloc이나 fballoc으로 할당한 데이터만 가능하다. 미리 할당을 해야 한다.)이다. 가져온 데이터가 버퍼의 크기보다 더 크다면 내부적으로 재할당된다.
len	Node에서 가져올 데이터의 길이가 저장된다.
flags	사용하지 않는다.

- 반환값

반환값	설명
>= 0	함수 호출에 성공한 경우이다.
< 0	함수 호출에 실패한 경우이다. tgerrno에 에러 코드가 설정된다.

- 오류

  tmax_grid_dequeue()가 정상적으로 수행되지 않을 경우 tgerrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TGEINVAL]	Key가 NULL이거나 key_len <=0일 경우 또는 Key의 길이가 128Byte가 넘어갈 경우 또는 flags에 적절하지 않은 값이 지정된 경우이다.
[TGEOS]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TGENOMEM]	메모리가 부족하여 할당에 실패한 경우 발생한다.
[TGEKEY]	Key를 정상적으로 인식하지 못하는 경우이다.
[TGENOKEY]	해당하는 Key가 없는 경우이다.
[TGENODATA]	입력된 데이터가 존재하지 않을 경우이다.
[TGECONF]	TmaxGrid가 설정되지 않은 경우이다.
[TGEENABLE]	TmaxGrid가 동작하지 않는 경우이다.
[TGEOTYPE]	data tpalloc으로 할당하지 않았을 경우 또는 가져온 데이터가 올바르지 않은 SDL 또는 FDL 데이터일 경우이다.

3.1.15. tmax_grid_destroy

Key를 삭제한다. value까지 함께 삭제한다. 기본적으로 Child Key가 존재하면 삭제되지 않지만, flags를 설정하면 persistent type에 한정하여 Child Key까지 함께 삭제할 수 있다. 또한 Key가 삭제될 때 watcher가 등록되어 있으면 설정된 이벤트를 통지 후 함께 삭제된다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	생성할 Key 이름이다. 127Byte를 넘을 수 없다.
keylen	Key의 길이이다.

파라미터	설명
flags	동작 방식을 결정한다. • TG_CHILDREN - Child Key까지 함께 삭제한다.

- 반환값

반환값	설명
>= 0	함수 호출에 성공한 경우이다.
< 0	함수 호출에 실패한 경우이다. tgerrno에 에러 코드가 설정된다.

- 오류

tmax_grid_destroy()가 정상적으로 수행되지 않을 경우 tgerrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TGEINVAL]	Key가 NULL이거나 key_len <=0일 경우 또는 Key의 길이가 128Byte가 넘어갈 경우 또는 flags에 적절하지 않은 값이 지정된 경우이다.
[TGEOS]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TGENOMEM]	메모리가 부족하여 할당에 실패한 경우 발생한다.
[TGEKEY]	Key를 정상적으로 인식하지 못하는 경우이다.
[TGESHMFULL]	TmaxGrid의 공유 메모리의 공간 부족할 경우 또는 환경설정 GQMAX 최댓값까지 Key가 생성되어서 더 이상 Key를 생성할 수 없는 경우이다.
[TGECONF]	TmaxGrid가 설정되지 않은 경우이다.
[TGEENABLE]	TmaxGrid가 동작하지 않는 경우이다.
[TGEDELCHL]	Child Key가 존재하여 삭제가 실패한 경우이다.
[TGEPROTO]	Child Key에 TG_QUEUE 또는 TG_LOCK 타입의 Node가 존재하거나 이들 타입의 Child Key를 삭제할 경우이다.
[TGENOKEY]	해당하는 Key가 없는 경우이다.

3.1.16. tmax_grid_enqueue

Key 이름으로 데이터를 입력한다. 해당하는 Key의 Child Key의 마지막 번호로 데이터가 추가된다. Key 이름은 반드시 TG_QUEUE 타입으로 생성되어야 한다. Key를 생성하지 않았을 경우에는 자동으로 Key를 생성하고 데이터를 입력한다. 순서가 보장된다. Child Key에 대해서 tmax_grid_get, tmax_grid_set API로 직접 데이터에 접근하는 것은 허용하지 않는다. 그러나 Key 이름에 대해서는 가능하다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	생성할 Key 이름이다. 127Byte를 넘을 수 없다.
keylen	Key의 길이이다.
type	해당하는 Key의 value type이다. tpalloc한 데이터의 경우 0을 입력하고 원시 타입의 경우 다음을 입력한다. • TMAX_GRID_INT • TMAX_GRID_FLOAT • TMAX_GRID_LONG • TMAX_GRID_SHORT • TMAX_GRID_DOUBLE • TMAX_GRID_STRING 원시 타입을 사용할 때는 tpalloc하지 않는다.
data	Key에 입력할 Value이다. tpalloc이나 fballoc으로 할당한 데이터만 가능하다. 미리 할당을 해야 한다.
len	data의 길이이다.
flags	동작 방식을 결정한다. • TG_TEMPORARY 생성한 클라이언트가 종료하면 자동으로 삭제한다. 만약 "/test"라는 키로 enqueue를 한 경우 enqueue의 대상인 "/test/0"만 삭제되고 부모 노드인 "/test"는 삭제되지 않는다.

- 반환값

반환값	설명
>= 0	함수 호출에 성공한 경우이다.
< 0	함수 호출에 실패한 경우이다. tgerrno에 에러 코드가 설정된다.

- 오류

tmax_grid_enqueue()가 정상적으로 수행되지 않을 경우 tgerrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TGEINVAL]	Key가 NULL이거나 key_len <=0일 경우 또는 Key의 길이가 128Byte가 넘어갈 경우 또는 flags에 적절하지 않은 값이 지정된 경우이다.
[TGEOS]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TGENOMEM]	메모리가 부족하여 할당에 실패한 경우 발생한다.

에러 코드	설명
[TGEKEY]	Key를 정상적으로 인식하지 못하는 경우이다.
[TGEMAXCHL]	환경설정의 Max Child 제한에 도달해서 더 이상 Child Key를 생성할 수 없는 경우이다.
[TGEPROTO]	Key 이름이 TG_QUEUE 타입으로 생성된 Node가 아닌 경우이다.
[TGESHMFULL]	TmaxGrid의 공유 메모리의 공간 부족할 경우 또는 환경설정 GQMAX 최댓값까지 Key가 생성되어서 더 이상 Key를 생성할 수 없는 경우이다.
[TGEDUPKEY]	Key 이름이 중복될 경우이다.
[TGECONF]	TmaxGrid가 설정되지 않은 경우이다.
[TGEENABLE]	TmaxGrid가 동작하지 않는 경우이다.
[TGEITYPE]	data가 잘못된 SDL이나 FDL일 경우 발생한다.

3.1.17. tmax_grid_get

key의 value를 가져오고 해당 key의 value는 삭제한다. value를 삭제하지 않고 가져오기 위해서는 flags를 지정해야 한다. 참고로 Key를 삭제하려면 tmax_grid_destroy 함수를 호출해야 한다.

TG_QUEUE 또는 TG_LOCK 타입의 Child Key의 경우에는 value를 가져올 수 없다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	생성할 Key 이름이다. 127Byte를 넘을 수 없다.
keylen	Key의 길이이다.
type	함수 수행 후 데이터의 type을 반환한다.
data	함수 수행 후 Value를 반환한다. tpalloc이나 fballoc으로 미리 할당해야 한다.
len	함수 수행 후 Value의 길이를 할당한다.
flags	동작 방식을 결정한다. • TG_GET_PEEK 데이터를 가져오고 해당 Node의 데이터를 삭제하지 않는다.

- 반환값

반환값	설명
>= 0	함수 호출에 성공한 경우이다.

반환값	설명
< 0	함수 호출에 실패한 경우이다. tgerrno에 에러 코드가 설정된다.

- 오류

  tmax_grid_get()가 정상적으로 수행되지 않을 경우 tgerrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TGEINVAL]	Key가 NULL이거나 key_len <=0일 경우 또는 Key의 길이가 128Byte가 넘어갈 경우 또는 flags에 적절하지 않은 값이 지정된 경우이다.
[TGEOS]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TGENOMEM]	메모리가 부족하여 할당에 실패한 경우 발생한다.
[TGEKEY]	Key를 정상적으로 인식하지 못하는 경우이다.
[TGECONF]	TmaxGrid가 설정되지 않은 경우이다.
[TGEENABLE]	TmaxGrid가 동작하지 않는 경우이다.
[TGEOTYPE]	data tpalloc으로 할당하지 않았을 경우 또는 가져온 데이터가 올바르지 않은 SDL이나 FDL 타입일 경우이다.

3.1.18. tmax_grid_get_child_with_index

Child Key의 정보를 담고 있는 TG_KEYLIST_T 핸들러에서 nth번째의 key에 대한 정보를 TG_KEYINFO_T 구조체에 저장한다.

- 프로토타입

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

- 파라미터

파라미터	설명
keylist	tmax_grid_get_children()을 호출하여 리턴받은 핸들러이다.
nth	참조할 Child Key의 번호이다.
keyinfo	Child Node의 정보를 저장할 구조체, 할당된 버퍼를 사용해야 한다.

- 반환값

반환값	설명
>= 0	함수 호출에 성공한 경우이다.
< 0	함수 호출에 실패한 경우이다. tgerrno에 에러 코드가 설정된다.

- 오류

  tmax_grid_get_child_with_index()가 정상적으로 수행되지 않을 경우 tgerrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TGEINVAL]	keylist가 NULL이거나 nth < 0 또는 keyinfo가 NULL인 경우이다.
[TGEOS]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TGENOMEM]	메모리가 부족하여 할당에 실패한 경우 발생한다.
[TGEITYPE]	잘못된 TG_KEYLIST_T 핸들러 값이 입력된 경우이다.
[TGELIMIT]	nth가 Child Node의 개수보다 더 큰 경우이다.

3.1.19. tmax_grid_get_children

Key 이름으로 Child Key 이름의 리스트를 참조한다. Child Key의 정보를 담고 있는 TG_KEYLIST_T 핸들러를 리턴한다. 실패할 경우에는 NULL이 리턴된다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	Key 이름이다. 127Byte를 넘을 수 없다.
keylen	Key의 길이이다.
child_count	동작 방식을 결정한다.

- 반환값

반환값	설명
NULL이 아닌 경우	TG_KEYLIST_T로 Key 이름 리스트의 시작 포인터를 반환한다. 더 이상 사용하지 않는다면 free를 반드시 수행해야 한다.
NULL인 경우	함수 호출에 실패한 경우이다. tgerrno에 에러 코드가 설정된다.

- 오류

  tmax_grid_get_children()가 정상적으로 수행되지 않을 경우 tgerrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TGEINVAL]	Key가 NULL이거나 key_len <=0일 경우 또는 Key의 길이가 128Byte가 넘어갈 경우 또는 flags에 적절하지 않은 값이 지정된 경우이다.
[TGEOS]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TGENOMEM]	메모리가 부족하여 할당에 실패한 경우 발생한다.
[TGEKEY]	Key를 정상적으로 인식하지 못하는 경우이다.
[TGECONF]	TmaxGrid가 설정되지 않은 경우이다.
[TGEENABLE]	TmaxGrid가 동작하지 않는 경우이다.
[TGENOKEY]	해당하는 Key가 없는 경우이다.

3.1.20. tmax_grid_is_exist

Key가 존재하는지 검사한다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	검사할 Key 이름이다. 127Byte를 넘을 수 없다.
keylen	Key의 길이이다.
flags	사용하지 않는다.

- 반환값

반환값	설명
0	Key가 존재하지 않는다.
1	Key만 존재하고 Value는 존재하지 않는다.
2	Key와 Value 모두 존재한다.
< 0	함수 호출에 실패한 경우이다. tgerrno에 에러 코드가 설정된다.

- 오류

  tmax_grid_is_exist()가 정상적으로 수행되지 않을 경우 tgerrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TGEINVAL]	Key가 NULL이거나 key_len <=0일 경우 또는 Key의 길이가 128Byte가 넘어갈 경우 또는 flags에 적절하지 않은 값이 지정된 경우이다.
[TGEOS]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TGENOMEM]	메모리가 부족하여 할당에 실패한 경우 발생한다.
[TGEKEY]	Key를 정상적으로 인식하지 못하는 경우이다.
[TGECONF]	TmaxGrid가 설정되지 않은 경우이다.
[TGEENABLE]	TmaxGrid가 동작하지 않는 경우이다.

3.1.21. tmax_grid_keylist_free

Child Key의 정보를 담고 있는 TG_KEYLIST_T 핸들러의 자원을 해제한다.

- 프로토타입

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

- 파라미터

파라미터	설명
keylist	tmax_grid_get_children()을 호출하여 리턴받은 핸들러이다.

- 반환값

반환값	설명
>= 0	함수 호출에 성공한 경우이다.
< 0	함수 호출에 실패한 경우이다. tgerrno에 에러 코드가 설정된다.

- 오류

  tmax_grid_keylist_free()가 정상적으로 수행되지 않을 경우 tgerrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TGEINVAL]	keylist가 NULL인 경우이다.
[TGEOS]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TGEITYPE]	잘못된 TG_KEYLIST_T 핸들러 값이 입력된 경우이다.

3.1.22. tmax_grid_lock

Key 이름으로 Lock을 수행한다. 다른 클라이언트 또는 서버에서 같은 Key 이름으로 Lock을 이미 수행한 경우 이들 프로세스에서 Unlock을 수행해서 자신의 차례가 돌아 올 때까지 타임아웃으로 지정한 시간 동안 대기한다.

Lock이 성공한 이후에는 반드시 Unlock을 수행해야 한다. 만약 프로그램이 Unlock을 수행하지 않고 종료할 경우 자동으로 Unlock 처리가 되며, 다음 차례의 프로세스에서 Lock을 획득한다. Key는 반드시 TG_LOCK으로 생성되어야 한다. TG_LOCK으로 생성한 key에는 data를 get/set 할 수 없다. Key를 생성하지 않고 해당 키에 Lock을 수행할 경우 자동으로 해당하는 이름의 Key를 생성한다. lock이 수행될 때에는 key에 자동으로 부여된 번호를 가지는 Child Key를 생성하고, Unlock이 호출될 때 생성되었던 Child Key를 삭제한다. Key 이름의 길이를 작게 줄 것을 권장한다. 한 프로세스 안에서 같은 Key 이름으로 재귀적으로 호출할 수 있으며, 이 경우에는 반드시 같은 횟수만큼 Unlock을 수행해야 한다. 또한 이 경우에는 단일한 Child Key만 생성된다. shared lock을 생성하려면 flags에 TG_SHARED_LOCK로 설정한다. shared lock의 경우 동시에 여러 노드가 lock을 획득할 수 있다. 이전 자식 노드가 unlock 되면 이후 연속된 자식 노드들이 shared lock일 경우 lock을 모두 한꺼번에 획득한다. 이 후 모든 shared lock이 unlock되면 대기하고 있던 TG_EXCLUSIVE_LOCK으로 설정된 자식 노드가 lock을 획득하게 된다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	생성할 Key 이름이다. 127Byte를 넘을 수 없다.
keylen	Key의 길이이다.
timeout	Lock을 획득할 수 있는 기다리는 시간이다. (단위 : 초)
flags	TG_EXCLUSIVE_LOCK, TG_SHARED_LOCK로 설정한다.

- 반환값

반환값	설명
>= 0	함수 호출에 성공한 경우이다.
< 0	함수 호출에 실패한 경우이다. tgerrno에 에러 코드가 설정된다.

- 오류

  tmax_grid_lock()가 정상적으로 수행되지 않을 경우 tgerrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TGEINVAL]	Key가 NULL이거나 key_len <=0일 경우 또는 Key의 길이가 128Byte가 넘어갈 경우 또는 flags에 적절하지 않은 값이 지정된 경우이다.
[TGEOS]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.

에러 코드	설명
[TGENOMEM]	메모리가 부족하여 할당에 실패한 경우 발생한다.
[TGEKEY]	Key를 정상적으로 인식하지 못하는 경우이다.
[TGEMAXCHL]	환경설정의 Max Child 제한에 도달해서 더 이상 Child Key를 생성할 수 없는 경우이다.
[TGEPROTO]	Key 이름이 TG_LOCK으로 생성된 Key가 아닌 경우이다.
[TGESHMFULL]	환경설정의 TGMAX 최댓값까지 Node가 생성되어서 더 이상 Lock을 위한 Child Node를 생성할 수 없는 경우이다.
[TGEMAXCHL]	환경설정의 TGMAX_CHILD 제한에 도달해서 더 이상 Lock을 위한 Child Node를 생성할 수 없는 경우이다.
[TGETIME]	timeout 동안 Lock 요청이 실패했다.
[TGECONF]	TmaxGrid가 설정되지 않은 경우이다.
[TGEENABLE]	TmaxGrid가 동작하지 않는 경우이다.

3.1.23. tmax_grid_set

Key에 Value를 입력한다. Key에 Value가 존재한다면 기존의 Value는 삭제되고 새로운 Value로 갱신된다. Key에 watcher가 등록되어 있을 경우 기존의 Value가 삭제될 때는 이벤트가 발생하지 않는다. TG_EVENT_SET_SELF 이벤트가 등록되어 있으면 이벤트를 통지한다. TG_QUEUE 또는 TG_LOCK 타입의 Key에도 Value 입력이 가능하다. 하지만 이들 타입의 Child Key에는 value를 입력할 수 없다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	생성할 Key 이름이다. 127Byte를 넘을 수 없다.
keylen	Key의 길이이다.

파라미터	설명
type	해당하는 Key의 value type이다. tpalloc한 데이터의 경우 0을 입력하고 원시 타입의 경우 다음을 입력한다. • TMAX_GRID_INT • TMAX_GRID_FLOAT • TMAX_GRID_LONG • TMAX_GRID_SHORT • TMAX_GRID_DOUBLE • TMAX_GRID_STRING 원시 타입을 사용할 때는 tpalloc하지 않는다.
data	입력할 Value이다. tpalloc으로 할당한 포인터나 실제 원시 데이터이다.
len	data의 길이이다.
flags	사용하지 않는다.

- 반환값

반환값	설명
>= 0	함수 호출에 성공한 경우이다.
< 0	함수 호출에 실패한 경우이다. tgerrno에 에러 코드가 설정된다.

- 오류

tmax_grid_set()가 정상적으로 수행되지 않을 경우 tgerrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TGEINVAL]	Key가 NULL이거나 key_len <=0일 경우 또는 Key의 길이가 128Byte가 넘어갈 경우 또는 flags에 적절하지 않은 값이 지정된 경우이다.
[TGEOS]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TGENOMEM]	메모리가 부족하여 할당에 실패한 경우 발생한다.
[TGEKEY]	Key를 정상적으로 인식하지 못하는 경우이다.
[TGEPROTO]	TG_QUEUE 또는 TG_LOCK 타입 Node의 Child Node에 데이터를 입력하려는 경우이다.
[TGESHMFULL]	TmaxGrid의 공유 메모리의 공간 부족할 경우 또는 환경설정 TGMAX 최댓값까지 Key가 생성되어서 더 이상 Key를 생성할 수 없는 경우이다.
[TGEITYPE]	data가 잘못된 SDL이나 FDL일 경우이다.
[TGECONF]	TmaxGrid가 설정되지 않은 경우이다.
[TGEENABLE]	TmaxGrid가 동작하지 않는 경우이다.

3.1.24. tmax_grid_unlock

Key 이름으로 Lock을 해제한다. 다음 차례의 Lock을 수행할 수 있다. 같은 Key 이름을 재귀적으로 Lock을 수행한 경우 반드시 같은 횟수만큼 Unlock을 수행해야 한다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	Lock Key 이름이다. 127Byte를 넘을 수 없다.
keylen	Key의 길이이다.
flags	사용하지 않는다.

- 반환값

반환값	설명
>= 0	함수 호출에 성공한 경우이다.
< 0	함수 호출에 실패한 경우이다. tgerrno에 에러 코드가 설정된다.

- 오류

  tmax_grid_unlock()가 정상적으로 수행되지 않을 경우 tgerrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TGEINVAL]	Key가 NULL이거나 key_len <=0일 경우 또는 Key의 길이가 128Byte가 넘어갈 경우 또는 flags에 적절하지 않은 값이 지정된 경우이다.
[TGEOS]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TGENOMEM]	메모리가 부족하여 할당에 실패한 경우 발생한다.
[TGEKEY]	Key를 정상적으로 인식하지 못하는 경우이다.
[TGECONF]	TmaxGrid가 설정되지 않은 경우이다.
[TGEENABLE]	TmaxGrid가 미동작한 경우이다.
[TGENOKEY]	Lock을 수행할 때 설정한 Key가 존재하지 않는 경우이다.

3.1.25. tmax_grid_set_watcher

해당 Key의 이벤트가 발생할 때 호출하는 함수를 등록한다. flags는 Bitwise OR 연산으로 여러 설정을 함께 적용할 수 있다.

watcher가 설정된 상태에서 다시 재설정하면 기존의 이벤트 설정은 새로운 값으로 대체된다. Callback을 NULL을 지정하면 watcher를 해제하며, 해당 key에 대해서 더 이상 이벤트를 받지 않는다.

Callback 함수에서 받을 수 있는 이벤트의 종류는 다음과 같다.

이벤트	설명
TG_EVENT_DELETE_CHILD	설명은 아래의 flags와 동일하다.
TG_EVENT_DELETE_SELF	설명은 아래의 flags와 동일하다.
TG_EVENT_CREATE_CHILD	설명은 아래의 flags와 동일하다.
TG_EVENT_SET_SELF	설명은 아래의 flags와 동일하다.
TG_EVENT_GET_SELF	설명은 아래의 flags와 동일하다.
TG_EVENT_RECOVERED	GQ 서버에 장애가 발생하여 복구되었다. 해당 이벤트를 수신하는 경우 사용자는 watcher를 다시 등록해야 한다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	Key 이름이다. 127Byte를 넘을 수 없다.
keylen	Key의 길이이다.
callback	Key의 이벤트가 발생할 때 이를 처리하게 될 함수의 포인터이다. 사용자는 'int CALLBACK(char *key, int keylen, int event_type, void *args);' 타입으로 함수를 구현한다. 각 Key마다 별도의 Callback 함수를 지정할 수 있다.
args	해당 Key에 대한 이벤트가 발생해서 Callback 함수가 호출될 때 함께 전달될 사용자 정의 인수이다.

파라미터	설명
flags	동작 방식을 결정한다. • TG_WATCHER_PERSISTENT watcher 등록 후 이벤트가 발생할 때 계속해서 해당 클라이언트에게 통지한다. • TG_WATCHER_ONCE watcher 등록 후 한 번만 통지하고 그 이후에는 통지하지 않는다. 만약 더 관심이 있다면 Callback 함수 호출 후 watcher를 재등록해야 한다. • TG_EVENT_DELETE_SELF 자신의 Key가 삭제되었을 때 이벤트를 수신한다. Callback 함수에서 해당 이벤트를 수신하게 되면 등록된 watcher가 삭제되므로, watcher를 재등록 해야 한다. • TG_EVENT_CREATE_CHILD Child Key가 생성되었을 때 이벤트를 수신한다. • TG_EVENT_SET_SELF 자신의 Key에 tmax_grid_set API를 통해 데이터가 set되었을 때 이벤트를 수신한다. 또한 tmax_grid_enqueue API를 통해 데이터가 입력되었을 때에도 수신한다. • TG_EVENT_GET_SELF 자신의 Key의 데이터가 get되었을 때 이벤트를 수신한다. 만약 tmax_grid_get API를 호출할 때 flags에 TG_GET_PEEK을 설정하면 이벤트는 발생하지 않는다. **[주의]** TG_WATCHER_PERSISTENT와 TG_WATCHER_ONCE는 단독으로 사용 할 수 없다. 반드시 아래의 이벤트와 Bitwise OR 연산으로 조합하여 적용해야만 한다.

• 반환값

반환값	설명
>= 0	함수 호출에 성공한 경우이다.
< 0	함수 호출에 실패한 경우이다. tgerrno에 에러 코드가 설정된다.

• 오류

tmax_grid_set_watcher()가 정상적으로 수행되지 않을 경우 tgerrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TGEINVAL]	Key가 NULL이거나 key_len <=0일 경우 또는 Key의 길이가 128Byte가 넘어갈 경우 또는 flags에 적절하지 않은 값이 지정된 경우이다.
[TGEOS]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TGENOMEM]	메모리가 부족하여 할당에 실패한 경우 발생한다.
[TGENOKEY]	Key 이름으로 생성된 Key가 존재하지 않는 경우 또는 watcher 해제할 때 Key 이름으로 설정하지 않았을 경우이다.
[TGECONF]	TmaxGrid가 설정되지 않은 경우이다.
[TGEENABLE]	TmaxGrid가 동작하지 않는 경우이다.

3.1.26. tmax_grid_wait_watcher

이벤트가 발생할 때까지 timeout으로 지정한 시간 동안 대기한다. 하나의 이벤트를 수신하면 수신된 이벤트의 Node로 지정된 Callback 함수가 호출되고 리턴된다.

- 프로토타입

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

- 파라미터

파라미터	설명
timeout	대기하는 시간이다. (단위 : 초)
flags	사용하지 않는 파라미터이다.

- 반환값

반환값	설명
>= 0	함수 호출에 성공한 경우이다.
< 0	함수 호출에 실패한 경우이다. tgerrno에 에러 코드가 설정된다.

- 오류

  tmax_grid_wait_watcher()가 정상적으로 수행되지 않을 경우 tgerrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TGEOS]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TGENOMEM]	메모리가 부족하여 할당에 실패한 경우 발생한다.
[TGETIME]	이벤트가 발생하지 않았을 경우 발생한다.

3.1.27. tmax_keylist_count

keylist 핸들로부터 키 리스트의 개수를 반환한다.

- 프로토타입

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

- 파라미터

파라미터	설명
keylist	tmax_sq_getkeylist() 또는 tmax_gri_getkeylist()로부터 전달받은 핸들이다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tmax_keylist_count()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEBADDESC]	잘못된 keylist 핸들이다.

3.1.28. tmax_keylist_free

keylist 핸들의 메모리나 기타 자원들을 해제한다.

- 프로토타입

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

- 파라미터

파라미터	설명
keylist	tmax_sq_getkeylist() 또는 tmax_gq_getkeylist()로부터 전달받은 핸들이다.

- 반환값

반환값	설명
-1이 아닌 값	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tmax_keylist_free()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEBADDESC]	잘못된 keylist 핸들이다.

3.1.29. tmax_keylist_getakey

keylist 핸들로부터 n번째 키 정보를 가져온다.

- 프로토타입

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

- 파라미터

파라미터	설명
keylist	**tmax_sq_getkeylist()** 또는 **tmax_gq_getkeylist()**로부터 전달받은 핸들이다.
nth	keylist 핸들로부터 n번째 키이다. n은 0에서부터 tmax_keylist_count() -1사이의 값이어야 한다.
keyinfo	n번째 키의 정보가 저장될 구조체로 자세한 설명은 표 이후의 내용을 참고한다.

  다음은 keyinfo에 대한 설명이다.

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

멤버	설명
long keylen	키의 크기이다. (단위: Byte)
long datalen	데이터의 크기이다. (단위: Byte)
time_t starttime	데이터가 저장 또는 업데이트된 시간이다.
char *key	키 값을 담고 있는 버퍼이다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tmax_keylist_getakey()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEBADDESC]	잘못된 keylist 핸들이다.
[TPELIMIT]	nth의 범위가 벗어났다.

3.1.30. tmax_sq_count

현재 SQ에 저장된 데이터 개수를 반환하는 함수이다.

- 프로토타입

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

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tmax_sq_count()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPENOENT]	현 세션을 위한 SQ가 존재하지 않는다.

3.1.31. tmax_sq_get

데이터를 세션 큐에 저장하는 함수로 SQ에서 데이터를 가져오기 위해 키를 지정하면 해당 키의 데이터를 가져온다. 기본적으로 tmax_sq_get()을 수행하면 큐에서 데이터가 삭제된다. 데이터를 보존하려면 **TPSQ_PEEK** flags를 설정해야 한다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	SQ에서 가져올 데이터를 위한 키 값을 저장할 버퍼이다.
keylenl	키 버퍼의 크기를 지정한다. (단위: Byte)
data	SQ에서 가져올 데이터 버퍼에 대한 포인터이다. 반드시 tpalloc()으로 할당된 버퍼에 대한 포인터이어야 한다.
lenl	SQ에서 가져온 데이터 버퍼의 크기가 저장된다. (단위: Byte)
flagsl	TPSQ_PEEK을 지정할 수 있다. 이 flags를 지정할 경우 데이터를 큐에서 삭제하지 않는다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tmax_sq_get()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPENOENT]	현 세션을 위한 SQ가 존재하지 않는다.
[TPEMATCH]	지정한 키의 데이터가 존재하지 않는다.

3.1.32. tmax_sq_getkeylist

현 세션 SQ의 키 리스트를 가져오기 위한 함수이다. 키 리스트는 SQ_KEYLIST_T 타입의 핸들이 반환되며, 각 키에 대한 정보는 **tmax_keylist_count()**, **tmax_keylist_getakey()**, **tmax_keylist_free()** 함수를 통해서 확인할 수 있다.

키 리스트는 Byte 단위로 ASCII 코드값을 비교하여 낮은 값 우선으로 정렬된다. 키 값을 지정할 경우 해당 키 값보다 크거나 같은 키 리스트가 정렬되어 조회된다. 키 값을 NULL로 지정할 경우 전체 키 리스트가 조회된다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	키 값을 저장할 버퍼이다.
keylenl	키 버퍼의 크기를 지정한다. (단위: Byte)

- 반환값

반환값	설명
SQ_KEYLIST_T	함수 호출에 성공한 경우이다.
NULL	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tmax_sq_getkeylist()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPENOENT]	현 세션을 위한 SQ가 존재하지 않는다.
[TPEMATCH]	지정한 키보다 크거나 같은 키가 존재하지 않는다.

3.1.33. tmax_sq_keygen

시스템 키 생성 및 가져오기를 위한 함수로 시스템 키 사이즈는 SQ_SYSKEY_SIZE(16Byte)이다. 버퍼 크기는 시스템 키 사이즈보다 크거나 같게 할당해야 한다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	생성된 시스템 키 값이 저장될 버퍼이다. 버퍼의 크기는 SQ_SYSKEY_SIZE(16Byte)보다 크게 할당해야 한다.
keylenl	키 버퍼의 크기를 지정한다. 반드시 SQ_SYSKEY_SIZE(16Byte)보다 커야 한다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.

반환값	설명
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tmax_sq_keygen()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPENOENT]	현 세션을 위한 SQ가 존재하지 않는다.

3.1.34. tmax_sq_purge

SQ의 데이터를 삭제하기 위한 함수로 키를 지정할 경우 해당 키의 데이터를 삭제하고, 키를 지정하지 않으면 모든 데이터를 삭제한다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	SQ에서 삭제할 데이터를 위한 키 값을 저장할 버퍼이다. NULL일 경우 현 세션 SQ의 모든 데이터를 삭제한다.
keylenl	키 버퍼의 크기를 지정한다. (단위: Byte)

- 반환값

반환값	설명
삭제된 데이터의 개수	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tmax_sq_purge()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPENOENT]	현 세션을 위한 SQ가 존재하지 않는다.

3.1.35. tmax_sq_put

서버 데이터를 SQ에 저장하는 함수로서 키와 데이터 값을 전달해야 한다.

- 프로토타입

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

- 파라미터

파라미터	설명
key	SQ에 저장할 데이터를 위한 키 값을 저장할 버퍼이다.
keylenl	키 버퍼의 크기를 지정한다. (단위: Byte)
data	SQ에 저장할 데이터를 담고 있는 버퍼로 반드시 **tpalloc()**으로 할당된 버퍼에 대한 포인터이어야 한다.
lenl	SQ에 저장할 데이터 버퍼의 크기를 지정한다. (단위: Byte)
flagsl	다음은 flagsl에 설정되는 값에 대한 설명이다. • TPSQ_UPDATE 키 값이 같을 경우 업데이트한다. flags가 설정되어 있지 않으면 TPEMATCH 에러가 반환된다. • TPSQ_SYSKEY **tmax_sq_keygen()**를 통해 생성된 시스템 키를 사용할 경우 설정한다. • TPSQ_KEYGEN 시스템 키를 자동으로 생성하여 저장한다. 키(key)는 SQ_SYSKEY_SIZE 크기(keylenl = SQ_SYSKEY_SIZE)로 할당해야 한다. 성공적으로 수행했을 경우 키에 생성된 키 값이 저장된다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tmax_sq_put()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPENOENT]	현 세션을 위한 SQ가 존재하지 않는다.
[TPEMATCH]	사용된 키 값이 이미 존재한다.
[TPELIMIT]	SQ 최대 개수를 초과했거나 SQ의 최대 키 개수를 초과하였다.

3.1.36. tmaxlastsvc

서버와 클라이언트에서 가장 마지막에 수행된 조회하는 함수로 에러가 발생한 서비스명 또는 최후로 루틴을 수행한 서비스명을 반환한다.

tpforward()나 **tprely()** 등을 사용하면 사용자가 호출했던 서비스 외에 여러 개의 다른 서비스 루틴을 수행한다. 여러 개의 서비스 루틴을 수행하는 중에 에러가 발생한 경우 사용자는 어떤 서비스 루틴에서 에러가 발생했는지 알 수 없다.

- 프로토타입

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

- 파라미터

파라미터	설명
idata, odata	tpcall에서 사용했던 send/rcv 버퍼를 사용한다.
flags	현재 버전에서는 지원하지 않으나 TPNOFLAGS로 설정한다.

- 반환값

반환값	설명
내부 버퍼 포인터	함수 호출에 성공한 경우이다.
NULL	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tmaxlastsvc()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다.

- 예제

```
#include <usrinc/atmi.h>
main(int argc, char *argv[])
{
```

```
    ...
    if(tpcall("TOUPPER", sndbuf, 0, &rcvbuf, &rcvlen, TPNOFLAGS)==-1){
       memset(svc, NULL, 16);
       strcpy((char *)svc, tmaxlastsvc(sndbuf, rcvbuf, TPNOFLAGS));
       printf("servicename = %s\n", svc);
       error processing
    }
    ...
}
```

3.1.37. tmaxreadenv

서버와 클라이언트에서 환경변수를 파일에서 접속할 시스템의 정보를 읽어서 환경변수에 새로운 값을 설정하는 함수이다.

Tmax 시스템과 접속하기 위해서는 몇 가지의 환경변수를 시스템에 등록해야 한다. 등록된 환경변수를 참조하여 **tpstart()** 함수를 사용해서 Tmax 시스템과 연결한다. 환경변수는 UNIX인 경우 **csh**는 <.cshrc>, **ksh**는 <.profile>, **DOS**인 경우에는 <autoexec.bat> 파일에 정의한다.

접속할 시스템이 2개 이상일 경우 환경변수에 2개의 시스템에 대한 정보를 등록할 수 없으므로 클라이언트는 상황에 따라 접속할 시스템을 변경해서 사용한다. Tmax 시스템과 접속하기 위한 정보를 환경변수에 설정하기 때문에 Tmax 시스템에 접속하기 전에 수행되어야 한다.

환경변수를 파일에 등록하는 방법은 "Tmax Administration Guide"를 참고한다.

- 프로토타입

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

- 파라미터

파라미터	설명
file	접속할 시스템의 환경 정보가 저장된 파일의 이름이다. 파일은 텍스트 형태로 일정한 형식에 맞게 등록되어 있어야 한다.
label	파일 내에 등록된 환경 정보의 구분자이다. 2개 이상의 시스템 정보를 하나의 파일에 등록할 경우에 각각의 시스템을 구별할 수 있는 값이다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	파일이 존재하지 않거나 label이 없는 경우이다.

- 예제

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

void main(int argc, char* argv[])
{
    tmaxreadenv(“tmax.env”, “tmax”);  /*”tmax.env”의 “tmax”절에서 환경변수를 얻음*/
    tpstart((TPSTART_T *)NULL);
    data process…
    tpend();
}
```

- 관련 함수

  tpstart()

3.1.38. tp_sleep

서버와 클라이언트에서 데이터의 도착을 초 단위로 기다리는 함수이다. 최대 sec 시간 동안 sleep하다가 그 안에 데이터가 도착하면 즉시 반환한다.

- 프로토타입

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

- 파라미터

파라미터	설명
sec	기다리려는 시간을 양의 정수값으로 입력한다. (단위 : 초)

- 반환값

반환값	설명
0	sec 시간까지 데이터가 도착하지 않는 경우이다.
-1	에러가 발생한 경우이다. tperrno에 에러 코드가 설정된다.
1	clh로부터 이벤트가 발생했다. tpacall에 대한 응답이나 비요청 메시지 등이 들어온 경우이다.
2	tmm으로부터 이벤트가 발생했다. clh로부터 이벤트는 발생하지 않았다. tpschedule()을 호출해서 tmm 이벤트를 처리해야 한다.
3	clh와 tmm 모두 이벤트가 발생했다. tpschedule()을 호출해서 tmm 이벤트를 처리해야 한다.

- 오류

  tp_sleep()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템에 에러가 발생한 경우이다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생한 경우이다.

- 관련 함수

  tpacall(), tpbroadcast(), tpgetrply(), tp_usleep(), tpsleep()

3.1.39. tp_usleep

서버와 클라이언트에서 데이터의 도착을 백만분의 1초 단위로 기다리는 함수로 최대 usec 시간 동안 sleep하다가 그 안에 데이터가 도착하면 즉시 반환한다.

- 프로토타입

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

- 파라미터

파라미터	설명
usec	기다릴 시간을 양의 정수값으로 입력해야 한다. (단위 : 1000000(백만)분의 1초)

- 반환값

반환값	설명
0	usec 시간까지 데이터가 도착하지 않는 경우이다.
-1	에러가 발생한 경우이다. tperrno에 에러 코드가 설정된다.
1	clh로부터 이벤트가 발생했다. tpacall에 대한 응답이나 비요청 메시지 등이 들어온 경우이다.
2	tmm으로부터 이벤트가 발생했다. clh로부터 이벤트는 발생하지 않았다. tpschedule()을 호출해서 tmm 이벤트를 처리해야 한다.
3	clh와 tmm 모두 이벤트가 발생했다. tpschedule()을 호출해서 tmm 이벤트를 처리해야 한다.

- 오류

  tp_usleep()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생한 경우이다.

- 예제

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

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

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

    buf=tpalloc(“STRING”, NULL, 0);
    if (buf==NULL) {error processing}
    data process…

    cd=tpconnect(“SERVICE”, buf, 0, TPRECVONLY);
    if (cd<0) { error processing }
    while(1)
    {
        ret=tp_usleep(2000000);
        if (ret<0) {error processing }
        if (ret==0)
            printf("waited 2sec..\n");
        else {
            printf("data received!\n");
            break;
        }
    }
    ret=tprecv(cd, &buf, &len, TPNOFLAGS, &revent);
    if (ret<0) { error processing }
    data process....
    tpend();
}
```

- 관련 함수

  tmax_get_db_usrname(), tmax_get_db_tnsname()

3.1.40. tpabort

서버와 클라이언트에서 전역 트랜잭션을 rollback하는 함수로 **tx_rollback()** 함수와 동일한 기능을 수행한다. tpabort()는 Tuxedo에서 사용한 함수를 Tmax 시스템에 그대로 적용하는 경우 사용한다. Tuxedo로 개발된 프로그램을 변경 없이 Tmax로 변환할 수 있도록 지원한다.

- 프로토타입

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

- 파라미터

파라미터	설명
flags	현재 버전에서는 지원하지 않으나 TPNOFLAGS로 설정한다.

- 반환값

  tx_rollback을 참고한다.

- 오류

  tx_rollback을 참고한다.

- 예제

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

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

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

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

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

    ret=tpcall(“SERVICE",(char *)buf,strlen(buf),(char **)&buf,&len,TPNOFLAGS);
    if (ret == -1)
    {
        ret = tpabort(TPNOFLAGS);
        if (ret < 0){ error processing }
        tpfree(buf);
        tpend();
        exit(1);
    }

    ret = tpcommit(TPNOFLAGS);
    if (ret < 0){ error processing }
    tpfree(buf);
    tpend();
```

```
}
```

- 관련 함수

  tx_rollback()

3.1.41. tpacall

서버와 클라이언트에서 비동기 서비스 요청을 송신하는 함수로 **svc**로 명명된 서비스에게 서비스 요청 메시지를 송신한다. 비동기 통신으로 메시지를 송신한 후에 결과를 받을 때까지 기다리지 않고 바로 반환된다. 결과는 **tpgetrply()**를 이용하여 응답을 받을 수도 있고, 또는 **tpcancel()**를 이용하여 응답을 취소할 수 있다.

tx_begin 호출 이후 tx_time이 지난 이후에 tpacall은 TPNOTRAN | TPNOREPLY를 flag에 설정한 호출만 성공하며 나머지는 모두 TPETIME 에러를 발생시킨다.

tpacall이 성공한 경우 반환값 범위의 제한은 int 값 범위 중 양의 정수 부분인 1부터 2147438647이다.

- 프로토타입

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

- 파라미터

파라미터	설명
svc	호출되는 서비스명으로 Tmax 응용 서버 프로그램에서 제공되는 것이어야 한다.
data	NULL 값이 아닌 경우 반드시 tpalloc()으로 할당된 버퍼에 대한 포인터이어야 한다.
len	송신되는 데이터 길이이다. • data가 가리키는 버퍼가 특별한 길이 명시가 필요없는 버퍼 유형(STRING, STRUCT, X_COMMON, X_C_TYPE)이라면 len은 무시되고 보통 0이 사용된다. • data가 가리키는 버퍼가 길이 명시가 반드시 필요한 버퍼 유형(X_OCTET, CARRAY, MULTI STRUCTURE)이라면 len은 0이 될 수 없다. • data가 NULL이라면 len은 무시되고 데이터 없이 서비스 요청이 송신된다. data의 유형(type)과 하위 유형(subtype)은 svc가 지원하는 유형이어야 한다. 서비스 요청이 트랜잭션 모드에서 송신되었다면 해당 응답은 반드시 수신되어야 한다.

파라미터	설명
flags	호출할 때 사용되는 옵션으로 호출 모드를 결정한다. flags로 사용 가능한 값은 다음과 같다. • TPBLOCK flags 없이 tpacall()이 사용되었다면, svc에 호출된 서비스가 없거나 잘못된 결과가 반환되었어도 정상적인 결과가 반환된다. tpgetrply()를 호출할 때 에러가 반환된다. TPBLOCK을 이용해 tpacall()을 호출할 경우 서비스 상태의 정상 여부를 확인할 수 있다. • TPNOTRAN 트랜잭션 모드에서 svc가 트랜잭션을 지원하지 않는 서비스라면, tpacall()이 트랜잭션 모드에서 호출되는 경우 flags는 반드시 TPNOTRAN으로 설정해야 한다. tpacall() 호출자가 트랜잭션 모드 상태에서 TPNOTRAN을 설정하여 svc 서비스를 요청하였다면, svc 서비스는 트랜잭션 모드에서 제외되어 수행된다. 트랜잭션 모드 내에서의 tpacall()을 호출할 때 TPNOTRAN로 설정되었어도 여전히 트랜잭션 타임아웃(timeout)에 영향을 받는다. 즉, 트랜잭션 타임아웃이 지난 이후의 TPNOTRAN을 적용한 tpcall도 호출하지 않고 실패를 한다는 의미이다. 예외적으로 TPNOTRAN|TPNOREPLY를 적용한 tpacall은 호출을 허용한다. TPNOTRAN으로 호출된 서비스가 실패하였을 경우 호출자의 트랜잭션에는 영향을 미치지 않는다. • TPNOREPLY tpacall()로 송신한 서비스 요청은 응답을 기다리지 않고 즉시 반환한다. 결과는 나중에 tpacall()에서 반환한 구별자를 이용하여 tpgetrply()의 결과를 수신한다. flags를 TPNOREPLY로 설정하면 서비스에 대한 응답을 받지 않는다고 설정된다. TPNOREPLY로 설정하면 tpacall()은 서비스가 정상적으로 호출되면 0을 반환한다. 함수 호출자가 트랜잭션 모드에 있을 경우에는 반드시 TPNOTRAN과 함께 설정해야만 TPNOREPLY를 사용할 수 있다. TPNOREPLY인 경우 서비스 상태의 정상 여부를 체크하기 위해서는 TPBLOCK을 함께 설정해야 한다. TPBLOCK을 설정하지 않으면 서비스가 NRDY인 경우에도 에러를 반환하지 않는다. • TPNOBLOCK 내부 버퍼가 송신할 메시지들로 가득 찬 경우와 같은 블로킹(blocking) 상황을 만나면 서비스 요청은 실패하도록 설정하는 flags이다. TPNOBLOCK flags 설정 없이 tpacall()이 호출된 경우 블로킹 상황이 발생하면 함수 호출자는 블로킹 상황이 풀리거나 타임아웃(트랜잭션 타임아웃 또는 블록 타임아웃)이 발생할 때까지 대기한다. • TPNOTIME 함수 호출자가 블록 타임아웃을 무시하고 응답이 수신될 때까지 무한정 대기하도록 설정하는 flags이다. 트랜잭션 타임아웃 내에서 tpacall()을 수행한 경우에는 여전히 트랜잭션 타임아웃이 적용된다. • TPSIGRSTRT 시그널(signal) 인터럽트를 수용하는 경우 사용하는 flags로 시스템 함수 호출이 방해될 때 시스템 함수 호출이 재실행된다. TPSIGRSTRT flag와 TPBLOCK flag가 설정되어 있으면 BLOCKTIME 시간동안 계속 응답을 기다린다. TPSIGRSTRT가 설정되지 않은 상태에서 시그널 인터럽트가 발생했다면 함수는 실패 처리되고 tperrno에 TPGOTSIG가 설정된다. • TPNOCALLBACK UCS의 usrmain()에서 tpacall()을 호출할 때 tpregcb()를 이용하여 콜백함수를 등록할 수 있다. flags를 TPNOCALLBACK로 설정하면 콜백함수에서 응답 처리하지 않고 tpgetrply()에서 응답을 처리한다.

- 반환값

반환값	설명
구별자(descriptor)	함수 호출에 성공한 경우이다. 반환된 구별자는 송신된 서비스 요청에 대한 응답을 수신하는 데 사용된다.
-1	함수 호출이 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpacall()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않은 경우 발생한다. 예를 들어 svc가 NULL이거나 data가 tpalloc()으로 할당되지 않은 버퍼를 가리키거나 또는 flags가 유효하지 않은 경우에 발생한다.
[TPENOENT]	svc라는 서비스가 존재하지 않아서 서비스를 요청할 수 없다.
[TPEITYPE]	data의 유형 및 하위 유형이 svc가 지원하지 않는 유형이다. 구조체인 경우 사용된 구조체가 SDLFILE 파일에 선언되어 있지 않은 경우 발생한다.
[TPELIMIT]	처리되지 않은 비동기성 서비스 요청 수가 최대 한계에 도달했기 때문에 호출자의 서비스 요청이 송신되지 않은 경우 발생한다.
[TPETIME]	타임아웃이 발생한 경우로 함수 호출자가 트랜잭션 모드인 경우 트랜잭션 타임아웃이 발생한 것이며 트랜잭션은 rollback된다. 함수 호출자가 트랜잭션 모드가 아닌 경우 TPNOTIME이나 TPNOBLOCK이 모두 설정되지 않은 상황에서 블록 타임아웃이 발생한다. 트랜잭션 타임아웃이 발생하는 경우 트랜잭션이 rollback될 때까지 새로운 서비스 요청을 송신한다거나 아직 수신되지 않은 응답을 기다리는 일은 모두 [TPETIME] 에러로 실패 처리된다.
[TPEBLOCK]	TPNOBLOCK이 설정된 상태에서 블로킹 상황이 발생하였다.
[TPGOTSIG]	TPSIGRSTRT가 설정되지 않은 상태에서 시그널이 수신되었다.
[TPEPROTO]	트랜잭션 모드에서의 TPNOREPLY 서비스 호출할 때 TPNOTRAN flags를 설정하지 않는 경우 등 부적절한 상황에서 발생한다.
[TPESYSTEM]	Tmax 시스템에 에러가 발생하였다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

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

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

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

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

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

- 관련함수

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

3.1.42. tpacallsvg

COUSIN으로 묶인 멀티 서버 그룹 환경에서 특정 서버 그룹에 속하는 서비스를 지정하여 비동기형 통신으로 서비스를 요청하는 함수이다. 특정 서버 그룹을 지정하여 서비스를 호출하는 것 이외에는 tpacall()과 동일하게 동작한다.

- 프로토타입

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

- 파라미터

  svgno를 제외한 다른 파라미터는 tpacall을 참고한다.

파라미터	설명
svgno	svgno에는 호출하려는 서비스가 속한 서버 그룹의 번호를 지정한다. 서버 그룹의 번호는 **tpgetsvglist()**을 통해서 알아낼 수 있다. -1로 설정했을 경우에는 기존의 tpacall()과 동일하게 동작한다. 서비스 내에서 또 다른 서비스를 호출할 경우 같은 서버 그룹에 속하는 서비스를 호출하려는 할 경우에는 **tpgetmysvgno()**를 이용하여 현재 자신이 속한 서버 그룹의 번호를 알아낸 후 동일한 서버 그룹에 속한 서비스를 호출할 수도 있다.

- 반환값

반환값	설명
구별자	함수 호출에 성공한 경우이다(client descriptor).
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  에러에 관한 자세한 내용은 tpacall을 참고한다.

- 예제

  <클라이언트 프로그램>

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

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

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

    ret = fbput(sndbuf, INPUT, sndata, 0);
    strcpy(svc, “FDLTOUPPER”);
    svg_list = (struct svglist *)tpgetsvglist(svc, 0);

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

```
    tpend();
    return 0;
}
```

- 관련 함수

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

3.1.43. tpalivechk

클라이언트의 연결 상태를 체크하는 함수로 소켓의 상태를 점검하는 역할을 하는 함수이다.

tmax_chk_conn() 함수에서 파라미터를 0으로 설정했을 때와 동일한 역할을 수행한다.

- 프로토타입

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

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpalivechk()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEPROTO]	아직 tpstart()가 수행되지 않았다.
[TPESYSTEM]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

main(int argc, char *argv[])
{
```

```
    FBUF    *sndbuf, *rcvbuf;
    int     chkno;
    char    sndata[30], rcvdata[30];
    char    input[10], outdata[10];
    long    sndlen, rcvlen;

    if ( (ret = tmaxreadenv( “tmax.env”,”TMAX” )) == -1 ){
        printf( “tmax read env failed\n” );
        exit(1);
    }
    if (tpstart((TPSTART_T *)NULL) == -1)
        error processing…
    if ((sndbuf = (FBUF *)tpalloc(“FIELD”, NULL, 0)) == NULL)
        error processing…
    if ((rcvbuf = (FBUF *)tpalloc(“FIELD”, NULL, 0)) == NULL)
        error processing…
    /* socket의 상태 점검 */
    chkno = tpalivechk();
    printf(“chkno = %d\n”, chkno);
    if(chkno < 0)
    {
        printf(“The situation of socket is bad\n”);
        printf(“errno = %d\tstrerror=%s\n”, tperrno, tpstrerror(tperrno) );
    }
    else if(chkno == 0)
    {
        printf(“The situation of socket is good\n”);
    }
    tpcall…
    data process…
    tpend();
}
```

- 관련 함수

  tmax_chk_conn()

3.1.44. tpalloc

서버와 클라이언트에서 유형 버퍼(typed buffer)를 할당하는 함수로 C 라이브러리의 malloc()이나 realloc() 또는 free()와 함께 사용될 수 없다. 예를 들어 tpalloc()으로 할당된 버퍼를 free()로 제거할 수 없다. 이 경우에는 **tpfree()**를 사용한다.

tpalloc()은 type으로 지정된 유형의 버퍼를 할당하고 그에 대한 포인터를 반환한다. 유형에 따라 subtype과 size는 선택적으로 지정할 수 있다. 일부 버퍼 유형은 사용되기 전에 초기화가 필요하기 때문에 tpalloc()은 버퍼를 할당한 후 이를 초기화하여 반환한다. 그러므로 호출자에게 반환된 버퍼는 즉시 사용 가능하다. 초기화 작업에 실패하면, tpalloc()은 버퍼를 0값으로 초기화하지 못하고 할당된 버퍼는 제거(free)된다.

메모리 동적 할당시 1 Gbyte 제한이 있다.

- 프로토타입

```
# include <atmi.h>
```

```
char * tpalloc (char *type, char *subtype, long size)
```

- 파라미터

파라미터	설명
type	버퍼 유형으로 다음 중에 하나를 지정한다. • STRING : NULL로 끝나는 문자열의 데이터를 전송할 때 사용한다. • CARRAY, X_OCTET : 길이가 지정된 문자형의 데이터를 전송할 때 사용한다. • STRUCT, X_C_TYPE : C 언어 구조체 타입의 데이터를 전송할 때 사용한다. • X_COMMON : char, int, long만 가능한 C 구조체일 때 사용한다. • FDL(FIELD 버퍼) : 데이터를 식별자와 식별자에 해당하는 값 형태로 저장할 때 사용한다.
subtype	type이 STRUCT, X_C_TYPE, X_COMMON인 경우에 반드시 subtype을 지정해야 한다. type의 처음 8Byte와 subtype의 처음 16Byte만이 사용되고 지정된 길이를 초과해서 사용하면 초과한 길이는 무시된다. 지정된 버퍼 유형이 하위 유형을 사용하지 않는다면, subtype은 무시되고 보통 NULL이 사용된다. 할당된 버퍼의 크기는 기본 크기(1024Byte) 이상이다.
size	버퍼 크기로 CARRAY와 X_OCTET에서는 반드시 지정하여야 하며 이를 제외하고는 생략 가능하다. 0으로 지정하면 각 버퍼의 기본 크기가 사용된다. STRING, STRUCT, X_C_TYPE, X_COMMON의 기본 크기는 1024Byte이다. CARRAY의 기본 크기는 0이지만 버퍼를 할당할 때에는 반드시 0보다 커야 한다.

- 반환값

반환값	설명
버퍼 포인터	함수 호출에 성공한 경우이다. 적절한 유형 버퍼에 대한 포인터를 반환한다.
NULL	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpalloc()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 NULL 형식인 경우에 발생한다.
[TPENOENT]	알 수 없는 유형 또는 하위 유형이다. STRUCT 버퍼 유형인 경우 하위 유형이(구조체의 태그명) SDLFILE에 존재하지 않는 경우에 발생한다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	메모리를 할당 받지 못하는 운영 시스템에 에러가 발생하였다.

에러 코드	설명
[TPEOTYPE]	요청되는 서버의 자료형이 구조체 버퍼인데 해당 서버가 컴파일할 때 구조체 파일과 함께 컴파일이 되지 않은 경우에 발생한다.

- 예제

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

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

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

    ret=tpcall(“SERVICE”, (char *)sndbuf, 0, (char **)&rcvbuf, &len, TPNOFLAGS);
    if (ret<0) {error processing }
    data process....
    tpfree((char *)buf);
    tpend();
}
```

- 관련함수

  tpfree(), tprealloc(), tptypes()

3.1.45. tpbegin

tx_set_transaction_timeout()과 **tx_begin()**의 기능을 한 번에 수행할 수 있는 함수이다. Tuxedo에서 사용하는 것과 동일한 형식을 가지므로 Tuxedo용으로 개발된 애플리케이션을 변환하지 않고도 사용할 수 있다. 함수의 기능은 tx_begin()와 동일하다.

- 프로토타입

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

- 파라미터

파라미터	설명
timeout	**tx_set_transaction_timeout()** 함수에 설정하는 값과 동일한 의미를 갖는다. 트랜잭션 타임아웃 시간을 입력한다. (단위 : 초)

파라미터	설명
flags	현재 버전에서는 지원하지 않으나 TPNOFLAGS로 설정한다.

- 반환값

  tx_set_transaction_timeout과 tx_begin을 참고한다.

- 예제

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

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }
    buf=tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }

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

    ret = tpcall("SERVICE", (char *)buf, strlen(buf),
                (char **)&buf, &len, TPNOFLAGS);
    if (ret == -1)
    {
        ret = tpabort(TPNOFLAGS);
        if (ret < 0){ error processing }
        tpfree(buf);
        tpend();
        exit(1);
    }
    ret = tpcommit(TPNOFLAGS);
    if (ret < 0){ error processing }
    tpfree(buf);
    tpend();
}
```

- 관련 함수

  tx_set_transaction_timeout(), tx_begin()

3.1.46. tpbroadcast

Tmax 시스템에 등록된 클라이언트들에게 요청하지 않은 메시지를 송신하는 함수이다. 메시지가 송신 가능한 클라이언트들은 **tpstart()**로 이미 Tmax 시스템에 연결되어 있어야 하며, 이때 클라이언트의 이름과 flags가 알맞게 정의되어야 한다. 비요청 메시지를 받기 위해서는 비요청 메시지를 받겠다는 정보를 주어야 한다. tpstart()를

사용하는 경우 TPSTART_T 구조체의 flags 값을 TPUNSOL_POLL이나 TPUNSOL_HND로 설정해야만 비요청 메시지를 받을 수 있다.

- 프로토타입

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

- 파라미터

파라미터	설명
nodename, usrname, cltname	대상 클라이언트를 선택하는 데 사용되는 논리적인 이름으로 63자 이내여야 한다. 이름 지정에는 물음표(?)나 애스터리스크(*) 등의 와일드 카드(wildcards) 문자들이 사용될 수 있다. NULL 값이 사용될 수 있는데, 이는 모든 클라이언트에 대응되는 와일드 카드로 동작한다. 길이가 0인 string 인수는 길이가 0인 클라이언트 이름과만 대응된다. cltname으로 사용되는 이름은 클라이언트가 tpstart()를 이용하여 처음 Tmax 시스템에 연결할 때 등록하는 클라이언트 이름이다.
data	반드시 tpalloc()에 의해 이전에 할당된 버퍼를 사용해야 한다.
len	송신할 데이터 길이이다. • data가 가리키는 버퍼가 특별한 길이 명시가 필요 없는 STRING, STRUCT, X_COMMON, X_C_TYPE 의 버퍼 유형이라면 len은 무시되고 0이 사용된다. • data가 NULL인 경우도 len은 무시된다.
flags	설정 가능한 값은 표 이후에 설명한다. flags로 사용 가능한 값에 대한 설명이다. • TPNOBLOCK 내부 버퍼가 송신 메시지들로 가득 찬 경우와 같은 블로킹(blocking) 상황을 만나면 요청이 송신되지 않는다. • TPNOTIME 함수 호출자가 블록 타임아웃을 무시하고 응답이 수신될 때까지 무한정 대기한다. 트랜잭션 타임아웃 내에서 **tpbroadcast()**를 한 경우에는 여전히 트랜잭션 타임아웃이 적용된다. • TPSIGRSTRT 시그널(signal) 인터럽트를 수용하는 경우 사용한다. 내부에서 시그널 인터럽트가 발생하여 시스템 함수 호출이 방해될 때 TPSIGRSTRT flag가 설정되어 있으면 BLOCKTIME 시간동안 계속 응답을 기다린다. TPSIGRSTRT flags가 설정되지 않은 경우 시그널 인터럽트가 발생했다면, 함수는 실패하고 tperrno에 TPGOTSIG가 설정된다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpbroadcast()가 정상적으로 수행되지 않은 경우 어떤 메시지도 클라이언트들에게 송신되지 않으며 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 식별자 길이가 너무 길거나 flags가 유효하지 않은 경우에 발생한다. nodename를 잘못 사용한 경우 tpbroadcast()가 실패하고 TPEINVAL을 반환하게 된다. 그러나 usrname이나 cltname이 잘못된 경우 아무에게도 메시지가 전달되지 않고 단순히 성공인 것으로 수행된다.
[TPETIME]	TPNOBLOCK이나 TPNOTIME이 설정되지 않은 상태에서 블록 타임아웃이 발생하였다.
[TPEBLOCK]	TPNOBLOCK이 설정된 상태에서 블로킹(blocking) 상황이 발생하였다.
[TPGOTSIG]	TPSIGRSTRT가 설정되지 않은 상태에서 시그널이 수신되었다.
[TPEPROTO]	tpbroadcast()가 부적절한 상황에서 호출되었다.
[TPESYSTEM]	Tmax 시스템에 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

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

    buf=tpalloc(“STRING”, NULL, 0);
    if (buf==NULL) { error processing }
    data process…

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

    tpfree(buf);
```

```
    tpend();
}
```

- 관련함수

  tpalloc(), tpend(), tpstart()

3.1.47. tpcall

서버와 클라이언트의 동기형 서비스 요청 송수신 함수로 동기형 통신으로 **svc**로 명명된 서비스에게 서비스 요청을 송신하고 이에 대한 응답을 수신한다. **tpacall()** 호출 후 연속적으로 **tpgetrply()**를 호출하는 것과 동일하게 처리된다.

tx_begin 호출 이후 tx_time이 지난 이후에 tpacall은 TPNOTRAN | TPNOREPLY를 flag에 설정한 호출만 성공하며 나머지는 모두 TPETIME 에러를 발생시킨다.

- 프로토타입

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

- 파라미터

파라미터	설명
svc	호출되는 서비스명으로 Tmax 응용 서버 프로그램에서 제공되고 있는 것이어야 한다.
idata	서비스 요청의 데이터에 대한 포인터이다. 반드시 이전에 tpalloc()에 의해 할당된 버퍼이어야 한다. idata의 유형(type)과 하위 유형(subtype)은 svc가 지원하는 유형이어야 한다.
ilen	송신할 데이터의 길이이다. • idata가 가리키는 버퍼가 특별한 길이 명시가 필요없는 버퍼 유형(STRING, STRUCT, X_COMMON, X_C_TYPE)인 경우 ilen은 무시되고 기본으로 0을 사용한다. • idata가 가리키는 버퍼가 길이 명시가 반드시 필요한 버퍼 유형(X_OCTET, CARRAY, MULTI STRUCTURE)인 경우 ilen은 0이 될 수 없다. • idata가 NULL인 경우 ilen은 무시된다.

파라미터	설명
*odata	*odata는 수신될 응답 버퍼에 대한 포인터로 버퍼는 *olen으로 반환된 길이 만큼의 응답이 수신된다. *odata는 반드시 이전에 tpalloc()에 의해 할당된 버퍼이어야 한다. 동일한 버퍼가 송신과 수신 역할을 모두 한다면, *odata는 idata의 주소로 설정되어야 한다. 응답 버퍼의 크기 변경 여부를 결정하기 위해서 tpcall()의 완료 전에 *odata로 할당된 응답 버퍼의 크기와 수신된 *olen을 비교한다. 수신된 *olen이 더 크다면 할당된 응답 버퍼의 크기가 증가되며, 그렇지 않으면 크기는 변경되지 않는다. idata와 *odata로 동일한 버퍼가 사용되어 tpcall()이 호출된 경우 *odata가 변경되었다면 idata가 가리키는 주소는 더 이상 유효하지 않다. *odata는 수신 데이터가 커서 변경될 수도 있고, 이 외에 다른 이유에 의해서도 변경될 수 있다. *olen이 0으로 반환되었다면, 어떤 데이터도 수신되지 않고 *odata와 *odata가 가리키는 버퍼 모두 아무런 변화가 없다. *odata나 olen이 NULL이 되는 것은 에러이다.
*olen	*odata에 반환될 응답에 대한 길이이다.

파라미터	설명
flags	호출할 때 사용되는 옵션으로 어떤 방식으로 통신할 것인지를 지정한다. flags로 사용 가능한 값은 다음과 같다. • TPNOTRAN tpcall() 호출자가 트랜잭션 모드 상태에서 이 flags를 설정하여 svc 서비스를 요청하였다면, svc 서비스는 트랜잭션 모드에서 제외되어 수행된다. 트랜잭션 모드에서 svc가 트랜잭션을 지원하지 않는 서비스라면, tpcall()이 트랜잭션 모드에서 호출되는 경우 flags는 반드시 TPNOTRAN으로 설정해야 한다. 트랜잭션 모드 내에서 tpcall()를 호출할 때 TPNOTRAN으로 설정되었어도 여전히 트랜잭션 타임아웃(timeout)에 영향을 받는다. 즉, 트랜잭션 타임아웃이 지난 이후의 TPNOTRAN을 적용한 tpcall도 호출하지 않고 실패를 한다는 의미이다. TPNOTRAN으로 호출된 서비스가 실패하였을 경우 호출자의 트랜잭션에는 영향을 미치지 않는다. • TPNOCHANGE TPNOBLOCK flags를 설정한 상태에서 내부 버퍼가 송신할 메시지들로 가득 찬 경우와 같은 블로킹(blocking) 상황을 만나면 서비스 요청은 실패한다. TPNOCHANGE은 **tpcall()**의 **Tx** 부분에만 적용된다. TPNOBLOCK flags 설정 없이 tpcall()을 호출할 때 블로킹 상황이 발생하면 함수 호출자는 블로킹 상황이 풀리거나 타임아웃(트랜잭션 타임아웃 또는 블록 타임아웃)이 발생할 때까지 대기한다. • TPNOTIME 함수 호출자가 블록 타임아웃을 무시하고 응답이 수신될 때까지 무한정 기다리겠다는 것을 의미한다. 트랜잭션 타임아웃 내에서 tpcall()을 한 경우에는 여전히 트랜잭션 타임아웃이 적용된다. • TPSIGRSTRT 시그널(signal) 인터럽트를 수용하는 경우 사용한다. 내부에서 시그널 인터럽트가 발생하여 시스템 함수 호출이 방해될 때 TPSIGRSTRT flag가 설정되어 있으면 BLOCKTIME 시간동안 계속 응답을 기다린다. TPSIGRSTRT flags가 설정되지 않은 경우 시그널 인터럽트가 발생했다면, 함수는 실패하고 tperrno에 TPGOTSIG가 설정된다.

• 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

• 오류

tpcall()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않거나 flags가 유효하지 않다. 예를 들어 svc가 NULL이거나 data가 tpalloc()으로 할당되지 않은 버퍼를 가리킨다.
[TPENOENT]	svc라는 서비스가 존재하지 않아서 서비스를 요청할 수 없다.
[TPEITYPE]	data의 유형 및 하위 유형이 svc가 지원하지 않는 유형이다.
[TPEOTYPE]	수신된 응답 버퍼의 유형이나 하위 유형이 호출자가 알지 못하는 유형이다. flags가 TPNOCHANGE로 설정되었는데 *odata가 가리키는 버퍼의 유형 및 하위 유형이 수신된 응답 버퍼의 유형 및 하위 유형과 맞지 않은 경우 *odata의 내용과 *olen은 모두 변경되지 않는다. 호출자가 트랜잭션 모드에서 서비스를 요청하였다면, 그 트랜잭션은 응답이 무시되었기 때문에 rollback된다.
[TPETRAN]	트랜잭션 서비스를 호출할 때 데이터 베이스에 문제가 발생하여 xa_start가 실패하였다.
[TPETIME]	타임아웃이 발생한 경우로 함수 호출자가 트랜잭션 모드에 있다면 트랜잭션 타임아웃이 발생하였고 그 트랜잭션은 rollback된다. 트랜잭션 모드가 아니고 TPNOTIME과 TPNOBLOCK 어느 것도 지정되지 않았다면, 블록 타임아웃이 발생한다. 이 두 경우에 *odata의 내용과 *olen은 변경되지 않는다. 트랜잭션 타임아웃이 발생하였다면 새로운 서비스 요청을 송신한다거나 응답을 대기하는 일은 트랜잭션이 rollback될 때까지 [TPETIME] 에러로 실패하게 된다.
[TPESVCFAIL]	서비스 요청에 대한 응답을 송신하는 서비스 루틴이 애플리케이션에서 에러가 발생하여 TPFAIL로 tpreturn()을 호출하였다. 서비스 응답이 수신되었다면 그 내용들은 *odata가 가리키는 버퍼를 통하여 사용될 수 있다. 트랜잭션 타임아웃이 발생해서 트랜잭션이 rollback되기 전에 다른 통신들이 시도될 수 있다. 그러한 통신들은 정상적으로 처리될 수도 있고, 또는 실패할 수도 있다. 통신이 정상적으로 수행되기 위해서는 TPNOTRAN이 설정되어야 한다. 호출자의 트랜잭션 모드에서 수행된 작업들은 트랜잭션을 완료할 때에 모두 rollback된다.

에러 코드	설명
[TPESVCERR]	서비스 루틴 수행 중이나 tpreturn()(예를 들어 잘못된 파라미터가 전달된 경우) 수행 중에 에러가 발생하였다. 에러가 발생하면 어떠한 응답 데이터도 반환되지 않고 *odata의 내용 또는 *olen 모두 변경되지 않는다. tpalloc로 생성되지 않은 버퍼를 사용하였거나 할당된 버퍼의 Tmax 헤더가 잘못된 포인터(memcpy 등)의 영향을 받았거나, tpacall이나 tpconnect의 cd로 반환하였을 경우 Recv 모드에서 서비스가 유효하지 않은 대화형 내용일 경우에 발생한다. 단, tpreturn을 시도하면 클라이언트는 TPESVCERR를 받는다. 클라이언트가 강제로 대화를 해제하여 서비스 프로그램이 TPEV_DISCOMN을 받는 것과 같은 TPEV_DISCOMN 이벤트가 발생하였을 경우 클라이언트는 서비스의 tpreturn에 TPESVCERR tperrno를 전송받는다. 함수 호출자가 트랜잭션 모드에 있을 경우 트랜잭션 타임아웃이 발생하기 전까지는 트랜잭션이 rollback되기 전에 다른 통신들이 시도될 수 있다. 그러한 통신들은 정상적으로 처리될 수도 있고, 또는 실패할 수도 있다. 이들이 제대로 수행되기 위해서는 TPNOTRAN이 설정되어야 한다. 호출자의 트랜잭션 모드에서 수행된 작업들은 트랜잭션이 완료되면 모두 rollback된다. Tmax 환경 파일에 서비스별로 SVCTIMEOUT을 설정할 수 있는데 서비스의 수행시간이 이 시간을 초과하게 되면 서비스는 수행을 멈추고 TPESVCERR를 반환한다. SVCTIMEOUT이 발생하면 tpsvctimeout()을 불러주는데 이 함수 내에서 버퍼 해제, 로깅 작업 등 업무별로 적당한 작업을 할 수 있다.
[TPEBLOCK]	TPNOBLOCK이 설정된 상태에서 블로킹 상황이 발생하였다.
[TPGOTSIG]	TPSIGRSTRT가 설정되지 않은 상태에서 시그널이 수신되었다.
[TPEPROTO]	tpcall()이 부적절한 상황에서 호출되었다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

void main(int argc, char *argv[])
{
    int ret;
    char *sndbuf, *rcvbuf;
    long sndlen,  rcvlen;

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

    sndbuf = (char *)tpalloc(“CARRAY”, NULL, 20);
    if (sndbuf==NULL) {error processing };

    rcvbuf = (char *)tpalloc(“CARRAY”, NULL, 20);
    if (rcvbuf==NULL) {error processing };
```

```
    data process....

    sndbuf=strlen(sndbuf);
    ret=tpcall("SERVICE", sndbuf, sndlen, &rcvbuf, &rcvlen, TPNOCHANGE);
    if (ret==-1) { error processing }

    data process....

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

- 관련 함수

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

3.1.48. tpcallsvg

특정 서버 그룹에 속하는 서비스를 서버와 클라이언트에서 호출하는 함수로 COUSIN으로 묶인 멀티 서버 그룹 환경에서 특정 서버 그룹에 속하는 서비스를 지정하여 동기형 통신으로 서비스 요청을 송신하고 이에 대한 응답을 수신할 수 있다. 특정 서버 그룹을 지정하여 서비스를 호출하는 것 이외에는 tpcall()과 동일하게 동작한다.

서비스 내에서 또 다른 서비스를 호출할 경우 같은 서버 그룹에 속하는 서비스를 호출하려는 할 경우에는 **tpgetmysvgno()**를 이용하여 현재 자신이 속한 서버 그룹의 번호를 알아낸 후 동일한 서버 그룹에 속한 서비스를 호출할 수도 있다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
int tpcallsvg (int svgno, char *svc, char *idata, long ilenl, char **odata,
               long *olen, long flagsl)
```

- 파라미터

  svgno 이외의 파라미터는 tpcall을 참고한다.

파라미터	설명
svgno	호출하려는 서비스가 속한 서버 그룹의 번호를 지정한다. 서버 그룹의 번호는 tpgetsvglist()로 서버 그룹의 일련 번호들을 알아낼 수 있다. -1로 설정했을 경우에는 tpcall()과 동일하게 동작한다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.

반환값	설명
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  에러에 관한 자세한 내용은 tpcall을 참고한다.

- 예제

  <클라이언트 프로그램>

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

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

    ret = tmaxreadenv( "tmax.env","TMAX" );
    ret = tpstart((TPSTART_T *)NULL);
    sndbuf = (FBUF *)tpalloc("FIELD", NULL, 0)};
    rcvbuf = (FBUF *)tpalloc("FIELD", NULL, 0));
    strcpy(sndata, argv[1]);
    ret = fbput(sndbuf, INPUT, sndata, 0);
    strcpy(svc, "FDLTOUPPER");
    svg_list = (struct svglist *)tpgetsvglist(svc, 0);

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

- 관련 함수

  tpcall(), tpacallsvg(), tpgetsvglist(), tpgetmysvgno()

3.1.49. tpcancel

서버와 클라이언트의 응답 취소 함수로 **tpacall()**이 반환한 호출 구별자인 cd를 취소한다.

- 프로토타입

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

- 파라미터

파라미터	설명
cd	tpacall()이 반환한 호출 구별자로 취소 대상을 설정한다. 전역 트랜잭션(global transaction)과 관련된 서비스는 취소할 수 없다. 서비스 응답이 성공적으로 취소되면, cd는 무효화되고 cd를 통해 받은 응답들도 모두 무시된다.

- 반환값

반환값	설명
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpcancel()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEBADDESC]	cd가 유효하지 않은 구별자이다.
[TPETRAN]	cd가 호출자의 전역 트랜잭션과 관련되어 있다. cd는 여전히 유효하고, 호출자의 현재 트랜잭션은 영향을 받지 않는다.
[TPEPROTO]	tpcancel()이 부적절한 상황에서 호출되었다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

```
#include <stdio.h>
#include <usrinc/atmi.h>
void main(int argc, char *argv[])
{
    char *test[2];
```

```
    int ret, i, cd[2];
    long len;

    if (argc != 4) { error processing }
    ret=tpstart((TPSTART_T *)NULL;
    if (ret==-1) { error processing }

    for (i=0; i<3; i++)
    {
        test[i] = tpalloc("STRING",NULL,0);
        if (test[I])==NULL} { error processing }
        strcpy(test[i],argv[i+1]);
        cd[i]=tpacall("SERVICE", test[i], 0, TPNOTIME);
    )

    ret=tpcancel(cd[1]);         /* 2번째 응답을 취소한다. */
    if (ret==-1) { error processing }
    for (i=0; i<3; i++)
    {
        ret=tpgetrply(&cd[i], (char **)&test[i], &len, TPNOTIME)
        if (ret==-1) printf("Can't rcv data from service of %d\n",cd[i]);
        else prtinf("%dth rcv data : %s\n", I+1, test[I]);
        tpfree(test[I]);
    }
    tpend();
}
```

- 관련함수

  tpacall()

3.1.50. tpcommit

서버와 클라이언트에서 전역 트랜잭션을 commit하는 함수로 **tx_commit()**과 동일한 기능을 수행한다.

tpcommit()은 Tuxedo에서 사용한 함수를 Tmax 시스템에 그대로 적용하기 위해서 사용하는 함수로 Tuxedo로 개발된 프로그램을 변경없이 Tmax로 변환할 수 있도록 지원한다.

- 프로토타입

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

- 파라미터

파라미터	설명
flags	의미가 없는 파라미터로 TPNOFLAGS로 설정한다.

- 반환값

tx_commit을 참고한다.

- 예제

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

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

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

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

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

    ret=tpcall("SERVICE",(char *)buf,strlen(buf),(char **)&buf,&len,TPNOFLAGS);
    if (ret == -1)
    {
        ret = tpabort(TPNOFLAGS);
        if (ret < 0){ error processing }
        tpfree(buf);
        tpend();
        exit(1);
    }
    ret = tpcommit(TPNOFLAGS);
    if (ret < 0){ error processing }

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

- 관련 함수

  tx_commit()

3.1.51. tpconnect

서버와 클라이언트에서 프로그램이 대화형 서비스 svc와 통신을 연결하는 함수이다. 통신은 동시에 수신 또는 송신만 가능한 반 이중(half-duplex) 형태이다. 연결을 설정하는 과정에서 함수 호출자는 서비스 루틴에게 데이터를 전달할 수 있다. 대화형 서비스는 TPSVCINFO 구조체를 통하여 data와 len을 수신하기 때문에 tpconnect()로 전달된 데이터들을 수신하기 위하여 **tprecv()**를 호출할 필요가 없다.

- 프로토타입

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

- 파라미터

파라미터	설명
svc	대화형 서비스의 서비스 이름을 지정한다.
data	호출자가 데이터를 전달하려는 경우 반드시 tpalloc()에 의해 이전에 할당된 버퍼이어야 한다. data의 유형(type)과 하위 유형(subtype)은 svc가 지원하는 유형이어야 한다.
len	전달할 데이터 길이를 지정한다. • data가 가리키는 버퍼가 특별한 길이 명시가 필요없는 버퍼 유형(STRING, STRUCT, X_COMMON, X_C_TYPE)이라면, len은 무시되고 보통 0이 사용된다. • data가 가리키는 버퍼가 길이 명시가 반드시 필요한 버퍼 유형(X_OCTET, CARRAY, MULTI STRUCTURE)이라면, len은 0이 될 수 없다. • data가 NULL이라면, len은 무시되고, 이 경우 아무런 데이터도 대화형 서비스에게 넘겨지지 않는다.

파라미터	설명
flags	flags로 사용 가능한 값은 다음과 같다. • TPNOTRAN tpconnect() 호출자가 트랜잭션 모드 상태에서 이 flags를 설정하여 svc 서비스를 요청하였다면, svc 서비스는 트랜잭션 모드에서 제외되어 수행된다. 트랜잭션 모드에서 svc가 트랜잭션을 지원하지 않는 서비스라면, tpconnect()가 트랜잭션 모드에서 호출되는 경우 flags는 반드시 TPNOTRAN으로 설정해야 한다. 트랜잭션 모드 내에서의 tpconnect()를 호출할 때, TPNOTRAN으로 설정되었어도 여전히 트랜잭션 타임아웃(timeout)에 영향을 받는다. TPNOTRAN으로 호출된 서비스가 실패하였을 경우 호출자의 트랜잭션에는 영향을 미치지 않는다. • TPSENDONLY 연결이 완료된 후 함수 호출자는 처음 데이터를 송신만 할 수 있고 요청된 서비스만 할 수 있도록 설정하는 flags이다. 호출자가 처음 통신 제어권을 가진다. TPSENDONLY나 TPRECVONLY 중 하나는 반드시 지정되어야 한다. • TPRECVONLY 연결이 완료된 후 함수 호출자는 데이터를 수신할 수만 있고 요청된 서비스가 처음 데이터를 송신을 시작하도록 하는 flags이다. 즉, 요청된 서비스가 처음 통신 제어권을 가진다. TPSENDONLY나 TPRECVONLY 중 하나는 반드시 지정되어야 한다. • TPNOTIME 함수 호출자가 블록 타임아웃을 무시하고 응답이 수신될 때까지 무한정 대기하겠다는 것을 의미한다. 트랜잭션 타임아웃 내에서 tpconnect()를 수행한 경우에는 여전히 트랜잭션 타임아웃이 적용된다. • TPSIGRSTRT 시그널(signal) 인터럽트를 수용하는 경우 사용한다. 내부에서 시그널 인터럽트가 발생하여 시스템 함수 호출이 방해될 때 TPSIGRSTRT flag가 설정되어 있으면 BLOCKTIME 시간동안 계속 응답을 기다린다. TPSIGRSTRT flags가 설정되지 않은 경우 시그널 인터럽트가 발생했다면, 함수는 실패하고 tperrno에 TPGOTSIG가 설정된다.

• 반환값

반환값	설명
구별자(descriptor)	함수 호출에 성공한 경우이다. 차후 연결을 위하여 참조될 구별자(descriptor)를 반환한다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

• 오류

tpconnect()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다. 특별한 언급이 없다면 함수 호출을 실패할 경우 호출자의 트랜잭션에 영향을 미치지 않는다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 svc와 data가 NULL인 경우 지시하는 버퍼가 tpalloc()으로 할당되지 않았거나, flags가 TPSENDONLY 또는 TPRECVONLY로 지정되지 않은 경우 또는 flags가 유효하지 않은 경우에 발생한다.
[TPENOENT]	svc라는 서비스가 존재하지 않아서 서비스를 요청할 수 없다.
[TPEITYPE]	data의 유형 및 하위 유형이 svc가 지원할 수 있는 유형 및 하위 유형이 아니다.
[TPELIMIT]	연결 개수가 최대 한계에 도달했기 때문에 서비스를 요청할 수 없다.
[TPETRAN]	트랜잭션 서비스를 호출할 때 데이터베이스에 문제가 발생하여 xa_start가 실패하였다.
[TPETIME]	타임아웃이 발생한 경우로 함수 호출자가 트랜잭션 모드에 있다면, 트랜잭션 타임아웃이 발생하고 그 트랜잭션은 rollback된다. 트랜잭션 모드가 아니고 TPNOTIME과 TPNOBLOCK 어느 것도 지정되지 않았다면, 블록 타임아웃이 발생하였다. 트랜잭션 타임아웃이 발생하였다면 새로운 서비스 요청은 트랜잭션이 rollback될 때까지 [TPETIME] 에러로 실패한다.
[TPEBLOCK]	TPNOBLOCK이 설정된 상태에서 블로킹 상황이 발생하였다.
[TPGOTSIG]	TPSIGRSTRT가 설정되지 않은 상태에서 시그널이 수신되었다.
[TPEPROTO]	tpconnect()가 부적절한 상황에서 호출되었다.
[TPESYSTEM]	Tmax 시스템에 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

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

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

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

    data process ....

    cd = tpconnect("SERVICE",sndbuf,0,TPRECVONLY);
    if (cd==-1) { error processing }
    data process....
```

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

- 관련함수

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

3.1.52. tpdeq

서버와 클라이언트에서 RQ로부터 데이터를 로드하는 함수로 **tpenq()**를 이용하여 서비스를 요청한 결과를 받거나 서비스명을 NULL로 해서 저장한 데이터를 읽는다. tpenq()를 호출할 때 flags에 TPNOREPLY를 설정하면 서비스는 결과를 받을 수 없다. 그러므로 트랜잭션 모드에서 tpdeq()를 수행 도중 에러가 발생해도 트랜잭션에는 영향을 미치지 않는다.

- 프로토타입

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

- 파라미터

파라미터	설명
qname	데이터를 저장할 RQ의 이름으로 config 파일에 등록된 이름이어야 한다.
svc	tpenq()를 호출하는 경우 svc에 전달한 이름이어야 한다. 서비스명으로 tpenq()를 호출한 경우 서비스가 자동 요청되고, 결과가 RQ에 저장된다. 서비스 결과를 받기 위해서는 tpdeq()도 동일한 서비스명을 입력해야 한다. 서비스명을 NULL로 tpenq()를 호출한 경우 tpdeq()도 동일한 서비스명을 입력한다. 서비스명이 NULL인 경우 서비스명에 상관없이 큐에 쌓여있는 모든 데이터를 하나씩 로드할 수 있다. tpenq()의 경우 에러나 시스템 장애로 인해 Fail 큐에 저장된 데이터를 tpdeq()하기 위해서는 svc에 _rq_sub_queue_name[TMAX_FAIL_QUEUE]를 주고 deq해야 한다.
*data	tpalloc()에 의해 할당된 버퍼에 대한 포인터이다. 함수가 성공적으로 반환되면 *data는 수신된 데이터가 저장된다.

파라미터	설명
len	tpdeq()가 성공적으로 수신한 데이터의 길이이다. tpdeq()는 필요하다면 응답 내용이 지정된 버퍼에 수신될 수 있도록 버퍼 크기를 증가시킨다. len은 *data의 데이터의 길이로 *data는 수신 데이터가 커서 변경될 수도 있고, 이 외에 다른 이유에 의해서도 변경될 수 있다. len이 호출 전 버퍼의 총 크기보다 크다면, len이 그 버퍼의 새로운 크기가 된다. len이 0으로 반환되었다면, 어떤 데이터도 수신되지 않고 *data와 len이 지시하는 버퍼 모두 아무런 변화가 없다. *data나 len이 NULL이 되는 것은 에러이다.
flags	flags에 사용 가능한 값은 다음과 같다. • TPRQS RQ에서 데이터를 가져올 때 사용된다. reply 큐로부터 서비스의 결과를 가져오기 위해서 설정된다. • TPFUNC 서비스별 RQ 데이터를 관리할 때 사용한다. flags가 설정되지 않았다면 처음 tpenq()를 통해 저장된 데이터가 제거된다. 데이터가 저장되기 전에 제거해야 할 경우에는 tpenq() 호출할 때 TPFUNC를 같이 설정한다. • TPBLOCK tpdeq() 호출할 때 블록 타임아웃 시간 동안 메시지가 올 때까지 기다린다. • TPNOTIME TPBLOCK과 함께 사용되면 블록 타임아웃 시간에 관계없이 응답이 올 때까지 기다리게 된다. • 0(zero) 데이터를 자신이 접속한 클라이언트의 버퍼에서 가져오려고 할 때 사용한다.

- 반환값

반환값	설명
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpdeq()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 데이터가 tpalloc()으로 할당되지 않은 버퍼를 가리키거나 또는 flags가 유효하지 않은 경우에 발생한다.

에러 코드	설명
[TPGOTSIG]	TPSIGRSTRT가 설정되지 않은 상태에서 시그널이 수신되었다.
[TPEMATCH]	서비스명이 잘못되었거나 제거할 데이터가 없는 경우 해당 조건을 만족하는 제거할 데이터를 찾지 못했다.
[TPENOENT]	존재하지 않는 qname이 사용되었다.
[TPEPROTO]	tpdeq()이 부적절한 상황에서 호출되었다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

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

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

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

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

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

- 관련함수

  tpenq(), tpqstat()

3.1.53. tpdeq_ctl

서버와 클라이언트에서 트랜잭션을 지원하며 RQ로부터 데이터를 로드하는 함수로 **tpenq_ctl()**를 이용하여 서비스를 요청한 결과나 서비스명을 NULL로 해서 저장한 데이터를 읽는 함수이다. tpenq_ctl()을 호출할 때 flags에 TPNOREPLY를 설정한 서비스는 결과를 받을 수 없다. tpdnq_ctl() 함수는 트랜잭션 모드에서 수행하면, 데이터를

RQ에서 로드하는 동안 트랜잭션으로 처리된다. 2번 이상의 tpdep_ctl()을 하나의 트랜잭션으로 묶은 경우 모든 데이터가 RQ에서 데이터를 로드하는 도중 에러가 발생하면 rollback된다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
int tpdeq_ctl(char *qname, char *svc, TMQCTL *ctl, char **data,
              long *len, long flags);
```

- 파라미터

파라미터	설명
qname	데이터를 저장할 RQ의 이름으로 환경 파일에 등록된 이름이어야 한다.
svc	tpenq_ctl() 호출하는 경우 svc에 전달한 이름이어야 한다. 서비스명으로 tpenq_ctl()를 호출한 경우 서비스가 자동 요청되고, 결과가 RQ에 저장된다. 이때 서비스 결과를 받기 위해서는 tpdeq_ctl()도 동일한 서비스명을 입력한다. 서비스명을 NULL로 tpenq_ctl()를 호출한 경우 tpdeq_ctl()도 서비스명을 NULL로 한다. 서비스명이 NULL인 경우 서비스명에 상관없이 큐에 쌓여있는 모든 데이터를 하나씩 deq할 수 있다. tpenq_ctl() 함수가 에러나 시스템 장애로 인해 Fail 큐에 저장된 데이터를 tpdeq_ctl()하려면 svc에 _rq_sub_queue_name [TMAX_FAIL_QUEUE]를 주고 deq해야 한다.
ctl	• * flags ◦ TPRQS_AUTOACK : deq하는 경우 별도로 ack을 주지 않아도 rqs에서 삭제되도록 한다. ◦ TPRQS_NOAUTOACK : deq하는 경우 rqs에서 삭제되지 않도록 한다. ◦ TPRQS_ACK_SUCCESS : ctl에 해당하는 q element를 ack을 주어 삭제하도록 한다. ◦ TPRQS_ACK_FAIL : deq하는 경우 rqs에서 삭제되지 않도록 한다. • * exp_time deq 이후 주어진 시간 이후까지 응답이 오지 않는다면 q element 가 다시 deq할 수 있는 상태로 돌아간다. (단위 : 초)
data	반드시 이전에 tpalloc()에 의해 할당된 버퍼에 대한 포인터이어야 한다. tpdeq_ctl()가 성공적으로 수행된 경우 수신된 데이터가 저장된다.

파라미터	설명
len	tpdeq_ctl()가 성공적으로 수신한 데이터의 길이이다. tpdeq_ctl()는 필요하다면 응답 내용이 지정된 버퍼에 수신될 수 있도록 버퍼 크기를 증가시킨다. *data는 수신 데이터가 커서 변경될 수도 있고, 이 외에 다른 이유에 의해서도 변경될 수 있다. 만약 len이 호출 전 버퍼의 총 크기보다 크다면 len이 그 버퍼의 새로운 크기가 된다. len이 0으로 반환되었다면 어떤 데이터도 수신되지 않고 *data와 len이 지시하는 버퍼 모두 아무런 변화가 없다. *data나 len이 NULL이 되는 것은 에러이다.
flags	데이터 처리 유형을 설정한다. 다음은 flags로 사용 가능한 값에 대한 설명이다. • TPRQS RQ에서 data를 가져올 때 사용된다. Reply 큐로부터 서비스의 결과를 가져오기 위해서 설정된다. • TPFUNC 서비스별 RQ data를 관리할 때 사용한다.TPFUNC flags가 설정되지 않았다면 처음 tpenq()를 통해 저장된 데이터가 제거(dequeued)된다. 데이터가 저장되기 전에 해야할 경우에는 tpenq()를 호출할 때 TPFUNC를 같이 설정한다. • TPBLOCK tpdeq_ctl()를 호출할 때 블록 타임아웃 시간 동안 메시지가 올 때까지 기다린다. • TPNOTIME TPBLOCK과 함께 사용되면 블록 타임아웃 시간에 관계 없이 응답이 올 때까지 기다린다. • 0(zero) 0(zero) flags는 데이터를 자신이 접속한 클라이언트의 버퍼에서 가져올 때 사용한다.

- 반환값

반환값	설명
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpdeq_ctl()가 정상 처리되지 않을 경우 tperrno에 아래 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	인수가 유효하지 않다. 예를 들어 데이터가 tpalloc()으로 할당되지 않은 버퍼를 가리키거나 또는 flags가 유효하지 않다.
[TPGOTSIG]	TPSIGRSTRT가 설정되지 않은 상태에서 시그널이 수신되었다.
[TPEMATCH]	서비스 이름이 잘못 되었거나 해당 조건을 만족하는 데이터를 찾지 못했다.
[TPENOENT]	존재하지 않는 qname이 사용되었다.
[TPEPROTO]	tpdeq_ctl()이 부적절한 상황에서 호출되었다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

main(int argc, char *argv[])
{
    char    *sndbuf, *rcvbuf;
    long    rcvlen, sndlen, revent;
    int ret, i, cd;
    TMQCTL   *ctl;

    ctl = (TMQCTL*)malloc(sizeof(TMQCTL));
    memset(ctl, 0, sizeof(TMQCTL));
    if (argc != 2) {
        printf("Usage: toupper_rq string\n");
        exit(1);
    }

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

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

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

    if ((rcvbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL) {
        printf("recvbuf alloc failed !\n");
        tpfree((char *)sndbuf);
```

```
        tpend();
        exit(1);
    }

    ret = tx_begin();
    if(ret < 0)
    {
        fprintf(stderr, “tx_begin() fail\n”);
        tpfree((char *)sndbuf);
        tpfree((char *)rcvbuf);
        exit(1);
    }

    strcpy(sndbuf, argv[1]);
    cd = tpdeq_ctl(“txrq1”, “TOUPPER”, ctl, &rcvbuf, &rcvlen, TPRQS );
    if (cd < 0)
    {
        printf(“tpdeq failed [%s]\n”, tpstrerror(tperrno) );
        tpfree((char *)sndbuf);
        tpfree((char *)rcvbuf);
        tpend();
        exit(1);
    }

    cd = tpdeq_ctl(“txrq1”, “TOUPPER”, ctl, &rcvbuf, &rcvlen, TPRQS );
    if (cd < 0)
    {
        printf(“tpdeq failed [%s]\n”, tpstrerror(tperrno) );
        tpfree((char *)sndbuf);
        tpfree((char *)rcvbuf);
        tpend();
        exit(1);
    }

    data process....
    ret = tx_commit();
    if(ret < 0)
    {
        fprintf(stderr, “tx_commit() fail\n”);
        tpfree((char *)sndbuf);
        tpfree((char *)rcvbuf);
        tx_rollback();
        exit(1);
    }
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
}
```

- 관련 함수

  tpenq_ctl(), tpqstat()

3.1.54. tpdiscon

서버와 클라이언트에서 대화형 통신의 연결을 종료하는 함수이다. **tpconnect()**으로 연결한 서비스일 경우 극히 예외적인 경우에 연결을 즉시 종료하고 연결된 상대방에게 **TPEV_DISCONIMM** 이벤트를 발생시킨다.

tpdiscon()은 단지 대화형 통신을 시작한 측에서만 호출할 수 있고 구별자(descriptor)를 제공한 서비스에서는 호출할 수 없다. 대화형 서비스와 통신하는 프로그램은 대화형 통신을 종료할 수 있는데, 올바른 결과를 보장하기 위해서는 서비스에서 tpreturn()으로 연결을 종료하는 것이 좋다.

tpdiscon()은 연결을 강제적으로 종료시킨다. 강제로 종료되는 경우 목적지에 전달되지 못한 일부 데이터는 분실될 수 있다. tpdiscon()은 연결된 상대방 프로그램이 호출자의 트랜잭션에 참여하고 있는 상황에서 호출될 수도 있다. 이 경우 트랜잭션은 취소되고 데이터는 상실될 수 있다. tpdiscon()을 호출하는 함수 호출자가 통신 제어권을 가져야 할 필요는 없다.

- 프로토타입

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

- 파라미터

파라미터	설명
cd	**tpconnect()**에서 반환한 참조 구별자(descriptor)이다.

- 반환값

반환값	설명
-1	함수 호출에 에러가 발생한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpdiscon()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEBADDESC]	cd가 유효하지 않거나 이미 대화형 서비스에서 사용되고 있는 구별자이다.
[TPETIME]	타임아웃이 발생하였다. cd는 더 이상 유효하지 않다.
[TPEPROTO]	tpdiscon()이 부적절한 상황에서 호출되었다.
[TPESYSTEM]	Tmax 시스템 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

void main(int argc, char *argv[])
```

```
{
    int ret, cd;
    char *buf
    long len, revent;

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

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

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

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

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

- 관련함수

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

3.1.55. tpenq

서버와 클라이언트에서 RQ에 데이터를 저장하는 함수로 Tmax 시스템은 시스템의 장애나 에러로 인한 서비스가 불가능한 상태에서도 RQ에 저장된 데이터는 정합성을 보장할 수 있다. RQ에 데이터를 저장해 두었다가 여러 가지 상황으로 시스템이 다운되고 복구 이후 다시 실행되면 이전에 처리하지 못한 데이터를 계속해서 처리할 수 있다.

tpcall()이나 **tpacall()**로 서비스를 요청한 경우 해당 서비스가 수행할 데이터가 누적되어 있다면 서비스를 요청한 데이터도 대기(waiting)한다. 이때 시스템의 장애나 에러로 인해 시스템이 다운되면 대기 중인 데이터는 분실된다. 이러한 문제점을 보안하고 데이터의 정합성을 보장할 수 있도록 tpenq()는 서비스를 요청하는 경우 데이터를 RQ에 저장한다. 트랜잭션 모드에서 수행해도 트랜잭션 모드에서 제외되기 때문에 트랜잭션 모드에서 함수를 수행 도중 에러가 발생해도 트랜잭션에는 영향을 미치지 않는다.

- 프로토타입

```
# include  <tmaxapi.h>
```

```
int  tpenq (char *qname, char *svc, char *data, long len, long flags)
```

- 파라미터

파라미터	설명
qname	qname은 데이터를 저장할 RQ의 이름으로 config 파일에 등록된 이름이어야 한다.
svc	데이터를 RQ에 저장하고, svc 이름이 NULL이 아니면 즉시 서비스를 요청한다. svc 이름이 NULL이면 데이터는 RQ에 저장되고 서비스는 수행되지 않는다. 이 경우 나중에 tpdeq()를 이용하여 서비스를 요청해야 한다. svc로 명명된 서비스가 없거나 또는 서비스를 수행하고 처리 결과를 받지 않은 상태에서 시스템 장애가 발생할 경우에는 이 데이터는 내부적으로 Fail 큐에 저장된다. 데이터는 **tpdeq()**로 서비스를 재요청하거나 에러 처리를 해야 한다.
data	NULL 값인 경우를 제외하고 반드시 tpalloc()으로 할당된 버퍼에 대한 포인터이어야 한다. data의 유형(type)과 하위 유형(subtype)은 svc가 지원하는 유형이어야 한다.
len	송신하는 데이터의 길이이다. • data가 가리키는 버퍼가 특별한 길이 명시가 필요없는 버퍼 유형(STRING, STRUCT, X_COMMON, X_C_TYPE)인 경우 len은 무시되고 0이 사용된다. • data가 가리키는 버퍼가 길이 명시가 반드시 필요한 버퍼 유형(X_OCTET, CARRAY, MULTI STRUCTURE)인 경우 len은 0이 될 수 없다. • data가 NULL인 경우 len은 무시되고 데이터 없이 서비스 요청이 송신된다.

파라미터	설명
flags	flags에 사용 가능한 값은 다음과 같다. • TPRQS svc가 NULL이 아닌 경우 svc로 명명된 서비스를 요청하고, 처리 결과를 RQ에 저장된다. 서비스 처리 결과는 **tpdeq()**를 이용해서 받는다. svc가 NULL인 경우 데이터는 단지 RQ에 저장되고 서비스는 수행하지 않는다. • TPNOREPLY svc가 NULL이 아닌 경우 svc로 명명된 서비스는 요청하지만, 처리 결과는 RQ에 저장하지 않겠다는 의미이다. svc가 NULL인 경우 데이터는 단지 RQ에 저장되고 서비스는 수행하지 않는다. • TPFUNC 서비스별 RQ 데이터를 관리할 때 사용한다. flags가 설정되지 않았다면 처음 tpenq()를 통해 저장된 데이터가 제거된다. 데이터가 저장되기 전에 제거해야 할 경우에는 tpenq() 호출할 때 TPFUNC를 같이 설정한다. • 0(zero) 서비스 처리 결과를 RQ에 저장하지 않고, 함수 호출자가 접속되어 있는 Tmax 시스템의 클라이언트 버퍼에 저장한다. 서비스는 RQ를 통해 요구하지만 결과는 tpcall()처럼 함수 호출자가 접속한 클라이언트의 버퍼에서 가져올 때 사용한다. 0(zero)flags가 설정되면 나중에 처리 결과를 받기 위해서는 tpdeq() 호출할 때 flags에 0(zero)을 설정해야 한다.

- 반환값

반환값	설명
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpenq()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 데이터가 tpalloc()으로 할당되지 않은 버퍼를 가리키거나, 또는 flags가 유효하지 않은 경우에 발생한다.
[TPENOENT]	존재하지 않는 qname이 사용되었다.
[TPEQFULL]	지속적인 서비스 결과로 지정된 큐의 크기를 넘는 경우에 발생한다.
[TPGOTSIG]	TPSIGRSTRT가 설정되지 않은 상태에서 시그널이 수신되었다.
[TPEPROTO]	tpenq()이 부적절한 상황에서 호출되었다.
[TPESYSTEM]	Tmax 시스템 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.

에러 코드	설명
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

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

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

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

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

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

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

- 관련함수

  tpdeq(), tpqstat()

3.1.56. tpenq_ctl

서버와 클라이언트에서 트랜잭션을 지원하며 RQ 데이터를 저장하는 함수이다. Tmax 시스템은 시스템의 장애나 에러로 인한 서비스가 불가능한 상태에서도 RQ에 저장된 데이터는 정합성을 보장할 수 있다. RQ에 데이터를 저장해 두었다가, 여러 가지 상황으로 시스템이 다운되고, 복구 이후 다시 실행되면 이전에 처리하지 못한 데이터를 계속해서 처리할 수 있도록 한다.

tpcall()이나 **tpacall()**로 서비스를 요청한 경우나 해당 서비스가 수행할 데이터가 누적된 경우에 방금 서비스를 요청한 데이터도 대기(wating)한다. 이 때 시스템의 장애나 에러로 인해 시스템이 다운되면 대기 중인 데이터는 분실된다. 이러한 문제점을 보안하고 데이터의 정합성을 보장할 수 있도록 tpenq_ctl()는 서비스를 요청할 때 데이터를 RQ에 저장하는 함수이다.

tpenq_ctl() 함수는 트랜잭션 모드에서 수행하면, 데이터를 RQ에 저장하기까지 트랜잭션으로 처리된다. 2번 이상의

tpenq_ctl()을 하나의 트랜잭션으로 묶은 경우 모든 데이터가 RQ에 저장되기까지 **tpdeq_ctl()**를 통해 데이터를 로드할 수 없으며, 저장되는 도중 에러가 발생하면 rollback된다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
int tpenq_ctl(char *qname, char *svc, TMQCTL *ctl, char *data, long len,
              long flags);
```

- 파라미터

파라미터	설명
qname	데이터를 저장할 RQ의 이름으로 환경 파일에 등록된 이름이어야 한다.
svc	데이터를 RQ에 저장하고, svc 이름이 NULL이 아니면 즉시 서비스를 요청한다. svc 이름이 NULL이면 데이터는 RQ에 저장되고 서비스는 수행되지 않는다. 이 경우 나중에 tpdeq_ctl()를 이용하여 서비스를 요청해야 한다. svc로 명명된 서비스가 없거나 또는 서비스를 수행하고 처리 결과를 받지 않은 상태에서 시스템 장애가 발생할 경우에 이 데이터는 내부적으로 Fail 큐에 저장된다. 이 데이터도 **tpdeq_ctl()**로 서비스를 재요청하거나 에러 처리를 해야 한다.
ctl	TMQCTL의 det_time를 사용하여 일정시간을 설정하면 그 시간 이후에 svc call한다. deq_time을 설정할 때는 현재시간에 delay_time를 설정해야 한다. delay_time을 3으로 설정하고 싶은 경우 'cur_time + 3'과 같이 설정해야 한다. TMQCTL의 기능 중 현재는 deq_time만 지원하며, deq_time는 트랜잭션과 함께 사용할 수 없다. flags 항목에 TPRQS_NON_PERSISTENT로 설정할 경우 메시지를 파일에 기록하지 않고 메모리에만 기록한다.
data	NULL인 경우를 제외하고 반드시 tpalloc()으로 할당된 버퍼에 대한 포인터이어야 한다. data의 유형(type)과 하위 유형(subtype)은 svc가 지원하는 유형들이어야 한다.
len	송신하는 데이터의 길이이다. • data가 가리키는 버퍼가 특별한 길이 명시가 필요 없는 버퍼 유형(STRING, STRUCT, X_COMMON, X_C_TYPE)이라면 len은 무시되고 보통 0이 사용된다. • data가 가리키는 버퍼가 길이 명시가 반드시 필요한 버퍼 유형(X_OCTET, CARRAY, MULTI STRUCTURE)이라면 len은 0이 될 수 없다. • data가 NULL인 경우 len은 무시되고 데이터 없이 서비스 요청이 송신된다.

파라미터	설명
flags	데이터 처리 유형을 설정한다. 다음은 flags에 설정 가능한 값에 대한 설명이다. • TPRQS svc가 NULL이 아닌 경우 svc로 명명된 서비스를 요청하고, 처리 결과를 RQ에 저장된다. 서비스 처리 결과를 받으려면 tpdeq_deq() 함수을 이용하여 받는다. svc가 NULL인 경우 데이터는 단지 RQ에 저장되고 서비스는 수행하지 않는다. • TPNOREPLY svc가 NULL이 아닌 경우 svc로 명명된 서비스는 요청하지만, 처리 결과는 RQ에 저장하지 않는다. svc가 NULL인 경우 데이터는 단지 RQ에 저장되고 서비스는 수행하지 않는다. • TPFUNC 서비스별 RQ 데이터를 관리할 때 사용한다. TPFUNC flags가 설정되지 않았다면 처음 tpenq_ctl()를 통해 저장된 데이터가 제거(dequeued)된다. 데이터가 저장되기 전에 해야할 경우에는 tpenq_ctl()를 호출될 때 TPFUNC를 같이 설정한다. • 0(zero) 서비스 처리 결과를 RQ에 저장하지 않고, 함수 호출자가 접속되어 있는 Tmax 시스템의 클라이언트 버퍼에 저장한다. 서비스는 RQ를 통해 요구하지만 결과는 tpcall()처럼 함수 호출자가 접속한 클라이언트의 버퍼에서 가져올 때 사용한다. 처리 결과를 받기 위해서 tpdeq_ctl()를 호출할 때 flags에 0(zero)을 설정해야 한다.

• 반환값

반환값	설명
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

• 오류

tpenq_ctl()가 정상 처리되지 않을 경우 tperrno에 아래 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	인수가 유효하지 않다. 예를 들어 data가 tpalloc()으로 할당되지 않은 버퍼를 가리키거나 또는 flags가 유효하지 않다.
[TPENOENT]	존재하지 않는 qname이 사용되었다.
[TPEQFULL]	지속적인 서비스 결과로 지정된 큐의 크기를 초과하는 경우에 발생한다.
[TPGOTSIG]	TPSIGRSTRT가 설정되지 않은 상태에서 시그널이 수신되었다.
[TPEPROTO]	tpenq_ctl()이 부적절한 상황에서 호출되었다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

main(int argc, char *argv[])
{
    char    *sndbuf, *rcvbuf;
    long    rcvlen, sndlen, revent;
    int ret, i, cd;
    TMQCTL   *ctl;
    long t;

    ctl = (TMQCTL*)malloc(sizeof(TMQCTL));
    memset(ctl, 0, sizeof(TMQCTL));
    time(&t);
    ctl->deq_time = (int)t + 3;

    if (argc != 2) {
        printf("Usage: toupper string\n");
        exit(1);
    }
    if ( (ret = tmaxreadenv( "tmax.env","TMAX" )) == -1 ){
        printf( "tmax read env failed\n" );
        exit(1);
    }
    if (tpstart((TPSTART_T *)NULL) == -1){
        printf("tpstart failed\n");
        exit(1);
    }
    if ((sndbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL) {
        printf("sendbuf alloc failed !\n");
        tpend();
        exit(1);
    }
    if ((rcvbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL) {
        printf("recvbuf alloc failed !\n");
        tpfree((char *)sndbuf);
        tpend();
        exit(1);
    }
    ret = tx_begin();
    if(ret < 0)
    {
        fprintf(stderr, "tx_begin() fail\n");
        tpfree((char *)sndbuf);
        tpfree((char *)rcvbuf);
        exit(1);
```

```
    }

    strcpy(sndbuf, argv[1]);
    cd = tpenq_ctl("txrq1", "TOUPPER", ctl, (char *)sndbuf, 0, TPRQS );
    if (cd < 0) {
        printf("tpenq failed [%s]\n", tpstrerror(tperrno));
        tpfree((char *)sndbuf);
        tpfree((char *)rcvbuf);
        tpend();
        exit(1);
    }
    cd = tpenq_ctl("txrq1", "TOUPPER", ctl, (char *)sndbuf, 0, TPRQS );
    if (cd < 0) {
        printf("tpenq failed [%s]\n", tpstrerror(tperrno));
        tpfree((char *)sndbuf);
        tpfree((char *)rcvbuf);
        tpend();
        exit(1);
    }
    data process....
    ret = tx_commit();
    if(ret < 0)
    {
        fprintf(stderr, "tx_commit() fail\n");
        tpfree((char *)sndbuf);
        tpfree((char *)rcvbuf);
        tx_rollback();
        exit(1);
    }

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

    tpend();
}
```

- 관련 함수

  tpdeq_ctl(), tpqstat()

3.1.57. tperrordetail

서버와 클라이언트에서 Tmax 시스템 호출할 때 발생한 오류의 자세한 정보를 얻을 때 사용하는 함수로 오류 심각성의 정도를 측정할 때 사용된다. 이런 경우 클라이언트가 적절한 조치를 취해 오류에 빠르게 대처할 수 있다. 시스템 단계의 오류가 발생하였을 경우에는 관리자에게 오류 수정을 요청해야 하며, 애플리케이션 단계에서 발생한 오류일 경우에는 개발자에게 문제 해결을 요청할 수 있다.

- 프로토타입

```
# include  <atmi.h>
int  tperrordetail(int errno)
```

- 파라미터

파라미터	설명
errno	현재는 사용하지 않고 에러 코드에 대한 정보는 tperrno로 판단한다.

- 반환값

반환값	설명
1	애플리케이션 단계의 오류가 발생한 경우이다.
2	시스템 단계 오류가 발생하는 경우이다.
-1	알 수 없는 오류가 발생한 경우이다.

- 예제

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

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

    data process…

    ret = tpcall(“SERVICE”, sndbuf, 20, &rcvbuf, &rcvlen, TPNOCHANGE);
    if (ret == -1){
        fprintf(stderr, “tpcall fail,,,[%d][%s]\n”,
                tperrordetail(tperrno), tpstrerror(tperrno));
        error processing
    }

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

3.1.58. tpextsvcinfo

서버와 클라이언트에서 **tpdeq()**로 RQ에서 데이터를 읽은 경우 해당 데이터에 대한 상세한 정보를 제공하는 함수이다.

- 프로토타입

```
# include <tmaxapi.h>
int  tpextsvcinfo (char *data, char *svc, , int *type, int *errcode )
```

- 파라미터

파라미터	설명
data	tpalloc()으로 할당되어 tpdeq()를 이용하여 RQ로부터 읽은 데이터가 저장되어 있는 포인터이다.
svc	해당 데이터의 서비스명을 받아오기 위한 포인터이다.
type	해당 데이터의 처리 결과를 나타낸다. 다음은 type에 설정 가능한 값에 대한 설명이다. • TPREQ(0) tpenq()를 수행할 때 두 번째 인자로 NULL을 지정한 경우 tpenq가 정상적으로 된 경우 이와 같이 type에 TPREQ가 설정된다. • TPFAIL(1) tpenq()를 수행할 때 두 번째 인자로 서비스명을 지정한 경우 서비스에서 tpreturn의 첫 번째 인자로 TPFAIL이 호출된 경우에 이와 같이 type에 TPFAIL이 설정된다. • TPSUCCESS(2) tpenq()를 수행할 때 두 번째 인자로 서비스명을 지정한 경우 서비스에서 tpreturn의 첫 번째 인자로 TPSUCCESS가 호출된 경우에 이와 같이 type에 TPSUCCESS가 설정된다. • TPERR(-1) tpenq()가 실패하여 Fail 큐에 송신된 경우 이와 같이 type에 TPERR이 설정된다.
errcode	에러가 발생하는 경우 해당되는 에러 코드 값이 저장된다.

- 반환값

반환값	설명
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpextsvcinfo()이 정상 처리되지 않을 경우 tperrno에 아래 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	인수가 유효하지 않다. 예를 들면 data나 svc가 NULL이다.
[TPEITYPE]	인수가 유효하지 않다. 예를 들면 data가 RQ에서 받아온 data가 아니다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 관련 함수

  tpenq(), tpdeq(), tpextsvcname()

3.1.59. tpextsvcname

서버와 클라이언트에서 **tpdeq()**로 RQ에서 데이터를 읽은 경우 해당 데이터의 서비스명을 알려고 할 때 사용한다. tpextsvcname()은 _Fail 큐에 저장되어 있는 데이터를 tpdeq()로 읽은 경우에 사용한다.

- 프로토타입

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

- 파라미터

파라미터	설명
data	tpdeq()를 이용하여 RQ로부터 읽은 데이터가 저장되어 있는 포인터로 tpalloc()으로 할당된다.
svc	해당 데이터의 서비스명을 받아오기 위한 포인터이다.

- 반환값

반환값	설명
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpextsvcname()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들면 data에 tpalloc()으로 할당되지 않은 버퍼가 전달되는 경우에 발생한다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

```
    long len;

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

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

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

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

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

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

- 관련함수

  tpenq(), tpdeq()

3.1.60. tpfree

서버와 클라이언트에서 유형 버퍼(typed buffer)에 할당된 메모리를 해제하는 함수로 이전에 **tpalloc()**이나 **tprealloc()**으로 얻어진 버퍼를 해제한다.

- 프로토타입

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

- 파라미터

파라미터	설명
ptr	tpalloc()이나 tprealloc()으로 얻어진 버퍼에 대한 포인터이다. ptr이 NULL이면 아무 일도 일어나지 않는다. ptr이 유형 버퍼를 가리키지 않거나, tpfree()로 이미 해제된 공간을 가리킨다면 그 결과는 알 수 없다. 서비스 루틴 내부에서 ptr이 서비스 루틴에게 전달된 버퍼라면 tpfree()는 버퍼를 해제하지 않고 그냥 반환한다. 버퍼를 제거하는 과정에서 일부 버퍼 유형은 관련 데이터나 상태 정보를 해제할 필요가 있다. tpfree()는 버퍼를 해제하기 전에 이와 관련된 정보도 제거한다. tpfree()가 수행되면 ptr은 XATMI 루틴에 파라미터로 전달될 수 없으며 어떤 다른 방식으로 사용될 수 없다.

- 반환값

  tpfree()는 함수 호출자에게 아무런 값도 반환하지 않는다.

- 예제

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

void main(int argc, char *argv[])
{
    int ret;
    struct data *buf;
    char *message, *message2;

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

    buf=(struct data *)tpalloc(“STRUCT”, “data”,0);
    if (buf==NULL) { error processing }
    message=tpalloc(“STRING”, NULL, 0);
    if (message==NULL) { error processing }

    message2=tpalloc(“CARRAY”, NULL, 20);
    if (message==NULL) { error processing }

    data process....
    tpfree((char *)buf);
    tpfree(message);
    tpfree((char *)message2);
    tpend();
}
```

- 관련함수

  tpalloc(), tprealloc()

tpfree()는 C 라이브러리의 malloc(), realloc() 또는 free()와 함께 사용될 수 없다.

> tpalloc()으로 할당된 버퍼는 free()로 해제할 수 없다.

3.1.61. tpget_timeout

서버와 클라이언트에서 블록 타임아웃 시간을 반환하는 함수로 서버에 설정되어 있는 서비스 제한시간, 현재 설정된 블록 타임아웃 시간을 확인할 때 사용된다.

tpset_timeout()로 서비스 제한시간을 설정한 경우 **tpget_timeout()**으로 확인할 수 있다. tpset_timeout() 함수로 블록 타임아웃을 설정하지 않은 경우 Tmax 환경 파일에 설정된 **BLOCKTIME**에 설정된 값을 확인할 수 있다.

- 프로토타입

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

- 반환값

  tpget_timeout()는 현재 설정된 블록 타임아웃을 초 단위로 반환하며, 에러를 반환하지는 않는다.

- 예제

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

void main(int argc, char *argv[])
{
    int ret;
    char *sndbuf, *rcvbuf;
    long sndlen,  rcvlen;

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

    sndbuf = (char *)tpalloc("CARRAY", NULL, 20);
    if (sndbuf==NULL) {error processing };
    rcvbuf = (char *)tpalloc("CARRAY", NULL, 20);
    if (rcvbuf==NULL) {error processing };
    data process....
    sndbuf=strlen(sndbuf);
    ret=tpset_timeout(4);
    if (ret==-1) { error processing }

    ret=tpget_timeout();
    printf("block time = %d\n", sec);

    ret=tpcall("SERVICE", sndbuf, sndlen, &rcvbuf, &rcvlen, TPNOCHANGE);
    if (ret==-1) { error processing }
    data process....
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
```

```
}
```

3.1.62. tpgetactivesvr

서버와 클라이언트에서 현재 활성화되어 있는 서버들의 목록을 알아내는 함수이다. 파라미터로 지정한 노드에서 현재 활성화되어 있는 서버들의 리스트를 얻을 수 있다.

- 프로토타입

```
#include <tmaxapi.h>
int tpgetactivesvr(char *nodename, char **outbufp);
```

- 파라미터

파라미터	설명
nodename	활성화되어 있는 서버들을 리스트로 알려고 할 때 해당 노드의 이름이다. 클라이언트에서 nodename이 NULL로 호출한 경우에는 현재 접속한 노드의 서버 리스트를 전달하고 서버에서 nodename을 NULL로 호출한 경우에는 서버가 기동된 노드의 서버 리스트를 전달한다. 서버 리스트는 사용자가 파라미터로 입력한 포인터인 outbufp에 저장된다.
outbufp	수신될 응답 버퍼에 대한 포인터로 서버 리스트가 저장된다. 서버 리스트는 tpgetactivesvr()에서 tpalloc()으로 생성한 CARRAY 유형의 버퍼이다. 사용을 완료한 후에는 반드시 tpfree()로 제거해야 한다. 서버 리스트는 NULL로 구분이 되는 문자열의 모임이다. 예를 들어 해당 노드에 svr10과 svr111이라는 2개의 서버가 있는 경우 서버 리스트에 담겨져 오는 내용은 { 's', 'v', 'r', '1', '0', NULL, 's', 'v', 'r', '1', '1', '1', NULL }이다.

- 반환값

반환값	설명
서버의 개수	함수 호출에 성공한 경우이다. 활성화되어 있는 서버의 개수를 반환한다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpgetactivesvr()가 정상적으로 수행하지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	잘못된 파라미터를 사용하였다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 클라이언트의 경우 대부분 네트워크 에러이다.

에러 코드	설명
[TPEOS]	운영 시스템에 에러가 발생하였다. 대부분 메모리 부족으로 인한 에러이다.

- 예제

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

main(int argc, char *argv[])
{
    char    *sndbuf, *rcvbuf, *data;
    long    rcvlen, sndlen;
    int     ret, n, i;
    char nodename[35];

    if (argc != 2) {
        printf(“Usage: toupper string\n”);
        exit(1);
    }
    if ( (ret = tmaxreadenv( “tmax.env”,”TMAX” )) == -1 ){
        printf( “tmax read env failed\n” );
        exit(1);
    }

    if (tpstart((TPSTART_T *)NULL) == -1){
        printf(“tpstart failed\n”);
       exit(1);
    }

    if ((rcvbuf = (char *)tpalloc(“STRING”, NULL, 0)) == NULL) {
        printf(“recvbuf alloc failed !\n”);
        tpfree((char *)sndbuf);
        tpend();
        exit(1);
    }

    strcpy(nodename, “starbj”);
    if((n = tpgetactivesvr(nodename, &rcvbuf)) < 0) {
        printf(“getactivesvr failed\n”);
        tpfree((char *)rcvbuf);
        tpend();
        exit(1);
    }
    printf(“Total %d servers\n”, n);
    data = rcvbuf;
    for (i = 0; i < n; i++) {
        printf(“ACTIVE[%s]\n”, data);
        data += strlen(data) + 1;
    }
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
}
```

3.1.63. tpgetcliaddr

Tmax 시스템에 접속된 클라이언트 중 clid에 해당하는 클라이언트의 IP 주소와 포트 번호를 얻는 함수이다.

이 함수는 IPv6 프로토콜 환경에서는 사용할 수 없다. IPv6 환경에서는 tpgetcliaddr_ipv6 함수를 사용한다.

- 프로토타입

```
#include <tmaxapi.h>
int tpgetcliaddr(int clid, int *ip, int *port, long flags);
```

- 파라미터

파라미터	설명
clid	조회할 clid이다.
ip	해당 클라이언트의 정보가 담긴다. IP 주소는 sockaddr_in 구조체에 있는 s_addr 필드이기 때문에 dot 형태로 바꾸기 위해서는 **inet_ntoa()**를 이용해야 한다.
port	해당 클라이언트의 정보가 담긴다.
flags	현재 버전에서는 지원하지 않으나 TPNOFLAGS로 설정한다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpgetcliaddr()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않은 경우로 NULL 형식이 전달되는 경우이다.
[TPEITYPE]	IPv6 프로토콜 환경에서 호출하는 경우이다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.

- 예제

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

TOUPPER(TPSVCINFO *msg)
```

```
{
        int             ret;
        int             clid;
        int             port;
        int*            ip;
        ...
        clid = tpgetclid();
        printf("clid = %d\n", clid);

        ret = tpgetcliaddr(clid, ip, &port, 0);
        if(ret < 0)
          error routine..

        printf("ip = %s, port = %d\n", inet_ntoa(ip), port);
        tpreturn(TPSUCCESS,0,(char *)msg->data, 0,0);
}
```

- 관련 함수

  tpgetclid()

3.1.64. tpgetcliaddr_ipv6

Tmax 시스템에 접속된 클라이언트 중 clid에 해당하는 클라이언트의 IP 주소와 포트 번호를 얻는 함수로 IPv6 환경에서 사용한다.

- 프로토타입

```
#include <tmaxapi.h>
#include <arpa/inet.h>
int tpgetcliaddr_ipv6(int clid, struct sockaddr_storage *saddr, long flags);
```

- 파라미터

파라미터	설명
clid	조회할 clid이다.
ipaddr	해당 클라이언트의 정보가 담긴다. 함수 호출이 성공하면 IP 주소와 포트 번호가 설정된다. sockaddr_storage 구조체의 ss_family 멤버를 통해 IP 프로토콜 버전을 확인할 수 있다. IP 주소를 문자열로 변경하기 위해서는 inet_ntop()를 이용해야 한다. 포트 번호는 ntohs()를 이용해야 올바른 값을 가져올 수 있다.
flags	현재 버전에서는 지원하지 않으나 TPNOFLAGS로 설정한다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpgetcliaddr_ipv6()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않은 경우로 NULL 형식이 전달되는 경우이다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.

- 예제

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

TOUPPER(TPSVCINFO *msg)
{
        int             ret;
        int             clid;
        int             portno;
        struct sockaddr_storage saddr;
        struct sockaddr_in *cli_sin4;
        struct sockaddr_in6 *cli_sin6;
        char ipaddrbuf[INET6_ADDRSTRLEN];
        const char *ipaddr = NULL;
        ...
        clid = tpgetclid();
        printf("clid = %d\n", clid);

        ret = tpgetcliaddr_ipv6(clid, &ipaddr, 0);
        if(ret < 0)
          error routine..

        if (saddr.ss_family == AF_INET) {
            cli_sin4 = (struct sockaddr_in *)&saddr;
            ipaddr = inet_ntop(AF_INET, &(cli_sin4->sin_addr), ipaddrbuf,
                     sizeof(ipaddrbuf));
            portno = ntohs(cli_sin4->sin_port);
        } else if (saddr.ss_family == AF_INET6) {
            cli_sin6 = (struct sockaddr_in *)&saddr;
            ipaddr = inet_ntop(AF_INET6, &(cli_sin6->sin6_addr), ipaddrbuf,
                     sizeof(ipaddrbuf));
            portno = ntohs(cli_sin6->sin6_port);
        }
        if (ipaddr == NULL)
            ipaddr = "unknown";

        printf("ip = %s, port = %d\n", ipaddr, portno);
        tpreturn(TPSUCCESS,0,(char *)msg->data, 0,0);
}
```

- 관련 함수

  tpgetclid()

3.1.65. tpgetclid

caller의 고유한 clid를 알 수 있는 함수이다. clid는 클라이언트, 서버가 가지는 고유한 값으로 클라이언트는 tpstart하는 시점에 clh로부터 부여받는 번호이고, 서버는 (spri + max client index)부터 시작하는 값을 부여받는다. 호출된 서비스에서 tpgetclid()를 호출하면 tpcall, tpacall을 호출한 caller의 clid가 리턴된다. clid는 도메인 시스템 내에서 유일한 번호이다. 여러 멀티 노드로 도메인 시스템이 구축되어 있어도 유일한 번호를 클라이언트, 서버에게 부여한다.

서버에서 사용하는 경우 일반적으로 서비스를 요청한 해당 클라이언트 ID를 구해서 **tpsendtocli()**에서 클라이언트로 메시지를 보내기 위해서 사용한다. 서비스 중간에 tpcall, tpacall 등으로 다른 서비스를 호출할 경우 피호출된 서비스에서는 real 클라이언트 clid를 가져올 수 없다.

클라이언트에서 사용하는 경우 클라이언트 자신의 clid를 리턴한다. 어느 서비스(tpforward를 통해 서비스가 relay되는 2차 서비스 포함)에서나 최초 tpcall을 시작시킨 clid의 값을 가져오는 함수로 tpgetfclid()를 제공한다.

- 프로토타입

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

- 반환값

반환값	설명
0 이상의 정수값	함수 호출에 성공하는 경우이다. 클라이언트의 번호에 해당하는 0 이상의 정수값을 반환한다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpgetclid()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	tpgetclid()가 부적절한 상황에서 호출되었다. 예를 들어 클라이언트 프로그램 내에 사용된 경우 발생한다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

SERVICE(TPSVCINFO *msg)
{
    int ret, clid;
    char *buf;
    buf = (char *)tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
```

```
    strcpy(buf, msg->data);
    data process…

    clid = tpgetclid();
    if (clid==-1) { error process }

    ret=tpsendtocli(clid, buf, strlen(buf), 0);
    if (ret==-1) { error processing }
    data process….

    tpreturn(TPSUCCESS,0,buf, strlen(buf), 0);
}
```

- 관련 함수

  tpsendtocli()

3.1.66. tpgetcliinfo

Tmax 시스템에 접속된 클라이언트 중 clid에 해당하는 클라이언트의 정보를 얻어 오는 함수이다.

- 프로토타입

```
#include <tmadmin.h>
int tpgetcliinfo(int clid, struct tmadm_cliinfo_body *info)
```

- 파라미터

파라미터	설명
clid	조회할 clid이다.
info	clid와 일치하는 client의 정보가 담기는 곳이다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

반환한 struct tmadm_cliinfo_body의 내용은 다음과 같다.

```
struct tmadm_cliinfo_body {
    int no; /* cli index */
    int clid; /* CLID */
    int clhno;
    int count;
    int idle;
    int reserve_int[3];
```

```
    char status[TMAX_NAME_SIZE];
    char addr[TMAX_IPADDR_SIZE];
    char usrname[TMAX_NAME_SIZE];
    int qpcount;
    int emaxcount;
    int reserved1;
    int reserved2;
};
```

멤버	설명
no	클라이언트의 번호이다.
clid	해당 클라이언트의 clid이다. 클라이언트 번호와는 다른 값이다.
clhno	클라이언트가 접속해 있는 CLH의 번호이다.
count	수행된 클라이언트의 요청 개수이다.
idle	클라이언트가 최초로 CLH에 접속한 시간으로부터 현재까지 흐른 시간이다. (단위: 초)
reserve_int	아직 사용하지 않는 항목이다.
status	클라이언트가 보낸 요청의 현재 상태를 의미한다. • RUNNING : 현재 요청된 서비스가 실행 중인 상태 • READY : 서비스 수행 정상 종료, 혹은 서비스 요청을 보내기 전 상태 • QUEUED : 요청이 대기 중인 상태
addr	클라이언트의 IP 주소이다.
usrname	tpstart할 때 인자로 넘겼던 tpinfo에 있는 클라이언트의 usrname 값이다.
qpcount	클라이언트가 보낸 요청들 중 purge된 요청들의 개수이다.
emaxcount	클라이언트가 보낸 요청들 중 queue의 max 값에 도달한 요청들의 개수이다.
reserved1	아직 사용하지 않는 항목이다.
reserved2	아직 사용하지 않는 항목이다.

- 오류

  tpgetcliinfo()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않은 경우로 NULL 형식이 전달되거나 clid가 잘못된 경우이다.
[TPEITYPE]	IPv6 프로토콜 환경에서 호출하는 경우이다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	동적 할당에 실패한 경우이다.

- 예제

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

PRINTCLIINFO(TPSVCINFO *msg)
{
   int i, ret, clid;
   struct tmadm_cliinfo_body cliinfo = {0, };

   clid = tpgetclid();
   if (clid < 0)
      error routine..

    ret = tpgetcliinfo(clid, &cliinfo);
    if (ret < 0)
       error routine..

     printf("INPUT : data=%s\n", msg->data);
     printf("CLIINFO : no:%d clid:%d clhno:%d count:%d idle:%d status:%s addr:%s usrname:%s\n",
             cliinfo.no, cliinfo.clid, cliinfo.clhno, cliinfo.count, cliinfo.idle,
             cliinfo.status, cliinfo.addr, cliinfo.usrname);

       ...

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

- 관련 함수

  tpgetclid()

3.1.67. tpgetctxt

함수를 호출하는 Thread에 현재 설정되어 있는 컨텍스트의 ID를 첫 번째 파라미터로 반환하는 함수이다.

MultiThread/MultiContext 서버에서는 Thread에 설정된 컨텍스트가 유효할 경우 1 이상의 값을 가져오게 되며, 컨텍스트가 설정되어 있지 않거나 유효하지 않을 경우에는 TPNULLCONTEXT(-2)를 가져오게 된다. 컨텍스트는 서비스 Thread에서 서비스 요청을 수행 중인 경우에만 유효하다. 만일 서비스 요청을 모두 처리하고 tpreturn()이 호출되면 해당 컨텍스트는 더 이상 유효하지 않으며, 사용자 생성 Thread에서는 컨텍스트를 더 이상 사용할 수 없다.

tpgetctxt 함수는 클라이언트와 서버 프로그램에서 작성 방법에 차이가 있으므로 아래의 서버와 클라이언트 예제를 구분하여 설명한다.

MultiThread/MultiContext 서버에서는 Singlecontext를 지원하지 않는다.

- 프로토타입

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

- 파라미터

파라미터	설명
ctxtid	함수를 호출한 시점의 현재 컨텍스트를 얻어온다. • multicontext 인 경우: 1보다 큰 값을 가져온다. • singlecontext인 경우: 0값을 가져온다.
flags	현재 버전에서는 지원하지 않으나 TPNOFLAGS로 설정한다.

- 반환값

반환값	설명
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpgetctxt()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	잘못된 파라미터가 설정되었다. 예를 들어 첫 번째 파라미터가 포인터 값이던가, 두 번째 파라미터가 0이 아닌 값으로 설정된 경우 발생한다.
[TPESYSTEM]	Tmax 시스템에 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제 - 클라이언트 프로그램

```
int newContext()
{
    int i;
    int id;
    i = tpstart(tpinfo);
    if (i < 0)
    {
       printf("\t[newContext]tpstart fail[%d][%s]\n",tperrno,tpstrerror(tperrno));
       tpfree((char *)tpinfo);
       return -1;
    }
    i = tpgetctxt(&id,TPNOFLAGS);
    if (i < 0)
    {
       printf("\t[newContext]tpgetctxt fail[%d][%s]\n",tperrno,tpstrerror(tperrno));
       return -1;
    }
    return id;
}
```

- 예제 - 서버 프로그램

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

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

    printf("MSERVICE service is started!");
    tpgetctxt(&param.ctxtid, TPNOFLAGS);
    param.svcinfo = svcinfo;

    pthread_create(&tid, NULL, THREAD_ROUTINE, &param);
    pthread_join(tid, NULL);

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

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

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

    if (tpsetctxt(param->ctxtid, TPNOFLAGS) == -1) {
        printf("tpsetctxt(%d) failed, [tperrno:%d]", param->ctxtid, tperrno);
        return NULL;
    }

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

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

    return NULL;
}
```

- 관련함수

  tpsetctxt()

3.1.68. tpgetenv

서버와 클라이언트에서 name이라는 이름으로 등록된 환경변수의 값을 반환하는 함수이다. 일반적으로 서버에서 사용할 수 있는 **getenv()**와 동일한 기능을 수행한다. Windows 클라이언트에서는 **autoexec.bat** 파일에 적용된 값 또는 **tmaxreadenv()**에 의해서 읽혀진 파일에 적용된 값 또는 Windows NT, Windows 2000에서는 환경변수에 설정되어 있는 값을 반환한다. 서버에서는 **셸 환경 파일** 또는 **envfile**에 설정된 값을 반환한다.

- 프로토타입

```
#include <tmaxapi.h>
char *tpgetenv(char *name)
```

- 파라미터

파라미터	설명
name	등록된 환경변수 이름을 설정한다.

- 반환값

반환값	설명
NULL	환경변수가 존재하지 않는 경우이다.
포인터	환경변수가 존재하는 경우이다. 설정된 환경변수의 값에 대한 포인터를 반환한다.

- 예제

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

main(int argc, char *argv[])
{
    char *TMAXD, *SDLFI;
    TMAXD=tpgetenv(“TMAXDIR”);
    SDLFI=tpgetenv(“SDLFILE”);
    printf (“tmaxdir : %s\nsdlfile : %s\n”,TMAXD,SDLFI);
}
```

- 관련 함수

  tpputenv()

3.1.69. tpgetlev

서버와 클라이언트에서 사용하는 함수로 트랜잭션 모드에 있는지의 여부를 확인한다.

- 프로토타입

```
#include <tuxatmi.h>
int  tpgetlev(void)
```

- 반환값

반환값	설명
1	트랜잭션 모드에 있는 경우이다.
0	트랜잭션 모드에 있지 않는 경우이다.

- 예제

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

EXEC SQL include sqlca.h;
EXEC SQL begin declare section;
struct emp *buf;
EXEC SQL end declare section;

DELETE(TPSVCINFO *msg)
{
    struct emp *buf;
    buf = (struct emp *)msg->data;
    data process...
    if (tpgetlev()) printf(“transaction mode\n”);
    else printf(« nontransaction mode\n »);
    EXEC SQL DELETE FROM emp
    WHERE empno = :buf->empno;
    data process...
    tpreturn(TPSUCCESS,0,buf, strlen(buf), 0);
}
```

3.1.70. tpgetnodelist

현재 접속된 Tmax 시스템의 노드의 목록을 가져오는 함수이다. 이때 반환된 nodelist 구조체의 주소는 메모리 해제를 수행해서는 안된다.

- 프로토타입

```
#include <tmaxapi.h>
struct nodelist *tpgetnodelist()
```

- 반환값

반환값	설명
NULL이 아닌 값	함수 호출에 성공한 경우이다. 노드들의 리스트가 담긴 구조체 리스트를 리턴한다.
NULL	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

반환한 struct nodelist와 그 안에 담긴 struct nodeinfo의 내용은 다음과 같다.

  - nodelist

```
struct nodelist {
    int count;
    struct nodeinfo *nodelist;
};
```

항목	설명
count	node들의 개수이다.
nodelist	node들의 정보가 담긴 구조체 nodeinfo의 리스트이다.

◦ nodeinfo

```
struct nodeinfo {
    char nodename[NODE_NAME_SIZE];
    int ipfamily;
    int ipaddr;
    struct in6_addr ipaddr6;
    int port;
};
```

항목	설명
nodename	node들의 개수이다.
ipfamily	AF_INET 또는 AF_INET6 값이 설정된다.
ipaddr	해당 node의 IP 주소이다. 만약 ipfamily에 설정된 값이 AF_INET이면 ipaddr 변수의 값을 IP 주소로 사용한다.
ipaddr6	해당 node의 IP 주소이다. AF_INET6이면 ipaddr6 변수의 값을 사용한다.
port	node의 port 정보이다.

- 오류

  tpgetnodelist()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생한 경우로 메모리 할당 실패 등이 여기에 해당된다.
[TPEPROTO]	클라이언트 모듈에서 해당 함수를 사용할 경우 tpstart를 하지 않은 상태로 사용했을 때 나는 에러이다.

- 예제

```
#include <stdio.h>
```

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

SVGCALL(TPSVCINFO *msg)
{
        char *svcname = msg->data;

        struct nodelist *node = tpgetnodelist();
        if (node == NULL)
           ...error process

        for (i = 0; i < node->count; i++) {
                printf("[%d] %16s : %x:%p\n", i, node->nodelist[i].nodename,
                        node->nodelist[i].ipaddr, node->nodelist[i].port);
        }

        ...

        struct svglist *svg = tpgetsvglist_bynode(node->nodelist[1].nodename, svcname, 0);
        if (svg == NULL)
           ...error process

        for (i = 0; i < svg->ns_entry; i++) {
                printf("[%d] %d(%x)\n", i, svg->s_list[i], svg->s_list[i]);
        }

        ...

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

- 관련 함수

  tpgetsvglist_bynode()

3.1.71. tpgetpeername

서버와 클라이언트에서 연결된 상대방의 소켓 주소를 얻어오는 함수로 Tmax 시스템에 연결이 완료된 후 상대방(노드)의 소켓 주소를 반환한다.

- 프로토타입

```
#include <tmaxapi.h>
int  tpgetpeername(struct sockaddr *name, int *namelen)
```

- 파라미터

파라미터	설명
name	주소가 저장된 구조체이다. IPv6 프로토콜 환경에서는 struct sockaddr_in6 구조체를 사용하여 주소 정보를 확인한다. 또한 struct sockaddr_storage 구조체를 사용하면 IPv4 와 IPv6 환경에서 모두 사용할 수 있다.

파라미터	설명
namelen	함수 호출전에 name 으로 전달하는 구조체의 크기로 초기화해야 한다. 리턴에 성공한 경우에는 실제로 name에 할당된 구조체의 크기가 저장된다.

- 반환값

반환값	설명
소켓 주소	함수 호출에 성공한 경우이다. 상대방의 소켓 주소가 반환된다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpgetpeername()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPINVAL]	파라미터가 유효하지 않다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.
[TPEITYPE]	namelen에 인자의 값인 name 구조체의 크기가 실제 저장될 구조체의 크기보다 작다.

- 예제

```
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <usrinc/atmi.h>
#inlcude <usrinc/tmaxapi.h>
...
DELETE(TPSVCINFO *msg)
{
    struct sockaddr_in cli;
    char ipAddr[16];
    int cli_len, ret;

    data process...
    memset((char *)&cli, 0, sizeof(cli));
    ret = tpgetpeername((struct sockaddr *)&cli, &cli_len);
    if (ret == -1){ error processing }
    else{
        memcpy(ipAddr, inet_ntoa(cli.sin_addr), 16);
    }
    printf("ip = %s , port = %d\n", ipAddr, cli.sin_port);
    data process...

    tpreturn(TPSUCCESS, 0, 0, 0, 0);
}
```

- 예제 - IPv6 프로토콜 환경

```
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <usrinc/atmi.h>
#inlcude <usrinc/tmaxapi.h>
...
DELETE(TPSVCINFO *msg)
{
    struct sockaddr_storage cli_saddr;
    struct sockaddr_in *cli_sin4;
    struct sockaddr_in6 *cli_sin6;
    char ipaddrbuf[INET6_ADDRSTRLEN];
    const char *ipaddr = NULL;
    int cli_len, ret;
    int portno;

    data process...
    memset((char *)&cli_saddr, 0, sizeof(cli_saddr));
    cli_len = sizeof(cli_saddr);
    ret = tpgetpeername((struct sockaddr *)&cli_saddr, &cli_len);
    if (ret == -1) {
        error processing
    }
    else {
        if (cli_saddr.ss_family == AF_INET) {
            cli_sin4 = (struct sockaddr_in *)&cli_saddr;
            ipaddr = inet_ntop(AF_INET, &(cli_sin4->sin_addr), ipaddrbuf,
                     sizeof(ipaddrbuf));
            portno = ntohs(cli_sin4->sin_port);
        } else if (cli_saddr.ss_family == AF_INET6) {
            cli_sin6 = (struct sockaddr_in *)&cli_saddr;
            ipaddr = inet_ntop(AF_INET6, &(cli_sin6->sin6_addr), ipaddrbuf,
                     sizeof(ipaddrbuf));
            portno = ntohs(cli_sin6->sin6_port);
        }
        if (ipaddr == NULL)
            ipaddr = "unknown";
    }
    printf("ip = %s , port = %d\n", ipaddr, portno);
    data process...

    tpreturn(TPSUCCESS, 0, 0, 0, 0);
}
```

- 관련함수

  tpgetpeer_ipaddr(), tpgetsockname(), tpstart()

3.1.72. tpgetrcahseqno

Tmax API인 tpgetsvrseqno API와 동일하게 RCAH 프로세스 번호를 반환하는 함수이다. 이 번호는 RCAL에서 순차적으로 부여한 번호이다.

- 프로토타입

```
#include <ucs.h>
int tpgetrcahseqno()
```

- 반환값

반환값	설명
0	RCAH 프로세스 순서 번호이다.

3.1.73. tpgetrcainfo

RCAH의 Thread 정보를 알려주는 함수이다.

- 프로토타입

```
#include <ucs.h>
void * tpgetrcainfo()
```

- 반환값

  RCA 헤더 파일의 RCAINFO에 대한 포인터를 반환한다.

3.1.74. tpgetrply

서버와 클라이언트에서 비동기적으로 요청한 서비스에 대한 응답을 수신하는 함수로 **tpacall()**로 요청한 서비스에 대한 응답을 수신한다.

- 프로토타입

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

- 파라미터

파라미터	설명
cd	tpacall()에 의해 반환된 호출 구별자를 가리킨다. 보통 cd와 일치하는 응답이 수신되거나 또는 타임아웃이 발생할 때까지 기다린다. 일반적으로 cd는 응답이 수신된 후에는 더 이상 유효하지 않다.
*data	반드시 이전에 tpalloc()에 의해 할당된 버퍼에 대한 포인터이어야 한다.

파라미터	설명
len	tpgetrply()가 성공적으로 수신한 데이터의 길이로 필요한 경우 응답 내용이 지정된 버퍼에 수신될 수 있도록 버퍼 크기를 증가시킨다. *data는 수신 데이터가 커서 변경될 수도 있고, 이 외에 다른 이유에 의해서도 변경될 수 있다. len이 호출 전 버퍼의 총 크기보다 크다면, len이 그 버퍼의 새로운 크기가 된다. len이 0으로 반환되는 경우 어떤 응답도 수신되지 않고 *data와 len이 지시하는 버퍼 모두 아무런 변화가 없다. *data나 len이 NULL이 되는 것은 에러이다.
flags	flags로 사용 가능한 값은 다음과 같다. • TPGETANY 입력값으로 설정한 cd를 무시하고, 이에 상관없이 수신 가능한 응답을 반환하도록 한다. cd는 반환된 응답에 대한 호출 구별자가 된다. 아무런 응답이 없으면 일반적으로 tpgetrply()는 응답이 도착할 때까지 대기한다. TPGETANY가 설정되지 않은 경우 특별한 언급이 없는 한 *cd는 무효화됨에 유의해야 한다. TPGETANY가 설정되었다면, cd는 에러가 발생한 응답에 대한 구별자가 된다. 응답이 반환되기 전에 에러가 발생했다면, cd는 0이 된다. 특별한 언급이 없다면, 호출자의 트랜잭션에 영향을 미치지 않는다. • TPNOCHANGE *data가 가리키는 버퍼의 유형은 변경되지 못한다. 수신된 응답 버퍼와 *data가 가리키는 버퍼의 유형이 다른 경우 *data의 버퍼 유형은 수신자가 인식할 수 있는 한도 내에서 수신된 응답 버퍼의 유형으로 변경되는데 TPNOCHANGE flags가 설정되는 경우에는 변경되지 않는다 . 수신된 응답 버퍼의 유형 및 하위 유형은 *data가 가리키는 버퍼의 유형 및 하위 유형과 반드시 일치해야 한다. • TPNOBLOCK 응답이 도착할 때까지 대기하지 않는다. 수신 가능한 응답이 있는 경우에는 반환을 한다. TPNOBLOCK flags가 지정되지 않았고 수신 가능한 응답이 없다면 함수 호출자는 응답이 도착하거나 또는 타임아웃(트랜잭션 타임아웃이나 블록 타임아웃)이 발생할 때까지 대기한다. • TPNOTIME 함수 호출자가 블록 타임아웃을 무시하고 응답이 수신될 때까지 무한정 대기한다. 트랜잭션 모드에서 tpgetrply()를 한 경우에는 트랜잭션 타임아웃이 적용된다. • TPSIGRSTRT 시그널(signal) 인터럽트를 수용하는 경우 사용한다. 내부에서 시그널 인터럽트가 발생하여 시스템 함수 호출이 방해될 때 TPSIGRSTRT flag가 설정되어 있으면 BLOCKTIME 시간동안 계속 응답을 기다린다. TPSIGRSTRT flags가 설정되지 않은 경우 시그널 인터럽트가 발생했다면, 함수는 실패하고 tperrno에 TPGOTSIG가 설정된다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다. tpreturn()으로 전달되는 tpurcode 전역변수는 tpgetrply()가 성공적으로 반환되었거나 tperrno가 [TPESVCFAIL]인 경우 애플리케이션에서 정의한 값을 갖게 된다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpgetrply()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 cd나 data, *data, len 등이 NULL이거나 또는 flags가 유효하지 않은 경우에 발생한다. cd가 NULL이 아니면 에러 발생 후에도 cd는 유효하며, 그에 대한 응답을 계속 기다린다.
[TPEBADDESC]	cd가 유효하지 않은 구별자이다.
[TPEOTYPE]	수신된 응답의 유형 또는 하위 유형이 호출자가 알지 못하는 유형이다. flags가 TPNOCHANGE로 설정되었는데, *data의 유형 및 하위 유형이 서비스가 송신한 응답의 것과 맞지 않는 경우로 *data의 내용과 *len은 모두 변경되지 않는다. 응답이 호출자의 트랜잭션 모드에서 수신되었다면, 그 트랜잭션은 응답이 무시되었기 때문에 rollback된다.
[TPETIME]	타임아웃이 발생한 경우로 함수 호출자가 트랜잭션 모드에 있다면, 트랜잭션 타임아웃이 발생하였고 그 트랜잭션은 rollback된다. 트랜잭션 모드가 아니고 TPNOTIME과 TPNOBLOCK 어느 것도 지정되지 않았다면, 블록 타임아웃이 발생하였다. 이 두 경우에, *data의 내용과 *len은 변경되지 않는다. 트랜잭션 타임아웃이 발생했다면, 새로운 서비스 요청을 송신한다거나 응답을 기다리는 일은 트랜잭션이 rollback될 때까지 [TPETIME] 에러로 실패하게 된다.
[TPESVCFAIL]	서비스 요청에 대한 응답을 송신하는 서비스 루틴이 애플리케이션에 에러가 발생하여 TPFAIL로 tpreturn()을 호출하였다. 서비스 응답이 수신되었다면, 그 내용들은 *data가 가리키는 버퍼를 통해 사용될 수 있다. 함수 호출자가 트랜잭션 모드에 있다면, 그 트랜잭션은 rollback된다. 트랜잭션 타임아웃이 발생하기 전까지는 트랜잭션이 rollback되기 전에 다른 통신들이 시도될 수 있다. 그러한 통신들은 정상적으로 처리될 수도 있고, 또는 실패할 수도 있다. 이들이 제대로 수행되기 위해서는 TPNOTRAN이 설정되어야 한다. 호출자의 트랜잭션 모드에서 수행된 작업들은 트랜잭션이 완료되면 모두 rollback된다.
[TPEBLOCK]	TPNOBLOCK이 설정된 상태에서 블로킹 상황이 발생하는 경우로 구별자(cd)는 유효하다.
[TPGOTSIG]	TPSIGRSTRT가 설정되지 않은 상태에서 시그널이 수신되었다.
[TPEPROTO]	tpgetrply()이 부적절한 상황에서 호출되었다.
[TPETRAN]	트랜잭션 서비스를 호출할 때 데이터베이스에 문제가 발생하여 xa_start가 실패하였다.
[TPESYSTEM]	Tmax 시스템에 에러가 발생하였다.

에러 코드	설명
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

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

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

    cd = tpacall("SERVICE", buf, 0, TPNOFLAGS);
    if (cd==-1) { error procesing }
    data process....

    ret=tpgetrply(&cd, &buf, &len, TPNOTIME);
    if (ret==-1) { error processing }
    data process....

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

- 관련함수

  tpacall(), tpalloc(), tpreturn()

3.1.75. tpgetsockname

서버와 클라이언트에서 Tmax 시스템 내부적으로 사용되는 소켓 주소를 얻는 함수이다.

- 프로토타입

```
#include <tmaxapi.h>
int  tpgetsockname (struct sockaddr  *name,  int  *namelen)
```

- 파라미터

파라미터	설명
name	소켓 주소를 얻을 메모리의 주소이다. IPv6 프로토콜 환경에서는 struct sockaddr_in6 구조체를 사용하여 주소 정보를 확인한다. 또한 struct sockaddr_storage 구조체를 사용하면 IPv4와 IPv6 환경에서 모두 사용할 수 있다.
namelen	함수 호출 전에 name으로 전달하는 구조체의 크기로 초기화해야 한다. 리턴에 성공한 경우에는 실제로 name에 할당된 구조체의 크기가 저장된다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpgetsockname()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.
[TPEITYPE]	namelen에 인자의 값인 name 구조체의 크기가 실제 저장될 구조체의 크기보다 작다.

- 예제

```
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <usrinc/atmi.h>
#inlcude <usrinc/tmaxapi.h>
...
DELETE(TPSVCINFO *msg)
{
    ...
    struct sockaddr_in cli;
    char ipAddr[16];
    int cli_len, ret, i;
    ...
    memset((char *)&cli, 0, sizeof(cli));

    ret = tpgetsockname((struct sockaddr *)&cli_saddr, &cli_len);
    if (ret == -1){
        error processing
    }
    else{
        memcpy(ipAddr, inet_ntoa(cli.sin_addr), 16);
    }
    printf("ip = %s , port = %d\n", ipAddr, cli.sin_port);
```

```
    ...
}
```

- 예제 - IPv6 프로토콜 환경

```
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <usrinc/atmi.h>
#inlcude <usrinc/tmaxapi.h>
...
DELETE(TPSVCINFO *msg)
{
    struct sockaddr_storage cli_saddr;
    struct sockaddr_in *cli_sin4;
    struct sockaddr_in6 *cli_sin6;
    char ipaddrbuf[INET6_ADDRSTRLEN];
    const char *ipaddr = NULL;
    int cli_len, ret;
    int portno;

    data process...
    memset((char *)&cli_saddr, 0, sizeof(cli_saddr));
    cli_len = sizeof(cli_saddr);


    ret = tpgetsockname((struct sockaddr *)&cli_saddr, &cli_len);
    if (ret == -1) {
        error processing
    }
    else {
        if (cli_saddr.ss_family == AF_INET) {
            cli_sin4 = (struct sockaddr_in *)&cli_saddr;
            ipaddr = inet_ntop(AF_INET, &(cli_sin4->sin_addr), ipaddrbuf, sizeof(ipaddrbuf));
            portno = ntohs(cli_sin4->sin_port);
        } else if (cli_saddr.ss_family == AF_INET6) {
            cli_sin6 = (struct sockaddr_in *)&cli_saddr;
            ipaddr = inet_ntop(AF_INET6, &(cli_sin6->sin6_addr), ipaddrbuf, sizeof(ipaddrbuf));
            portno = ntohs(cli_sin6->sin6_port);
        }
        if (ipaddr == NULL)
            ipaddr = "unknown";
    }
    printf("ip = %s , port = %d\n", ipaddr, portno);
    ...
}
```

- 관련함수

  tpgetpeer_ipaddr(), tpgetpeername(), tpstart()

3.1.76. tpgetsprlist

서버 프로세스 단위로 호출하기 위해 해당 서비스가 속한 서버 프로세스의 인덱스를 가져오는 함수로 해당 서비스가 속한 서버 리스트의 starti, endi를 제공한다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
int tpgetsprlist(char *svc, int svgno, int *starti, int *endi, long flagsl);
```

- 파라미터

파라미터	설명
svc	호출할 서비스를 설정한다.
svgno	해당 서비스가 속해 있는 서버 그룹의 번호로 **tpgetsvglist()**를 호출하거나 **tmadmin**의 cfg –g를 통해서 알 수 있다.
starti	해당 서비스가 속해 있는 서버의 첫 번째 프로세스의 인덱스 번호이다.
endi	마지막 프로세스의 인덱스 번호이다. 예를 들면 starti가 36이며, endi가 40일 경우 프로세스의 일련 번호는 36부터 40까지가 되고 서버의 MIN 값이 5이고, MAX 값이 10인 경우 tpgetsprlist()하면 MAX 값인 10개의 인덱스를 가져온다.
flagsl	현재 버전에서는 지원하지 않으나 TPNOFLAGS나 0으로 설정한다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다. starti, endi를 얻어올 수 있다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpgetsprlist()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러코드	설명
[TPESYSTEM]	Tmax 시스템에 에러가 발생하였다.
[TPEOS]	운영 시스템에 에러가 발생하였다.
[TPEBLOCK]	블로킹 상황이 발생하였다.

- 예제

```
main(int argc, char *argv[])
{
```

```
    int     ret;
    int     starti = 0, endi = 0;

    if (argc != 2) {
       printf("Usage: argv[1] string\n");
       exit(1);
       if((ret = tmaxreadenv( "tmax.env","TMAX" )) == -1 ){
          printf("<%-15s> tmaxreadenv fail [%s]", __FILE__, tpstrerror(tperrno));
          exit(1);
       }

    if (tpstart((TPSTART_T *)NULL) == -1){
        printf("<%-15s> tpstart fail [%s]", __FILE__, tpstrerror(tperrno));
        exit(1);
    }

    /* tpgetsprlist in client */
    ret = tpgetsprlist(argv[1], 2, &starti, &endi, 0);
    if (ret < 0) {
        printf("<%-15s> tpgetsprlist fail [%s]", __FILE__, tpstrerror(tperrno));
        exit(1);
    }
    else {
       printf("Received Message : starti[%d], endi[%d]\n", starti, endi);
    }
    tpend();
}
```

3.1.77. tpgetsvglist

서버와 클라이언트에서 해당 서비스가 속하는 서버 그룹과 이 서버 그룹의 COUSIN으로 설정된 서버 그룹들에 대한 정보를 제공하는 함수이다. 반환한 구조체에는 서버 그룹의 수와 서버 그룹 일련 번호들의 배열이 저장되어 있다.

- 프로토타입

```
#include <tmaxapi.h>
struct svglist* tpgetsvglist(char *svc, long flags)
```

- 파라미터

파라미터	설명
svc	알고자하는 서비스의 이름이다.
flags	현재 버전에서는 지원하지 않으나 TPNOFLAGS으로 설정한다.

- 반환값

반환값	설명
0 이상의 정수	함수 호출에 성공한 경우이다. 서버 프로세스의 일련번호에 해당하는 0 이상의 정수값을 반환한다.

반환값	설명
NULL	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

에러가 발생하면 NULL을 반환하고 tperrno에 상황에 해당하는 값이 설정된다. 그렇지 않으면 서비스 호출에 실패한 서버 그룹 리스트를 struct svglist 에 설정하여 반환한다.

반환한 struct svglist의 내용은 다음과 같다.

```
struct svglist {
    int ns_entry;
    int nf_entry;
    int *s_list;
    int *f_list;
};
```

멤버	설명
ns_entry	해당 서비스가 속하는 서버 그룹의 수이다.
nf_entry	실패한 서버 그룹의 수이다.
s_list	서버 그룹 일련번호들의 배열에 대한 포인터로 nf_entry와 f_list는 다른 용도로 사용되며 각각 0과 NULL값을 가진다.
f_list	실패한 서버 그룹 일련번호의 배열이다.

- 오류

  tpgetsvglist()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생한 경우로 메모리 할당 실패 등이 여기에 해당된다.

3.1.78. tpgetsvglist_bynode

지정한 서비스를 제공하는 서버그룹 중에 지정한 노드명에 해당하는 서버그룹 목록을 가져온다. tpgetsvglist()와 동일하지만 노드명 인자에 따라 서버그룹 목록의 내용이 달라진다.

- 프로토타입

```
#include <tmaxapi.h>
struct svglist *tpgetsvglist_bynode(char *nodename, char *svc, long flags)
```

- 파라미터

파라미터	설명
nodename	svc에 지정한 서비스를 제공하는 노드명을 입력한다. NULL 을 입력할 경우 tpgetsvglist()와 동일한 동작을 수행한다. 만약 잘못된 노드명을 입력하는 경우 TPEINVAL 또는 TPENOENT 오류가 발생한다.
svc	서비스명을 입력한다.
flags	현재는 사용되지 않는다.

- 반환값

반환값	설명
NULL 이 아닌 값	함수 호출에 성공한 경우이다. 지정한 노드명에 해당하는 서그룹 목록이 리턴된다.
NULL	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

에러가 발생하면 NULL을 반환하고 tperrno에 상황에 해당하는 값이 설정된다. 그렇지 않으면 서비스 호출에 실패한 서버 그룹 리스트를 struct svglist에 설정하여 반환한다.

반환한 struct svglist의 내용은 다음과 같다.

```
struct svglist {
    int ns_entry;
    int nf_entry;
    int *s_list;
    int *f_list;
};
```

멤버	설명
ns_entry	해당 서비스가 속하는 서버 그룹의 수이다.
nf_entry	실패한 서버 그룹의 수이다.
s_list	서버 그룹 일련번호들의 배열에 대한 포인터로 nf_entry와 f_list는 다른 용도로 사용되며 각각 0과 NULL값을 가진다.
f_list	실패한 서버 그룹 일련번호의 배열이다.

- 오류

tpgetsvglist_bynode()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생한 경우로 메모리 할당 실패 등이 여기에 해당된다.
[TPEPROTO]	클라이언트의 경우 tpstart()가 수행되지 않은 상태에서 함수를 호출한 경우이다.
[TPEINVAL]	노드명이 잘못된 경우이다.

에러 코드	설명
[TPENOENT]	서비스명과 노드명이 실제 Tmax 시스템에 존재하지 않는다.

- 예제

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

SVGCALL(TPSVCINFO *msg)
{
        char *svcname = msg->data;

        struct nodelist *node = tpgetnodelist();
        if (node == NULL)
           ...error process

        for (i = 0; i < node->count; i++) {
                printf("[%d] %16s : %x:%p\n", i, node->nodelist[i].nodename,
                        node->nodelist[i].ipaddr, node->nodelist[i].port);
        }

        ...

        struct svglist *svg = tpgetsvglist_bynode(node->nodelist[1].nodename, svcname, 0);
        if (svg == NULL)
           ...error process

        for (i = 0; i < svg->ns_entry; i++) {
                printf("[%d] %d(%x)\n", i, svg->s_list[i], svg->s_list[i]);
        }

        ...

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

- 관련 함수

  tpgetnodelist()

3.1.79. tpgprio

서버와 클라이언트에서 요청받은 서비스의 우선순위를 보여주는 함수이다. 클라이언트의 tpgprio()는 마지막으로 송신한 요청의 우선순위를 반환하고 서버의 tpgprio()는 마지막으로 수신된 요청의 우선순위를 반환한다.

tpgprio()는 **tpcall()**이 호출된 후에 호출되며 수신한 요청의 우선순위를 반환한다. 또한 요청받은 서비스의 우선순위를 확인하기 위해 서비스 루틴 내에서 호출될 수 있다. 우선순위는 요청된 메시지의 서비스 우선순위를 나타내는 값으로 0에서 100까지이며 각 서비스의 기본값은 50이다. **tpsprio()**와 **tpgprio()**를 사용하여 요청된 메시지의 서비스 순위를 관리할 수 있다.

- 프로토타입

```
#include <tuxfatmi.h>
int tpgprio (void)
```

- 반환값

반환값	설명
우선순위	함수 호출에 성공한 경우로 서비스 우선순위를 반환한다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpgprio()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPENOENT]	tpgprio()가 호출된 후에도 수신된 요청이 없거나 요청이 없는 대화형 서비스 상태에서 호출되었다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

SERVICE(TPSVCINFO *msg)
{
    int ret, prio;
    prio = tpgprio();
    if (prio==-1) { error processing }
    if (prio > 50) tpforward("SERVICE1", msg->data, msg->len, 0);
    else  tpforward("SERVICE2", msg->data, msg->len, 0);
}
```

- 관련함수

  tpacall(), tpcall(), tpsprio()

3.1.80. tpmcall

서버와 클라이언트에서 COUSIN으로 묶인 모든 서버 그룹 서버의 서비스를 호출하는 함수이다. COUSIN으로 묶인 여러 개의 서버 그룹이 있는 Tmax 환경에서는 **tpcall()**을 하는 경우 Tmax의 라우팅 방식에 따라 하나의 서버 그룹에 속한 서비스를 호출한다. 그러나 tpmcall()을 사용하면 COUSIN으로 묶인 서버 그룹의 해당 서비스를 모두 호출하므로 클라이언트 입장에서는 multicasting을 수행하게 된다.

tpmcall()을 사용하면 내부적으로 COUSIN에 속한 서버 그룹의 서비스로 tpacall(..., TPNORPLY)를 수행한 것과 동일한 방식으로 동작한다. tpmcall()은 트랜잭션의 범주에 들지 않는다.

- 프로토타입

```
#include <tmaxapi.h>
struct svglist *tpmcall(char *qname, char *svc, char *data, long len, long flags)
```

- 파라미터

파라미터	설명
qname	qname을 설정하면 RQ를 거쳐 서비스를 호출한다. 현재 버전에서는 지원하지 않는다.
svc, data, len	tpcall()에서 사용되는 첫 번째 ~ 세 번째 파라미터와 동일하다.
flags	현재 버전에서는 지원하지 않으나 TPNOFLAGS로 설정한다.

- 반환값

  에러가 발생하면 NULL을 반환하고 tperrno에 상황에 해당하는 값이 설정된다. 그렇지 않으면 서비스 호출에 실패한 서버 그룹 리스트를 struct svglist 에 설정하여 반환한다.

  반환한 struct svglist의 내용은 다음과 같다.

```
struct svglist {
    int ns_entry;
    int nf_entry;
    int *s_list;
    int *f_list;
};
```

멤버	설명
ns_entry	tpmcall()에 성공한 서버 그룹의 수이다.
nf_entry	tpmcall()에 실패한 서버 그룹의 수이다.
s_list	tpmcall()에 성공한 서버 그룹 일련번호의 배열이다. nf_entry와 f_list는 다른 용도로 사용되며 각각 0과 NULL 값을 가진다.
f_list	tpmcall()에 실패한 서버 그룹 일련번호의 배열이다.

- 오류

  tpmcall()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 svc가 NULL이거나 data가 tpalloc()으로 할당되지 않은 버퍼를 가리키거나, flags가 유효하지 않는 경우에 발생한다.
[TPENOENT]	설정한 qname이 Tmax에 등록되지 않았다.

에러 코드	설명
[TPEITYPE]	data의 유형 및 하위 유형이 svc가 지원하지 않는 유형이다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다. 메모리 할당 에러 등이 여기에 해당된다.

- 예제

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

main(int argc, char *argv[])
{
    char    *sndbuf;
    int     ret;
    …

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

    sndbuf = (char *)tpalloc(“STRING”, NULL, 0);
    if (sndbuf == NULL){ error processing }

    ret = tpmcall(NULL, “TPMCALL”, sndbuf, 0, TPNOFLAGS);
    if (ret == NULL){  error processing  }
    …
}
```

- 관련함수

  tpcall(), tpgetsvglist(), tpmcallx()

3.1.81. tpmcallx

tpmcallx()는 기존 tpmcall()의 확장 기능 제공을 목적으로 하는 함수로 기존과 달리 COUSIN 서버 그룹의 서비스들로부터 모두 응답을 받을 때까지 기다린다. 또한 응답 성공 여부가 저장되는 svglist 구조체에 r_list가 추가되었으며 flags에 몇 가지 항목이 추가되었다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
struct svglistx* tpmcallx(char *svc, char *data, long len, long flags)
```

- 파라미터

파라미터	설명
svc	호출되는 서비스 이름으로 Tmax 응용 서버 프로그램에서 제공되고 있는 것이어야 한다.

파라미터	설명
data	서비스 요청의 데이터에 대한 포인터이다. 반드시 이전에 tpalloc()에 의해 할당된 버퍼이어야 한다. data의 유형(type)과 하위 유형(subtype)은 svc가 지원하는 유형이어야 한다.
len	송신할 데이터의 길이를 지정한다. • data가 가리키는 버퍼가 특별한 길이 명시가 필요없는 버퍼 유형(STRING, STRUCT, X_COMMON, X_C_TYPE)인 경우 len은 무시되고 기본으로 0을 사용한다. • data가 가리키는 버퍼가 길이 명시가 반드시 필요한 버퍼 유형(X_OCTET, CARRAY, MULTI STRUCTURE)인 경우 len은 0이 될 수 없다. • data가 NULL인 경우 len은 무시된다.
flags	호출할 때 사용되는 옵션으로 어떤 방식으로 통신할 것인지를 지정한다. flags로 사용 가능한 값은 아래와 같다. • TPNOREPLY 송신만 수행한다. 이 flags를 설정할 경우 기존의 tpmcall()과 유사하게 동작한다. • TPBLOCK 송신이 CLH까지 성공적으로 전달되었는지를 확인한다. • TPNOTIME 수신할 때 블록 타임아웃 시간을 무시하고 무한으로 대기한다.

- 반환값

에러가 발생하면 NULL을 반환하고 tperrno에 상황에 해당하는 값이 설정된다. 그렇지 않으면 서비스 호출에 실패한 서버 그룹 리스트를 struct svglistx에 설정하여 반환한다.

반환한 struct svglistx의 내용은 다음과 같다.

```
struct svglistx {
     int ns_entry; /* number of entries of s_list */
     int nf_entry; /* number of entries of f_list */
     int nr_entry; /* number of entries of r_list */
     int *s_list; /* list of server group numbers */
     int *f_list; /* list of tperrno of each server group */
     int *r_list; /* list of tpurcode of each server group */
};
```

멤버	설명
ns_entry	tpmcallx()에 성공한 서버 그룹의 수이다.
nf_entry	tpmcallx()에 실패한 서버 그룹의 수이다.

멤버	설명
nr_entry	r_list가 설정된 서버 그룹의 수이다.
s_list	tpmcallx()에 성공한 서버 그룹 일련번호의 배열이다.
f_list	tpmcallx()에 실패한 서버 그룹 일련번호의 배열이다.
r_list	tpurcode가 설정된 서버 그룹 일련번호의 배열이다.

- 오류

  tpmcallx()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 svc가 NULL이거나 data가 tpalloc()으로 할당되지 않은 버퍼를 가리키거나, flags가 유효하지 않다.
[TPENOENT]	설정한 qname이 Tmax에 등록되지 않았다.
[TPEITYPE]	data의 유형 및 하위 유형이 svc가 지원하지 않는 유형이다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템의 에러가 발생하였다. 메모리 할당 에러 등이 여기에 해당된다.

- 예제

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <usrinc/atmi.h>
#include <usrinc/tmaxapi.h>
main(int argc, char *argv[])
{
     char *sndbuf, *rcvbuf;
     long rcvlen, sndlen;
     struct svglistx svglist;
     struct svglistx *psvglist;
     int ret, i;

     psvglist = &svglist;
     if (argc != 2) {
          printf("Usage: toupper string\n");
          exit(1);
     }

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

     if (tpstart((TPSTART_T *)NULL) == -1){
          printf("tpstart failed[%s]\n",tpstrerror(tperrno));
          exit(1);
     }
```

```
     if ((sndbuf = (char *)tpalloc("STRING", NULL, 0)) == NULL)
     {
          printf("sendbuf alloc failed !\n");
          tpend();
          exit(1);
     }

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

     strcpy(sndbuf, argv[1]);
     psvglist = tpmcallx("TOUPPER", sndbuf, 0, TPBLOCK);
     if(psvglist == NULL)
     {
          printf("tpmcall is failed[%s]\n",
          tpstrerror(tperrno));
          tpfree((char *)sndbuf);
          tpfree((char *)rcvbuf);
          tpend();
          exit(1);
     }

     printf("send data: %s\n", sndbuf);
     printf("ns_entry = %d\n", psvglist->ns_entry);
     printf("nf_entry = %d\n", psvglist->nf_entry);
     printf("nr_entry = %d\n", psvglist->nr_entry);

     for(i=0; i<psvglist->ns_entry; i++)
          printf("psvglist->s_list[%d] = %d\n", i, psvglist->s_list[i]);
     for(i=0; i<psvglist->nf_entry; i++)
          printf("psvglist->f_list[%d] = %d\n", i, psvglist->f_list[i]);
     for(i=0; i<psvglist->nr_entry; i++)
          printf("psvglist->r_list[%d] = %d\n", i, psvglist->r_list[i]);

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

- 관련함수

  tpcall(), tpgetsvglist(), tpmcall()

3.1.82. tpnotify

서버에서 지정된 클라이언트에 비요청 메시지를 송신하는 함수이다.

- 프로토타입

```
#include <atmi.h>
int tpnotify(CLIENTID *id, char *data, long len, long flags);
```

- 파라미터

파라미터	설명
id	TPSVCINFO 구조체에 저장되어 있는 클라이언트 ID에 대한 포인터이다.
data	반드시 이전에 tpalloc()에 의해 할당된 버퍼이어야 한다.
len	송신하는 버퍼의 길이이다. • data의 길이 명시가 필요없는 버퍼를 가리키는 경우 len은 무시되고 보통 0이 사용된다. • data의 길이 명시가 반드시 필요한 버퍼를 가리키는 경우 len은 0이 될 수 없다. • data가 NULL인 경우 len은 무시된다.
flags	flags로 사용 가능한 값은 다음과 같다. • TPACK 반드시 이전에 tpalloc()에 의해 할당된 버퍼이어야 한다. • TPNOBLOCK 데이터가 도착할 때까지 기다리지 않는다. 수신 가능한 데이터가 있으면 이를 반환한다. TPNOBLOCK flags가 지정되지 않고 수신 가능한 데이터가 없으면 호출자는 데이터가 도착할 때까지 기다린다. • TPNOTIME TPNOTIME flags는 함수 호출자가 블록 타임아웃을 무시하고 응답이 수신될 때까지 무한정 대기하는겠다는 것을 의미한다. 그러나 트랜잭션 타임아웃 내에서 WinTmaxSend()을 한 경우에는 트랜잭션 타임아웃이 적용된다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpnotify()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다.

에러 코드	설명
[TPENOENT]	대상 클라이언트가 존재하지 않거나, 비요청 메시지 핸들러가 설정되지 않았을 경우 발생한다.
[TPETIME]	타임아웃이 발생하였다. 함수 호출자가 트랜잭션 모드에 있다면 트랜잭션 타임아웃이 발생하고 그 트랜잭션은 rollback된다. 트랜잭션 모드가 아니고 TPNOTIME과 TPNOBLOCK 어느 것도 지정되지 않았다면 블록 타임아웃이 발생한다.
[TPEPROTO]	tpnotify()가 부적절한 상황에서 호출되었다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

TOUPPER(TPSVCINFO* msg) {
    int i, n;
    char    rdata[100];
    FBUF    *stdata;

    stdata=(FBUF *)msg->data;
    memset(rdata, 0x00, sizeof(rdata));
    if (fbget(stdata, INPUT, rdata, 0) == -1){
    printf("fbget failed errno = %d\n", fberror);
    }

    for (i = 0; i < msg->len; i++) {
        rdata[i] = toupper(rdata[i]);
    }
    if (fbput(stdata, OUTPUT, rdata, 0) == -1)
        printf("fbput failed\n");

    i = tpnotify(&(msg->cltid), (char*)stdata, 0, TPACK);
    printf("i= %d\n", i);
    tpreturn(TPSUCCESS, 0, (char *)stdata, 0, 0);
}
```

3.1.83. tppost

서버와 클라이언트에서 특정 사건을 발생시키고 메시지를 전달하는 함수이다. tppost()는 이름이 eventname인 사건에 **tpsubscribe()**로 등록된 모든 클라이언트와 서버 프로세스에게 사건의 발생을 알리고 필요할 경우 메시지를 전달한다.

- 프로토타입

```
# include <tmaxapi.h>
```

```
int tppost(char *eventname, char *data, long len, long flags)
```

- 파라미터

파라미터	설명
eventname	NULL로 끝나는 63자 이내의 문자열로 발생할 사건의 이름을 의미한다. 와일드 카드 문자나 partial-matching 등은 지원되지 않는다.
data	보낼 메시지에 대한 포인터로 tpalloc()에 의해 할당된 버퍼이어야 한다.
len	송신하는 버퍼의 길이이다. • data의 길이 명시가 필요없는 버퍼를 가리키는 경우 len은 무시되고 보통 0이 사용된다. • data의 길이 명시가 반드시 필요한 버퍼를 가리키는 경우 len은 0이 될 수 없다. • data가 NULL인 경우 len은 무시된다.
flags	현재 TPNOTIME만 의미가 있다.

- 반환값

반환값	설명
양수	함수 호출에 성공한 경우이다.
음수	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tppost()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생한 경우로 클라이언트의 경우 네트워크 에러가 가장 빈번하다.
[TPEOS]	OS 레벨의 에러 발생한 경우로 메모리 할당과 같은 문제를 포함한다.
[TPEINVAL]	잘못된 파라미터를 사용하는 경우이다.
[TPENOENT]	tpsubscribe()의 경우 TPEVCTL 구조체의 qname이나 svc에 해당하는 RQ나 서버가 존재하지 않는 경우이다.
[TPEPROTO]	클라이언트에서 tpsubscribe()를 수행하는 경우 ctl이 NULL이 아닌 경우나 서버에서 수행하는 경우 ctl이 NULL인 경우 발생한다.
[TPETIME]	타임아웃이 발생한다.

- 관련함수

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

3.1.84. tpputenv

서버와 클라이언트에서 환경변수 값을 재설정하는 함수로 string으로 입력된 "이름=값"을 환경변수에 적용한다. 기존의 환경변수가 존재한다면 수정하고 환경변수가 존재하지 않는다면 새로 추가한다.

서버에서 수행 가능한 **putenv()**와 동일한 기능을 수행하는 함수이다. 클라이언트에서는 **autoexec.bat** 파일에 적용된 값을 수정하거나 새로 추가하고 서버에서는 각각의 셸 환경 파일에 적용된 값을 수정하거나 존재하는 값이 없다면 추가한다.

- 프로토타입

```
#include <tmaxapi.h>
int tpputenv(char *string)
```

- 파라미터

파라미터	설명
string	환경변수에 설정할 값이다. ("이름=값")

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
음수	함수 호출에 실패한 경우이다. 해당 환경변수를 적용할 수 있는 충분한 메모리를 확보하지 못한 경우에 발생한다.

- 예제

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

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

    SDLFI=tpgetenv("SDLFILE");
    if (SDLFI=NULL) ret=tpputenv("SDLFILE=/tmax/sample/sdl/test.sdl");
    if (ret<0) { error processing }

    SDLFI=tpgetenv("SDLFILE");
    printf ("tmax sdlfile : %s\n",SDLFI);
}
```

- 관련 함수

  tpgetenv()

3.1.85. tpqstat

서버와 클라이언트에서 사용되는 함수로 RQ에 저장된 데이터의 통계를 요청한다. RQ는 내부적으로 _fail queue, _request queue, _reply queue의 3부분으로 구성되어 있다. flags 값을 이용하여 각 큐에 저장된 데이터 통계를 구할 수 있다.

- 프로토타입

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

- 파라미터

파라미터	설명
qname	Tmax 환경 파일에 등록된 RQ 이름을 나타낸다.
flags	대상 데이터의 type을 설정한다. flags로 사용 가능한 값은 다음과 같다. • 0(TMAX_ANY_QUEUE) _fail queue, _request queue, _reply queue의 데이터 통계를 낼 때 사용한다. • 1(TMAX_FAIL_QUEUE) : _fail queue의 데이터 통계를 낼 때 사용한다. • 2(TMAX_REQ_QUEUE) : _request queue의 데이터 통계를 낼 때 사용한다. • 3(TMAX_RPLY_QUEUE) : _reply queue의 데이터 통계를 낼 때 사용한다.

- 반환값

반환값	설명
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpqstat()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 qname이 NULL이거나 qname에 해당하는 큐가 없거나 flags가 유효하지 않은 경우에 발생한다.
[TPEPROTO]	tpqstat()가 부적절한 상황에서 호출되었다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

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

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

    buf = (char *)tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
    strcpy(buf, argv[1]);

    data process…

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

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

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

    }
    printf("\n");
    tpfree((char *)buf);
    tpend();
}
```

- 관련함수

  tpenq(), tpdeq()

3.1.86. tpqsvcstat

서버와 클라이언트에서 사용되는 함수로 RQ에 저장된 데이터 중 지정한 서비스에 대한 통계를 요청한다. RQ는 내부적으로 _fail queue, _request queue, _ reply queue의 3부분으로 구성되어 있다.

flags값을 이용하여 각 큐에 저장된 데이터 통계를 구할 수 있다.

- 프로토타입

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

- 파라미터

파라미터	설명
qname	Tmax 환경 파일에 등록된 RQ 이름을 설정한다.
svc	통계 정보를 얻을 서비스 이름으로 NULL인 경우 **tpqstat()**과 동일한 의미를 갖는다.
flags	데이터 통계의 유형을 설정한다. 다음은 flags에 설정 가능한 값에 대한 설명이다. • 0(TMAX_ANY_QUEUE) : _fail queue, _request queue, _reply queue의 데이터 통계를 낼 때 사용한다. • 1(TMAX_FAIL_QUEUE) : _fail queue의 데이터 통계를 낼 때 사용한다. • 2(TMAX_REQ_QUEUE) : _request queue의 데이터 통계를 낼 때 사용한다. • 3(TMAX_RPLY_QUEUE) : _reply queue의 데이터 통계를 낼 때 사용한다.

- 반환값

반환값	설명
-1 이외의 값	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpqscvstat()가 정상 처리되지 않을 경우 tperrno에 아래 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	인수가 유효하지 않다. 예를 들어 qname이 NULL이거나, qname에 해당하는 큐가 없거나 type이 유효하지 않은 경우이다.
[TPEPROTO]	tpqscvstat()가 부적절한 상황에서 호출되었다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 관련 함수

  tpenq(), tpdeq(), tpqstat()

3.1.87. tprealloc

서버와 클라이언트에서 유형 버퍼의 재할당하는 함수로 ptr이 가리키는 버퍼의 크기를 Byte로 재할당하고, 버퍼가 변경되었을 경우 새 버퍼에 대한 포인터를 반환한다.

tpalloc()과 동일하게 버퍼의 크기는 기본 사이즈(1024Byte) 이상이다. 버퍼의 유형은 재할당된 후에도 동일하게 유지된다. 함수가 성공적으로 반환되면, 반환된 포인터가 버퍼를 참조하기 위해서 사용되고 ptr은 더 이상 사용될 수 없다. 재할당된 버퍼의 크기가 축소되는 경우 원래 ptr의 내용은 보장할 수 없다.

일부 버퍼 유형들은 사용되기 전에 초기화될 필요가 있다. tprealloc()은 버퍼 재할당 이후 다시 초기화하여 반환한다. 그렇기 때문에 호출자에게 반환된 버퍼는 즉시 사용 가능하다. 버퍼가 재초기화에 실패하면 tprealloc()은 NULL을 반환하고 ptr이 지시하는 버퍼의 내용은 유효하지 않다.

- 프로토타입

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

- 파라미터

파라미터	설명
ptr	할당할 버퍼에 대한 포인터이다.
size	할당할 버퍼의 크기이다.

- 반환값

반환값	설명
버퍼 포인터	함수 호출에 성공한 경우이다. 적절한 유형 버퍼에 대한 포인터를 반환한다.
NULL	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tprealloc()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 ptr이 가리키는 버퍼가 tpalloc()으로 할당된 것이 아닌 경우이다.
[TPEPROTO]	tprealloc()이 부적절한 상황에서 호출되었다.
[TPENOENT]	ptr이 가리키는 버퍼가 tpalloc()으로 할당되지 않은 버퍼이다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

```
#include <stdio.h>
#include <usrinc/atmi.h>
void main(int argc, char *argv[])
{
    char *buf;
    buf=tpalloc("STRING",NULL,10);
    if (buf==NULL) { error processing }
    buf=tprealloc(buf,20);  /* ok */
    if (buf==NULL) { error processing }

    buf="test";
    buf=tprealloc(buf,30);  /*error : TPEINVAL */
    if (buf==NULL) { error processing }
}
```

- 관련함수

  tpalloc(), tpfree(), tptypes()

C 라이브러리의 **malloc()**, **realloc()** 또는 **free()**와 함께 사용될 수 없다. 예를 들어 tprealloc()으로 할당된 버퍼를 free()로 해제할 수 없다.

3.1.88. tprecv

서버와 클라이언트에서 대화형 통신을 하는 경우 메시지를 수신하는 함수로 대화형 통신으로 연결된 상대방 프로그램으로부터 송신된 데이터를 수신하기 위해 사용된다. tprecv()는 클라이언트나 서버 중 통신 제어권을 갖지 않은 프로그램에서만 사용될 수 있다.

- 프로토타입

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

- 파라미터

파라미터	설명
cd	데이터를 수신할 연결을 지정하는 것으로 **tpconnect()**나 TPSVCINFO 매개변수 중 하나로부터 반환된 구별자(descriptor)이다.
data	tpalloc()에 의해 이전에 할당된 버퍼에 대한 포인터 주소이다. 함수가 성공적으로 반환되면 *data는 수신된 데이터를 가리킨다.
len	데이터의 길이를 나타낸다. len이 호출 전 버퍼의 총 크기보다 더 크다면 버퍼의 새로운 크기는 len이 된다. len이 0이면 어떤 데이터도 수신되지 않았고, *data나 *data가 가리키는 버퍼 모두 아무런 변화없다. *data나 len이 NULL이면 data는 에러이다.

파라미터	설명
flags	flags로 사용 가능한 값은 다음과 같다. • TPNOCHANGE 수신된 응답 버퍼와 *data가 가리키는 버퍼의 유형이 다르다면 *data의 버퍼 유형은 수신자가 인식할 수 있는 한도 내에서 수신된 응답 버퍼의 유형으로 변경된다. 그러나 이 flags가 설정되었다면 *data가 가리키는 버퍼의 유형은 변경되지 못한다. 수신된 응답 버퍼의 유형 및 하위 유형은 *data가 가리키는 버퍼의 유형 및 하위 유형과 반드시 일치해야 한다. • TPNOBLOCK 데이터가 도착할 때까지 기다리지 않는다. 수신 가능한 데이터가 있으면 이를 반환한다. TPNOBLOCK flags가 지정되지 않고 수신 가능한 데이터가 없으면 호출자는 데이터가 도착할 때까지 대기한다. • TPNOTIME 함수 호출자가 블록 타임아웃을 무시하고 응답이 수신될 때까지 무한정 대기한다는 것을 의미한다. 그러나 트랜잭션 타임아웃 내에서 tprecv()를 한 경우에는 여전히 트랜잭션 타임아웃이 적용된다. • TPSIGRSTRT 시그널(signal) 인터럽트를 수용하는 경우 사용한다. 내부에서 시그널 인터럽트가 발생하여 시스템 함수 호출이 방해될 때 TPSIGRSTRT flag가 설정되어 있으면 BLOCKTIME 시간동안 계속 응답을 기다린다. TPSIGRSTRT flags가 설정되지 않은 경우 시그널 인터럽트가 발생했다면, 함수는 실패하고 tperrno에 TPGOTSIG가 설정된다.

파라미터	설명
revent	다음은 revent에 반환되는 이벤트 유형에 대한 설명이다. • TPEV_DISCONIMM 대화 종속자에게 수신되는 이 이벤트는 대화 시작자가 tpdiscon()으로 대화형 연결을 강제로 종료했음을 의미한다. 또한 이 이벤트는 통신 에러, 예를 들어 서버, 노드, 네트워크 장애 등으로 인하여 연결이 끊어졌을 때 대화 시작자나 종속자에게 반환될 수 있다. 이것은 강제적인 연결 해제 통보이기 때문에 전달 중인 데이터는 분실될 수도 있다. 2개의 프로그램이 동일한 트랜잭션에 참여하고 있었다면 그 트랜잭션은 rollback된다. 대화형 통신에 사용되었던 구별자(cd)는 더 이상 유효하지 않다. • TPEV_SENDONLY 연결된 상대방 프로그램 측에서 통신 제어권을 포기하였다. TPEV_SENDONLY 이벤트의 수신자는 데이터를 송신할 수는 있지만 수신자가 제어권을 넘길 때까지는 어떤 데이터도 수신할 수 없다. • TPEV_SVCERR 대화 시작자에게 수신되는 이 이벤트는 대화 종속자가 tpreturn() 수행 중에 에러가 발생하였음을 알린다. 예를 들어 tpreturn()에 잘못된 파라미터들이 전달되었거나, 서비스가 다른 종속자들과 연결을 유지하고 있는 동안에 tpreturn()이 호출되었을 수 있다. 이 경우 반환 코드나 데이터 일부는 사용이 불가능하다. 대화형 연결이 종료되고 cd는 더 이상 유효하지 않다. 만약 이 이벤트가 수신자의 트랜잭션 과정에서 발생했다면, 그 트랜잭션은 rollback된다. • TPEV_SVCFAIL 대화 시작자에게 수신되는 이 이벤트는 상대편인 대화 종속자 서비스가 애플리케이션에서 실패로 서비스를 종료하였음을 알린다. TPFAIL로 tpreturn()을 호출하였다. 대화 종속자 서비스가 tpreturn()을 호출했을 때 통신 제어권을 가지고 있었다면, 서비스는 연결된 상대방에게 데이터를 전달할 수 있다. 서비스가 종료하면서 서버 프로세스는 대화형 연결을 끊는다. 그러므로 cd는 더 이상 유효하지 않다. 만약 이 이벤트가 수신자의 트랜잭션 과정에서 발생하였다면 그 트랜잭션은 rollback된다. • TPEV_SVCSUCC 대화 시작자에게 수신되는 이 이벤트는 상대편인 대화 종속자 서비스가 성공적으로 종료하였음을 알린다. TPSUCCESS로 tpreturn()을 호출하였다.

• 반환값

revent 값이 TREV_SVCSUCC이거나 TREV_SVCFAIL인 경우 tpreturn()으로 전달되는 tpurcode 전역 변수는 애플리케이션에서 정의한 값을 갖게 된다. 그렇지 않으면 -1을 반환하고, tperrno에 에러 상황에 해당하는 값이 설정된다. 에러 없이 이벤트가 존재한다면 tprecv()는 -1을 반환하고, tperrno는 [TPEEVENT]가 된다.

• 오류

tprecv()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEBADDESC]	구별자(cd)가 유효하지 않다.
[TPEBLOCK]	TPNOBLOCK가 설정된 상태에서 블로킹 상황이 발생하였다.
[TPEEVENT]	이벤트가 발생하였고 revent로 이벤트 유형을 알 수 있다.
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 *data가 가리키는 버퍼가 tpalloc()으로 되지 않았거나 flags가 유효하지 않은 경우이다.
[TPEOS]	운영 시스템에 에러가 발생하였다.
[TPEOTYPE]	입력된 버퍼의 유형이나 하위 유형이 호출자가 알지 못하는 것이거나, flags로 TPNOCHANGE가 설정되었는데 *data의 유형 및 하위 유형이 입력되는 버퍼의 유형 및 하위 유형과 일치하지 않는다. 이런 경우 *data의 내용과 *len은 모두 아무 변화 없다. 대화형 통신이 트랜잭션의 한 부분이라면, 그 트랜잭션은 응답이 무시되었기 때문에 rollback된다. 에러가 발생하면, cd에 대한 이벤트는 무시되고 대화형 통신 상태는 보장할 수 없다. 따라서 호출자는 대화형 통신을 종료해야 한다.
[TPEPROTO]	tprecv()가 부적절한 상황에서 호출되었다. 예를 들어 송신자 모드에서 사용한 경우에 발생한다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPETIME]	타임아웃이 발생한 경우로 함수 호출자가 트랜잭션 모드에 있다면, 트랜잭션 타임아웃이 발생하였고 그 트랜잭션은 rollback된다. 트랜잭션 모드가 아니고 TPNOTIME과 TPNOBLOCK 어느 것도 지정되지 않았다면, 블록 타임아웃이 발생하였다. 이 두 경우에 *data의 내용과 len은 변경되지 않는다. 트랜잭션 타임아웃이 발생하였다면, 트랜잭션이 rollback될 때까지 대화형 통신으로 메시지를 송신하거나 수신하는 일 또는 새로운 연결을 시작하는 일은 모두 [TPETIME] 에러로 실패하게 된다.
[TPGOTSIG]	TPSIGRSTRT가 설정되지 않은 상태에서 시그널이 수신되었다.

- 예제

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

main(int argc,char* argv[])
{
    int ret, cd;
    struct dat *buf;
    long revent, len;
    if (argc!=3) {error processing }
    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf=(struct dat *)tpalloc(“STRUCT”, “dat”, 0);
    if (buf==NULL) { error processing }
    strcpy(buf->sdata, argv[1]);
    data process….
```

```
    cd=tpconnect("SERVICE", buf, 0, TPSENDONLY);
    if (cd==-1) { error processing }
    strcpy(buf->sdata, argv[2]);
    data process….

    ret=tpsend(cd, buf, 0,TPRECVONLY,&revent);
    if (ret==-1) { error processing }
    ret=tprecv(cd,(char**)&buf,&len,TPNOTIME,&revent);
    if (ret==-1) { error processing }
    data process....

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

- 관련함수

  tpalloc(), tpconnect(), tpdiscon(), tpsend()

3.1.89. tprecvfromcli

서버에서 클라이언트로 메시지를 요청하고 받는 함수이다. 이 기능은 tpcall과 같이 블로킹으로 동작하며, clid는 tpgetclid로 얻어온 clid를 사용하며, 다른 인자는 tpcall과 동일하게 사용된다.

- 프로토타입

```
# include <tmaxapi.h>
int tprecvfromcli(int clid, char *idata, long ilenl, char **odata, long *olen, long flagsl)
```

- 파라미터

파라미터	설명
clid	tpgetclid로 얻어온 clid 값이다.
idata	서비스 요청의 데이터에 대한 포인터이며, tpcall의 idata와 동일하게 동작한다.
ilenl	송신할 데이터의 길이이며, tpcall의 ilenl과 동일하게 동작한다.
odata	*odata는 수신될 응답 버퍼에 대한 포인터로 버퍼는 *olen으로 반환된 길이 만큼의 응답이 수신되며, tpcall의 odata와 동일하게 동작한다.
olen	*odata에 반환될 응답에 대한 길이이며, tpcall의 olen과 동일하게 동작한다.
flags	호출할 때 사용되는 옵션이다. TPNOTIME이나 TPNOBLOCK, 혹은 둘 다 사용할 경우 TPRECVFROMCLIFLAG flag를 사용 가능하다. 이 이외의 flag는 지원하지 않는다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tprecvfromcli()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	아래와 같은 상황에 발생한다. • olen가 NULL인 경우이다. • odata가 NULL인 경우이다. • flags 파라미터에 TPRECVFROMCLIFLAG가 아닌 flag가 들어온 경우이다. • clid 값이 올바르지 않은 경우이다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOTYPE]	odate에 할당된 버퍼의 header에 있는 magic number가 현재 magic number와 일치하지 않는 경우이다.
[TPECLOSE]	클라이언트가 종료된 상태이면 리턴되는 에러이다.
[TPESVRDOWN]	clid 값이 다른 노드의 클라이언트이고, 또 해당 노드가 다운된 상태일 경우 리턴되는 에러이다.

3.1.90. tpreissue

서버와 클라이언트에서 사용되는 함수로 해당 RQ의 Fail 큐에 쌓인 요청 데이터를 다시 Request 큐에 넣어준다. **tpenq()** 등으로 RQ를 통한 서비스 수행 중 네트워크 불안정이나 기타 서버 측의 오류로 서비스 수행을 실패하여 Fail 큐에 쌓인 클라이언트의 서비스 요청 데이터를 다시 Request 큐에 넣어주는 함수이다. 저장된 데이터를 해당 서비스로 전달해서 결과를 Reply 큐에 저장한다.

- 프로토타입

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

- 파라미터

파라미터	설명
qname	Tmax 시스템에 등록된 RQ의 이름이다.
filter	현재 지원하지 않으며 NULL로 설정해야 한다.
flags	현재 버전에서는 지원하지 않으나 TPNOFLAGS으로 설정한다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpreissue()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 qname이 NULL이거나 현재 지원하지 않는 필터나 flags에 위에서 언급한 값 이외의 값을 설정했을 경우 발생한다.
[TPENOENT]	qname에 해당하는 RQ가 존재하지 않는다.
[TPESYSTEM]	Tmax 시스템에 에러가 발생한 경우로 네트워크 상태가 불량한 경우 발생한다.

- 관련 함수

  tpenq(), tpdeq()

3.1.91. tpremoteconnect

리모트와 TCP로 연결하는 함수로서 sec가 주어지면 해당 시간 동안 연결을 시도한다. 시간 초과의 경우 연결 오류가 발생한다.

- 프로토타입

```
#include <ucs.h>
int tpremoteconnect(char *host, int portno, int sec)
```

- 반환값

반환값	설명
0	소켓 번호이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 상황에 해당되는 값이 설정된다.

3.1.92. tpscmt

서버와 클라이언트에서 사용되는 함수로 환경 파일에 설정된 트랜잭션 commit 방식 설정을 변경한다.

- 프로토타입

```
# include <tuxatmi.h>
```

```
int  tpscmt(long  flags)
```

- 파라미터

파라미터	설명
flags	다음 값 중 원하는 값을 설정한다. • 1: 트랜잭션 제어를 TP_CMT_LOGGED로 설정한다. • 2: 트랜잭션 제어를 TP_CMT_COMPLETE로 설정한다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다.

- 예제

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

int main(int argc, char *argv[])
{
    ...
    ret = tpstart((TPSTART_T *)NULL);
    if (ret == -1) {
        error processing
    }
    ...
    ret = tpscmt(TP_CMT_COMPLETED);
    if (ret < 0){
        error processing
    }

    ret = tpbegin();
    if (ret < 0){
        error processing
    }
    ...
   ret= tpcall(“SERVICE1”,(char*)buf,strlen(buf),(char **)&get,&rlen,TPNOFLAGS);
    if (ret == -1)
    {
       error processing
    }
    ...
    ret= tpcall(“SERVICE2",(char*)buf,strlen(buf),(char **)&get,&rlen,TPNOFLAGS);
    if (ret == -1)
    {
        error processing
    }

    ret = tpcommit();
```

```
    if (ret < 0) {
        error processing
    }
    ...
}
```

3.1.93. tpsend

서버와 클라이언트에서 사용하는 함수로 대화형 통신에서 상대방 프로그램에게 데이터를 송신한다. 호출자는 반드시 통신 제어권을 가지고 있어야 한다.

- 프로토타입

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

- 파라미터

파라미터	설명
cd	데이터가 송신되는 연결을 지정하는 것으로 **tpconnect()**나 **TPSVCINFO** 매개변수로부터 반환되는 구별자이다.
data	**tpalloc()**에 의해 할당된 버퍼이다. 애플리케이션 데이터가 아무 것도 송신되지 않을 경우(예를 들어 어떤 데이터도 전달하지 않고 통신 제어권만을 넘기는 경우)에는 data는 NULL이 될 수 있다. data의 유형 및 하위 유형은 연결된 상대방이 인식할 수 있는 유형 및 하위 유형이어야 한다.
len	송신하는 버퍼의 길이로 data가 길이 명시가 필요없는 버퍼를 가리키면 len은 무시되고 보통 0이 사용된다. data가 길이 명시가 반드시 필요한 버퍼를 가리킨다면 len은 0이 될 수 없다.

파라미터	설명
flags	flags로 사용 가능한 값은 다음과 같다. • TPNOBLOCK 내부 버퍼가 전달될 메시지들로 꽉 찬 경우와 같은 블로킹 상황이 발생하면 데이터와 이벤트들은 송신되지 않는다. TPNOBLOCK flags 설정 없이 tpsend()를 호출할 때, 블로킹 상황이 발생하면 호출자는 타임아웃(트랜잭션 또는 블록 타임아웃 중 하나)이 발생하거나 상황이 완화될 때까지 기다린다. • TPNOTIME 함수 호출자가 블록 타임아웃을 무시하고 응답이 수신될 때까지 무한정 대기하는 것을 의미한다. 트랜잭션 타임아웃 내에서 tpsend()를 사용한 경우에는 여전히 트랜잭션 타임아웃이 적용된다. • TPRECVONLY 호출자가 데이터를 송신한 후 통신 제어권을 상대방에게 넘긴다는 의미이다. 호출자는 다시 통신 제어권을 넘겨 받기 전까지 tpsend()를 호출할 수 없다. 대화 상대방 수신자는 tprecv()로 데이터를 수신하면서 통신 제어권을 갖게 됨을 의미하는 TPEV_SENDONLY 이벤트를 수신하게 된다. 수신자 또한 다시 통신 제어권을 상대방에게 넘기기 전까지 tprecv()를 호출할 수 없다. • TPSIGRSTRT 시그널(signal) 인터럽트를 수용하는 경우 사용한다. 내부에서 시그널 인터럽트가 발생하여 시스템 함수 호출이 방해될 때 TPSIGRSTRT flag가 설정되어 있으면 BLOCKTIME 시간동안 계속 응답을 기다린다. TPSIGRSTRT flags가 설정되지 않은 경우 시그널 인터럽트가 발생했다면, 함수는 실패하고 tperrno에 TPGOTSIG가 설정된다.

파라미터	설명
revent	구별자인 cd에 대한 이벤트가 존재한다면 tpsend()는 실패 처리되고 데이터는 송신되지 않는다. 이벤트 유형은 revent로 반환된다. revent에 전달되는 이벤트들은 다음과 같다. • TPEV_DISCONIMM 대화 종속자에게 수신되는 이 이벤트는 대화 시작자가 tpdiscon()을 사용하여 연결을 강제로 종료하였다는 것을 의미한다. 또한 이 이벤트는 통신 에러(예를 들어 서버, 노드, 네트워크 장애 등)로 인하여 연결이 끊어졌을 때에도 반환된다. • TREV_SVCERR 대화 시작자에게 수신되는 이 이벤트는 아래의 TPEV_SVCFAIL 상황 이외의 경우에 대화 종속자가 통신 제어권 없이 tpreturn()을 수행하였음을 알린다. • TREV_SVCFAIL 대화 시작자에게 수신되는 이 이벤트는 대화 종속자가 통신 제어권 없이 tpreturn()을 수행하였으며, 이때 tpreturn()은 아무 데이터 없이 TPFAIL로 호출되었다. rval은 TPFAIL이며 data는 NULL로 수행되었음을 알린다.

• 반환값

반환할 경우 revent가 TREV_SVCFAIL이면, tpurcode 전역변수는 tpreturn()를 호출할 때에 전달된 rcode 값이 된다 .

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

• 오류

tpsend()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 data가 가리키는 버퍼가 tpalloc()으로 할당되지 않았거나 또는 flags가 유효하지 않은 경우에 발생한다.
[TPEBADDESC]	구별자(cd)가 유효하지 않다.
[TPETIME]	타임아웃이 발생한 경우로 함수 호출자가 트랜잭션 모드에 있다면 트랜잭션 타임아웃이 발생하고 그 트랜잭션은 rollback된다. 트랜잭션 모드가 아니고 TPNOTIME과 TPNOBLOCK 어느 것도 지정되지 않았다면 블록 타임아웃이 발생한다. 이 두 경우에 *data의 내용과 len은 변경되지 않는다. 트랜잭션 타임아웃이 발생하였다면 트랜잭션이 rollback될 때까지 대화형 통신으로 메시지를 송신하거나 수신하는 일 또는 새로운 연결을 시작하는 일은 모두 [TPETIME] 에러로 실패하게 된다.

에러 코드	설명
[TPEEVENT]	이벤트가 발생한 경우로 에러가 발생하면 data는 송신되지 않고, 이벤트 유형은 revent로 반환된다.
[TPEBLOCK]	TPNOBLOCK이 설정된 상태에서 블로킹 상황이 발생하였다.
[TPGOTSIG]	TPSIGRSTRT가 설정되지 않은 상태에서 시그널이 수신되었다.
[TPEPROTO]	tpsend()가 부적절한 상황에서 호출되었다. 예를 들어 수신자 모드에서 사용하는 경우에 발생한다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

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

    if (argc!=3) {error processing }
    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }

    buf=(struct dat *)tpalloc(“STRUCT”, “dat”, 0);
    if (buf==NULL) { error processing }
    strcpy(buf->sdata, argv[1]);
    data process….

    cd=tpconnect(“SERVICE”, buf, 0, TPSENDONLY);
    if (cd==-1) { error processing }
    strcpy(buf->sdata, argv[2]);
    data process….

    ret=tpsend(cd, buf, 0,TPRECVONLY,&revent);
    if (ret==-1) { error processing }

    ret=tprecv(&cd,(char**)&buf,&len,TPNOTIME,&revent);
    if (ret==-1) { error processing }
    data process....

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

- 관련함수

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

3.1.94. tpset_timeout

서버와 클라이언트에서 사용되는 함수로 서버에 설정되어 있는 서비스 제한시간인 블록 타임아웃을 설정한다. tpset_timeout()으로 서비스 제한시간을 설정한 경우 설정한 시간 동안 서비스 요청에 대한 응답을 기다린다. 설정한 시간이 지나도록 응답을 수신하지 못하면 타임아웃 에러가 발생하고, 요청한 서비스에 대한 응답을 기다리지 않고 서비스 실패로 반환된다.

tpset_timeout()는 해당 함수가 호출된 이후의 서비스 요청들에 적용된다. 다시 tpset_timeout()이 호출되거나, 클라이언트나 서버 프로세스가 종료될 때까지 함수는 유효하다. tpset_timeout()이 사용되지 않는다면 블록 타임아웃으로 Tmax 환경 파일에 설정된 **BLOCKTIME**이 적용된다.

- 프로토타입

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

- 파라미터

파라미터	설명
sec	블록 타임아웃 시간을 초 단위로 설정한다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpset_timeout()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

void main(int argc, char *argv[])
{
    int ret;
    char *sndbuf, *rcvbuf;
    long sndlen,  rcvlen;
```

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

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

    sndbuf=strlen(sndbuf);
    ret=tpset_timeout(4);
    if (ret==-1) { error processing }

    ret=tpcall("SERVICE", sndbuf, sndlen, &rcvbuf, &rcvlen, TPNOCHANGE);
    if (ret==-1) { error processing }
    data process....

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

3.1.95. tpsetctxt

현재 컨텍스트를 설정하는 함수로 클라이언트 프로그램과 서버 프로그램에서 다음과 같은 작성 방법의 차이가 있다.

- **클라이언트 프로그램**

  함수를 사용하여 하나의 클라이언트가 기존에 생성된 다른 context를 현재 클라이언트에 할당할 수 있고 대부분의 ATMI 함수들은 per-context 기반으로 되어 있다. 클라이언트의 현재 context를 알기 위해서는 tpgetctxt()라는 함수를 사용하여 반환되는 값으로 확인할 수 있다.

  클라이언트는 여러 개의 context를 사용할 수 있지만 해당하는 순간에는 단 하나의 context만을 갖게 된다. 예를 들면 context1에서 tpacall()을 한 경우 다른 context를 사용했다 하더라도 tpgetrply()를 정상적으로 하기 위해서는 tpgetrply()를 하는 시점에서는 context1을 현재 context로 설정해 주어야 한다.

- **서버 프로그램**

  서비스 Thread는 서비스를 처리할 때 context를 할당받아서 사용하게 되지만, 사용자 생성 Thread는 자신만의 context가 존재하지 않는다. 대부분의 ATMI 함수들은 context가 할당되어야만 동작할 수 있다. 따라서 사용자 생성 Thread는 필요한 경우에는 서비스 Thread의 context를 공유해서 사용해야 한다. 사용자 생성 Thread는 tpsetctxt 함수를 사용하여 다른 서비스 Thread의 context를 공유할 수 있다.

  tpsetctxt를 호출한 사용자 생성 Thread는 서비스 Thread와 context 정보를 공유한다. 예를 들어 서비스 Thread에서 tpacall()을 호출하면 그 이후 사용자 생성 Thread에서 tpgetrply()를 통해 요청에 대한 응답을 받는 것이 가능하다.

  tpsetctxt() 함수는 서비스 Thread에서는 사용할 수 없다. 서비스 Thread는 기본적으로 자신의 context를 가지고 있으며, 다른 context로 교체하여 사용할 수 없다. 따라서 서비스 Thread에서 tpsetctxt() 함수를 호출하면 TPEPROTO 에러 코드와 함께 에러를 반환하게 된다.

  사용자 생성 Thread에서 이 함수를 통해 서비스 Thread의 context를 공유한 경우에는 반드시 서비스

Thread가 tpreturn()을 호출하기 전에 먼저 사용자 생성 Thread에서 tpsetctxt(TPNULLCONTEXT)를 호출해야 한다. 즉, 서비스 Thread가 tpreturn()을 호출하는 시점에 다른 사용자 생성 Thread가 서비스 Thread의 context를 공유하지 않도록 변경해야 한다. 이를 지키지 않으면 tpreturn()은 실패하게 되고 클라이언트로 TPESVCERR 에러 코드를 반환하게 된다. 따라서 반드시 동기화 등을 통해 이들 Thread 사이에서의 프로세스 흐름을 제어해야 한다. tpsetctxt() 함수의 ctxtid 파라미터는 서비스 Thread에서 tpgetctxt() 함수를 호출해서 얻은 Context-ID를 사용한다.

프로그램에 따라 작성 방법에 다른 것에 유의하고, tpsetctxt() 함수의 기본적인 정보는 다음과 같다.

- 프로토타입

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

- 파라미터

파라미터	설명
ctxtid	함수를 호출한 시점의 현재 컨텍스트를 설정한다. 설정 가능한 컨텍스트는 클라이언트 프로그램에서는 tpstart()를 사용하여 새로운 컨텍스트가 생성된 ID이고, 서버 프로그램에서는 서비스 Thread의 Context-ID이다. TPNULLCONTEXT를 비롯한 다른 사용 가능한 컨텍스트로 설정할 수 있으나 TPINVALIDCONTEXT로는 설정할 수 없다.
flags	현재 버전에서는 지원하지 않으나 TPNOFLAGS이나 0으로 설정한다.

- 반환값

반환값	설명
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpsetctxt()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEPROTO]	서버 프로그램의 경우에 서비스 Thread에서 호출한 경우이거나 파라미터로 전달된 Context ID가 유효하지 않는 값인 경우 등 부적절한 상황에서 함수를 호출한 경우 발생한다.
[TPEINVAL]	잘못된 파라미터가 설정된 경우로 ctxtid에 0 또는 TPINVALIDCONTEXT가 설정된 경우나 flags 값에 0 이외의 값이 설정된 경우이다. 클라이언트 프로그램에서는 tpstart()를 수행하기 전에 이 함수를 호출한 경우 발생한 경우로 tpstart()를 호출할 때 버퍼의 flags를 TPMULTICONTEXTS로 설정하지 않은 상태에서 이 함수를 호출한 경우 발생한다. ctxtid가 TPINVALIDCONTEXT이거나 0으로 설정된 경우에 발생한다.

에러 코드	설명
[TPENOENT]	ctxtid에 설정된 값이 설정 가능한 컨텍스트가 아닌 경우 발생한다.
[TPESYSTEM]	Tmax 시스템에 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제 - 클라이언트 프로그램

```
int altContext(int id)
{
    int i;
    int ret;
    ret = tpsetctxt(id, TPNOFLAGS);
    if (ret < 0)
    {
       printf("\t[altContext]tpsetctxt fail[%d][%s]\n"tperrno,
              tpstrerror(tperrno));
       tpfree((char *)tpinfo);
       return -1;
    }
    return 1;
}
```

- 예제 - 서버 프로그램

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

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

    printf("MSERVICE service is started!");
    tpgetctxt(&param.ctxtid, TPNOFLAGS);
    param.svcinfo = svcinfo;

    pthread_create(&tid, NULL, THREAD_ROUTINE, &param);
    pthread_join(tid, NULL);

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

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

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

```
    if (tpsetctxt(param->ctxtid, TPNOFLAGS) == -1) {
        printf("tpsetctxt(%d) failed, [tperrno:%d]", param->ctxtid, tperrno);
        return NULL;
    }

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

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

    return NULL;
}
```

- 관련함수

  tpgetctxt()

3.1.96. tpsetfd

소켓 FD를 UCS 프로세스의 외부 소켓 스케줄러에 등록하는 함수로 UCS 방식 프로세스를 사용한 소켓 FD를 켤 때 사용된다. UCS 스케줄러는 TMM, CLH뿐만 아니라 해당 소켓 FD에 도착한 메시지도 같이 검사한다. 사용자가 지정한 소켓에 메시지가 도착했을 경우 tpschedule()은 별도의 처리없이 정상 결과(UCS_USER_MSG)를 반환하며 어떤 소켓에 메시지가 도착했는지를 알기 위해서는 아래에 설명된 tpisetfd()를 사용해야 한다.

- 프로토타입

```
#include <ucs.h>
int  tpsetfd (int  fd)
```

- 파라미터

파라미터	설명
fd	등록할 소켓 FD를 설정한다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpsetfd()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <usrinc/ucs.h>
...
#define SERV_ADDR “168.126.185.129”
#define SERV_PORT 1500

int fd_read(int, char *, int);
extern int errno;

int usermain(int argc, char *argv[])
{
    ...
    int listen_fd, n, newfd;
    struct sockaddr_in my_addr, child_addr;
    socklen_t child_len;
    buf = tpalloc(“STRING”, NULL, 0);
    if (buf == NULL){
        error processing
    }

    memset((void *)&my_addr, NULL, sizeof(my_addr));
    memset((void *)&child_addr, NULL, sizeof(child_addr));
    listen_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (listen_fd == -1){
        error processing
    }

    my_addr.sin_family = AF_INET;
    inaddr = inet_addr(SERV_ADDR);
    my_addr.sin_port = htons((unsigned short)SERV_PORT);

    if (inaddr != -1){
        memcpy((char *)&my_addr.sin_addr, (char *)&inaddr, sizeof(inaddr));
    }
    ret = bind(listen_fd, (struct sockaddr *)&my_addr, sizeof(my_addr));
    if (ret == -1){
        error processing
    }
    ret = listen(listen_fd, 5);
    if (ret == -1){
        error processing
    }

    ret = tpsetfd(listen_fd);
    if (ret == -1){
        error processing
    }
```

```
    ...

    while(1) {
        n = tpschedule(10);
        ...
        if (n == UCS_USER_MSG){
            if (tpissetfd(listen_fd)) {
                child_len = sizeof(child_addr);
                newfd = accept(listen_fd, &child_addr, &child_len);
                if (newfd == -1){
                    error processing
                }

            ret = tpsetfd(newfd);
            if (ret == -1){
                error processing
            }
        }

        if (tpissetfd(newfd)){
            /* 소켓으로부터 버퍼를 읽는다 */
            fd_read(newfd, buf, 1024);
            ret = tpcall("SERVICE", (char *)buf, sizeof(buf), (char **)&buf,
                          (long *)&rlen, TPNOFLAGS);
            if (ret == -1){
                error processing
            }
            ...
            tpclrfd(newfd);
            close(newfd);
        }
        ...
    }
    return 1;
}
```

- 관련 함수

  tpclrfd(), tpissetfd()

3.1.97. tpsetsvctimeout

서버에서 사용되는 함수로 서버에 설정되어 있는 서비스 타임아웃 시간을 설정한다. tpsetsvctimeout()으로 서비스 시간을 설정한 경우 이 함수가 호출된 이후 설정한 시간(초) 동안 서비스 요청에 대한 응답을 기다린다. 설정한 시간이 지나도록 응답을 수신하지 못하면 타임아웃 에러가 발생하고, 요청한 서비스에 대한 응답을 기다리지 않고 서비스 실패로 반환된다.

- 프로토타입

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

- 파라미터

파라미터	설명
sec	서비스 타임아웃 시간을 초 단위로 설정한다.
flags	현재에는 사용하지 않는다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpsetsvctimeout()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

TOUPPER(TPSVCINFO *msg)
{
        int             i;

        if ( tpsetsvctimeout(10, 0) < 0 )
              ;
            //error handle code
        printf("TOUPPER service is started!\n");
        sleep(15);
        printf("INPUT : data=%s\n", msg->data);

        for (i = 0; i < msg->len; i++)
            msg->data[i] = toupper(msg->data[i]);
        printf("OUTPUT: data=%s\n", msg->data);
        tpreturn(TPSUCCESS,0,(char *)msg->data, 0,0);
}
```

3.1.98. tpsleep

서버와 클라이언트에서 사용되는 함수로 데이터가 도착할 때까지 대기하도록 한다. 최대 타임아웃 시간 동안 sleep하다가 그 안에 데이터가 도착하면 즉시 반환한다.

- 프로토타입

```
#include <tmaxapi.h>
int  tpsleep (struct  timeval  *timeout)
```

- 파라미터

  timeout은 timeval이라는 구조체에 대한 포인터로서 다음과 같이 구성되어 있다(UNIX 시스템의 <sys/time.h>을 참고한다).

```
struct timeval{
      long    tv_sec    /* seconds unit */
      long    tv_usec  /* 0.000001 (micro seconds) unit */
};
```

- 반환값

반환값	설명
양의 정수	타임아웃 시간에 데이터가 도착한 경우이다.
0	타임아웃 시간까지 데이터가 도착하지 않은 경우이다.
-1	함수 수행 중에 에러가 발생한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpsleep()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

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

    struct timeval sl_time, *sleep;
    sl_time.tv_sec =3;
    sl_time.tv_usec=500000;
```

```
    sleep=&sl_time;

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

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

    cd=tpacall("SERVICE", buf, 20, TPNOFLAGS);
    if (cd==-1)
       { error processing }

    ret=tpsleep(sleep); /* 3.5초간 대기 */
    if (ret==-1)
       { error processing }
    if (ret==0)
       printf("waited 3.5sec\n");

    ret=tpgetrply(&cd, &buf, &len, TPNOTIME);
    if (ret==-1)
        { error processing }
    data process....

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

- 관련 함수

  tp_sleep(), tp_usleep(), tpacall(), tpbroadcast(), tpgetrply()

3.1.99. tpspracall

tpgetsprlist()를 통해 얻어온 서버 프로세스의 인덱스 중 특정 프로세스에게 서비스를 호출하는 함수로 비동기 통신으로 TPBLOCK flag를 설정하지 않으면 메시지를 송신한 후 결과를 받을 때까지 기다리지 않고 바로 반환한다. 요청이 쌓이면 clh에서 큐에 쌓인 채 대기한다.

결과는 나중에 **tpgetrply()**를 이용하여 응답을 받을 수도 있고 **tpcancel()**를 이용하여 응답을 취소할 수도 있다. MULTICLH=N인 서버는 지원하지 않는다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
int tpspracall(char *svcname, int spri, char *data, long lenl, long flagsl);
```

- 파라미터

파라미터	설명
svcname	데이터를 전달할 서비스를 설정한다.

파라미터	설명
spri	데이터를 전달할 프로세스 번호를 설정한다.
data	서비스 요청의 데이터에 대한 포인터로 반드시 이전에 tpalloc()에 의해 할당된 버퍼이어야 한다. idata의 유형(type)과 하위 유형(subtype)은 svc가 지원하는 유형이어야 한다.
lenl	송신되는 데이터 길이이다.
flagsl	flags로 사용 가능한 값은 다음과 같다. • TPBLOCK tpspracall 요청 후 그에 대한 응답이 올 때까지 대기한다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpspracall()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPENOENT]	해당 인덱스의 서버 프로세스가 존재하지 않는다.
[TPESYSTEM]	Tmax 시스템에 에러가 발생하였다.
[TPEOS]	운영 시스템에 에러가 발생하였다.
[TPEPROTO]	CLH가 CONV=Y인 서버로 서비스 요청을 전달한 경우이거나, 혹은 MULTICLH=N인 경우에 요청을 받은 CLH에 인자로 받은 spri를 가진 서버가 연결되어 있지 않은 경우에 발생한다.
[TPENOREADY]	인자로 받은 spri에 해당하는 서버가 NOT_READY 상태일 경우에 발생한다.

- 예제

```
...
main(int argc, char *argv[])
{
    char    *sndbuf, *rcvbuf;
    long    sndlen, rcvlen;
    int     ret, cd;
    int     starti = 0, endi = 0, spri = 0;
    if (argc != 2) {
        printf(“Usage: argv[1] string\n”);
        exit(1);
    }

    if ( (ret = tmaxreadenv( “tmax.env”,”TMAX” )) == -1 ){
```

```
        printf("<%-15s> tmaxreadenv fail [%s]", __FILE__, tpstrerror(tperrno));
        exit(1);
    }

    if (tpstart((TPSTART_T *)NULL) == -1){
        printf("<%-15s> tpstart fail [%s]", __FILE__, tpstrerror(tperrno));
        exit(1);
    }

    if ((sndbuf = (char*)tpalloc("CARRAY", 0, 0)) == NULL) {
        printf("<%-15s> sndbuf tpalloc fail [%s]", __FILE__, tpstrerror(tperrno));
        exit(1);
    }

    if ((rcvbuf = (char*)tpalloc("CARRAY", 0, 0)) == NULL) {
        printf("<%-15s> sndbuf tpalloc fail [%s]", __FILE__, tpstrerror(tperrno));
        tpfree((char*)sndbuf);
        exit(1);
    }

    /* tpgetsprlist in client */
    ret = tpgetsprlist(argv[1], 2, &starti, &endi, 0);
    if (ret < 0) {
        printf("<%-15s> tpgetsprlist fail [%s]", __FILE__, tpstrerror(tperrno));
        exit(1);
    } else {
        /*printf("Received Message from Server :
        starti[%d], endi[%d]\n", starti, endi);*/
    }
    strcpy((char*)sndbuf, "tpspracall test");

    /* tpspracall in client */
    for(spri = starti; spri <= endi; spri++) {
        cd = tpspracall(argv[1], spri, sndbuf, strlen(sndbuf), 0);
        if(cd < 0) {
            printf("<%-15s> tpsprcall fail [%s]", __FILE__, tpstrerror(tperrno));
            exit(1);
        } else {
            ret = tpgetrply(&cd, &rcvbuf, &rcvlen, 0);
            if(ret < 0) {
                printf("Received Message from %d spri: tpgetrply fail[%s]\n",
                        spri, tpstrerror(tperrno));
            } else {
                printf("Received Message from %d spri: msg[%s], len[%d]\n",
                        spri, rcvbuf, rcvlen);
            }
        }
    }
    tpend();
}
```

3.1.100. tpspracall2

DYN 타입을 지원하기 위한 api로 STD, STD_DYN 타입 모두 사용 가능하다. **tpgetsprlist()**를 통해 얻어온 starti로부터 spri 순번을 지정하여 해당 서버 프로세스에게 서비스를 호출하는 함수로 비동기 통신으로 TPBLOCK flag를 설정하지 않으면 메시지를 송신한 후 결과를 받을 때까지 기다리지 않고 바로 반환한다. 요청이 쌓이면

clh에서 큐에 쌓인 채 대기한다.

결과는 나중에 **tpgetrply()**를 이용하여 응답을 받을 수도 있고 **tpcancel()**를 이용하여 응답을 취소할 수도 있다. MULTICLH=N인 서버는 지원하지 않는다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
int tpspracall2(char *svcname, int startspri, int nth, char *data, long lenl, long flagsl);
```

- 파라미터

파라미터	설명
svcname	데이터를 전달할 서비스를 설정한다.
startspri	tpgetsprlist() 통해 얻은 starti 값을 설정한다.
nth	spri의 순번을 설정한다.
data	서비스 요청의 데이터에 대한 포인터로 반드시 이전에 tpalloc()에 의해 할당된 버퍼이어야 한다. idata의 유형(type)과 하위 유형(subtype)은 svc가 지원하는 유형이어야 한다.
lenl	송신되는 데이터 길이이다.
flagsl	flags로 사용 가능한 값은 다음과 같다. • TPBLOCK tpspracall 요청 후 그에 대한 응답이 올 때까지 대기한다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpspracall()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPENOENT]	해당 인덱스의 서버 프로세스가 존재하지 않는다.
[TPESYSTEM]	Tmax 시스템에 에러가 발생하였다.
[TPEOS]	운영 시스템에 에러가 발생하였다.
[TPEPROTO]	CLH가 CONV=Y인 서버로 서비스 요청을 전달한 경우이거나, 혹은 MULTICLH=N인 경우에 요청을 받은 CLH에 인자로 받은 spri를 가진 서버가 연결되어 있지 않은 경우에 발생한다.

에러 코드	설명
[TPENOREADY]	인자로 받은 spri에 해당하는 서버가 NOT_READY 상태일 경우에 발생한다.

- 예제

```
...
main(int argc, char *argv[])
{
    char    *sndbuf, *rcvbuf;
    long    sndlen, rcvlen;
    int     ret, cd;
    int     starti = 0, endi = 0, spri = 0;
    if (argc != 2) {
        printf("Usage: argv[1] string\n");
        exit(1);
    }

    if ( (ret = tmaxreadenv( "tmax.env","TMAX" )) == -1 ){
        printf("<%-15s> tmaxreadenv fail [%s]", __FILE__, tpstrerror(tperrno));
        exit(1);
    }

    if (tpstart((TPSTART_T *)NULL) == -1){
        printf("<%-15s> tpstart fail [%s]", __FILE__, tpstrerror(tperrno));
        exit(1);
    }

    if ((sndbuf = (char*)tpalloc("CARRAY", 0, 0)) == NULL) {
        printf("<%-15s> sndbuf tpalloc fail [%s]", __FILE__, tpstrerror(tperrno));
        exit(1);
    }

    if ((rcvbuf = (char*)tpalloc("CARRAY", 0, 0)) == NULL) {
        printf("<%-15s> sndbuf tpalloc fail [%s]", __FILE__, tpstrerror(tperrno));
        tpfree((char*)sndbuf);
        exit(1);
    }

    /* tpgetsprlist in client */
    ret = tpgetsprlist(argv[1], 2, &starti, &endi, 0);
    if (ret < 0) {
        printf("<%-15s> tpgetsprlist fail [%s]", __FILE__, tpstrerror(tperrno)
    );
    exit(1);
    } else {
        /*printf("Received Message from Server :
         starti[%d], endi[%d]\n", starti, endi);*/
    }
    strcpy((char*)sndbuf, "tpspracall test");

    /* tpspracall2 in client */
    cd = tpspracall2(argv[1], starti, 2, send_data, send_len, TPNOFLAGS);
    if(cd < 0) {
        printf("<%-15s> tpsprcall fail [%s]", __FILE__, tpstrerror(tperrno));
        exit(1);
    } else {
        ret = tpgetrply(&cd, &rcvbuf, &rcvlen, 0);
```

```
        if(ret < 0) {
            printf("Received Message from %d spri: tpgetrply fail[%s]\n",
                    spri, tpstrerror(tperrno));
        } else {
            printf("Received Message from %d spri: msg[%s], len[%d]\n",
                    spri, rcvbuf, rcvlen);
        }
    }
    tpend();
}
```

3.1.101. tpsprio

서버와 클라이언트에서 사용되어 서비스 요청의 우선순위를 설정하는 함수로 이후에 전송될 tpacall()의 순번을 설정한다. 설정된 순번은 이후에 수신된 요청에만 영향을 준다. 순번이란 요청된 메시지의 서비스 우선순위를 나타내는 값이다. 순번은 0에서 100까지이며 각 서비스의 기본값은 50이다.

tpsprio()와 **tpgprio()**를 사용하여 요청된 메시지의 서비스 순위를 제어할 수 있다.

- 프로토타입

```
#include <tuxatmi.h>
int tpsprio (int prio, long flags)
```

- 파라미터

파라미터	설명
prio	-50에서 50 사이의 정수가 될 수 있다. prio 값과 현재 순번을 더한 값은 다음 요청에 사용된다.
flags	현재 버전에서는 지원하지 않으나 TPNOFLAGS나 0으로 설정한다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpsprio()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

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

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

    buf = (char *)tpalloc(“STRING”, NULL, 0);
    if (buf==NULL) {error processing };
    ret=tpsprio(30,0);
    if (ret==-1) { error processing }

    ret=tpcall(“SERVICE”, buf, 0, &buf, &len, TPNOFLAGS);
    if (ret==-1) { error processing }

    data process....

    ret=tpsprio(-20,0);
    if (ret==-1) { error processing }

    ret=tpcall(“SERVICE”, buf, 0, &buf, &len, TPNOFLAGS);
    if (ret==-1) { error processing }
    data process....

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

- 관련 함수

  tpacall(), tpcall(), tpgprio()

3.1.102. tpstrerror

서버와 클라이언트에서 사용되는 함수로 에러 번호에 해당하는 메시지를 출력한다. Tmax 함수 이용 중 에러가 발생할 경우 해당 에러 코드는 tperrno라는 전역변수에 설정된다. tpstrerror() 함수는 tperrno에 설정된 에러에 대한 메시지를 출력하는 함수이다.

- 프로토타입

```
# include <atmi.h>
char *tpstrerror (int tperrno)
```

- 파라미터

파라미터	설명
tperrno	에러 메시지를 출력하려는 에러 코드이다.

- 반환값

반환값	설명
에러 메시지	에러 코드에 대한 메시지가 있는 경우이다.
NULL	에러 코드에 대한 메시지가 없는 경우이다.

- 예제

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

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

    tpinfo = (TPSTART_T *)tpalloc(“TPSTART”, NULL, sizeof(TPSTART_T));

    if (tpinfo==NULL) { error processing }
    strcpy(tpinfo->dompwd, “tuxedo”);

    if (tpstart(tpinfo) == -1){
        printf(“tpstart fail , err = %s\n”, tpstrerror(tperrno));
        exit(1);
    }

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

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

3.1.103. tpsubqname

서버와 클라이언트에서 사용되는 함수로 서브 큐 번호에 해당하는 큐의 이름을 반환한다.

- 프로토타입

```
# include <tmaxapi.h>
char *tpsubqname(int type)
```

- 파라미터

tpqstat(), tpqsvcstat() 등에 사용된 것과 동일한 의미로 서브 큐 번호에 해당하는 이름은 다음과 같다.

서브 큐 번호	설명
0	RQ_ANY
1	RQ_FAIL
2	RQ_REQ
3	RQ_RPLY

- 반환값

반환값	설명
서브 큐 이름	해당 서브 큐 번호에 해당하는 서브 큐가 있는 경우이다.
NULL	해당 서브 큐 번호에 해당하는 서브 큐의 이름이 없는 경우이다.

3.1.104. tpsubscribe

서버와 클라이언트에서 사용되는 함수로 특정 사건의 발생시점에 클라이언트나 서버로부터 데이터를 전달받기 위한 요청을 등록한다. eventname에 해당하는 사건이 발생하면 알려 달라는 것을 Tmax 시스템에 등록하는 함수이다.

- 프로토타입

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

- 파라미터

파라미터	설명
eventname	NULL로 끝나는 63자 이내의 문자열로 메시지를 받을 사건의 이름을 지정한다. 와일드 카드 문자나 partial-matching 등은 지원되지 않으며 전체 문자열이 일치하는 이름의 사건만을 등록할 수 있다.
filter	앞으로 확장을 위하여 예약해 놓은 것으로 NULL을 지정한다.
ctl	사건이 발생하는 경우 메시지를 받아오는 방식을 지정하는 구조체로 tpsubscribe()의 주체에 따라 다른 동작을 한다. 클라이언트에서 **tpsubscribe()**를 사용한 경우에 ctl은 항상 NULL이어야 하며 메시지는 클라이언트에게 unsolicited 데이터 형태로 전달된다. 클라이언트는 **tpsubscribe()**이나 **tpgetunsol()** 등의 함수를 사용하여 전달된 데이터를 처리한다.
flags	현재 TPNOTIME만 의미가 있다.

서버에서 **tpsubscribe()**를 행하는 경우에는 ctl이 NULL이면 안되고 아래와 같은 내용을 가져야 한다.

```
struct tpevctl {
```

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

멤버	설명
ctl_flags	앞으로 확장을 위하여 생성된 것으로 현재는 0으로 채워야한다.
post_flags	앞으로 확장을 위하여 생성된 것으로 현재는 0으로 채워야한다.
qname	qname을 사용할 경우 메시지는 tpenq(qname, NULL, data, len, TPNOFLAGS)를 통해 RQ에 저장된다.
svc	svc를 사용할 경우 메시지는 tpacall(svc, data, len, TPNOREPLY)과 유사한 방식으로 서버에게 전달되며 서버의 반환값은 무시된다. qname과 svc는 둘 중 한 가지만을 사용한다.

- 반환값

반환값	설명
descriptor	함수 호출에 성공한 경우이다. tpunsubscribe()를 호출할 때에 사용될 descriptor를 반환한다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpsubscribe()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 클라이언트의 경우 네트워크 에러가 가장 빈번히 발생한다.
[TPEOS]	OS 레벨에서 발생하는 에러로 메모리 할당과 같은 문제를 포함한다.
[TPEINVAL]	잘못된 파라미터를 사용하는 경우 발생하는 에러이다.
[TPENOENT]	TPEVCTL 구조체의 qname이나 svc에 해당하는 RQ나 서버가 존재하지 않는 경우이다.
[TPEPROTO]	클라이언트와 서버에서 ctl이 NULL이 아닌 경우이다.
[TPETIME]	타임아웃이 발생한 경우이다.

- 관련함수

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

3.1.105. tptypes

서버와 클라이언트에서 사용되는 함수로 버퍼의 유형 및 하위 유형에 대한 정보를 제공한다. tptypes()는 ptr로 데이터 버퍼에 대한 포인터를 입력받아, type과 subtype에 각각 버퍼의 유형과 하위 유형을 반환한다.

- 프로토타입

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

- 파라미터

파라미터	설명
ptr	반드시 tpalloc()으로 할당된 버퍼를 가리켜야 한다.
type, subtype	NULL이 아니라면 가리키고 있는 문자열에 각각 버퍼의 type과 subtype 이름을 갖는다. subtype이 존재하지 않으면 subtype이 가리키는 배열은 NULL 스트링을 포함하고 있다. 이름이 최대 길이라면 문자열은 NULL로 끝나지 않는다(최대 길이는 type은 8자, subtype은 16자이다). type의 처음 8Byte와 subtype의 처음 16Byte만이 유효한 값을 갖는다.

- 반환값

반환값	설명
버퍼크기	함수 호출에 성공하는 경우이다.
-1	함수 호출에 실패하는 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tptypes()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 ptr이 가리키는 버퍼가 tpalloc()으로 할당되지 않은 경우에 발생한다.
[TPEPROTO]	함수가 부적절한 상황에서 호출되었다.
[TPESYSTEM]	Tmax 시스템 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include “../sdl/demo.s”
main(int argc, char *argv[])
```

```
{
    int ret;
    struct sel_o *rcvbuf;
    char type[9], subtype[17];
    long size;

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1) { error processing }
    buf=(struct sel_o*)tpalloc("STRUCT","sel_o",0);
    if (buf==NULL) {error processing };

    size =tptypes((char*)buf, type,subtype);
    if (size==-1) {error processing };
    printf ("buf : size %d, type %s, subtype %s\n\n", size, type, subtype);

    /*rcvbuf : size 1024, type STRUCT, subtype sel_o */
    data process...
    tpfree((char *)buf);
    tpend();
}
```

- 관련함수

  tpalloc(), tpfree(), tprealloc()

3.1.106. tpunsubscribe

서버와 클라이언트에서 사용하는 함수로 **tpsubscribe()**로 등록한 특정 사건에 대한 요청을 해제한다. 등록된 모든 요청이 해제되는 경우 Tmax 시스템은 해당 사건의 테이블을 삭제한다.

- 프로토타입

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

- 파라미터

파라미터	설명
sd	tpsubscribe()로 등록할 때 받아온 반환값이다.
flags	현재 TPNOTIME만 의미가 있다.

- 반환값

반환값	설명
양수	함수 호출에 성공한 경우이다.
음수	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpunsubscribe()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템에서 에러가 발생하였다. 클라이언트의 경우 네트워크 에러가 가장 많이 발생한다.
[TPEOS]	OS 레벨의 에러 발생한 경우로 메모리 할당과 같은 문제를 포함한다.
[TPEINVAL]	잘못된 파라미터를 사용한 경우 발생한다.
[TPETIME]	타임아웃이 발생한 경우이다.

- 관련함수

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

3.1.107. tuxgetenv

서버와 클라이언트에서 사용하는 함수로 환경변수 값을 반환한다. 서버에서 사용되는 **getenv()**와 같은 기능을 한다. 클라이언트에서는 **autoexec.bat** 파일에 적용된 값을 서버에서는 각 셸 환경 파일에 적용된 값을 반환한다.

- 프로토타입

```
#include <tuxfml.h>
char *tuxgetenv(char *name)
```

- 파라미터

파라미터	설명
name	반환받을 환경변수를 설정한다.

- 반환값

반환값	설명
환경변수 값 포인터	환경변수가 존재하는 경우이다.
NULL	해당 환경변수가 존재하지 않는 경우이다.

- 예제

```
...
#include <tuxinc/tuxatmi.h>
int main(int argc, char *argv[])
{
    ...
    char hostAddr[20];
```

```
    ...
    memset(hostAddr, NULL, sizeof(hostAddr));
    memcpy(hostAddr, tuxgetenv(“TMAX_HOST_ADDR”), sizeof(hostAddr));
    printf(“host address = %s\n”, hostAddr);
    ...
}
```

- 관련 함수

  tuxputenv()

3.1.108. tuxputenv

서버와 클라이언트에서 사용하는 함수로 환경변수를 적용한다. string 형태로 입력된 "이름 = 값"을 환경변수에 적용한다. 설정할 값이 기존의 환경변수가 존재하는 경우 해당 환경변수 값을 변경한다. 환경변수가 없을 경우에는 새 환경변수를 추가한다. 서버에서 사용하는 **putenv()**와 같은 기능을 한다. 클라이언트에서는 **autoexec.bat** 파일에 적용된 값을 서버에서는 각 셸 환경 파일에 적용된 값을 설정한다.

- 프로토타입

```
#include <tuxfml.h>
int tuxputenv(char *string)
```

- 파라미터

파라미터	설명
string	환경변수에 설정할 값을 입력한다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
음수	환경변수가 충분한 메모리 공간을 확보하지 못한 경우이다.

- 예제

```
 ...
#include <tuxinc/tuxatmi.h>
int main(int argc, char *argv[])
{
    ...
    char hostAddr[20];
    char envVal[20];
    ...
    ret = tuxputenv(“A=10”);
    if (ret < 0){
        printf(“tuxputenv fail...[%d]\n”, tperrno);
```

```
    }
    memset(envVal, NULL, sizeof(envVal));
    memcpy(envVal, tuxgetenv("A"), sizeof(envVal));
    printf("envVal = %s\n", envVal);
    ...
}
```

- 관련 함수

  tuxgetenv()

3.1.109. tuxreadenv

서버와 클라이언트에서 사용되는 함수로 파일의 환경변수를 읽어오는 함수이다. Tmax 시스템에 접속하기 위해서는 환경변수가 등록되어 있어야 한다. 클라이언트는 등록된 환경변수를 참조하여 tpstart()를 이용해 Tmax 시스템에 접속한다. DOS는 <autoexec.bat> 파일에 환경변수를 정의하고, UNIX는 csh에 <.cshrc>를 ksh에 <.profile>을 정의한다. 둘 이상의 시스템에 접속해야 하는 경우에는 상황에 따라 시스템을 변경하는데 이런 경우에는 해당되는 두 시스템에 관한 정보를 환경변수에 등록할 수 없기 때문에 클라이언트는 각 환경변수를 파일 형태로 등록한다.

tuxreadenv()는 접속한 시스템에 관한 정보를 파일로부터 읽어들이고 환경변수를 새로 설정하는 함수이기 때문에 Tmax 시스템에 접속하기 전에 시스템에 대한 정보가 환경변수에 미리 등록되어 있어야 한다. 따라서 함수는 Tmax에 접속하기 전에 실행시켜야 한다.

- 프로토타입

```
#include <tuxfml.h>
int tuxreadenv (char *file, char *label)
```

- 파라미터

파라미터	설명
file	시스템의 접속 환경에 대한 정보가 저장되어있는 파일의 이름을 나타낸다. 이 파일은 미리 작성된 텍스트 포맷에 따라 등록된다.
label	파일에 등록된 환경설정 정보에 대한 구별자이다. 이 값으로 한 파일에 2개 이상의 시스템 정보가 등록된 경우 각 시스템을 구별할 수 있다.

- 반환값

반환값	설명
1	sec 초까지 데이터가 수신되지 않는 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tuxreadenv()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파일이 존재하지 않거나 label이 없는 경우이다.

- 예제

```
...
#include <tuxinc/tuxatmi.h>
int main(int argc, char *argv[])
{
    ...
    ret = tuxreadenv( “tmax.env”,”TMAX” );
    if (ret == -1){
        error processing
    }
    ret = tpinit((TPINIT *)NULL);
    if (ret == -1){
        error processing
    }
    ...
    ret = tpcall(“SERVICE”, buf, 0, &buf, &rcvlen, TPNOFLAGS);
    if (ret == -1){
        error processing
    }
    ...
    tpterm();
}
```

- 관련 함수

  tpstart()

파일에 환경변수를 등록하는 방법에 대한 내용은 "Tmax Administration Guide"를 참고한다.

3.1.110. tx_begin

서버와 클라이언트에서 사용하는 함수로 호출하면 전역 트랜잭션을 시작되고 함수 호출자는 트랜잭션 모드가 된다. 호출 프로세스가 트랜잭션을 시작하기 전에, 먼저 **tx_open()**으로 자원 관리자와 연결되어 있어야 한다. tx_begin()은 호출자가 이미 트랜잭션 모드에 있거나 또는 tx_open()이 호출되지 않았다면 실패하게 되고, [TX_PROTOCOL_ERROR]를 반환한다.

트랜잭션이 시작되면 호출 프로세스는 현재 트랜잭션을 완료하기 위해서 **tx_commit()**이나 **tx_rollback()**를 호출한다. 트랜잭션을 시작하기 위하여 반드시 tx_begin()을 직접적으로 호출할 필요가 없는 연속(chaining) 트랜잭션의 경우도 존재한다. 자세한 내용은 tx_commit과 tx_rollback을 참고한다.

- 프로토타입

```
#include <tx.h>
```

```
int  tx_begin (void)
```

- 반환값

반환값	설명
TX_OK	함수 호출에 성공한 경우이다.
음수	함수 호출에 실패한 경우이다.

- 오류

  tx_begin()이 정상적으로 수행되지 않을 경우 다음의 에러 코드를 반환한다.

에러 코드	설명
[TX_OUTSIDE]	현재 호출 프로세스가 외부의 전역 트랜잭션에 참여하고 있기 때문에 트랜잭션 관리자가 전역 트랜잭션을 시작할 수 없다. 그러한 작업들이 모두 완료되어야만 전역 트랜잭션을 시작할 수 있다. 참여하고 있는 트랜잭션에는 영향을 주지 않는다.
[TX_PROTOCOL_ERROR]	함수가 부적절한 상황에서 호출되었다. 예를 들어 호출자가 이미 트랜잭션 모드에 있는 경우에 발생한다. 현재 트랜잭션에는 영향을 주지 않는다.
[TX_ERROR]	트랜잭션 관리자 또는 리소스 관리자가 트랜잭션을 시작하는 중에 일시적으로 에러가 발생하였다. 에러가 반환되면 호출자는 더 이상 트랜잭션 모드에 있지 않다. 에러의 정확한 원인은 제품의 특성에 따라 결정된다.
[TX_FAIL]	치명적인 에러가 발생하여 트랜잭션 관리자나 리소스 관리자는 더이상 애플리케이션을 위하여 작업을 실행할 수 없다. 에러가 반환되면 호출자는 더 이상 트랜잭션 모드에 있지 않다. 에러의 정확한 원인은 제품의 특성에 따라 다르다.

- 예제

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

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

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

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

    data process.…

    ret=tx_set_transaction_timeout( 5 );
    if (ret<0) { error processing }

    ret=tx_begin();
```

```
    if (ret<0) { error processing }

    cd = tpconnect("SERVICE", buf, 20, TPRECVONLY);
    if (cd==-1) { error processing }

    ret = tprecv(cd, (char **)&buf, &len,TPNOFLAGS, &revent)};
    if (ret < 0 && revent != TPEV_SVCSUCC)
        tx_rollback();
    else
        tx_commit();

    data process....

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

- 관련함수

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

3.1.111. tx_commit

서버와 클라이언트에서 사용하는 함수로 전역 트랜잭션을 commit한다.

transaction_control이 TX_UNCHAINED인 경우 tx_commit()이 반환할 때 호출자는 더 이상 트랜잭션 모드에 있지 않다. transaction_control 특성이 TX_CHAINED인 경우 tx_commit()이 반환할 때 호출자는 새로운 트랜잭션을 위하여 트랜잭션 모드로 남아 있는다. transaction_control 특성에 대한 자세한 내용은 tx_set_transaction_control을 참고한다.

- 프로토타입

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

- 반환값

반환값	설명
TX_OK	함수 호출에 성공한 경우이다.
음수	함수 호출에 실패한 경우 에러 코드를 반환한다.

- 오류

  tx_commit()이 정상적으로 수행되지 않을 경우 다음의 에러 코드를 반환한다.

에러 코드	설명
[TX_NO_BEGIN]	transaction_control 특성이 TX_CHAINED일 때에만 발생하는 에러로 트랜잭션이 성공적으로 commit되었다. 그러나 새로운 트랜잭션은 시작될 수 없고 호출자는 더 이상 트랜잭션 모드에 있지 않다.
[TX_ROLLBACK]	트랜잭션이 rollback되었다. transaction_control 특성이 TX_CHAINED라면 새로운 트랜잭션이 시작된다.
[TX_ROLLBACK_NO_BEGIN]	transaction_control 특성이 TX_CHAINED일 때에만 발생하는 에러로 트랜잭션이 rollback되었다. 이 에러가 발생하는 경우 새로운 트랜잭션은 시작될 수 없고 호출자는 더이상 트랜잭션 모드에 있지 않다.
[TX_HAZARD]	에러때문에 트랜잭션이 일부는 commit되거나 일부는 rollback된다. transaction_control 특성이 TX_CHAINED라면 새로운 트랜잭션이 시작된다.
[TX_HAZARD_NO_BEGIN]	transaction_control 특성이 TX_CHAINED일 때에만 발생하는 에러로 트랜잭션이 일부는 commit되거나 일부는 rollback된다. 새로운 트랜잭션은 시작될 수 없고 호출자는 더 이상 트랜잭션 모드에 있지 않다.
[TX_PROTOCOL_ERROR]	함수가 부적절한 상황에서 호출되었다. 예를 들어 호출자가 트랜잭션 모드에 있지 않는 경우에 발생한다. 트랜잭션과 관련된 호출자의 상태는 변함없다.
[TX_FAIL]	치명적인 에러가 발생하여 트랜잭션 관리자나 리소스 관리자가 더 이상 애플리케이션을 위해 작업할 수 없다. 에러의 정확한 원인은 제품의 특성에 따라 다르다. 트랜잭션과 관련된 호출자의 상태는 알 수 없다.

- 예제

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

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

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

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

    data process.…

    ret=tx_set_transaction_timeout( 5 );
    if (ret<0) { error processing }

    ret=tx_begin();
    if (ret<0) { error processing }

    cd = tpconnect("SERVICE", buf, 20, TPRECVONLY);
    if (cd==-1) { error processing }

    ret = tprecv(cd, (char **)&buf, &len,TPNOFLAGS, &revent)};
```

```
    if (ret < 0 && revent != TPEV_SVCSUCC)
       tx_rollback();
    else
       tx_commit();

    data process....
    tpfree((char *)buf);
    tpend();
}
```

- 관련함수

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

3.1.112. tx_info

서버와 클라이언트에서 전역 트랜잭션 정보를 반환하는 함수로 info가 가리키는 구조체를 통하여 전역 트랜잭션 정보를 알려준다. 또한, 호출자가 현재 트랜잭션 모드에 있는지 여부를 알려주는 값을 반환한다.

- 프로토타입

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

- 파라미터

  info가 NULL이 아니라면 info가 가리키는 TXINFO 구조체는 전역 트랜잭션에 관한 정보가 된다.

  TXINFO 구조체는 아래와 같이 구성되어 있다.

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

  tx_info()가 트랜잭션 모드에서 호출된다면 xid는 현재 트랜잭션 branch id가 되고 transaction_state는 현재 트랜잭션의 상태가 된다. 호출자가 트랜잭션 모드에 있지 않다면, xid는 NULL XID가 된다. 자세한 내용은 <tx.h>를 참고한다.

  호출자가 트랜잭션 모드에 있는 것과 관계없이 when_return, transaction_control 그리고 transaction_timeout은 commit_return의 현재 설정과 transaction_control 특성 그리고 초 단위의 트랜잭션 타임아웃 값을 포함한다.

  반환된 트랜잭션 타임아웃 값은 다음 트랜잭션이 시작될 때부터 사용된다. 현재 트랜잭션이 시작된 후에 tx_set_transaction_timeout() 호출을 통해 트랜잭션 타임아웃값을 변경했을 수도 있기 때문에 호출자의 현재

전역 트랜잭션에 대한 타임아웃 값이 아닐 수도 있다. info가 NULL이라면 TXINFO 구조체는 반환되지 않는다.

같은 전역 트랜잭션 내에서 계속적인 tx_info() 호출은 동일한 gtrid(전역 트랜잭션 구분자)의 XID 제공을 보장한다. 하지만 반드시 동일한 bqual(로컬 트랜잭션 구분자)을 보장하지는 않는다. XID는 동일하지 않을 수 있다.

- 반환값

반환값	설명
1	호출자가 트랜잭션 모드에 있는 경우 함수 호출에 성공한 경우이다.
0	호출자가 트랜잭션 모드에 있지 않은 경우 함수 호출에 성공한 경우이다.
음수	함수 호출에 실패한 경우로 에러 코드를 반환한다.

- 오류

tx_info()가 정상적으로 수행되지 않을 경우 다음의 에러 코드를 반환한다.

에러 코드	설명
[TX_PROTOCOL_ERROR]	함수가 부적절한 상황에서 호출되었다. 예를 들어 호출자가 아직 tx_open()을 호출하지 않은 경우에 발생한다.
[TX_FAIL]	치명적인 에러가 발생하여 트랜잭션 관리자는 더 이상 애플리케이션을 위한 작업을 수행할 수 없다. 에러의 정확한 원인은 제품의 특성에 따라 다르다.

- 예제

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

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

    buf = (char *)tpalloc("CARRAY", NULL, 20);
    if (buf==NULL) {error processing };
    data process....
    ret=tx_begin();
    if (ret<0) { error processing }

    ret=tpcall("SERVICE", buf, 20, (char **)&buf, &len, TPNOFLAGS);
    if (ret==-1) { error processing }

    if (tx_info(&info)==1) printf("In transaction \n");
    else printf("Not in transaction \n");
```

```
    if (strncmp(buf, “err”, 3)==0) tx_rollback();
    else  tx_commit();

    data process....
    tpfree((char *)buf);
    tpend();
}
```

- 관련함수

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

3.1.113. tx_rollback

서버와 클라이언트에서 사용하는 함수로 전역 트랜잭션을 rollback한다.

transaction_control 특성이 TX_UNCHAINED인 경우 tx_rollback()이 반환할 때 호출자는 더 이상 트랜잭션 모드에 있지 않다. transaction_control 특성이 TX_CHAINED인 경우 tx_rollback()이 반환할 때 호출자는 새로운 트랜잭션을 위하여 트랜잭션 모드로 남아 있는다. transaction_control 특성에 대한 자세한 내용은 tx_set_transaction_control을 참고한다.

- 프로토타입

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

- 반환값

반환값	설명
TX_OK	함수 호출에 성공한 경우이다.
음수	함수 호출에 실패한 경우로 에러 코드를 반환한다.

- 오류

  tx_rollback()이 정상적으로 수행되지 않을 경우 다음의 에러 코드를 반환한다.

에러 코드	설명
[TX_NO_BEGIN]	transaction_control 특성이 TX_CHAINED일 때에만 발생하는 에러로 트랜잭션이 rollback된다. 새로운 트랜잭션은 시작될 수 없고 호출자는 더 이상 트랜잭션 모드에 있지 않다.
[TX_MIXED]	트랜잭션이 일부는 commit되었고 일부는 rollback된다. transaction_control 특성이 TX_CHAINED라면 새로운 트랜잭션을 시작한다.
[TX_MIXED_NO_BEGIN]	transaction_control 특성이 TX_CHAINED일 때에만 발생하는 에러로 트랜잭션이 일부는 commit되고 일부는 rollback된다. 새로운 트랜잭션은 시작될 수 없고 호출자는 더 이상 트랜잭션 모드에 있지 않다.

에러 코드	설명
[TX_HAZARD]	에러로 인해 트랜잭션이 일부는 commit되거나 일부는 rollback된다. transaction_control 특성이 TX_CHAINED라면 새로운 트랜잭션이 시작된다.
[TX_HAZARD_NO_BEGIN]	transaction_control 특성이 TX_CHAINED일 때에만 발생하는 에러로 트랜잭션이 일부는 commit되거나 일부는 rollback된다. 새로운 트랜잭션은 시작될 수 없고 호출자는 더 이상 트랜잭션 모드에 있지 않다.
[TX_COMMITTED]	트랜잭션이 독자적으로 commit된다. transaction_control 특성이 TX_CHAINED인 경우 새로운 트랜잭션이 시작된다.
[TX_COMMITTED_NO_BEGIN]	transaction_control 특성이 TX_CHAINED일 때만 발생될 수 있는 에러로 트랜잭션이 독자적으로 commit된다. 새로운 트랜잭션은 시작될 수 없고 호출자는 더 이상 트랜잭션 모드에 있지 않다.
[TX_PROTOCOL_ERROR]	함수가 부적절한 상황에서 호출되었다. 예를 들어 호출자가 트랜잭션 모드에 있지 않은 경우에 발생한다. 트랜잭션과 관련된 호출자의 상태는 변함없다.
[TX_FAIL]	치명적인 에러가 발생하여 트랜잭션 관리자나 리소스 관리자는 더 이상 애플리케이션을 위한 작업을 실행할 수 없다. 에러의 정확한 원인은 제품의 특성에 따라 다르다. 트랜잭션과 관련된 호출자의 상태는 알 수 없다.

- 예제

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

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

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

    buf = (char *)tpalloc("CARRAY", NULL, 20);
    if (buf==NULL) {error processing };
    data process.…
    ret=tx_set_transaction_timeout( 5 );
    if (ret<0) { error processing }

    ret=tx_begin();
    if (ret<0) { error processing }

    cd = tpconnect("SERVICE", buf, 20, TPRECVONLY);
    if (cd==-1) { error processing }
    ret = tprecv(cd, (char **)&buf, &len,TPNOFLAGS, &revent)};
    if (ret < 0 && revent != TPEV_SVCSUCC)  tx_rollback();
    else    tx_commit();
    data process....

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

```
}
```

- 관련함수

  tx_begin(), tx_set_transaction_control(), tx_set_transaction_timeout()

3.1.114. tx_set_commit_return

서버와 클라이언트에서 commit_return 특성을 설정하는 함수로 when_return 값으로 반환한다.

commit_return 특성은 tx_commit()이 함수 호출자에게 제어권을 반환하는 방식을 결정한다. tx_set_commit_return()은 함수 호출자가 트랜잭션 모드에 있는 것과 관계없이 호출 가능하다. 설정은 tx_set_commit_return() 재호출로 인해 변경될 때까지 유효하게 적용된다. commit_return 특성에 대한 초기 설정은 실행에 따라 다르다.

- 프로토타입

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

- 파라미터

  when_return으로 사용 가능한 값은 다음과 같다.

설정값	설명
TX_COMMIT_DECISION_LOGGED	tx_commit()이 2PC(2 Phase Commit) 프로토콜 중 첫 번째 단계에서 로깅된 후 두 번째 단계는 완료되기 전에 반환한다. tx_commit()이 호출자에게 보다 빠르게 응답할 수 있지만, 트랜잭션이 독자적(heuristic)인 결과를 갖게 될 위험이 있고, 그런 경우에 호출자는 tx_commit()으로부터 반환된 코드로 발생된 상황을 제대로 알 수 없다. 정상적인 경우에 첫 번째 단계에서 트랜잭션을 commit하기로 한 트랜잭션 참여자는 두 번째 단계에서 제대로 commit하게 된다. 그러나 네트워크나 노드 장애가 길게 지속되는 등의 비정상적인 경우에는 2개 단계의 완료가 가능하지 않을 수 있으며, 독자적인 결과를 초래할 수도 있다. 트랜잭션 관리자는 옵션으로 이 특성을 지원하지 않도록 선택할 수 있으며, 이때 이 값을 지원하지 않음을 나타내는 [TX_NOT_SUPPORTED] 값으로 반환한다.
TX_COMMIT_COMPLETED	2PC 프로토콜이 완전하게 종료된 후에 tx_commit()이 반환한다. 설정은 트랜잭션이 독자적인(heuristic) 결과를 갖게 되었거나 또는 그럴 가능성을 알리는 반환 코드를 tx_commit()의 호출자에게 전달한다. 트랜잭션 관리자는 옵션으로 이 특징을 지원하지 않도록 선택할 수 있으며, 이 값을 지원하지 않음을 나타내는 [TX_NOT_SUP-PORTED] 값으로 반환한다.

- 반환값

  성공적으로 작업이 완료된 경우 tx_set_commit_return()은 음이 아닌 값의 [TX_OK]를 반환한다.

when_return이 TX_COMMIT_COMPLETED 또는 TX_COMMIT_DECISION_LOGGED으로 설정되지 않았다면, 함수는 음이 아닌 값으로 [TX_NOT_SUPPORTED]를 반환하고, commit_return 특성은 현재 적용되고 있는 값이 여전히 유효하다. 트랜잭션 관리자는 when_return을 최소한 TX_COMMIT_COMPLETED나 TX_COMMIT_DECISION_LOGGED 중의 하나로는 설정해야 한다.

함수 호출에 실패한 경우 에러 코드를 반환한다.

- 오류

tx_set_commit_return()는 commit_return 특성 설정 변경 없이 음수값 중 하나를 반환한다.

에러 코드	설명
[TX_EINVAL]	when_return이 TX_COMMIT_COMPLETED나 TX_COMMIT_DECISION_LOGGED로 설정되어 있지 않다.
[TX_PROTOCOL_ERROR]	함수가 부적절한 상황에서 호출되었다. 예를 들어 호출자가 아직 tx_open()을 호출하지 않은 경우이다.
[TX_FAIL]	치명적인 에러가 발생하여 트랜잭션 관리자는 더 이상 애플리케이션을 위한 작업을 수행할 수 없다. 에러의 정확한 원인은 제품의 특성에 따라 다르다.

- 예제

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

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

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

    buf = (char *)tpalloc("CARRAY", NULL, 20);
    if (buf==NULL) {error processing };
    ret = tx_set_transaction_timeout(5);
    if (ret < 0){ error processing }

    ret = tx_set_commit_return(TX_COMMIT_COMPLETED);
    if (ret < 0){ error processing }

    ret = tx_begin();
    if (ret < 0){ error processing }

    data process....
    ret = tpcall("SERVICE1", (char *)buf, strlen(buf), (char **)&buf, &len,
                  TPNOFLAGS);
    if (ret == -1)
    {
        tx_rollback();
        error processing
    }

    data process...
```

```
    ret = tpcall("SERVICE2", (char *)buf, strlen(buf), (char **)&buf, &len,
                  TPNOFLAGS);
    if (ret == -1)
    {
        tx_rollback();
        error processing
    }

    data process ….

    ret = tx_commit();
    if (ret < 0) { error processing }

    data process...

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

- 관련함수

  tx_commit(), tx_open(), tx_info()

3.1.115. tx_set_transaction_control

서버와 클라이언트에서 transaction_control 특성을 control 값으로 설정하는 함수이다.

transaction_control 특성은 **tx_commit()**과 **tx_rollback()**이 호출자에게 반환하기 전에 새로운 트랜잭션을 시작할지 여부를 결정한다. tx_set_transaction_control()는 애플리케이션이 트랜잭션 모드에 있는지 여부와 관계없이 호출 가능하다. 이 설정은 tx_set_transaction_control() 재호출에 의해 변경될 때까지 유효하게 적용된다.

- 프로토타입

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

- 파라미터

  control로 사용 가능한 값은 다음과 같다.

설정값	설명
TX_UNCHAINED	tx_commit()과 tx_rollback()이 함수 호출자에게 반환하기 전에 새로운 트랜잭션을 시작하지 않도록 한다. 이 경우 호출자는 새로운 트랜잭션을 시작하려면 tx_begin()을 실행해야 한다. transaction_control 특성의 초기 설정값이다.
TX_CHAINED	tx_commit()과 tx_rollback()이 호출자에게 반환하기 전에 새로운 트랜잭션을 시작하도록 한다.

- 반환값

반환값	설명
TX_OK	함수 호출에 성공한 경우이다.
음수	함수 호출에 실패한 경우로 에러 코드를 반환한다.

- 오류

tx_set_transaction_control()은 기존의 transaction_control 특성 변경없이 다음의 에러 코드를 반환한다.

에러 코드	설명
[TX_EINVAL]	control 파라미터가 TX_UNCHAINED나 TX_CHAINED로 설정되지 않았다.
[TX_PROTOCOL_ERROR]	함수가 부적절한 상황에서 호출되었다. 예를 들어 함수 호출자가 아직 tx_open()을 호출하지 않은 경우에 발생한다.
[TX_FAIL]	치명적인 에러가 발생하여 트랜잭션 관리자는 더 이상 애플리케이션을 위한 작업을 수행할 수 없다. 에러의 정확한 원인은 제품의 특성에 따라 다르다.

- 예제

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

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

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

    buf = (char *)tpalloc("CARRAY", NULL, 20);
    if (buf==NULL) {error processing };
    ret = tx_set_transaction_timeout(5);
    if (ret < 0){ error processing }

    ret = tx_set_transaciont_control(TX_UNCHAINED);
    if (ret < 0){ error processing }
    ret = tx_begin();
    if (ret < 0) { error processing }
    data process....

    ret = tpcall("SERVICE1", (char *)buf, strlen(buf), (char **)&buf, &len,
                 TPNOFLAGS);
    if (ret == -1)
    {
        tx_rollback();
        error processing
    }

    data process...
    ret = tpcall("SERVICE2", (char *)buf, strlen(buf), (char **)&buf, &len,
```

```
                    TPNOFLAGS);
    if (ret == -1)
    {
        tx_rollback();
        error processing
    }
    data process ….

    ret = tx_commit();
    if (ret < 0) { error processing }
    data process...

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

- 관련함수

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

3.1.116. tx_set_transaction_timeout

서버와 클라이언트에서 transaction_timeout 설정 함수로 transaction_timeout 특성을 타임아웃 값으로 설정한다. 설정된 값은 트랜잭션 타임아웃이 발생하기 전에 트랜잭션을 완료해야 하는 시간으로 **tx_begin()**과 **tx_commit()** 또는 **tx_begin()**과 **tx_rollback()** 사이의 시간이다.

tx_set_transaction_timeout()는 함수 호출자가 트랜잭션 모드에 있는지와 상관없이 호출 가능하다. tx_set_transaction_timeout()이 트랜잭션 모드에서 호출된다면 새로운 타임아웃 값은 다음 트랜잭션 때부터 적용된다.

- 프로토타입

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

- 파라미터

파라미터	설명
timeout	트랜잭션 타임아웃이 발생하기 전까지 허용된 시간을 초 단위 숫자로 지정한다. 설정값은 시스템별로 정의된 long 타입의 최댓값까지 설정 가능하다. 초기값은 0으로 설정되고 타임아웃 제한이 없음을 의미한다.

- 반환값

반환값	설명
TX_OK	함수 호출에 성공한 경우이다.
음수	함수 호출에 실패한 경우로 에러 코드를 반환한다.

- 오류

  tx_set_transaction_timeout()은 기존의 transaction_timeout 값의 변경 없이 다음의 에러 코드를 반환한다.

에러 코드	설명
[TX_EINVAL]	지정된 타임아웃 값이 유효하지 않다.
[TX_PROTOCOL_ERROR]	함수가 부적절한 상황에서 호출되었다. 예를 들어 호출자가 아직 tx_open()을 호출하지 않은 경우에 발생한다.
[TX_FAIL]	치명적인 에러가 발생하여 트랜잭션 관리자가 더 이상 애플리케이션을 위한 작업을 수행할 수 없다. 에러의 정확한 원인은 제품의 특성에 따라 다르다.

- 예제

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

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

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

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

    ret=tx_set_transaction_timeout( 5 );
    if (ret<0) { error processing }

    ret=tx_begin();
    if (ret<0) { error processing }
    ret = tpcall("SERVICE", (char *)buf, strlen(buf), (char **)&buf, &len,
                  TPNOFLAGS);
    if (ret == -1) {
        tx_rollback();
        error processing
    }
    data process ….
    ret = tx_commit();
    if (ret < 0) { error processing }

    data process...
    tpfree((char *)buf);
    tpend();
}
```

- 관련함수

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

3.1.117. ulogsync

userlog()를 이용하여 ulog의 내용을 디스크의 메모리 버퍼에 있는 <ulog.dat> 파일에 저장하는 함수이다.

- 프로토타입

```
# include  <userlog.h>
int ulogsync(void)
```

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. 에러가 발생해도 tperrno에 저장되는 값은 없다.

- 예제

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

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

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

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

    data process....

    cd = tpacall("SERVICE", (char **)buf, 20, TPNOFLAGS);
    if (cd==-1) { error processing }

    ret=tpgetrply(&cd, &buf, &len, TPNOTIME);
    if (ret==-1){ error processing }

    ret=userlog(" error : %s\n", tpstrerror(errno));
    if (ret==-1) { error processing }
    tpfree((char *)buf);

    ret=ulogsync();
    if (ret==-1) { error processing }

    tpend();
}
```

3.1.118. userlog

프로그램에 필요한 정보를 <ulog.날짜> 파일로 작성하는 함수이다. 프로그램 Logic 에러를 찾기 위해서 프로그램의 중간 중간에 사용자가 필요한 정보를 저장할 때 사용한다.

<ulog.날짜> 파일의 위치는 설정 파일의 ULOGDIR 항목에 정의한 디렉터리에 생성된다. 서버와 클라이언트 모두 이 함수를 사용 가능하며, 모두 동일한 파일에 저장된다.

userlog() 함수가 호출되어도 즉시 데이터를 파일에 저장하지 않고 버퍼에 일정량이 저장되어야만 실제로 파일에 저장한다. 파일에 즉시 저장하기 위해서는 **ulogsync()**를 사용한다. userlog() 함수와 UserLog() 함수는 C 언어에서 제공하는 printf() 함수와 동일한 포맷을 사용하므로 escape sequence character를 그대로 사용할 수 있다.

userlog()를 사용할 때 로그 파일은 프로그램에서 첫 번째 userlog() 함수가 호출될 때 ULOGDIR 디렉터리에 생성되며 그 이후에는 변하지 않는다는 것에 주의한다.

첫 번째로 호출된 userlog()에 의해 이 함수가 어느 디렉터리에 로그를 남길지 결정된다. 따라서 userlog()를 한 번이라도 호출한 후에 클라이언트의 환경 파일(ULOGDIR)을 변경해도 파일이 새로 생성되거나 파일의 위치가 변경되지 않는다.

- 프로토타입

```
# include <userlog.h>
int userlog(const char *fmt, ...)
```

- 파라미터

파라미터	설명
fmt	사용자가 남길 로그 문자열이다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. 에러가 발생해도 tperrno에 저장되는 값은 없다.

- 예제

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

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

```
    char *buf;

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

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

    cd = tpacall("SERVICE", (char **)buf, 20, TPNOFLAGS);
    if (cd==-1) { error processing }

    ret=tpgetrply(&cd, &buf, &len, TPNOTIME);
    if (ret==-1){

    ret=userlog("error no: %d, urcode : %d", tperrno, tpurcode");
    if (ret==-1) { error processing }

    data process..
    tpfree((char *)buf);
    tpend();
}
```

3.1.119. UserLog

서버와 클라이언트에서 사용하는 함수로 메모리 버퍼를 사용하지 않고 ulog 파일을 직접 작성한다. UserLog()는 호출 즉시 데이터를 저장하는 **ulogsync()**를 포함한다. 용도는 userlog() 함수와 같다. 버퍼링 사이즈는 8KB로 제한된다.

- 프로토타입

```
# include <userlog.h>
int UserLog (const char *fmt, ...)
```

- 파라미터

파라미터	설명
fmt	사용자가 남길 로그 문자열이다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. 에러가 발생해도 tperrno에 저장되는 값은 없다.

- 예제

```
#include <stdio.h>
```

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

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

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

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

    strcpy(input, buf);
    ret=tpcall("SERVICE", buf, 20, (char **)&buf, &len, TPNOFLAGS);
    if (ret==-1) { error processing }

    data process ….

    UserLog("input %s -> output : %s\n",input, buf);

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

3.2. 서버 함수

3.2.1. _tmax_check_license

AnyLink 및 OpenFrame에서 해당 라이선스가 발급되었는지를 체크할 수 있는 함수이다.

- 프로토타입

```
int _tmax_check_license(char *tmaxdir, int subprod);
```

- 파라미터

파라미터	설명
tmaxdir	현재 Tmax 시스템이 설치되어 있는 홈 디렉터리($TMAXDIR)를 설정한다.

subprod	AnyLink와 OpenFrame 중 어느 제품에 대하여 체크할 것인지를 설정한다.
	
	다음은 subprod에 사용하는 상수에 대한 정의이다.
	
	`#define SUB_PROD_ANYLINK 0x00200000`
	`#define SUB_PROD_OPENFRAME 0x00400000`

- 반환값

반환값	설명
1	해당 라이선스가 발급된 상태이다.
-1	해당 라이선스가 발급되지 않은 상태이다.

3.2.2. _tmax_event_handler

SVRTYPE이 EVT_SVR인 경우 SLOG가 발생하면 호출되는 콜백 함수이다. SLOG가 발생했을 때 더 자세한 정보를 조회할 경우 이용한다. SLOG 이벤트가 발생하는 경우 해당 함수가 호출되며 해당 함수의 파라미터에 자세한 정보가 담겨진다. SVRTYPE이 EVT_SVR인 서버 프로그램 안에 정의할 수 있으며, 한 노드당 하나의 EVT_SVR 타입의 서버를 정의한다.

해당 함수가 호출되는 빈도는 환경 파일 NODE 절의 TMMOPT에 [–h] 옵션을 통해서 이벤트 핸들러 로그 레벨을 설정한다. NODE 절 TMMOPT의 [–h] 옵션에 대한 자세한 설명은 "Tmax Administration Guide"를 참고한다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
int _tmax_event_handler(char *progname, int pid, int tid, char *msg, int flags);
```

- 파라미터

파라미터	설명
progname	이벤트를 발생시킨 프로그램명이다.
pid	프로세스 ID이다.
tid	Thread ID이다(예비용).
msg	이벤트 메시지이다.
flags	예비용으로 현재 버전에서는 지원하지 않는다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다.

- 예제

<환경 파일>

```
*DOMAIN
tmax1         SHMKEY = 78650, MINCLH = 1, MAXCLH = 3,
              TPORTNO = 8650, BLOCKTIME = 30
```

```
*NODE
Tmaxh4        TMAXDIR = "/data1/starbj81/tmax",
              APPDIR  = "/data1/starbj81/tmax/appbin",
              PATHDIR = "/data1/starbj81/tmax/path",
              TLOGDIR = "/data1/starbj81/tmax/log/tlog",
              ULOGDIR = "/data1/starbj81/tmax/log/ulog",
              SLOGDIR = "/data1/starbj81/tmax/log/slog",
              TMMOPT = "-h i", SMSUPPORT = Y, SMTBLSIZE = 1000

*SVRGROUP
svg1          NODENAME = "@HOSTNAME@"

*SERVER
evtsvr        SVGNAME = svg1, SVRTYPE = EVT_SVR
```

+ <서버 프로그램>

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

int tpsvrinit(int argc, char *argv[])
{
    printf("[EVTHND] started\n");
    return 1;
}

int svrdone()
{
    printf("[EVTHND stopped\n"];
    return 1;
)

int _tmax_event_handler(char *program, int pid, int tid, char *msg, int flags)
{
    time_t t1;
    struct tm *tm;

    time(&t1);
    tm = localtime(&t1);
    printf("[EVTHND] %s.%d.%02d%02d%02d:%s\n", program, pid, tm->tm_hour,
             tm->tm_min, tm->tm_sec, msg);
    return 0;
}
```

<Makefile.evt>

```
# Server makefile
TARGET  = $(COMP_TARGET)
APOBJS  = $(TARGET).o
NSDLOBJ = $(TMAXDIR)/lib64/sdl.o

LIBS    = -lsvrevt -lnodb
```

```
OBJS    = $(APOBJS) $(SVCTOBJ)
SVCTOBJ = $(TARGET)_svctab.o
CFLAGS  = -Ae +DA2.0W +DD64 +DS2.0 -O -I$(TMAXDIR)

APPDIR  = $(TMAXDIR)/appbin
SVCTDIR = $(TMAXDIR)/svct
LIBDIR  = $(TMAXDIR)/lib64

#
.SUFFIXES : .c

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

#
# server compile
#

$(TARGET): $(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)
```

3.2.3. _tmax_main

서버에서 사용하는 함수로 사용자 프로그램에 main()이 포함되어 있는 경우에 사용한다. 서버 프로세스를 만들 때 사용자 프로그램에서는 main()이 포함되어 있으면 안 된다. 부득이하게 사용자가 작성한 라이브러리 내에 main()이 포함되어 있고 이를 수정할 수 없는 경우에는 Tmax 서버 라이브러리 내의 main() 루틴을 호출하여 처리할 수 있도록 제공되는 함수가 _tmax_main() 함수이다.

이 함수를 사용하기 위해 서버 프로그램 외에 <***.c> 파일을 별도로 하나 생성하여 이 파일 안에 main() 함수를 생성한다. 이 함수 내에서 _tmax_main()을 호출하면 사용자가 작성한 라이브러리에 있는 main()이 호출되지 않고 Tmax 서버 라이브러리에 있는 main() 함수가 호출된다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
int _tmax_main(int argc, char *argv[])
```

- 파라미터

main() 함수와 동일하다.

파라미터	설명
argc	argv 의 배열 수이다.
argv	사용자가 CLOPT 절에 설정한 인자 목록이다.

- 반환값

  정수값으로 main() 함수와 동일하다.

- 예제

  다음은 함수 사용하는 과정에 대한 설명이다.

  1. 사용자는 다음과 같이 라이브러리를 작성한다.

     ```
     #include <stdio.h>

     int main(int argc, char* argv[])
     {
         printf(“my lib main called.\n”);
     }
     ```

  2. 함수를 컴파일한 후 <libmysvr.so> 파일을 생성한다.

     사용자가 작성한 라이브러리의 main()을 사용하지 않고 Tmax 서버 라이브러리 내의 main()을 사용하기 위해 <mysvr.c>파일을 생성하고 파일 안에 main()을 생성한다.

     main()에서 _tmax_main()을 호출한다. 현재 main()은 사용자가 작성한 라이브러리, Tmax 서버 라이브러리, <mysvr.c>의 3가지 main() 함수가 존재한다. 하지만 Tmax 시스템을 기동하면 제일 먼저 서버 프로그램과 함께 컴파일된 <mysvr.c>의 main() 함수가 호출되기 때문에 함수 내에서 호출된 Tmax 서버 라이브러리 내의 main() 함수가 호출되고, 사용자가 작성한 서버 라이브러리 내의 main()은 호출되지 않는다.

     ```
     #include <stdio.h>

     int main(int argc, char *argv[])
     {
         printf(“My main is called\n”);
         _tmax_main(argc, (char **)argv);
         return 0;
     }
     ```

  3. 작성된 <***.c> 파일을 컴파일하여 오브젝트 파일을 생성한 후 서버 프로그램과 함께 링크시켜 하나의 실행 파일을 생성한다.

3.2.4. CliWatcherCallback

client 종료를 통지받을 콜백함수이며, 사용자가 함수를 작성한 뒤 tpsetcliwatcher 함수의 callback 인자에 넣어준다. 콜백 함수가 호출되는 시점은 tpschedule()이 호출되고 클라이언트의 종료 이벤트가 감지되는 시점에 호출된다.

CliWatcherCallback() 함수의 기본적인 정보는 다음과 같다.

- 프로토타입

```
#include <usrinc/ucs.h>
typedef void (*CliWatcherCallback)(int clid, int reason, void *args)
```

- 파라미터

파라미터	설명
clid	종료된 client의 clid이다.
reason	클라이언트가 종료된 원인을 명시한다. tmaxgw_setcliwatcher()의 경우 요청을 했으나 clh에서 감지 못하는 경우에도 이 콜백함수를 통해 응답을 준다.
args	tpsetcliwatcher()나 tmaxgw_setcliwatcher()에서 사용자가 정의한 args 변수의 값이다. 사용자 정의 데이터를 전달하는 용도로 사용된다. 해당 args를 사용할 때는 반드시 메모리 할당/해제에 주의해야 한다.

- 예제

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

void logout_callback(int clid, int reason, void *args) {
        printf("[%s] clid:%#x, args:%p, reason:%d\n", "CALLBACK", clid, args, reason);
}

LOGIN(TPSVCINFO *msg) {
        int clid, ret;
        time_t curtime = time(NULL);

        clid = tpgetclid();
        ret = tpsetcliwatcher(clid, logout_callback, (void *)curtime, TPNOFLAGS);

        printf("[%s] clid:%#x, curtime:%ld, set_watch:%d(%s), data:%s\n",
                        msg->name, clid, curtime, ret, tpstrerror(tperrno), msg->data);

        tpreturn(TPSUCCESS, 0, NULL, 0, 0);
}
```

- 관련함수

tpsetcliwatcher(), tpclrcliwatcher()

3.2.5. tmadmin

Tmax 시스템 관리 툴인 tmadmin에서 조회할 수 있는 통계정보를 출력하는 함수로 프로세스 내부에서 동적으로 사용이 가능하다. 서버 프로세스 내부에서만 사용 가능한 함수이다.

- 프로토타입

```
# include <tmadmin.h>
int tmadmin(int cmd, void *arg, int opt, long flags);
```

- 파라미터

파라미터	설명
cmd	다음 항목이 설정 가능하다. TMADM_DISCON, TMADM_CLIINFO, TMADM_QPURGE, TMADM_BOOT, TMADM_DOWN, TMADM_SUSPEND, TMADM_RESUME, TMADM_RESTAT, TMADM_SVC_STAT, TMADM_SPR_STAT, TMADM_SVR_STAT, TMADM_SVC_STAT_EX, TMADM_SPR_STAT_EX, TMADM_TMAX_INFO, TMADM_DOMAIN_CONF, TMADM_NODE_CONF, TMADM_SVG_CONF, TMADM_SVR_CONF, TMADM_SVC_CONF, TMADM_SMTRC, TMADM_CHTRC, TMADM_CHLOG, TMADM_TMGW_STAT, TMADM_NTMGW_STAT, TMADM_TMMS_STAT, TMADM_CLHS_STAT, TMADM_TMS_STAT, TMADM_HMS_STAT, TMADM_HMSCLI_STAT, TMADM_RESTART, TMADM_SET, TMADM_TG_INFO
arg	대부분 각 cmd에 맞는 구조체의 포인터를 지정해야 하며 데이터 버퍼의 크기와 offset 등을 지정할 수 있다.
opt	각 cmd별 옵션 flags들이 설정될 수 있다. 자세한 내용은 표 이후에 설명한다.
flags	예비용으로 현재는 사용하지 않는다.

다음은 cmd별 opt에 설정할 수 있는 옵션 flags 목록이다.

◦ TMADM_DISCON

옵션 flags	설명
TPNOFLAGS	클라이언트가 서비스를 요청 중이면 종료하지 않는다.

옵션 flags	설명
TMADM_FFLAG	클라이언트가 서비스를 요청 중이면 종료한다.

◦ TMADM_CLIINFO

옵션 flags	설명
TPNOFLAGS	해당 노드의 모든 클라이언트 정보를 전달한다.
TMADM_SFLAG	해당 노드의 모든 클라이언트 수를 전달한다(arg →header.num_left).

◦ TMADM_QPURGE

옵션 flags	설명
TMADM_SFLAG	서비스 큐를 purge한다.
TMADM_VFLAG	서버 큐를 purge한다.

◦ TMADM_BOOT(TMADM_DOWN)

옵션 flags	설명
TMADM_SFLAG	arg→args.name1을 서버명으로 사용하여 해당 서버를 boot/down한다.
TMADM_GFLAG	arg→args.name2를 서버 그룹명으로 사용하여 해당 서버 그룹의 서버를 boot/down한다.
TMADM_TFLAG	arg →args.name1을 서버 그룹명으로 사용하여 해당 서버 그룹의 TMS를 boot/down한다.
TMADM_USFLAG	tmboot/tmdown의 -S 옵션과 같은 의미이다. struct tmadm_boot의 args.count에 값을 지정하지 않는다.TMADM_USFLAG 와 TMADM_GFLAG를 함께 지정하는 경우 해당 서버그룹의 서버들만 적용된다.

다음은 tmadmin()의 TMADM_BOOT(TMADM_DOWN)에서 사용가능한 flags 조합이다.

- TMADM_SFLAG : -s 옵션과 동일하고 count를 지정해야 한다.
- TMADM_SFLAG | TMADM_GFLAG : -s -g 옵션과 동일하고, count를 지정해야 한다.
- TMADM_TFLAG : -t 옵션과 동일하고, count를 지정해야 한다.
- TMADM_USFLAG : -S 옵션과 동일하다.
- TMADM_USFLAG | TMADM_GFLAG : -S -g 옵션과 동일하다.
- 위의 모든 경우 | TMADM_FFLAG : -i 옵션과 동일하다. TMADM_DOWN 에서만 사용 가능하다.

◦ TMADM_SUSPEND(TMADM_RESUME)

옵션 flags	설명
TMADM_SFLAG	서비스를 suspend/resume한다.
TMADM_VFLAG	서버를 suspend/resume한다.

◦ TMADM_RESTAT

옵션 flags	설명
TMADM_VFLAG	해당 서버의 통계 정보를 초기화한다.
TMADM_AFLAG	모든 서버의 통계 정보를 초기화한다.

◦ TMADM_SVC_STAT (TMADM_SPR_STAT, TMADM_SVR_STAT)

옵션	설명
TPNOFLAGS	해당 노드의 모든 통계치를 전달한다.
TMADM_SFLAG	arg→header.opt_char에 지정된 이름에 matching되는 통계치만 전달한다.
TMADM_CFLAG	arg→header.opt_int에 지정된 CLH 번호에 해당되는 CLH의 통계치만 전달한다.

◦ TMADM_SVC_STAT_EX (TMADM_SPR_STAT_EX)

옵션	설명
TPNOFLAGS	해당 노드의 모든 통계치를 전달한다. TMADM_SVC_STAT의 항목에 fail_count, error_count, mintime, maxtime 항목이 추가되었다. tmadmin 에서 stat 명령의 -x 옵션과 같은 기능을 제공한다.
TMADM_SFLAG	arg→header.opt_char에 지정된 이름에 matching되는 통계치만 전달한다.
TMADM_CFLAG	arg→header.opt_int에 지정된 CLH 번호에 해당되는 CLH의 통계치만 전달한다.

◦ TMADM_SMTRC

옵션 flags	설명
TPNOFLAGS	tmadm_smtrc 구조체를 사용한다.
TMADM_AFLAG	tmadm_smtrcall 구조체를 사용한다. 기존 정보 외에 spri, reserved, curtime, svctime, ucputime, scputime 등의 정보를 추가로 제공한다.

◦ TMADM_TMAX_INFO, TMADM_DOMAIN_CONF, TMADM_NODE_CONF, TMADM_SVG_CONF, TMADM_SVR_CONF, TMADM_SVC_CONF

옵션 flags	설명
TPNOFLAGS	TPNOFLAGS만 사용 가능하다.

- 반환값

반환값	설명
0보다 큰 수	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tmadmin()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 NULL 형식인 경우에 발생한다. 두 번째 파라미터인 구조체의 멤버 중 tmadmin_header의 멤버변수에 넣을 수 있는 값인 size와 offset, opt_char가 잘못된 경우에도 발생한다.
[TPENOREADY]	TMADM_SVC_STAT/SPR_STAT/SVR_STAT 명령어에서 TMADM_CFLAG를 사용한 경우 해당 CLH가 NOT READY 상태인 경우 발생한다. TMADM_DISCON 명령어에서 해당 클라이언트가 존재하지 않거나, 서비스 요청 중인 상태이면서 TMADM_FFLAG를 지정하지 않은 경우에 발생한다.
[TPEOS]	운영 시스템에 에러가 발생하였다. 주로 내부 버퍼 allocation 에러인 경우가 많다.
[TPESYSTEM]	Tmax 시스템 에러로 주로 통신 장애인 경우가 많다.
[TPESVCFAIL]	TMADM_QPURGE, TMADM_BOOT/DOWN, TMADM_SUSPEND/RESUME 명령어가 TPEINVAL, TPEOS, TPESYSTEM 이외의 이유로 실패한 경우에 발생한다.

- 예제

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

SERVICE(TPSVCINFO *msg)
{
   char cmd, *buf, *sndbuf;
   int len;
   long sndlen;

   cmd = msg->data[0];
   switch (cmd)
   {
      case TMADM_TMAX_INFO:
         len = tmaxinfo(&buf);
         break;
      case TMADM_SVC_STAT:
         len = svcstat(&buf);
         break;
      case TMADM_SVC_STAT_EX:
         len = svcstatex(&buf);
         break;
      case TMADM_SPR_STAT:
         len = sprstat(&buf);
         break;
      case TMADM_SPR_STAT_EX:
         len = sprstatex(&buf);
         break;
      case TMADM_QPURGE:
         len = qpurge(&buf);
         break;
      default:
         len = -1;
```

```
         break;
   }
   if (len < 0){ error processing }
   sndbuf = (char *)tpalloc("STRING", NULL, len + 1);
   if (sndbuf==NULL) {error processing }
   memcpy(sndbuf, buf, len);
   sndlen = len;
   tpreturn(TPSUCCESS, 0, sndbuf, sndlen, TPNOFLAGS);
}

int qpurge(char **buf)
{
   int n;
   /*n=tmadmin( TMADM_QPURGE, "TOUPPER",
   TMADM_SFLAG, TPNOFLAGS);*/
   n = tmadmin(TMADM_QPURGE, "toupper", TMADM_VFLAG, TPNOFLAGS);
   if (n < 0){ error processing   }
   ...
}

int tmaxinfo(char **buf)
{
   struct tmadm_tmax_info *info;
   ...
   size = sizeof(struct tmadm_tmax_info) +
   (MAX_NODE - 1) * sizeof (struct tmadm_node_summary);
   info = (struct tmadm_tmax_info *) malloc(size);
   if (info == NULL)}{ error processing }
   memset(info, 0x00, size);
   info->header.version = _TMADMIN_VERSION;
   info->header.size = size;

   n = tmadmin(TMADM_TMAX_INFO, info, TPNOFLAGS, TPNOFLAGS);
   if (n < 0)}{ error processing }
   ...

   /* 구조체 info의 내용을 정리하여 *buf에 넣는다.
      *buf의 크기를 반환한다. */
}

int svcstat(char **buf)
{
   struct tmadm_svc_stat info;
   struct tmadm_svc_stat *stat;
   ...
   memset(&info, 0x00, sizeof(struct tmadm_svc_stat));
   info.header.version = _TMADMIN_VERSION;
   info.header.size = sizeof(struct tmadm_svc_stat);
   n = tmadmin(TMADM_SVC_STAT, &info, TPNOFLAGS, TPNOFLAGS);
   if (n < 0){ error processing }

   svccount = info.header.num_entry + info.header.num_left;
   size = sizeof(struct tmadm_svc_stat)+ ((svccount - 1) *
          sizeof (struct tmadm_svc_stat_body));
   stat = (struct tmadm_svc_stat *) malloc(size);
   if (stat == NULL){ error processing }

   memset(stat, 0x00, size);
   stat->header.version = _TMADMIN_VERSION;
```

```
   stat->header.size = size;

   n = tmadmin(TMADM_SVC_STAT, stat, TPNOFLAGS, TPNOFLAGS);
   if (n < 0){ error processing }
   ...

   /* 구조체 stat의 내용을 정리하여 *buf에 넣는다.
      *buf의 크기를 반환한다. */
}
int svcstatex(char **buf)
{
   struct tmadm_svc_stat_ex info;
   struct tmadm_svc_stat_ex *stat;
   ...
   memset(&info, 0x00, sizeof(struct tmadm_svc_stat_ex));
   info.header.version = _TMADMIN_VERSION;
   info.header.size = sizeof(struct tmadm_svc_stat_ex);
   n = tmadmin(TMADM_SVC_STAT_EX, &info, TPNOFLAGS, TPNOFLAGS);
   if (n < 0){ error processing }

   svccount = info.header.num_entry + info.header.num_left;
   size = sizeof(struct tmadm_svc_stat_ex)+ ((svccount - 1) *
          sizeof (struct tmadm_svc_stat_body_ex));
   stat = (struct tmadm_svc_stat_ex *) malloc(size);
   if (stat == NULL){ error processing }

   memset(stat, 0x00, size);
   stat->header.version = _TMADMIN_VERSION;
   stat->header.size = size;

   n = tmadmin(TMADM_SVC_STAT_EX, stat, TPNOFLAGS, TPNOFLAGS);
   if (n < 0){ error processing }
   ...

   /* 구조체 stat의 내용을 정리하여 *buf에 넣는다.
      *buf의 크기를 반환한다. */
}

int sprstat(char **buf)
{
   struct tmadm_spr_stat info;
   struct tmadm_spr_stat *stat;
   ...
   memset(&info, 0x00, sizeof(struct tmadm_spr_stat));
   info.header.version = _TMADMIN_VERSION;
   info.header.size = sizeof(struct tmadm_spr_stat);
   n = tmadmin(TMADM_SPR_STAT, &info, TPNOFLAGS, TPNOFLAGS);
   if (n < 0){ error processing }

   sprcount = info.header.num_entry + info.header.num_left;
   size = sizeof(struct tmadm_spr_stat) + (sprcount - 1) *
          sizeof (struct tmadm_spr_stat_body);
   stat = (struct tmadm_spr_stat *) malloc(size);
   if (stat == NULL)}{ error processing }

   memset(stat, 0x00, size);
   stat->header.version = _TMADMIN_VERSION;
   stat->header.size = size;
```

```
   n = tmadmin(TMADM_SPR_STAT, stat, TPNOFLAGS, TPNOFLAGS);
   if (n < 0))){ error processing }
   ...

   /* 구조체 stat의 내용을 정리하여 *buf에 넣는다.
      *buf의 크기를 반환한다. */
}
int sprstatex(char **buf)
{
   struct tmadm_spr_stat_ex info;
   struct tmadm_spr_stat_ex *stat;
   ...
   memset(&info, 0x00, sizeof(struct tmadm_spr_stat_ex));
   info.header.version = _TMADMIN_VERSION;
   info.header.size = sizeof(struct tmadm_spr_stat_ex);
   n = tmadmin(TMADM_SPR_STAT_EX, &info, TPNOFLAGS, TPNOFLAGS);
   if (n < 0){ error processing }

   sprcount = info.header.num_entry + info.header.num_left;
   size = sizeof(struct tmadm_spr_stat_ex) + ((sprcount - 1) *
          sizeof (struct tmadm_spr_stat_body_ex));
   stat = (struct tmadm_spr_stat_ex *) malloc(size);
   if (stat == NULL)}{ error processing }

   memset(stat, 0x00, size);
   stat->header.version = _TMADMIN_VERSION;
   stat->header.size = size;

   n = tmadmin(TMADM_SPR_STAT_EX, stat, TPNOFLAGS, TPNOFLAGS);
   if (n < 0))){ error processing }
   ...

   /* 구조체 stat의 내용을 정리하여 *buf에 넣는다.
      *buf의 크기를 반환한다. */
}
```

3.2.6. tmax_get_db_passwd

현재 Tmax가 접속하고 있는 데이터베이스의 username에 대한 비밀번호를 조회하는 함수이다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
int tmax_get_db_passwd (char *svgname, char *passwd, int type)
```

- 파라미터

파라미터	설명
svgname	비밀번호를 조회할 서버 그룹명을 지정한다. 서버 그룹은 반드시 XA 서버 그룹이어야 한다.
passwd	결과 정보가 담겨져 반환된다. passwd은 반환되는 정보를 담을 수 있을 만한 충분한 크기로 할당되어 있어야 한다.

파라미터	설명
type	현재 사용하고 있는 DBMS를 구별하기 위해 지정한다. 다음 중에 하나를 지정해야 한다. • ORACLE_TYPE • SYBASE_TYPE • INFORMIX_TYPE • DB2_TYPE

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
–1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tmax_get_db_passwd()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 서버 그룹명을 잘못 지정했거나, type을 usrinc/tmaxapi.h에 정의된 값 이외의 값으로 정의한 경우 또는 type에 INFORMIX_TYPE을 설정한 경우에 발생한다.
[TPEITYPE]	svgname에 NON-XA 서버 그룹을 지정했을 경우나 Tmax 환경 파일의 OPENINFO 정보가 잘못되었을 경우 이 에러가 발생한다.

- 예제

<환경 파일의 OPENINFO>

```
svg4    NODENAME = @HOSTNAME@, DBNAME = ORACLE,
OPENINFO =“ORACLE_XA+Acc=P/scott/tiger+SesTm=60+Sqlnet=tmaxi1”,
TMSNAME = tms_ora
```

<서버 프로그램>

```
#include <stdio.h>
...
SVC_GETPASSWORD( TPSVCINFO *msg )
{
    char passwd[30];
    printf(“\n   => use right..\n”);
    ret = tmax_get_db_passwd(“svg1”, passwd, ORACLE_TYPE);
    if(ret < 0)
```

```
        printf("tmax_get_db_passwd fail[%s]\n",tpstrerror(tperrno));
    else
        printf("\ntmax_get_db_passwd = %s\n", passwd);
    tpreturn( TPSUCCESS, 0, rcvbuf, 0, 0 );
}
```

<결과>

```
tmax_get_db_passwd = tiger
```

- 관련 함수

  tmax_get_db_usrname(), tmax_get_db_tnsname()

3.2.7. tmax_get_db_tnsname

현재 Tmax가 접속하고 있는 데이터베이스의 tnsname를 조회하는 함수이다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
int tmax_get_db_tnsname (char *svgname, char *tnsname, int type)
```

- 파라미터

파라미터	설명
svgname	tnsname을 알고자 하는 서버 그룹명을 지정한다. 서버 그룹은 반드시 XA 서버 그룹이어야 한다.
tnsname	결과 정보가 담겨져 반환된다. tnsname은 반환되는 정보를 담을 수 있을 만한 충분한 크기로 할당되어 있어야 한다.
type	현재 사용하고 있는 DBMS를 구별하기 위한 값이다. 다음 중에 하나를 지정해야 한다. • ORACLE_TYPE • SYBASE_TYPE • INFORMIX_TYPE • DB2_TYPE

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.

반환값	설명
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tmax_get_db_tnsname()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않은 경우로 서버 그룹명을 잘못 지정했거나, type을 usrinc/tmaxapi.h에 정의된 값 이외의 값으로 정의한 경우 또는 type에 INFORMIX_TYPE을 설정한 경우에 발생한다.
[TPEITYPE]	svgname에 NON-XA 서버 그룹을 지정했을 경우나 Tmax 환경 파일의 OPENINFO 정보가 잘못되었을 경우 이 에러가 발생한다.

- 예제

  <환경 파일의 OPENINFO>

```
svg4    NODENAME = @HOSTNAME@, DBNAME = ORACLE,
        OPENINFO = ORACLE_XA+Acc=P/scott/tiger+SesTm=60+Sqlnet=tmaxi1”,
        TMSNAME = tms_ora
```

  <서버 프로그램>

```
#include <stdio.h>
...
SVC_GETTNSNAME( TPSVCINFO *msg )
{
    char tnsname[30];
    printf(“\n   => use right..\n”);
    ret = tmax_get_db_tnsname(“svg1”, tnsname, ORACLE_TYPE);
    if(ret < 0)
        printf(“tmax_get_db_tnsname fail[%s]\n”,tpstrerror(tperrno));
    else
        printf(“\ntmax_get_db_tnsname = %s\n”, passwd);
    tpreturn( TPSUCCESS, 0, rcvbuf, 0, 0 );
}
```

  <결과>

```
tmax_get_db_tnsname = tmaxi1
```

- 관련 함수

  tmax_get_db_usrname(), tmax_get_db_tnsname()

3.2.8. tmax_get_db_usrname

현재 Tmax가 접속하고 있는 데이터베이스의 username을 조회하는 함수이다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
int tmax_get_db_usrname (char *svgname, char *usrname, int type)
```

- 파라미터

파라미터	설명
svgname	username을 알고자 하는 서버 그룹명을 지정한다. 서버 그룹은 반드시 XA 서버 그룹이어야 한다.
usrname	결과 정보가 담겨져 반환된다. 반환되는 정보를 담을 수 있을 만한 충분한 크기로 할당되어 있어야 한다.
type	현재 사용하고 있는 DBMS를 구별하기 위한 값이다. 다음 중에 하나를 지정해야 한다. • ORACLE_TYPE • SYBASE_TYPE • INFORMIX_TYPE • DB2_TYPE

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tmax_get_db_usrname()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않은 경우로 서버 그룹명을 잘못 지정했거나, type을 usrinc/tmaxapi.h에 정의된 값 이외의 값으로 정의한 경우 또는 type에 INFORMIX_TYPE을 설정한 경우에 발생한다.
[TPEITYPE]	svgname에 NONXA 서버 그룹을 지정했을 경우나 Tmax 환경 파일의 OPENINFO 정보가 잘못되었을 경우에 발생한다.

- 예제

<환경 파일의 OPENINFO>

```
svg4    NODENAME = @HOSTNAME@, DBNAME = ORACLE,
        OPENINFO = ORACLE_XA+Acc=P/scott/tiger+SesTm=60+Sqlnet=tmaxi1”,
        TMSNAME = tms_ora
```

<서버 프로그램>

```
#include <stdio.h>
...
SVC_GETUSRNAME( TPSVCINFO *msg )
{
    char usrname[30];
    printf(“\n   => use right..\n”);
    ret = tmax_get_db_usrname(“svg1”, usrname, ORACLE_TYPE);
    if(ret < 0)
        printf(“tmax_get_db_usrname fail[%s]\n”,tpstrerror(tperrno));
    else
        printf(“\ntmax_get_db_usrname = %s\n”, usrname);
    tpreturn( TPSUCCESS, 0, rcvbuf, 0, 0 );
}
```

<결과>

```
tmax_get_db_usrname = scott
```

- 관련 함수

  tmax_get_db_passwd(), tmax_get_db_tnsname()

3.2.9. tmax_get_svccnt

자신이 속한 서버의 서비스 개수를 반환하는 함수로 tmax_get_svclist() 함수에서 서비스 목록을 가져오기 위한 버퍼의 크기를 할당하는 데 사용할 수 있다.

- 프로토타입

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

- 반환값

반환값	설명
서비스 개수	함수 호출에 성공한 경우이다. 자신이 속한 서비스의 개수를 반환한다.

3.2.10. tmax_get_svclist

자신이 속한 서버의 서비스 목록을 가져오기 위한 함수이다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
int tmax_get_svclist(char *buf, int bufsize)
```

- 파라미터

파라미터	설명
buf	다음의 크기만큼 할당한 버퍼로 자신이 속한 서버가 가지는 서비스명을 받아온다. `서비스의 개수(tmax_get_svccnt()의 반환값) x XATMI_SERVICE_NAME_LENGTH`
bufsize	다음의 값이다. `서비스의 개수(tmax_get_svccnt()의 반환값) x XATMI_SERVICE_NAME_LENGTH`

- 반환값

반환값	설명
0	함수가 호출에 성공한 경우이다.
-1	함수가 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tmax_get_svclist()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	버퍼의 크기가 (서비스 개수 x XATMI_SERVICE_NAME_LENGTH)보다 작거나 NULL인 경우이다.

- 예제

  <서버 프로그램>

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

GETSVC(TPSVCINFO *msg)
{
    int     i;
    int len, ret, size;
```

```
    char *buf;
    char *ptr;

    printf("GETSVC service is started!\n");
    len = tmax_get_svccnt();
    if (len < 0) {
        printf("tmax_get_svccnt fail[%d]\n", tperrno);
        tpreturn(TPFAIL, 0, 0, 0, 0);
    }
    printf("SVCCNT = %d\n", len);

    size = len * XATMI_SERVICE_NAME_LENGTH;
    buf = malloc(size);
    if (buf == NULL) {
        printf("buf is NULL\n");
        tpreturn(TPFAIL, 0, 0, 0, 0);
    }
    ret = tmax_get_svclist(buf, size);
    if (ret < 0) {
        printf("tmax_get_svclist fail[%d]\n", tperrno);
        tpreturn(TPFAIL, 0, 0, 0, 0);
    }
    for (i = 0; i < len; i++) {
        ptr = buf;
        printf("%dth SVC[%s]\n", i, ptr);
        buf += XATMI_SERVICE_NAME_LENGTH;
    }
    free(buf);
    tpreturn(TPSUCCESS,0,(char *)msg->data, 0,0);
}
```

3.2.11. tmax_is_restarted

서버에서만 사용할 수 있는 함수로 Tmax AP 서버 루틴 내에서 자신이 속한 서버 프로세스가 비정상적으로 종료된 후 재기동되었는지의 여부를 판단할 수 있도록 하는 함수이다.

다음의 경우 함수가 재기동되었다고 판단한다.

- AP Runtime 에러(비정상 종료)
- 타임아웃: tpreturn(TPEXIT)
- tmdown -i -s AP를 실행한 후 재기동하였을 경우

tmax_is_restarted 함수에 대한 기본적인 설명은 다음과 같다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
int tmax_is_restarted(void);
```

- 반환값

반환값	설명
1	자신이 속한 서버 프로세스가 비정상 종료 후 재기동된 경우이다.
0	재기동되지 않은 일반 AP 서버일 경우이다.

- 예제

  <서버 프로그램>

```
ISRESTARTED(TPSVCINFO *msg)
{
    int ret;
    ret = tmax_is_restarted()
    if(ret == 1)
        printf(“restarted server process\n”);
    else
        printf(“normal server process\n”);
    tpreturn(TPSUCCESS, 0, (char *)NULL, 0, 0);
}
```

3.2.12. tmax_is_xa

현재 자신이 속한 서버가 XA인지 NON-XA인지 체크하는 함수이다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
int tmax_is_xa(void);
```

- 반환값

반환값	설명
XA_MODE(1)	현재 자신이 속한 서버가 XA 서버일 경우이다.
NONXA_MODE(0)	XA 모드가 아닌 경우이다.

- 예제

```
#include <stdio.h>
SVC_XA( TPSVCINFO *msg )
{
    ...
    ret = tmax_is_xa();
    if (ret < 0)
        printf(“\ntmax_is_xa func fail [%s]\n”, tpstrerror(tperrno));

    if(ret == 1) strcpy(mode, « XA_MODE »);
    else if(ret == 0) strcpy(mode, “NONXA_MODE”);
    else strcpy(mode, “unknown”);
```

```
    printf(“ => [SVC_XA] result of tmax_is_xa() : %s\n\n”, mode);

    tpreturn( TPSUCCESS, 0, rcvbuf, 0, 0 );
}
```

3.2.13. tmax_my_svrinfo

서버 프로세스의 시스템 설정 정보를 획득하는 함수이다. tmax_my_svrinfo()를 통해 얻어낼 수 있는 각종 정보는 서버 프로세스의 정적인 정보로서 프로세스 동작 중의 각종 상태값을 얻어내기 위해서는 **tmadmin()**를 사용해야 한다.

- 프로토타입

```
#include <tmaxapi.h>
int tmax_my_svrinfo (TMAXSVRINFO*)
```

- 파라미터

  Tmax 시스템 환경 파일에 등록되는 각각의 서버 프로세스는 환경 파일에서의 계층적인 정보(노드명, 서버 그룹명, 서버명 등)를 가지고 있다. 또한 한 노드에서 유일한 값으로 관리되는 서버 그룹 인덱스, 서버 인덱스, 서버 프로세스 일련 번호 등도 확인할 수 있다. 개발자는 이와 같은 값을 프로세스 관리 및 구분 등 여러 가지 용도로 사용할 수 있다.

  다음은 <tmaxapi.h>에 설정된 TMAXSVRINFO 구조체 정의이다.

```
typedef struct {
    int nodeno; /* node index */
    int svgi;   /* server group index;unique in the node */
    int svri;   /* server index; unique in the node */
    int spri;   /* server process index; unique in the node */
    int spr_seqno; /* server process seqno ; unique in the server */
    int min, max;    /* min/max server process number */
    int clhi;         /*  for RDP only, corresponding CLH id */
    char nodename[TMAX_NAME_SIZE];
    char svgname[TMAX_NAME_SIZE];
    char svrname[TMAX_NAME_SIZE];
    char reserved_char[TMAX_NAME_SIZE];
    /* for more detail use tmadmin API */
} TMAXSVRINFO;
```

멤버	설명
nodeno	Tmax 시스템에서 하나의 도메인 내에서 유일한 값으로 관리하는 노드 인덱스이다.
svgi	하나의 노드에서 유일한 값으로서 서버 그룹 인덱스이다.
svri	하나의 노드에서 유일한 값으로서 서버 인덱스이다.
spri	하나의 노드에서 유일한 값으로서 서버 프로세스 인덱스이다.
spr_seqno	서버 프로세스의 일련 번호이다.

멤버	설명
min , max	Tmax 시스템 환경 파일에 설정된 프로세스의 최소 개수와 최대 개수를 의미한다.
clhi	서버 프로세스의 타입이 RDP인 경우에만 해당되는 필드로서 해당 RDP 프로세스에 대응하는 CLH 번호를 나타낸다.
nodename	Tmax 시스템 환경 파일에 설정된 노드명이다.
svgname	Tmax 시스템 환경 파일에 설정된 서버 그룹명이다.
svrname	Tmax 시스템 환경 파일에 설정된 서버명이다.
reserved_char	현재는 사용되지 않고 향후 사용을 위한 예비용 파라미터이다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	서버 프로세스의 정보를 저장할 변수가 지정되지 않은 경우이다.

- 예제

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

SERVICE(TPSVCINFO *msg)
{
    int n;
    TMAXSVRINFO *info;

    info = (TMAXSVRINFO *)malloc(sizeof(TMAXSVRINFO));
    if (info == NULL) {
        tpreturn(TPFAIL, -1, NULL, 0, TPNOFLAGS);
    }
    n = tmax_my_svrinfo(info);
}
```

- 관련 함수

  tmadmin()

3.2.14. tmax_my_rminfo

자기 자신이 속한 서버 그룹의 OPENINFO 정보를 가져온다.

- 프로토타입

```
#include <tmaxapi.h>
int tmax_my_rminfo(RM_INFO_T *rminfo, int max_elem)
```

- 파라미터

파라미터	설명
rminfo	RM 정보를 저장할 구조체 배열 버퍼의 포인터이다.
max_elem	구조체 배열의 요소 개수이다. MRM은 서버 그룹을 최대 16개까지 지정 가능하다.

다음은 rminfo에 대한 설명이다.

```
Structure rminfo {
     int rmid
     int flags
     char dbname[DBNAME_SIZE + 1]
     char openinfo[RM_STRING_SIZE + 1]
     char closeinfo[RM_STRING_SIZE + 1]
};
```

멤버	설명
rmid	RM의 ID이다.
flags	현재는 사용하지 않는다.
dbname[DBNAME_SIZE + 1]	서버 그룹에 정의한 DB 이름이다. 대부분의 경우 ORACLE, TIBERO, DB2_64, SYBASE, INFOMIX, ALTIBASE, DB2_STATIC, MQ의 이름을 가진다. 사용자가 빌드로 정의한 이름을 가질 수도 있다. 그런 경우 $TMAXDIR/config/RM 파일에 지정한 이름을 사용해야 한다.
openinfo[RM_STRING_SIZE + 1]	openinfo를 가져온다.
closeinfo[RM_STRING_SIZE + 1]	closeinfo를 가져온다.

- 반환값

반환값	설명
rmcount	반환된 rm 수를 가져온다.

3.2.15. tmgetsmgid

서버에서 사용하는 SysMaster Trace 기능을 지원하기 위한 함수로 현재 자신의 GID를 얻어온다. 사용자는 GID를 이용하여 시스템 단위의 업무 추적을 가능하게 할 수 있다.

함수가 정상적으로 실행되기 위해서는 환경 파일 NODE 절의 SMSUPPORT 항목이 'Y'로 설정되어 있어야 한다. tmgetsmgid() 함수 외에 **tmadmin** 툴의 (st –p –x) 명령어를 이용하여 서버 프로세스의 GID를 알아낼 수도 있다.

GID의 구조는 다음과 같다. 함수가 성공적으로 수행하고 난 후에는 gid→gid1, gid→gid2, gid→seqno 항목이 담겨진 후 반환된다.

항목	설명
GID1(4Byte)	제품 내의 클라이언트별 고유 번호로(WebtoB의 경우 cli id) domain id, node id, hth #, slot id 등으로 제품에 접속한 클라이언트를 구별하기 위한 번호이다.
GID2 (4Byte)	상위 3Byte는 seq #, 하위1Byte는 제품 고유 ID로 구성된다.
SEQNO (4Byte)	상위 2Byte는 비동기 호출할 때 branch #로 사용되고 하위 2Byte는 모든 호출할 때 seq #로 사용한다.

- 프로토타입

```
#include <usrinc/tmadmin.h>
int tmgetsmgid(tmax_smgid_t *gid);
```

- 파라미터

반환값	설명
gid	제품 내의 클라이언트별 고유 번호이다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우로 gid가 NULL로 입력되는 경우이다.

- 예제

<svr02.c : TPNOFLAGS 사용>

```
#include <stdio.h>
#include <stdlib.h>
#include <usrinc/atmi.h>
#include <usrinc/tmadmin.h>
#include “../sdl/demo.s”

GETGID(TPSVCINFO *msg)
{
    tmax_smgid_t smgid;
    int ret;
    char *buf;

    buf = (char *)tpalloc(“STRING”, NULL, 0);
    if(buf == NULL)
        tpreturn(TPFAIL, -1, NULL, 0, 0);
    ret = tmgetsmgid(&smgid);
    memcpy(buf, (char *)&smgid.gid1, 4);
```

```
    memcpy(buf+4, (char *)&smgid.gid2, 4);
    memcpy(buf+8, smgid.seqno, 4);
    tpreturn(TPSUCCESS, 0, (char *)buf, strlen(buf), 0);
}

SMTRACE(TPSVCINFO *msg)
{
    struct tmadm_smtrc *smtrc;
    int max = 10, size;
    int gid1, gid2, n, i;
    struct smtrace *ptr;
    char *buf;

    buf = (char *)tpalloc("CARRAY", NULL, 0);
    if(buf == NULL)
       tpreturn(TPFAIL, -1, NULL, 0, 0);
    ptr = (struct smtrace *)msg->data;
    gid1 = ptr->gid1;
    gid2 = ptr->gid2;

    /*
    printf("SMTRACE start: %x %x\n", gid1, gid2);
    */

    size = sizeof(struct tmadm_smtrc) + (max-1) * sizeof(struct tmadm_smtrc_body);
    smtrc = (struct tmadm_smtrc *)malloc(size);
    if(smtrc == NULL)
    {
        printf("smtrc is null\n");
        tpreturn(TPFAIL, -1, NULL, 0, 0);
    }

    memset(smtrc, 0x00, size);
    smtrc->header.version = _TMADMIN_VERSION;
    smtrc->header.size = size;
    smtrc->header.reserve_int[0] = gid1;
    smtrc->header.reserve_int[1] = gid2;
    n = tmadmin(TMADM_SMTRC, smtrc, TPNOFLAGS, TPNOFLAGS);
    if(n < 0)
    {
        free(smtrc);
        tpreturn(TPFAIL, -1, NULL, 0, 0);
    }
    /*
    printf("smtrc->header.num_entry = %d\n", smtrc->header.num_entry);
    */
    for(i=0; i<smtrc->header.num_entry; i++)
    {
        sprintf(buf, "SMTRACE[%d] : gid[%x-%x-%x] seqno[%x] clhno[%x] status
        [%s] name[%s]\n",i, gid1, gid2, ptr->seqno,
        smtrc->trc[i].seqno,
        smtrc->trc[i].clhno,
        smtrc->trc[i].status,
        smtrc->trc[i].name);
    }

    free(smtrc);
    tpreturn(TPSUCCESS, 0, (char *)buf, strlen(buf), 0);
}
```

+ <svr02_a.c : TMADM_AFLAG 사용>

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

GETGID_A(TPSVCINFO *msg)
{
        tmax_smgid_t smgid;
        int ret;
        char *buf;

        buf = (char *)tpalloc("STRING", NULL, 0);
        if(buf == NULL)
           tpreturn(TPFAIL, -1, NULL, 0, 0);

        ret = tmgetsmgid(&smgid);

        memcpy(buf, (char *)&smgid.gid1, 4);
        memcpy(buf+4, (char *)&smgid.gid1, 4);
        memcpy(buf+8, smgid.seqno, 4);

        tpreturn(TPSUCCESS, 0, (char *)buf, strlen(buf), 0);
}

SMTRACE_A(TPSVCINFO *msg)
{
        struct tmadm_smtrcall *smtrcall;
        int max = 10, size;
        int gid1, gid2, n, i;
        struct smtrace *ptr;
        char *buf;

        buf = (char *)tpalloc("CARRAY", NULL, 0);
        if(buf == NULL)
           tpreturn(TPFAIL, -1, NULL, 0, 0);

        ptr = (struct smtrace *)msg->data;
        gid1 = ptr->gid1;
        gid2 = ptr->gid2;

/*
        printf("SMTRACE start: %x %x\n", gid1, gid2);
*/
        size = sizeof(struct tmadm_smtrcall) +
              (max-1) * sizeof(struct tmadm_smtrcall_body);
        smtrcall = (struct tmadm_smtrcall *)malloc(size);
        if(smtrcall == NULL)
        {
                printf("smtrcall is null\n");
                tpreturn(TPFAIL, -1, NULL, 0, 0);
        }

        memset(smtrcall, 0x00, size);
        smtrcall->header.version = _TMADMIN_VERSION;
        smtrcall->header.size = size;
```

```
        smtrcall->header.reserve_int[0] = gid1;
        smtrcall->header.reserve_int[1] = gid2;

        n = tmadmin(TMADM_SMTRC, smtrcall, TMADM_AFLAG, TMADM_AFLAG);
        if(n < 0)
        {
                free(smtrcall);
                tpreturn(TPFAIL, -1, NULL, 0, 0);
        }
        printf("smtrcall->header.num_entry = %d\n", smtrcall->header.num_entry);
        printf("smtrcall->header.num_left = %d\n", smtrcall->header.num_left);
        n = 0;
        for(i=0; i<smtrcall->header.num_entry + smtrcall->header.num_left; i++)
        {
                sprintf(buf+n, "SMTRACE[%d] : gid[%x-%x-%x] seqno[%x] clhno[%x]
                        status[%s] name[%s] spri[%d] curtime[%ld], svctime[%ld],
                        ucputime[%ld], scputime[%ld]\n",
                        i, gid1, gid2, ptr->seqno,
                        smtrcall->trc[i].seqno,
                        smtrcall->trc[i].clhno,
                        smtrcall->trc[i].status,
                        smtrcall->trc[i].name,
                        smtrcall->trc[i].spri,
                        smtrcall->trc[i].curtime.tv_sec,
                        smtrcall->trc[i].svctime.tv_sec,
                        smtrcall->trc[i].ucputime.tv_sec,
                        smtrcall->trc[i].scputime.tv_sec);
                n = n + strlen(buf);
        }
        free(smtrcall);
        tpreturn(TPSUCCESS, 0, (char *)buf, strlen(buf), 0);
}
```

<svr01.c>

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/tmadmin.h>
#include “../sdl/demo_sdl.h”

SDLTOUPPER(TPSVCINFO *msg)
{
    int     i, ret, cd;
    struct  smtrace *stdata;
    tmax_smgid_t smgid;
    char *buf;
    long rcvlen;

    buf = (char *)tpalloc(“CARRAY”, NULL, 0);
    ret = tmgetsmgid(&smgid);
    if(ret < 0)
        tpreturn(TPFAIL, -1, NULL, 0, 0);
    stdata = (struct smtrace *)msg->data;
    stdata->gid1 = smgid.gid1;
    stdata->gid2 = smgid.gid2;
    stdata->seqno = smgid.seqno;
```

```
    cd = tpacall(“SMTRACE”, msg->data, 0, 0);
    ret = tpgetrply(&cd, (char **)&buf, (long *)&rcvlen, 0);
    if(ret < 0)
        tpreturn(TPFAIL, -1, NULL, 0, 0);
    sleep(20);
    tpreturn(TPSUCCESS,0,(char *)buf, strlen(buf),0);
}
```

< client.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <usrinc/atmi.h>
#include <usrinc/tmadmin.h>
#include “../sdl/demo.s”

main(int argc, char *argv[])
{
    struct  smtrace *sndbuf, *rcvbuf;
    long    rcvlen, sndlen;
    int     ret;

    if (tpstart((TPSTART_T *)NULL) == -1){
        printf(“tpstart failed\n”);
        exit(1);
    }
    ...
    if (tpcall(“SDLTOUPPER”,(char *)sndbuf,0, (char **)&rcvbuf, &rcvlen, 0) == 1)
   {
        printf(“Can’t send request to service SDLTOUPPER =>\n”);
        tpfree((char *)sndbuf);
        tpfree((char *)rcvbuf);
        tpend();
        exit(1);
    }
    printf(“rcvbuf = %s\n”, rcvbuf);
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
}
```

tmadmin 툴에 대한 자세한 설명은 "Tmax Administration Guide"를 참고한다.

3.2.16. TPADMNTOI

tmadmin으로부터 admntoi 명령으로 들어오는 요청에 대해서 처리한다. 함수 원형만 존재하며 사용자가 이 원형대로 내부 처리 로직을 구현해야 한다. 구현 후에는 tpregancb API를 이용하여 등록을 해야 그 때부터 tmadmin에서 admnoti 명령어 사용 시

- 프로토타입

```
# include <tmaxapi.h>
typedef int (*TPADMNOTI)(int seqno, int len, char *args);
```

- 파라미터

파라미터	설명
seqno	tmadmin의 an 명령 요청에 대한 순번이다.
len	notification message의 문자열 길이이다.
args	notification message 문자열이 저장된 버퍼이다. 해당 버퍼는 return 되면 서버라이브러리에 의해 free 된다.

- 반환값

반환값	설명
0 이상	함수 호출에 성공한 경우이다.
음수	함수 호출에 실패한 경우이다.

- 예제

```
    ...
int admnoti(int reqno, int len, char *args);
void foo();
void bar();
TMAXSVRINFO svrinfo;

int tpsvrinit(int argc, char *argv[])
{
   tmax_my_svrinfo(&svrinfo);
   if (tpregancb(admnoti) < 0)
      error processing;
}

int admnoti(int reqno, int len, char *args)
{
   printf("svg[%s] svr[%s] spri[%d] reqno[%d] message[%s]",
      svrinfo.svgname, svrinfo.svrname, svrinfo.spri, reqno, args);
   if (strncmp(args, "foo", 3) == 0) {
      foo();
   } else if (strncmp(args, "bar", 3) == 0) {
      bar();
   } else {
      return -1;
   }
   return 0;
}
```

- 관련 함수

tpretgancb, tpunregancb

3.2.17. tmget_smtrclog

로깅 데이터를 구조체 버퍼에 저장하는 함수로 입력할 때 버퍼의 최대 로깅 건수를 설정하며 출력할 때 저장된 로깅 건수가 저장된다.

- 프로토타입

```
#include <usrinc/tmadmin.h>
int tmget_smtrclog(void *handle, tmax_smtrclog_t *buf, int *count)
```

- 파라미터

항목	설명
handle	로깅 서비스에서 전달받은 데이터의 포인터(msg→data)이다.
buf	로깅 데이터를 가져오기 위한 버퍼이다.
count	로깅 건수가 저장된다.

로깅 정보 구조체(tmax_smtrclog_t)는 아래와 같다.

< usrinc/tmadmin.h>

```
/* SysMaster Trace Log structure */
typedef struct {
        tmax_smgid_t gid;
        int     clhno;
        char    status[TMAX_NAME_SIZE];
        char    name[TMAX_NAME_SIZE];
        int     spri;
        int     reserved;
        struct timeval curtime;
        struct timeval svctime;
        struct timeval ucputime;
        struct timeval scputime;
} tmax_smtrclog_t;
```

멤버	설명
gid	SysMaster GID이다.
clhno	요청을 받은 CLH 번호이다.
status[TMAX_NAME_SIZE]	서비스의 상태이다.
name[TMAX_NAME_SIZE]	서비스명이다.

멤버	설명
spri	서버 프로세스의 인덱스이다.
reserved	현재 사용하지 않는다.
curtime	현재시간이다.
svctime	서비스 수행시간이다.
ucputime	user cpu time이다.
scputime	system cpu time이다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tmget_smtrclog()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	인자가 잘못 입력되었을 경우이다.
[TPESYSTEM]	노드가 SysMaster 설정이 되어 있지 않을 경우나 SysMaster log service 설정이 되어 있지 않을 경우 발생한다.
[TPELIMIT]	입력한 count가 실제 handle에 있는 건수보다 작을 경우 발생한다.

- 예제

<svr.c>

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

SMLOGSERVICE(TPSVCINFO *msg)
{
        tmax_smtrclog_t *smtrclog;
        int ret, count=0, i;
        char *buf;

        smtrclog = (tmax_smtrclog_t *)tpalloc("CARRAY", NULL, 1024);
        if(smtrclog == NULL)
        {
                printf("smtrclog tpalloc fail [%s]\n", tpstrerror(tperrno));
        }
        buf = (char *)tpalloc("STRING", NULL, 0);
        if(buf == NULL)
           tpreturn(TPFAIL, -1, NULL, 0, 0);
```

```
        memset(buf, 0x00, 1024);
        memset(smtrclog, 0x00, 1024);
        count = tmget_smtrclog_count((char *)msg->data);
        printf("\n############################\n\n");
        printf("tmget_smtrclog_count = %d\n", count);

        count = 100;
        ret = tmget_smtrclog(msg->data, smtrclog, &count);
        printf("count = %d\n", count);
        for(i=0; i<count; i++)
        {
           printf("smtrclog[%d].gid = %d-%d-%d\n", i, smtrclog[i].gid.gid1,
                   smtrclog[i].gid.gid2, smtrclog[i].gid.seqno);
           printf("smtrclog[%d].clhno = %d\n", i, smtrclog[i].clhno);
           printf("smtrclog[%d].status = %s\n", i, smtrclog[i].status);
           printf("smtrclog[%d].name = %s\n", i, smtrclog[i].name);
           printf("smtrclog[%d].spri = %d\n", i, smtrclog[i].spri);
           printf("\n");
        }
        printf("############################\n\n");
        strcpy(buf, "success\n");
        tpreturn(TPSUCCESS, 0, (char *)buf, 0, 0);
}
```

3.2.18. tmget_smtrclog_count

데이터의 개수를 반환하는 함수로 현재 로깅할 데이터의 개수를 반환한다.

- 프로토타입

```
##include <usrinc/tmadmin.h>
int tmget_smtrclog_count(void *handle)
```

- 파라미터

항목	설명
handle	로깅 서비스에서 전달받은 데이터의 포인터(msg→data)이다.

- 반환값

반환값	설명
현재 로깅된 데이터의 개수	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tmget_smtrclog_count()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	인자가 잘못 입력되었을 경우이다.
[TPESYSTEM]	노드가 SysMaster 설정이 되어 있지 않을 경우나 SysMaster log service 설정이 되어 있지 않을 경우 발생한다.

- 예제

  tmget_smtrclog의 예제를 참고한다.

3.2.19. tpadvertise

서버 프로세스가 제공하는 서비스를 서버에 등록(advertise)하는 함수로 특정 서비스명을 등록할 수 있다. 특정 서비스를 등록할 경우 TMM이 관리하는 서버별 서비스의 이름 테이블에 해당 서비스명을 등록할 수 있다. 등록된 서비스가 호출될 경우 func이 가리키는 함수가 호출되어 실행된다. 등록은 서버 프로세스로 하여금 그 서버가 제공하는 새로운 서비스를 등록할 수 있도록 한다.

Tmax 5 SP1 Fix4(r7000), Tmax 5 SP2 Fix1 이전 버전에서는 환경설정에 등록되어 있었던 서비스명에 대해서만 처리가 가능하고, 새로운 서비스를 등록하거나 해제할 수 없었다. 언급된 버전 부터 새로운 서비스에 대해서도 등록/해제가 가능해졌다.

다음은 각각의 다양한 경우에 동작하는 방식에 대한 설명이다.

- **환경설정 파일에 등록된 서비스**

  tpunadvertise 수행을 했었다면 다시 서비스할 수 있는 상태로 변경한다. 다른 서버에서 tpadvertise를 실행하면 실패한다.

- **mksvr로 등록된 서비스**

  tpunadvertise 수행을 했었다면 다시 서비스할 수 있는 상태로 변경한다. 다른 서버에서 tpadvertise를 실행하면 실패한다. 서버 프로세스가 모두 종료하면 tpadvertise했던 서비스는 모두 자동 삭제된다.

- **새롭게 등록하는 서비스**

  새로운 서비스를 등록한다. tpunadvertise 수행을 했었다면 다시 서비스할 수 있는 상태로 변경한다. 다른 서버에서 tpadvertise를 실행하면 실패한다. 서버 프로세스가 모두 종료하면 tpadvertise했던 서비스는 모두 자동 삭제된다. 새롭게 등록하는 서비스의 경우 제약사항은 SVCTIMEOUT과 AUTOTRAN을 설정할 수 없다.

다음은 함수 사용법과 예제에 대한 설명이다.

- 프로토타입

```
#include <atmi/usrinc.h>
int tpadvertise(char svcname, void (func)(TPSVCINFO *));
```

- 파라미터

파라미터	설명
svcname	등록할 서비스명이다.
(func)(TPSVCINFO *)	호출할 함수명이다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpadvertise()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	서비스명이 NULL이거나 func이 NULL이다.
[TPELIMIT]	서비스명이 지정된 길이를 초과하여 등록되었다. 또는 해당 노드의 MAXSVC 개수에 도달하여 더이상 새로운 서비스를 등록할 수 없다.
[TPEMATCH]	해당 서비스가 이미 다른 함수로 advertise되었다. 또는 다른 서버에서 이미 등록한 서비스를 등록하려는 경우이다.
[TPEPROTO]	해당 함수가 부적절한 상황에서 호출되었다.
[TPESYSTEM]	Tmax 시스템에 에러가 발생하였다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

SVC2TN_NEWSVC1( TPSVCINFO *msg )
{
    ...
}

SVC2TN_NEWSVC2( TPSVCINFO *msg )
{
    ...
}

SVC2TN_1( TPSVCINFO *msg )
{
    int     ret;
    char    input[MAXLEN];
    void    (*func)(TPSVCINFO*);

    memset(input, 0x00, MAXLEN);
    strncpy(input, msg->data, msg->len);
```

```
    func = SVC2TN_NEWSVC1;
    ret = tpadvertise(input, func);
    if (ret < 0) {
        tpreturn(TPFAIL, 0, (char*)msg->data, msg->len, 0);
    }
    tpreturn(TPSUCCESS, 0, (char*)msg->data, msg->len, 0);
}
```

advertise / unadvertise 함수에 대한 자세한 내용은 Tmax Administration Guide의

3.2.20. tpcancelctx

tpsavectx()로 저장된 클라이언트 정보들 중 해당 구조체의 내용을 취소하는 함수이다. tprelay()를 수행하지 않아도 서비스 루틴이 종료되면 정상적으로 결과가 반환된다.

tpgetctx()는 서비스 루틴 내에서만 사용할 수 있다.

- 프로토타입

```
#include <ucs.h>
int tpcancelctx(CTX_T *ctxp);
```

- 파라미터

파라미터	설명
ctxp	라이브러리 내부에 저장된 CTX_T 구조체의 내용을 삭제한다. • CTX_T 구조체의 정의 `typedef struct { int version[4]; char data[CTX_USR_SIZE - 16]; } CTX_T;`

- 예제

```
RELAY_SVC(TPSVCINFO *msg) {
    .....
    ctxp = (CTX_T *)tpsavectx();
    ret=tpcancelctx(ctxp);
    if (ret<0) {
          error process routine
     }
     .....
     tpreturn(TPFAIL, sqlca.sqlcode, NULL, 0, TPNOFLAGS);
}
```

3.2.21. tpchkclid

ID에 해당하는 클라이언트가 해당 서버 프로세스가 위치한 노드에 접속한 상태인지를 확인하는 함수이다. 주로 RDP 방식의 서버 프로그램을 개발할 때 서비스 루틴에서 접속한 클라이언트 ID를 저장하고 usermain() 루틴에서 tpsendtocli()로 메시지를 보내는 경우에 사용하면 불필요한 에러를 사전에 방지할 수 있다.

RDP 방식에서는 서버 프로세스가 위치한 노드에 직접 연결된 상태가 아니면 tpsendtocli()를 사용할 수 없다.

- 프로토타입

```
#include <tmaxapi.h>
int tpchkclid(int clid)
```

- 파라미터

파라미터	설명
clid	클라이언트 번호로 **tpgetclid()**를 사용해서 얻어내는 값이다.

- 반환값

반환값	설명
-2	해당 클라이언트가 로컬 노드에 접속된 클라이언트가 아닌 경우이다.
-1	해당 클라이언트가 접속되어 있지 않은 경우이다.
1	해당 클라이언트가 정상적으로 접속된 경우이다.

- 오류

  tpchkclid()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	클라이언트가 로컬 노드에 접속되어 있지 않은 경우이거나 입력된 클라이언트 번호가 잘못된 값이다.
[TPENOREADY]	클라이언트가 정상적으로 접속되어 있지 않은 상태이다.

- 예제

```
int _discon(char **buf)
{
    int clid, n;
    clid = tpgetclid();
    n = tpchkclid(clid);
    if (n < 0) {
        printf("Invalid Client\n");
```

```
        return -1;
    }
   ...
}
```

- 관련 함수

  tpgetclid()

3.2.22. tpclrcliwatcher

tpsetcliwatcher()으로 등록한 clid에 대해서 종료 이벤트 감지를 취소한다. tpclrcliwatcher() 함수가 호출되고 나면 클라이언트가 종료되었을 때 더이상 콜백은 호출되지 않는다.

- 프로토타입

```
#include <usrinc/ucs.h>
int tpclrcliwatcher(int clid, void **pargs, int flags)
```

- 파라미터

파라미터	설명
clid	종료 이벤트 감지를 취소할 clid를 입력한다.
pargs	tpsetcliwatcher()에서 입력한 args 변수의 값을 되돌려준다.
flags	현재 사용되지 않는다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpclrcliwatcher()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 clid 값이 클라이언트가 아니거나, 인자가 잘못되었다.
[TPENOENT]	입력한 clid에 대한 종료 이벤트를 찾지 못했다. 이미 콜백이 호출되었거나 해제가 완료된 상태일 수 있다.
[TPENOTREADY]	clid에 해당하는 클라이언트가 접속되지 않았다.
[TPEOS]	메모리 할당이 실패하여 요청이 실패하였다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.

- 예제

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

LOGOUT(TPSVCINFO *msg) {
        int clid, ret;
        void *args;
        time_t curtime;

        clid = tpgetclid();
        ret = tpclrcliwatcher(clid, &args, TPNOFLAGS);

        printf("[%s] clid:%#x, args:%p, clr_watch:%d(%s), data:%s\n",
                        msg->name, clid, args, ret, tpstrerror(tperrno), msg->data);

        tpreturn(TPSUCCESS, 0, NULL, 0, 0);
}
```

- 관련함수

  CliWatcherCallback(), tpsetcliwatcher()

3.2.23. tpclrfd

UCS 방식 프로세스 내부의 fdset의 소켓 FD를 off시키는 데 사용되는 함수로 UCS 방식 서버 프로세스의 외부 소켓을 스케줄링하는 경우에 사용한다.

- 프로토타입

```
#include <ucs.h>
int  tpclrfd (int fd)
```

- 파라미터

파라미터	설명
fd	off할 내부 fdset의 소켓이다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.

반환값	설명
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpclrfd()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <usrinc/ucs.h>
….
#define SERV_ADDR “168.126.185.129”
#define SERV_PORT 1500

int fd_read(int, char *, int);
extern int errno;

int usermain(int argc, char *argv[])
{
    ...
    int listen_fd, n, newfd;
    struct sockaddr_in my_addr, child_addr;
    socklen_t child_len;
    buf = tpalloc(“STRING”, NULL, 0);
    if (buf == NULL){ error processing }

    memset((void *)&my_addr, NULL, sizeof(my_addr));
    memset((void *)&child_addr, NULL, sizeof(child_addr));
    listen_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (listen_fd == -1){ error processing }
    my_addr.sin_family = AF_INET;
    inaddr = inet_addr(SERV_ADDR);
    my_addr.sin_port = htons((unsigned short)SERV_PORT);
    if (inaddr != -1){
        memcpy((char *)&my_addr.sin_addr, (char *)&inaddr, sizeof(inaddr));
    }

    ret = bind(listen_fd, (struct sockaddr *)&my_addr, sizeof(my_addr));
    if (ret == -1){ error processing }
    ret = listen(listen_fd, 5);
    if (ret == -1){ error processing }

    tpsetfd(listen_fd);
    ...
    while(1) {
        n = tpschedule(10);
```

```
        ...
        if (n == UCS_USER_MSG){
        if (tpissetfd(listen_fd)) {
            child_len = sizeof(child_addr);
            newfd = accept(listen_fd, &child_addr, &child_len);
            if (newfd == -1){ error processing }
            tpsetfd(newfd);
        }

        if (tpissetfd(newfd)){
            /* 소켓으로부터 버퍼를 읽는다 */
            fd_read(newfd, buf, 1024);
            ret = tpcall("SERVICE", (char *)buf, sizeof(buf), (char **)&buf,
                         (long *)&rlen, TPNOFLAGS);
            if (ret == -1){ error processing }
                ...
            ret = tpclrfd(newfd);
            if (ret == -1){ error processing }
            close(newfd);
        }
        ...
   }
   return 1;
}
```

- 관련 함수

  tpissetfd()

3.2.24. tpclrfd_w

UCS 방식 프로세스 내부의 Writable FDSET의 소켓 FD를 off시키는 데 사용되는 함수로 UCS 방식 서버 프로세스의 외부 소켓을 스케줄링하는 경우에 사용한다. UCS 방식 서버 프로세스의 FDSET에서 인자값으로 주어진 소켓 FD를 제거시키는 함수이다.

tpclrfd() 함수가 Readable FDSET에서 제거하는 반면에, 이 함수는 Writable FDSET에서 제거한다. UCS 방식 프로세스의 외부 소켓 스케줄링에 사용된다. FD는 Writable FDSET에서 제거할 소켓 FD 값이다.

tpissetfd_w(), tpsetfd_w(), tpclrfd_w() 함수는 tpschedule() / tpuschedule()의 UCS 스케줄러에 TMM, CLH 뿐 아니라 외부 호스트 / 클라이언트와 연결을 맺고 통신하는 사용자 소켓 FD까지도 스케줄링할 수 있다. 사용자가 지정한 소켓 FD에 보낼 메시지가 준비되면 tpschedule()은 UCS_USER_MSG를 반환하며, 어떤 소켓 FD에 메시지가 준비되었는지 알기 위해서는 tpissetfd_w()를 사용한다.

tpissetfd(), tpsetfd(), tpclrfd(), tpissetfd_w(), tpsetfd_w(), tpclrfd_w() 등의 소켓 FD 관련 매크로 함수들은 일반 네트워크 프로그램에서 사용하는 FD_SET, FD_CLR, FD_ISSET과 유사하다.

- 프로토타입

```
#include <ucs.h>
int  tpclrfd_w (int fd)
```

- 파라미터

파라미터	설명
fd	off할 내부 fdset의 소켓이다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpclrfd_w()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

  tpissetfd, tpsetfd, tpclrfd 함수의 예제를 참고한다.

- 관련 함수

  tpissetfd(), tpsetfd(), tpclrfd()

3.2.25. tpforward

서버에서 서비스 요청을 또 다른 서비스 루틴으로 전달하는 함수로 자신의 서비스 처리를 종료하고 클라이언트의 요청을 svc 서비스 루틴으로 전달한다.

tpforward()는 서비스 루틴에서 마지막으로 호출되는 것으로 **tpreturn()**처럼 작동한다. tpreturn()과 마찬가지로 tpforward()가 Tmax 시스템으로 정상적으로 반환되기 위해서 tpforward()는 Tmax 시스템이 제어하는 서비스 루틴 내에서 호출되어야 한다.

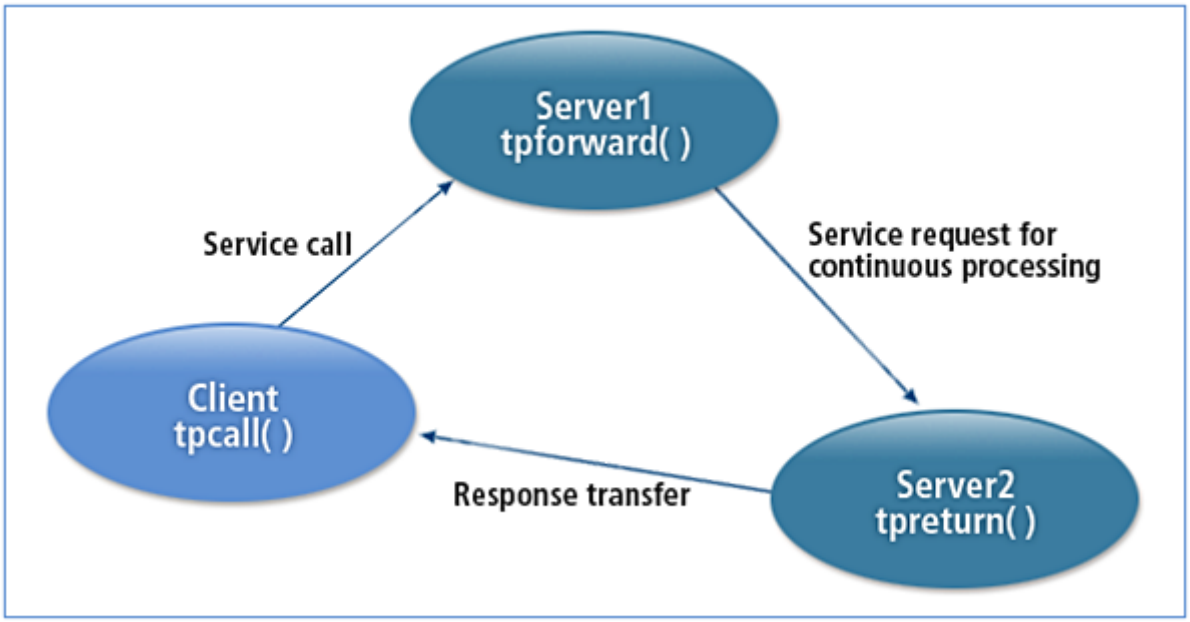
data가 가리키는 데이터를 사용하여 svc로 지정된 서비스에게 요청을 전달한다. 요청을 전달하는 서비스 루틴은 어떤 응답도 수신하지 않는다. 요청이 전달된 후 서비스 루틴은 Tmax 시스템에게로 반환한다. 그리고 서버는 자유롭게 다른 작업을 수행할 수 있다. tpforward()는 요청자로부터 아무 응답도 기대하지 않기 때문에 특별한 에러없이 어떤 서비스에게든지 전달될 수 있다.

서비스 루틴이 트랜잭션 모드에 있다면 그 트랜잭션은 트랜잭션 시작자(originator)가 tx_commit() 또는 tx_rollback() 중의 하나를 실행하여 트랜잭션을 완료할 때 비로소 완료된다. tpforward()는 tpreturn()과 마찬가지로 트랜잭션을 완료하지 않는다. 트랜잭션이 서비스 루틴 내에서 tx_begin()을 사용하여 시작된 것이라면 그 트랜잭션은 tpforward()의 호출 전에 tx_commit() 또는 tx_rollback() 둘 중의 하나로 먼저 완료되어야 한다. tpforward()로 연결된 모든 서비스들은 모두가 트랜잭션 모드이거나, 아니면 모두가 트랜잭션 모드가 아니어야

한다. 최종적으로 전달된 서버 프로세스가 tpreturn()을 이용하여 처음 서비스를 요청한 클라이언트에게 응답을 보낸다. tpforward()는 응답을 기다리고 있는 요청자에게 응답을 송신하는 책임을 다른 서버 프로세스에게 전가하는 것으로 멀티 노드 간에도 서비스가 이루어진다.

tpforward()는 서비스 루틴이 요청한 모든 서비스들에 대한 응답을 받은 후 호출되어야 한다. 수신되지 않은 응답에 대한 구별자들은 무효화되고 전달 요청은 전송되지 않는다. 대화형 서비스에서는 tpforward()가 호출될 수 없다.

다음은 tpforward 함수 흐름에 대한 그림이다.



tpforward

- 프로토타입

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

- 파라미터

파라미터	설명
svc	버퍼를 받을 서비스명이다.
data	NULL이 아니라면 tpalloc()에 의해 이전에 할당된 버퍼를 가리켜야 한다. 버퍼가 서비스 루틴에 송신된 것과 동일한 버퍼라면 Tmax 시스템이 이 버퍼에 대한 처리 책임을 갖는다. 서비스 루틴 작성자가 이 버퍼를 해제하려고 하면 이는 실패 처리된다. 그러나 tpforward()로 송신되는 버퍼가 서비스 호출할 때에 전달된 것과 동일한 버퍼가 아니라면 tpforward()가 그 버퍼를 해제한다.

파라미터	설명
len	송신될 데이터의 길이이다. data가 특별한 길이 명시가 필요없는 버퍼를 가리킨다면 (예를 들어, STRUCT 타입 버퍼), len은 무시되고 0이 된다. data가 NULL이라면 len은 무시되고 데이터 길이가 0인 요청이 송신된다. 서비스 루틴 작성자는 tpforward() 호출 후 다시 제어권을 획득할 수 없기 때문에 TPSIGRSTRT가 암시적으로 정의된 형태의 블로킹 송신이 사용된다. tpforward() 수행 중에 시그널이 발생하여 수행이 중지되더라도 나중에 재수행되며, 블로킹 상황을 만나더라도 타임아웃 발생 전까지는 기다린 후 송신한다.
flags	현재 버전에서는 지원하지 않으나 TPNOFLAGS로 설정한다.

- 반환값

  서비스 루틴은 호출자인 Tmax 시스템에서 어떤 값도 반환하지 않는다. 서비스 루틴은 void로 선언된다.

- 오류

  tpforward()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESVCERR]	유효하지 않은 버퍼를 사용하는 경우, 유효한 tpacall(), tpconnect()의 반환값으로 cd를 반환하는 경우 , 대화형 통신에서 tpreturn() 대신 tpforward()를 사용한 경우, TPEV_DISCONIMM 이벤트가 발생한 경우, 트랜잭션 모드에서 XA operation이 실패한 경우(tx_begin(), tx_rollback(), tx_commit()) 경우에 발생한다.
[TPETIME]	서비스 루틴 작업 중이나 또는 요청을 전송하는 중에 트랜잭션 타임아웃이 발생한 경우에 TPETIME 에러가 반환된다.

- 예제

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

SWITCH(TPSVCINFO *msg)
{
    int switch;
    char *buf;

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

    strcpy(buf, msg->data);
    data process…

    if (switch>5)
       tpforward(“SERVICE1”, buf, 0, 0);
    else
       tpforward(“SERVICE2”, buf, 0, 0);
}
```

- 관련 함수

  tpalloc(), tpconnect(), tpreturn()

3.2.26. tpgetctx

현재 클라이언트의 정보를 사용자가 선언하고 할당한 CTX_T 구조체에 복사해오는 함수이다. tpgetctx()를 사용했을 경우 tprelay()로 이 정보를 사용하지 않을 경우 해당 서비스 루틴이 완료되더라도 클라이언트는 계속 응답을 기다리게 된다.

tpgetctx()로 얻어진 정보는 tpcancelctx()로 취소할 수 없으므로 반드시 tprelay()를 사용해야 하고 서비스 루틴 내에서만 사용할 수 있다.

- 프로토타입

```
#include <tmaxapi.h>
int tpgetctx (CTX_T *ctxp)
```

- 파라미터

파라미터	설명
ctxp	tpsavectx()로 저장된 클라이언트의 정보를 CTX_T 구조체로 받아온다.

- 예제

```
RELAY_SVC(TPSVCINFO *msg)
{
    CTX_T *ctxp;
    ctxp=(CTX_T *)malloc(sizeof(CTX_T);
    ....
    ret = tpgetctx(ctxp);
    if (ret<0) {
        error process routine
    }
    .....
}
```

3.2.27. tpgetdbsessionid

RM 세션 정보를 얻기 위한 함수이다. RM의 세션 정보를 서비스에서 처리하는 경우 매번 처리해야 하는 불편함을 없애기 위하여 콜백 함수(tpsetdbsessionid)를 이용하여 RM 세션 정보를 설정할 수 있으며, 사용자는 tpgetdbsessionid를 통해서 현재 연결된 세션 ID를 얻어낼 수 있다.

- 프로토타입

```
#include <tmaxapi.h>
```

```
char * tpgetdbsessionid(int flags);
```

- 파라미터

파라미터	설명
flags	현재 버전에서는 지원하지 않으나 TPNOFLAGS로 설정한다.

- 반환값

반환값	설명
현재 연결된 세션 ID	함수 호출에 성공하는 경우이다.

- 예제

```
#include <usrinc/tmaxapi.h>
...
EXEC SQL include sqlca.h;
EXEC SQL begin declare section;
...
char  h_ssid[20];
EXEC SQL end declare section;

int tpsetdbsessionid(char dbsessionid[MAX_DBSESSIONID_SIZE], int flags)
{
    EXEC SQL SELECT TO_CHAR(USERENV('sessionid')) into :h_ssid FROM dual;
    if ( sqlca.sqlcode != 0 ){
        printf( "getting session id fail = %d\n",sqlca.sqlcode );
        return -1;
    }
    printf("RM session id = %s\n", h_ssid);
    memset(dbsessionid, 0x00, MAX_DBSESSIONID_SIZE);
    strcpy(dbsessionid, h_ssid);
    return 0;
}

FDLINS( TPSVCINFO *msg )
{
    char *ssid;
    int ret;
    int flags = 0;

    printf(" >>> current RM session id = %s\n", h_ssid);
    ssid = tpgetdbsessionid(flags);
    tpreturn( ... );
}
```

3.2.28. tpgetfclid

해당 함수를 호출한 서비스가 tpforward 또는 tprelay를 통해 호출된 N차 서비스일 경우 최소 서비스를 시작시킨 클라이언트 번호를 가져오는 함수이다.

- 프로토타입

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

- 반환값

반환값	설명
0 이상의 정수값	함수 호출에 성공하는 경우이다. 클라이언트의 번호에 해당하는 0 이상의 정수값을 반환한다.
-1	함수 호출에 실패한 경우이다. 함수를 호출한 시점에 서비스가 실행중이 아니라면 실패한다.

- 예제

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

SERVICE(TPSVCINFO *msg)
{
    int ret, clid;
    char *buf;

    buf = (char *)tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
    strcpy(buf, msg->data);
    data process…

    clid = tpgetfclid();
    if (clid==-1) { error process }

    ret=tpsendtocli(clid, buf, strlen(buf), 0);
    if (ret==-1) { error processing }
    data process….

    tpreturn(TPSUCCESS,0,buf, strlen(buf), 0);
}
```

- 관련 함수

  tpsendtocli()

3.2.29. tpgetmaxsvr

서버 프로세스의 최대 실행 개수를 반환하는 함수로 설정 파일의 SERVER 절에 정의한 서버 프로세스의 최대 실행 개수인 MAX 항목의 값을 얻어온다. 항상 자신의 서버 프로세스의 최대 개수만을 알 수 있다.

- 프로토타입

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

- 반환값

반환값	설명
정수값	함수 호출에 성공한 경우이다. 서버 프로세스의 최대 실행 개수를 반환한다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpgetmaxsvr()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

SERVICE(TPSVCINFO *msg)
{
    char *buf;
    buf = msg->data;

    data process...
    printf(“maxsvr : %d\n”,tpgetmaxsvr());
    data process...
    tpreturn(TPSUCCESS,0,buf, strlen(buf), 0);
}
```

- 관련 함수

  tpgetminsvr()

3.2.30. tpgetmaxuser

서버 프로세스가 속한 노드의 최대 동시 접속자 수를 반환하는 함수이다.

- 프로토타입

```
#include <tmaxapi.h>
```

```
int  tpgetmaxuser(void)
```

- 반환값

반환값	설명
정수	함수가 정상적으로 수행된 경우이다. 최대 동시 접속자 수를 반환한다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpgetmaxuser()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

SERVICE(TPSVCINFO *msg)
{
    char *buf;
    buf = msg->data;
    data process...
    printf("maxusr : %d\n",tpgetmaxusr());
    data process...
    tpreturn(TPSUCCESS,0,buf, strlen(buf), 0);
}
```

3.2.31. tpgetminsvr

서버 프로세스의 최소 실행 개수를 반환하는 함수로 설정 파일의 SERVER 절에 정의한 서버 프로세스의 최소 실행 개수인 MIN 항목 값을 반환한다. 항상 자신의 서버 프로세스의 최소 개수만을 알 수 있다.

- 프로토타입

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

- 반환값

반환값	설명
정수	함수 호출에 성공한 경우이다. 서버 프로세스의 최소 실행 개수를 반환한다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpgetminsvr()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

SERVICE(TPSVCINFO *msg)
{
    char *buf;
    buf = msg->data;
    data process...
    printf(“minsvr : %d\n”,tpgetminsvr());
    data process...
    tpreturn(TPSUCCESS,0,buf, strlen(buf), 0);
}
```

- 관련 함수

  tpgetmaxsvr()

3.2.32. tpgetmynode

서버에서 특정 노드명과 노드 번호를 얻는 함수이다.

- 프로토타입

```
#include <tmaxapi.h>
char *tpgetmynode(int *nodeno)
```

- 파라미터

파라미터	설명
nodeno	노드명은 반환값인 char *에 반환되고 노드 번호는 nodeno에 설정된다.

- 반환값

반환값	설명
노드명	함수 호출에 성공한 경우이다.
NULL	함수 호출에 실패한 경우이다. 에러가 발생되어도 tperrno에는 에러 번호가 설정되지 않는다.

- 예제

```
#include <usrinc/tmaxapi.h>
SERVICE(TPSVCINFO *msg)
{
    int i, clid, n;
    char *nodename;
    ...
    for (i = 0; i < msg->len; i++)
        msg->data[i] = toupper(msg->data[i]);
    ...
    if (nodename == NULL){  error processing }
    nodename = tpgetmynode((int *)&n);
    if (nodename == NULL){
        error processing
    }
    else {
        printf("TOUPPER SERVICE node(%dth node) name = %s\n", n, nodename);
    }
    ...
    tpreturn(TPSUCCESS,0,(char *)msg->data, 0,TPNOFLAGS);
}
```

3.2.33. tpgetmysvgno

서버에서만 사용할 수 있는 함수로 현재 자신이 속해 있는 서버 그룹의 번호를 알려준다. 보통 COUSIN으로 묶인 멀티 서버 그룹 환경에서 특정 서버 그룹에 속하는 서비스를 지정하여 서비스 요청을 송신하는 **tpcallsvg()**나 **tpacallsvg()**와 함께 사용되며, 자신이 속한 서버 그룹으로 호출하는 경우에 사용된다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
int tpgetmysvgno(void);
```

- 반환값

반환값	설명
그룹번호	함수 호출에 성공한 경우이다. 해당 함수를 호출한 서비스가 속한 서버 그룹의 번호를 반환한다.

- 예제

  <서버 프로그램>

```
FDLTOUPPER1(TPSVCINFO *msg)
{
    FBUF    *sndbuf, *rcvbuf;
    int ret, i, mysvgno;
    char sndata[30], rcvdata[30];
    char svc[XATMI_SERVICE_NAME_LENGTH];

    sndbuf = (FBUF *)tpalloc("FIELD", NULL, 0)};
    rcvbuf = (FBUF *)tpalloc("FIELD", NULL, 0))

    /* tpcallsvg (mysvgno) */
    ret = func_tpcallsvg_myno(sndbuf, rcvbuf, msg->name);
    if(ret < 0)
        tpreturn(TPFAIL, 0, (char *)NULL, 0, 0);
    tpreturn(TPSUCCESS, 0, (char *)rcvbuf, 0, 0);
}

int func_tpcallsvg_myno(FBUF *sndbuf, FBUF *rcvbuf, char *svc)
{
    int ret;
    char sndata[30], rcvdata[30];
    long rcvlen;
    int svgno;

    strcpy(sndata, "starbj81");
    ret = fbput(sndbuf, INPUT, sndata, 0);
    svgno = tpgetmysvgno();
    ret = tpcallsvg(svgno, "FDLTOUPPER2", (char *)sndbuf, 0, (char **)&rcvbuf,
                    &rcvlen, 0);
    if(ret < 0)
    {
      printf("tpcallsvg[%s] failed![%d][%s]\n",svc,tperrno, tpstrerror(tperrno));
      return -1;
    }
    fbprint(rcvbuf);
    fbinit(sndbuf, 1024);
    fbinit(rcvbuf, 1024);
    return 0;
}
```

- 관련 함수

  tpcallsvg(), tpacallsvg()

3.2.34. tpgetmysvrid

서버 프로세스 ID를 알려주는 함수로 서버 프로세스 ID는 설정 파일에 있는 SERVER 절의 서버 프로세스를 실행할 때 부여하는 번호이다. 같은 서버 프로세스를 여러 개 실행해도 다른 서버 프로세스 번호가 부여된다. 반환되는 값은 0부터이며 동작하는 서버 프로세스의 개수에 따라서 하나씩 증가된 값이 사용된다. tpgetmysvrid() 함수는 서비스

루틴(서버 프로세스)에서만 사용할 수 있다.

- 프로토타입

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

- 반환값

반환값	설명
정수	함수 호출에 성공한 경우이다. 서버 프로세스의 ID에 해당하는 0 이상의 정수 값을 반환한다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpgetmysvrid()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEPROTO]	클라이언트 프로그램에 사용되는 것과 같이 함수가 부적절한 상황에서 호출되었다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

SERVICE(TPSVCINFO *msg)
{
    char *buf;
    buf = msg->data;
    data process...
    printf("mysvr : %d\n",tpgetmysvrid());
    data process...
    tpreturn(TPSUCCESS,0,buf, strlen(buf), 0);
}
```

3.2.35. tpgetnodename

서버에서 지정된 노드명을 얻는 함수이다.

- 프로토타입

```
#include <tmaxapi.h>
```

```
char *tpgetnodename(int nodeno)
```

- 파라미터

파라미터	설명
nodeno	노드명을 조회할 노드 번호를 전달한다.

- 반환값

반환값	설명
노드명	함수 호출에 성공한 경우이다. 정상 처리된 경우 노드명을 반환한다.
NULL	함수 호출에 실패한 경우이다. tperrno에는 에러 번호가 설정되지 않는다.

- 예제

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

SERVICE(TPSVCINFO *msg)
{
    int i, clid, n;
    char *nodename;
    ...
    for (i = 0; i < msg->len; i++)
        msg->data[i] = toupper(msg->data[i]);

    nodename = (char *)tpgetnodename(n);
    if (nodename == NULL){
        error processing
    }
   else {
       printf("%dth node name(original node name) = %s\n", n, nodename);
   }
   ...
   tpreturn(TPSUCCESS,0,(char *)msg->data, 0,TPNOFLAGS);
}
```

3.2.36. tpgetnodeno

서버에서 nodename을 갖는 노드의 번호를 얻는 함수이다.

- 프로토타입

```
#include <tmaxapi.h>
int tpgetnodeno(char *nodename)
```

- 파라미터

파라미터	설명
nodename	노드 번호를 얻을 노드명을 전달한다.

- 반환값

반환값	설명
노드 번호	함수 호출에 성공한 경우이다. 정상 처리된 경우 노드 번호를 반환한다.
-1	함수 호출에 실패한 경우이다. tperrno에는 에러 번호가 설정되지 않는다.

- 예제

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

SERVICE(TPSVCINFO *msg)
{
    int i, clid, n;
    char *nodename;
    ...
    for (i = 0; i < msg->len; i++)
        msg->data[i] = toupper(msg->data[i]);

    n = tpgetnodeno(“tmax”);
    if (n < 0){
        error processing
    }
    else {
        printf(“%s’s node no(original node no) = %d\n”, “tmax”, n);
    }
    ...
    tpreturn(TPSUCCESS,0,(char *)msg->data, 0,TPNOFLAGS);
}
```

3.2.37. tpgetorgclh

해당 클라이언트가 현재 접속되어 있는 CLH 번호를 조회하는 함수이다.

- 프로토타입

```
#include <tmaxapi.h>
int tpgetorgclh(int clid)
```

- 파라미터

파라미터	설명
clid	현재 접속되어 있는 클라이언트 ID를 설정한다.

- 반환값

반환값	설명
CLH 번호	함수 호출에 성공한 경우이다. 정상 처리된 경우 해당 클라이언트가 현재 접속되어 있는 CLH 번호를 반환한다.
-1	함수 호출에 실패한 경우이다. tperrno에는 에러 번호가 설정되지 않는다.

- 예제

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

SERVICE(TPSVCINFO *msg)
{
    int i, clid, n;
    ...
    for (i = 0; i < msg->len; i++)
       msg->data[i] = toupper(msg->data[i]);

    n = tpgetorgclh((int)*(msg->cltid.clientdata));
    if (n < 0){
        error processing
    }
    else {
        printf(”clh number = %d\n”, n);
    }
    ...
    tpreturn(TPSUCCESS,0,(char *)msg->data, 0,TPNOFLAGS);
}
```

3.2.38. tpgetorgnode

해당 클라이언트가 접속된 노드 번호를 반환하는 함수이다.

- 프로토타입

```
#include <tmaxapi.h>
int tpgetorgnode(int clid)
```

- 파라미터

파라미터	설명
clid	현재 접속되어 있는 클라이언트 ID를 설정한다.

- 반환값

반환값	설명
노드 번호	함수 호출에 성공한 경우이다. 정상 처리된 경우 노드 번호를 반환한다.
-1	함수 호출에 실패한 경우이다. tperrno에는 에러 번호가 설정되지 않는다.

- 예제

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

SERVICE(TPSVCINFO *msg)
{
    int i, clid, n;
    char *nodename;
    ...
    for (i = 0; i < msg->len; i++)
        msg->data[i] = toupper(msg->data[i]);

    n = tpgetorgnode((int)*(msg->cltid.clientdata));
    if (n < 0){
        error processing
    }else {
        printf("original node number = %d\n", n);
    }
    ...
    tpreturn(TPSUCCESS,0,(char *)msg->data, 0,TPNOFLAGS);
}
```

3.2.39. tpgetpeer_ipaddr

서버에서 연결된 상대방의 소켓 주소를 얻어오는 함수로 Tmax 시스템에 연결이 완료된 후 상대방(노드)의 소켓 주소를 반환한다. 이 함수는 클라이언트가 하나의 서비스를 직접적으로 호출한 경우만 상대방의 IP 주소를 알 수 있다.

만약 클라이언트가 호출한 서비스가 다른 서비스를 호출하거나 다른 서비스로 서비스를 전달시키는 등 간접적으로 다른 서비스들을 거치는 경우에는 쓰레기 값이 들어올 수 있다.

- 프로토타입

```
#include <tmaxapi.h>
int  tpgetpeer_ipaddr(struct sockaddr *name,  int *namelen)
```

- 파라미터

파라미터	설명
name	주소가 저장된 구조체이다. IPv6 프로토콜 환경에서는 struct sockaddr_in6 구조체를 사용하여 주소 정보를 확인한다. 또한 struct sockaddr_storage 구조체를 사용하면 IPv4와 IPv6 환경에서 모두 사용할 수 있다.
namelen	함수 호출 전에 name으로 전달하는 구조체의 크기로 초기화해야 한다. 리턴에 성공한 경우에는 실제로 name에 할당된 구조체의 크기가 저장된다.

- 반환값

반환값	설명
소켓 주소	함수 호출에 성공한 경우이다. 상대방의 소켓 주소가 반환된다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpgetpeer_ipaddr()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPINVAL]	파라미터가 유효하지 않다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

```
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <usrinc/atmi.h>
#inlcude <usrinc/tmaxapi.h>
...

DELETE(TPSVCINFO *msg)
{
    struct sockaddr_in cli;
    char ipAddr[16];
    int cli_len, ret;

    data process...
    memset((char *)&cli, 0, sizeof(cli));
    ret = tpgetpeer_ipaddr((struct sockaddr *)&cli, &cli_len);
    if (ret == -1){
        error processing
    }
    else {
        memcpy(ipAddr, inet_ntoa(cli.sin_addr), 16);
    }
    printf(“ip = %s , port = %d\n”, ipAddr, ntohs(cli.sin_port));
    data process...

    tpreturn(TPSUCCESS, 0, 0, 0, 0);
}
```

다음은 IPv6 프로토콜 환경에서 사용 예이다.

```
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <usrinc/atmi.h>
#inlcude <usrinc/tmaxapi.h>
```

```
...

DELETE(TPSVCINFO *msg)
{
    struct sockaddr_storage cli_saddr;
    struct sockaddr_in *cli_sin4;
    struct sockaddr_in6 *cli_sin6;
    char ipaddrbuf[INET6_ADDRSTRLEN];
    const char *ipaddr = NULL;
    int cli_len, ret;
    int portno;

    data process...
    memset((char *)&cli_saddr, 0, sizeof(cli_saddr));
    cli_len = sizeof(cli_saddr);
    ret = tpgetpeer_ipaddr((struct sockaddr *)&cli_saddr, &cli_len);
    if (ret == -1){
        error processing
    }
    else {
        if (cli_saddr.ss_family == AF_INET) {
            cli_sin4 = (struct sockaddr_in *)&cli_saddr;
            ipaddr = inet_ntop(AF_INET, &(cli_sin4->sin_addr), ipaddrbuf,
                     sizeof(ipaddrbuf));
            portno = ntohs(cli_sin4->sin_port);
        } else if (cli_saddr.ss_family == AF_INET6) {
            cli_sin6 = (struct sockaddr_in *)&cli_saddr;
            ipaddr = inet_ntop(AF_INET6, &(cli_sin6->sin6_addr), ipaddrbuf,
                     sizeof(ipaddrbuf));
            portno = ntohs(cli_sin6->sin6_port);
        }
        if (ipaddr == NULL)
            ipaddr = "unknown";
    }
    printf(“ip = %s , port = %d\n”, ipaddr, portno);
    data process...

    tpreturn(TPSUCCESS, 0, 0, 0, 0);
}
```

- 관련함수

  tpgetpeername(), tpgetsockname(), tpstart()

3.2.40. tpgetsvcname

서비스 인덱스로부터 서비스명을 가져오는 함수로 응답 메시지가 폐기될 때 호출되는 Loss 서비스에서 TPSVCINFO의 cltid.clientdata[3]의 서비스 인덱스 값을 파라미터로 사용한다. 함수는 Tmax 서버에서만 사용할 수 있으며, Tmax 클라이언트에서는 사용할 수 없다. 반환되는 버퍼는 내부 Static 버퍼이므로 버퍼의 내용을 직접 변경하지 말고 별도의 버퍼에 복사하여 사용한다.

- 프로토타입

```
#include <tmaxapi.h>
```

```
char* tpgetsvcname(int svc_index)
```

- 파라미터

파라미터	설명
svc_index	서비스의 인덱스 값이다.

- 반환값

반환값	설명
서비스명	함수 호출에 성공한 경우이다. 서비스명이 저장된 버퍼의 주소값을 반환한다.
NULL	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpgetsvcname()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	svc_index가 잘못된 범위로 전달된 경우에 발생한다.
[TPENOENT]	해당하는 svc_index의 svc 이름이 존재하지 않는다.

- 예제

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

SVC01(TPSVCINFO *msg)
{
   int i;

   printf("\nSVC01 service is started!\n");
   printf("INPUT : data=%s\n", msg->data);

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

   printf("OUTPUT: data=%s\n", msg->data);
   sleep(10);
   tpreturn(TPSUCCESS,0,(char *)msg->data, 0,0);
}

LOSS_SVC(TPSVCINFO *msg)
{
   long svcindex;
   char *svcname;

   printf("\nLOSS_SVC service is started!\n");
   printf("INPUT : data = %s\n", msg->data);
   printf("TPERROR  : %d\n", msg->cltid.clientdata[1]);
```

```
   printf("TPURCODE : %d\n", msg->cltid.clientdata[2]);

   svcindex = msg->cltid.clientdata[3];
   printf("SVC INDEX Of Discarded Message : %ld\n", svcindex);

   svcname = tpgetsvcname((int)svcindex);
   if(NULL == svcname) {
      printf("tpgetsvcname is failed!!\n");
   } else {
      printf("SVC Name Of Discarded Message  : %s\n", svcname);
   }
   tpreturn(TPSUCCESS, 0, (char*)msg->data, 0, 0);
}
```

3.2.41. tpgetsvrseqno

같은 서버 프로세스들 간의 서버 프로세스에 대한 일련 번호를 반환한다. 환경설정 파일에 있는 SERVER 절의 MIN, MAX 항목과 관련되어 하나 이상 동작하는 서버 프로세스들을 구분하는 데 사용된다. 반환되는 값은 0부터이며, 동작하는 서버 프로세스의 개수에 따라서 하나씩 증가된 값이 사용된다.

예를 들어 'svr1'이란 이름의 서버 프로세스가 5개 동작하고 있다면, 각각 0부터 4까지의 일련번호가 부여된다. 3번에 해당하는 서버 프로세스가 종료되었다면 **tmboot -s**나 **autorestart**에 의해 다시 기동되었을 때 3번을 갖게 된다.

- 프로토타입

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

- 반환값

반환값	설명
0 이상의 정수	함수 호출에 성공한 경우이다. 서버 프로세스의 일련 번호에 해당하는 0 이상의 정수값을 반환한다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpgetsvrseqno()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

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

SERVICE(TPSVCINFO *msg)
{
    char *buf;
    printf("\nINPUT : %s\t processing svr no : %d\n", msg->data, tpgetsvrseqno());
    data process....
    tpreturn(TPSUCCESS, buf, 0, );
}
```

3.2.42. tpissetfd

서버에서 UCS 프로세스에서 소켓 FD로 데이터가 도착했는지를 검사하는 함수로 UCS 방식 서버 프로세스의 외부 소켓을 스케줄링하는 데 사용한다.

- 프로토타입

```
#include <ucs.h>
int  tpissetfd (int fd)
```

- 파라미터

파라미터	설명
fd	테스트할 fdset 내부의 FD를 설정한다.

- 반환값

반환값	설명
양수	메시지가 도착했을 경우이다.
0	메시지가 도착하지 않은 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpissetfd()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#include <errno.h>
#include <usrinc/ucs.h>
...

#define SERV_ADDR “168.126.185.129”
#define SERV_PORT 1500

int fd_read(int, char *, int);
extern int errno;

int usermain(int argc, char *argv[])
{
    ...
    int listen_fd, n, newfd;
    struct sockaddr_in my_addr, child_addr;
    socklen_t child_len;

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

    memset((void *)&my_addr, NULL, sizeof(my_addr));
    memset((void *)&child_addr, NULL, sizeof(child_addr));

    listen_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (listen_fd == -1){ error processing }

    my_addr.sin_family = AF_INET;
    inaddr = inet_addr(SERV_ADDR);
    my_addr.sin_port = htons((unsigned short)SERV_PORT);
    if (inaddr != -1)
        memcpy((char *)&my_addr.sin_addr, (char *)&inaddr, sizeof(inaddr));

        ret = bind(listen_fd, (struct sockaddr *)&my_addr, sizeof(my_addr));
        if (ret == -1){ error processing }
        ret = listen(listen_fd, 5);
        if (ret == -1){ error processing }

        tpsetfd(listen_fd);
        ...
        while(1) {
            n = tpschedule(10);
            ...
            if (n == UCS_USER_MSG){
                if (tpissetfd(listen_fd)) {
                    child_len = sizeof(child_addr);
                    newfd = accept(listen_fd, &child_addr, &child_len);
                    if (newfd == -1){ error processing }
                    tpsetfd(newfd);
            }
            if (tpissetfd(newfd)){
                /* 소켓으로부터 버퍼를 읽는다 */
                fd_read(newfd, buf, 1024);
                ret = tpcall(“SERVICE”, (char *)buf, sizeof(buf), (char **)&buf,
                             (long *)&rlen, TPNOFLAGS);
                if (ret == -1){ error processing }
                ...
                tpclrfd(newfd);
                close(newfd);
            }
```

```
        ...
        }
    }
    return 1;
}
```

- 관련함수

  tpissetfd(), tpsetfd()

3.2.43. tpissetfd_w

UCS 방식 서버 프로세스의 FDSET을 검사하여 파라미터값으로 주어진 소켓 FD에 보낼 데이터가 있는지 확인하는 함수이다. tpissetfd() 함수가 Readable FDSET을 검사하는 반면, 이 함수는 Writable FDSET을 검사한다. UCS 방식 프로세스의 외부 소켓 스케줄링에 사용된다. FD는 Writable FDSET에서 검사할 소켓 FD 값이다.

- 프로토타입

```
#include <ucs.h>
int tpissetfd_w(int fd)
```

- 파라미터

파라미터	설명
fd	테스트할 fdset 내부의 FD를 설정한다.

- 반환값

반환값	설명
양수	메시지가 도착했을 경우이다.
0	메시지가 도착하지 않은 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpissetfd()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	함수 호출에 성공한 경우이다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 관련함수

  tpissetfd(), tpsetfd()

3.2.44. tpprechk

RM의 상태 점검을 위한 사용자 콜백 함수로 Tmax 시스템 접속 전에 호출된다. RM의 상태를 확인하는 용도로만 사용해야 하며 그 외의 초기화 과정은 **tpsvrinit()**에서 처리해야 한다. tpprechk()에서 블록 상태가 되면 서비스는 READY 서버는 NOREADY 상태가 된다. **tmdown**의 [-i] 옵션을 사용하여 프로세스를 종료시킬 수 있다.

tpsvrinit()에서 RM 상태를 확인하는 도중 블록 상태에 빠지면 서비스 스케줄링이 되고 사용자는 서비스 타임아웃까지 대기하는데 이런 문제를 해결하기 위해 tpprechk()를 사용해야 한다.

- 프로토타입

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

- 반환값

반환값	설명
양수	함수 호출에 성공한 경우이다.
음수	함수 호출에 실패한 경우이다. 해당 서버를 재기동한다.

이 함수 안에서 tpsleep()를 사용하는 경우 TmaxTrace에서 tpsleep()을 로깅할 때 공유 메모리 초기화 과정 중이면 'server name'이 '$svr'로 표기된다. 또한 이 시점에 cll/tmm과 연결이 되어 있지 않다면 로깅 과정에서 TPEPROTO 에러가 발생한다.

3.2.45. tpregancb

tmadmin으로부터 admntoi 명령으로 들어오는 요청에 대해서 처리한다.

- 프로토타입

```
# include <tmaxapi.h>
int  tpregancb (TPADMNOTI)
```

- 파라미터

파라미터	설명
TPADMNOTI	콜백 함수이다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpregancb()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	callback에 지정된 함수 주소가 잘못되었다.

- 예제

```
    ...

int admnoti(int reqno, int len, char *args);
void foo();
void bar();
TMAXSVRINFO svrinfo;

int tpsvrinit(int argc, char *argv[])
{
   tmax_my_svrinfo(&svrinfo);
   if (tpregancb(admnoti) < 0)
      error processing;
}

int admnoti(int reqno, int len, char *args)
{
   printf("svg[%s] svr[%s] spri[%d] reqno[%d] message[%s]",
      svrinfo.svgname, svrinfo.svrname, svrinfo.spri, reqno, args);
   if (strncmp(args, "foo", 3) == 0) {
      foo();
   } else if (strncmp(args, "bar", 3) == 0) {
      bar();
   } else {
      return -1;
   }
   return 0;
}
```

- 관련 함수

  TPADMNOTI, tpunregancb

3.2.46. tpregcb

서버에서 UCS의 비동기형 요청에 대한 응답을 받는 루틴을 설정하는 함수로 UCS 방식 프로세스가 서버 프로그램으로부터 응답을 받았을 때 이를 처리할 루틴을 설정한다. UCS 방식 서버 프로세스에 **tpgetrply()** 대신 사용된다.

- 프로토타입

```
# include <ucs.h>
```

```
int  tpregcb (UcsCallback)
```

- 파라미터

파라미터	설명
UcsCallback	UCS에서 비동기형 요청에 대한 응답을 처리하는 콜백 함수를 지정한다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpregcb()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

void reply_receive(UCSMSGINFO *reply);
DUMMY(TPSVCINFO *msg)
{
    data process ….
}

int usermain(int argc, char *argv[])
{
    int ret;
    char *buf

    ret = tpregcb(reply_receive);
    if (ret == -1){ error processing }
    buf=tpalloc(“STRING”, NULL, 0);
    if (buf==NULL) { error processing }
    data process…
    while(1)
    {
        ...
        tpschedule(3);
        cd = tpacall(“SERVICE”, buf, strlen(buf), TPNOFLAGS);
        if (cd < 0) { error processing }
```

```
        ...
    }
}

void reply_receive(UCSMSGINFO *reply)
{
    printf("data....%s\n", reply->data);
}
```

- 관련 함수

  tpunregcb()

3.2.47. tprelay

UCS 형태의 서버 프로세스에서만 사용 가능한 함수로 서비스를 요청한 클라이언트에 대한 정보를 담아서 또 다른 서비스를 요청하는 방식으로 멀티 노드 간에도 서비스가 이루어진다. 이런 형식을 취하면 tprelay()를 통해서 호출된 서비스는 마치 클라이언트가 자기를 직접 호출한 것으로 인식하고 서비스의 수행 결과는 최초 서비스를 요청한 클라이언트에게 전달된다.

서비스 수행 결과를 호출한 클라이언트에게 전달할 수 있으므로 UCS 프로세스에서 간단한 구성을 통해 빠른 응답을 유도할 수 있다. 대체적으로 2~3번의 서비스를 호출해야만 결과를 얻어낼 수 있는 프로그램 루틴인 대외기간 업무와 연동하여 서비스를 처리하는 경우에 사용하는 것이 좋다.

만약 tpsavcctx() 또는 tpgetctx()를 통해 클라이언트 정보를 저장한 이후에 tprelay()를 통해 다른 서비스를 요청하지 않은 상태에서 서버 프로세스가 종료되는 경우에는 자동으로 서비스 호출자에게 에러 응답이 전달된다. 에러 응답의 전달과 관련해서는 환경설정 SERVER 절의 CTX_EREPLY 옵션을 참고한다. 이러한 동작은 Tmax 5 SP2 이후 버전에서부터 지원되며, 이전 버전에서는 이와 같은 상황에서 서비스 호출자에게 에러 응답이 전달되지 않았다.

- 프로토타입

```
#include <ucs.h>
int  tprelay(char *svc, char *data, long len, long flags, CTX_T *ctxp);
```

- 파라미터

파라미터	설명
svc	Tmax 환경 파일에 등록된 서비스명을 지정한다.
data	서비스를 호출할 때 전달되는 데이터로 NULL이 아닌 경우는 반드시 tpalloc()으로 할당된 버퍼를 사용해야 한다.
len	보내는 데이터의 길이를 지정한다. CARRAY, X_OCTET, 구조체 배열 타입일 경우에는 반드시 설정해야 한다.
flags	현재 지원하지 않는 파라미터로 TPNOFLAGS를 설정한다.
ctxp	tpgetctx() 또는 tpsavectx()로 받아온 정보 구조체이다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tprelay()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 ctxp가 NULL이거나 잘못된 버퍼를 사용한 경우에 발생한다.
[TPESYSTEM]	Tmax 시스템적인 에러가 발생하였다.

- 예제

```
...
#include <stdio.h>
#include <usrinc/ucs.h>
CTX_T *ctx = NULL;

DUMMY(TPSVCINFO *msg)
{
    data process ….
}

usermain(int argc, char *argv[])
{
    int ret, i;
    char *rcvbuf, *sndbuf;
    long sndlen;

    rcvbuf = (char *)tpalloc(“CARRAY”,NULL, 1024);
    if (rcvbuf == NULL){ error processing }
    i = 0;

    while(1) {
        tpschedule(1);
        if (ctx != NULL)
        {
            i++;
            if ((sndbuf = (char *)tpalloc(“CARRAY”,NULL, 1024)) == NULL)
            { error processing }
            else
            {
                 ...
                 ret = tprelay(“TPRETURN”, sndbuf, sndlen, 0, ctx);
                 if (ret==-1) { error processing }
                 data process...
                 ctx = NULL;
                 tpfree(sndbuf);
             }
```

```
        }
    }
}

int RELAY(TPSVCINFO *rqst)
{
   ...
   ctx = tpsavectx();
   tpreturn(TPSUCCESS, 0, rqst->data, rqst->len, 0);
}
```

- 관련 함수

  tpreturn(), tpforward()

3.2.48. tpresumetx

현재 중지된 전역 트랜잭션을 재개하는 함수로 중지된 전역 트랜잭션을 **tpresumetx(), tpsuspendtx()**를 통해 재개시킬 수 있다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
int tpresumetx(TRANID *tranid, int flags)
```

- 파라미터

파라미터	설명
tranid	tranid는 TRANID 구조체의 포인터이어야 하며 **tpresumetx(), tpsuspendtx()**를 호출할 때 사용했던 구조체의 포인터와 동일해야 한다.
flags	현재 버전에서는 지원하지 않으나 TPNOFLAGS로 설정한다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpresumetx()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	tranid 가 NULL 로 입력되었거나 flags가 TPNOFLAGS가 아닌 경우에 발생한다.
[TPEPROTO]	트랜잭션을 resume할 수 있는 상황이 아닌 경우에 발생한다.

에러 코드	설명
[TPETRAN]	트랜잭션 재시작을 실패한 경우에 발생한다.

- 관련 함수

  tpsuspendtx()

3.2.49. tpreturn

서버의 서비스를 종료하는 함수로 서비스 루틴의 완료를 의미한다. C 언어에서의 반환 문장과 같은 역할을 한다. tpreturn()이 호출되면 서비스 루틴은 Tmax 시스템에게 반환된다. Tmax 시스템으로 정상적으로 반환되기 위해서는 tpreturn()은 Tmax 시스템이 제어하는 서비스 루틴 내에서 호출되는 것이 바람직하다.

tpreturn() 함수는 서비스의 응답 메시지를 송신한다. 응답을 수신할 프로그램이 **tpcall()**, **tpgetrply()**, **tprecv()**로 응답을 기다리고 있다면 그 응답은 tpreturn() 호출이 성공한 후 수신자의 버퍼를 통해 전달된다.

tpreturn()은 대화형 서비스들이 대화형 연결을 종료할 수 있게 한다. 서비스 루틴은 직접 tpdiscon()을 호출할 수 없다. 올바른 결과를 보장하려면 대화형 서비스에 연결된 프로그램은 tpdiscon()을 호출하지 않는 것이 좋다. 그보다 대화형 서비스에서의 완료 통보 즉, tpreturn() 함수를 사용하여 송신되는 TPEV_SVCSUCC나 TPEV_SVCFAIL과 같은 이벤트들을 기다려야 한다.

서비스 루틴이 트랜잭션 모드에 있는데 해당 서비스를 호출한 클라이언트나 서비스가 명시적으로 트랜잭션을 시작하지 않으면(tx_begin을 사용하지 않으면) tpreturn은 트랜잭션의 한 부분으로서 TPSUCCESS인 경우 commit 또는 rollback된다. 서비스는 동일한 트랜잭션(Global Transaction)의 한 부분으로서 여러 번 호출될 수도 있다. 그래서 tx_begin을 사용한 트랜잭션 시작자가 tx_commit 또는 tx_rollback 중 하나를 호출하여 트랜잭션을 완료하기 전까지는 완전히 commit 또는 rollback되지 않는다.

tpreturn() 함수는 서비스 루틴에서 요청된 서비스들로부터 모든 응답을 수신한 후에 호출되어야 한다. 그렇지 않으면 서비스 특성에 따라 [TPESVCERR] 에러나 TPEV_SVCERR 이벤트가 서비스 루틴과 통신하는 프로그램에게 반환된다. 수신되지 않은 응답들은 Tmax 시스템에 의해 자동으로 무시된다. 또한 이런 응답들에 사용되는 구별자(descriptor)들은 무효화된다.

tpreturn() 함수는 대화형 통신에 사용되는 서비스에서 시작된 모든 연결을 종료한 후에 호출되어야 한다. 그렇지 않으면 서비스 특성에 따라 [TPESVCERR] 에러나 TPEV_SVCERR 이벤트 중의 하나가 서비스 루틴과 통신하는 프로그램에게 반환된다. 또한 강제적인 연결 해제 이벤트(TPEV_DISCONIMM)가 서비스와 연결된 모든 연결 종속자들에게 전달된다.

대화형 통신에서 서비스 루틴이 tpreturn()을 호출할 때 통신 제어권을 가지고 있지 않았다면 2가지 결과가 발생할 수 있다.

1. 서비스 루틴이 TPFAIL의 rval과 NULL의 data로 tpreturn()를 호출하였다면 대화 시작자에게 TPEV_SVCFAIL 이벤트가 전달된다.
2. 이와 다른 형태의 tpreturn()이 호출된다면 대화 시작자에게 TPEV_SVCERR 이벤트가 전달된다. 대화형 서비스는 서비스에서 시작하지 않은 대화형 연결을 하나만 가지고 있기 때문에 Tmax 시스템에서 데이터나 이벤트가 송신되어야 할 구별자를 알고 있다. 따라서 구별자(descriptor)는 tpreturn()에 파라미터로 전달되지 않는다.
   - 프로토타입

```
# include <atmi.h>
void  tpreturn (int  rval, long  rcode, char  *data, long  len, long  flags)
```

◦ 파라미터

파라미터	설명
rval	다음은 rval로 사용 가능한 값이다. • TPSUCCESS 서비스가 성공적으로 종료되었다. 데이터가 존재하고 tpreturn() 수행 중에 에러가 발생하지 않는다면 데이터는 송신된다. 호출자가 트랜잭션 모드에 있다면 이 트랜잭션의 한 부분을 commit이 가능한 상태로 결정한다. 트랜잭션이 최종적으로 완료될 때 그 트랜잭션에 속한 나머지 서비스들이 모두 성공적으로 완료되어 commit이 가능한 상태라면 commit을 하고, 하나라도 실패하면 rollback된다. tpreturn()에 대한 호출이 반드시 전체 트랜잭션을 완료하는 것이 아님에 유의해야 한다. 또한 호출자가 TPSUCCESS로 반환하더라도 대기 중인 응답이나 대화형 연결이 존재하거나 또는 서비스 내에서 행해진 어떤 작업이 트랜잭션을 rollback되도록 했다면 서비스 실패로 메시지가 송신된다. 응답의 수신자가 [TPESVCERR] 표시 또는 TPEV_SVCERR 이벤트를 수신한다. 서비스 루틴 내에서 트랜잭션이 rollback되면 rval은 TPFAIL로 설정됨에 유의해야 한다. 대화형 서비스에서 TPSUCCESS로 반환되면 TPEV_SVCSUCC 이벤트가 발생된다. • TPFAIL 서비스가 애플리케이션의 실패로 종료되었다. 응답을 수신하는 프로그램에 에러가 반환된다. 응답을 수신하는 호출이 실패하고 수신자는 [TPSVCFAIL] 값이나 TPEV_SVCFAIL 이벤트를 수신한다. 이 값은 데이터를 송신할 수 없다. TPFAIL 호출자가 트랜잭션 모드에 있고 autotransaction인 경우 tpreturn()은 트랜잭션을 rollback한다. 트랜잭션이 이미 Rollback 상태로 결정되어 있을 수도 있다. • TPEXIT 서비스 호출 후 반환할 경우 서버 프로세스를 강제 종료하고자 할 때 사용된다. tpexit()로 종료된 프로세스는 TMM에 의해 다시 시동된다. • TPDOWN TPEXIT와 비슷하나 TPDOWN으로 종료된 프로세스는 TMM에 의해 다시 기동되지 않는다. • TMSUCCESS TPSUCCExSS와 동일하다. • TMFAIL TPFAIL과 동일하다. 설명에 존재하지 않는 rval 값은 모두 TPFAIL로 간주된다.

파라미터	설명
rcode	애플리케이션에서 사용자에 의해 정의되는 반환값 rcode는 서비스 응답을 수신하는 프로그램에게 송신된다. 이 코드는 rval의 값과 상관없이 응답이 클라이언트로 무사히 송신될 수 있는 한(수신하는 호출이 성공하거나 [TPSVCFAIL]로 반환하거나 또는 TPEV_SVCSUCC 또는 TPEV_SVCFAIL 이벤트들 중 하나를 수신하는 한) 송신된다. rcode 값은 수신자에게 전역변수 **tpurcode**로 전달된다.
data	송신되는 응답 데이터를 가리킨다. data가 NULL이 아니라면 반드시 이전에 tpalloc()에 의하여 할당된 버퍼를 가리켜야 한다. 이것이 서비스 루틴에 전달된 것과 동일한 버퍼라면 Tmax 시스템에서 처리를 담당한다. 따라서 서비스 루틴 작성자는 이 버퍼에 대한 제거 여부를 신경 쓸 필요가 없다. 실제로 사용자가 이 버퍼를 제거하려 한다면 이는 실패한다. 그러나 tpreturn()으로 전달되는 버퍼가 서비스가 발생할 때와 동일한 버퍼가 아니라면 tpreturn()은 이 버퍼를 해제한다.
len	송신된 데이터 길이이다. data가 길이 명시가 필요 없는 버퍼를 가리킨다면 len은 무시되고 보통 0이 사용된다. data가 길이 명시가 필요한 버퍼를 가리킨다면 len은 0이 될 수 없다. data가 NULL이면 len은 무시된다. 이 경우 서비스를 호출한 프로그램이 응답을 기대하고 있다면, 아무 데이터도 없는 응답이 송신된다. 응답이 기대되지 않는다면 tpreturn()은 필요에 따라 data를 제거하고, 송신하는 응답 없이 반환한다.
flags	사용되지 않으며 반드시 0으로 설정하도록 한다. 서비스가 대화형이라면 데이터가 전달되지 않는 경우는 다음의 2가지이다. • tpreturn()을 호출할 경우에 대화형 연결이 이미 종료된 경우로 호출자가 TPEV_DISCONIMM 이벤트를 수신하였다. 이 경우 tpreturn()은 단순히 서비스 루틴을 종료하고, 트랜잭션 모드에 있다면 현재 트랜잭션을 rollback한다. 이 경우 호출자의 데이터는 전달될 수 없다. • 호출자가 통신 제어권을 갖고 있지 않다면 위에서 언급한 것처럼 TPEV_SVCFAIL 또는 TPEV_SVCERR 중 하나가 대화 시작자에게 송신된다. 대화 시작자가 수신하는 이벤트에 관계없이 어떤 데이터도 전달되지 않는다. 그러나 대화 시작자가 TPEV_SVCFAIL 이벤트를 수신하였다면 반환 코드는 시작자의 tpurcode 변수로 이용 가능하다.

◦ 반환값

서비스 루틴은 호출자인 Tmax 시스템에게 어떤 값도 반환하지 않는다. 서비스 루틴은 tpreturn()을 사용하여 종료되는 것이 원칙이다. 서비스 루틴이 tpreturn()을 사용하지 않고 예를 들어, C 언어의 반환 문장 등을 사용하여 반환한다면 서버는 서비스 요청자에게 서비스 에러를 반환한다. 대화형 통신을 위해 유지되어 있는 연결이 강제적으로 종료되고, 비동기적으로 기다리고 있는 응답들이 모두 무시된다.

서버가 트랜잭션 모드에 있었다면 그 트랜잭션은 rollback된다. 또한 tpreturn()이 서비스 루틴 외부에서 사용되었을 경우(예를 들어 서비스가 아닌 루틴에서 사용된 경우) 이는 아무런 일도 하지 않고 단순히 반환만 한다.

◦ 오류

tpreturn()이 서비스 루틴을 종료시키기 때문에 파라미터를 처리하는 중에 에러가 발생하면 호출자인 서비스 루틴에게 전달되지 않는다. 에러들은 다음과 같이 전달된다.

구분	설명
동기와 비동기 통신	tpcall() 또는 tpgetrply()으로 서비스 결과를 수신하는 프로그램에 대해서는 tperrno에 [TPESVCERR]이 전달된다.
대화형 통신	tpsend()나 tprecv()를 사용하는 프로그램에 대해서는 TPEV_SVCERR 이벤트를 발생시킨다.

◦ 예제

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

SERVICE1(TPSVCINFO *msg)
{
    char *buf;
    long len;

    buf=tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processsing }
    buf=msg->data;
    data process....

    ret=tpcall("SERVICE2", buf, sizeof(buf), &buf, &len, TPNOFLAGS);
    if (ret==-1) { error processing }
    data process....

    if (buf !="SUCCESS") {
        printf("svc fail..\n");
        tpreturn (TPFAIL, -1, NULL,0,0);
    }
    else {
        tpreturn(TPSUCCESS, 0, msg->data, msg->len, 0);
    }
}
```

◦ 관련 함수

tpalloc(), tpcall(), tpconnect(), tpdiscon(), tpgetrply(), tprecv(), tpsend()

3.2.50. tpsavectx

UCS 프로세스에서 사용되며 클라이언트의 정보를 내부적으로 관리하도록 한다. tpsavectx()는 **tprelay()** 함수와 함께 사용된다. tprelay()는 처리 결과를 다른 서비스로 전달하는 역할을 수행한다. 일반적인 서비스 프로그램에서 tpforward 형식으로 다른 서비스를 부른 것과 동일하게 동작한다. 결과적으로 호출된 서비스에서는 처리된 결과값을 해당 클라이언트에게 전달한다.

tpsavectx() 함수는 대외기관 업무와 같이 다른 프로토콜이 이용되며 시간이 많이 소요되어 채널이 묶일 수 있는 가능성이 많은 경우에 사용된다.

일반적으로 사용되는 형태는 다음과 같다.

```
클라이언트 → svc1 → svc2(service, tpsavectx) → 대외기관
클라이언트 ← svc3 ← svc2(usermain, tprelay) ← 대외기관
```

1. 클라이언트가 svc1에게 원하는 서비스를 요청한다.
2. svc1에서는 tpforward(...TPNOREPLY)를 이용해서 svc2를 호출한다.
3. svc2는 UCS 프로세스에 존재하는 서비스이며 서비스 루틴 내에서 tpsavectx()를 사용하여 클라이언트의 정보를 저장하며 대외기관과 통신한다.
4. 결과는 usermain에서 수신하며 이를 tprelay()를 통해서 svc3에게 전달한다. svc3는 svc2에서 tpforward로 자신을 호출한 것으로 간주하여 최종 결과를 주어진 클라이언트에게 전달한다.

svc1에서 서비스를 TPNOREPLY로 전달하고 있기 때문에 채널이 묶이는 것을 방지할 수 있어 작은 수의 업무 프로세스로도 많은 클라이언트를 감당할 수 있다. 또한 하나의 UCS 프로세스로서 송수신 프로세스 역할을 겸함으로써 비교적 단순한 시스템을 구성할 수 있다. 이는 시스템 관리 측면에서도 효율적이다.

- 프로토타입

```
#include <ucs.h>
CTX_T * tpsavectx(void)
```

- 반환값

반환값	설명
CTX_T	함수 호출에 성공한 경우이다.
NULL	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpsavectx()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEPROTO]	tpsavectx() 함수는 반드시 서비스 루틴 내에서 사용해야 한다. 서비스 루틴이 아닌 곳에 사용되는 경우에는 TPEPROTO 에러가 발생한다. **tpsvrinit()** 또는 **tpsvrdone()**에서는 사용될 수 없다.
[TPESYSTEM]	메모리 할당 오류가 발생하였다.

- 예제

```
...
#include <stdio.h>
```

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

CTX_T *ctx = NULL;

usermain(int argc, char *argv[])
{
    int ret, i;
    char *rcvbuf, *sndbuf;
    long sndlen;

    rcvbuf = (char *)tpalloc("CARRAY",NULL, 1024);
    if (rcvbuf == NULL){ error processing }

    i = 0;
    while(1) {
        tpschedule(1);
        if (ctx != NULL)
        {
            i++;
            if ((sndbuf = (char *)tpalloc("CARRAY",NULL, 1024)) == NULL)
            { error processing }
            else
            {
                ...
                ret = tprelay("TPRETURN", sndbuf, sndlen, 0, ctx);
                if (ret==-1) { error processing }
                data process...
                ctx = NULL;
                tpfree(sndbuf);
            }
        }
    }
}

int RELAY(TPSVCINFO *rqst)
{
    ...
    ctx = tpsavectx();
    tpreturn(TPSUCCESS, 0, rqst->data, rqst->len, 0);
}
```

- 관련 함수

  tpreturn(), tpforward(), tprelay()

3.2.51. tpschedule

UCS 형태의 서버 프로세스에서만 사용 가능한 함수로 UCS 서버 프로세스에서 데이터의 도착을 기다린다. 최대 타임아웃 시간 동안 sleep하다가 그 안에 데이터가 도착하면 즉시 반환한다.

tpschedule() 함수는 데이터가 도착했을 때 해당되는 서비스가 자동적으로 수행되고 난 후에 반환된다. 그러므로 데이터가 도착한 후에 사용자가 임의로 서비스를 수행하면 안 된다.

서비스는 무조건 시스템에 의해서 항상 수행되므로 UCS 형태의 서비스 프로그램이라도

이점을 주의해야 한다.

- 프로토타입

```
#include <ucs.h>
int  tpschedule(int timeout)
```

- 파라미터

파라미터	설명
timeout	기다릴 시간을 초 단위로 입력해야 한다. • -1 : 데이터가 도착했는지 체크만 한 후 즉시 반환한다. • 0 : 데이터가 도착할 때까지 무한 대기한다.

- 반환값

반환값	설명
양의 정수	함수가 수행에 성공해서 데이터가 도착한 경우이다.
-1	타임아웃 시간까지 데이터가 도착하지 않은 경우 또는 함수가 수행에 실패해서 에러가 발생한 경우이다. 타임아웃 시간까지 데이터가 도착하지 않으면 -1을 반환하고, tperrno에13번(TPETIME)이 설정된다. 그 외의 경우 tperrno에 에러 코드가 설정된다.

- 오류

tpschedule()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.
[TPETIME]	타임아웃 시간까지 데이터가 도착하지 않았다.
[TPEPROTO]	함수가 부적절한 상황에서 호출되었을 경우이다. 예를 들어 서비스 내에서 호출하는 경우 발생한다.

- 예제

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include <usrinc/ucs.h>
int usermain(int argc, char *argv[])
{
    ...
    while(1)
    {
```

```
        ...
        tpschedule(3);
        ret = tpcall("SERVICE", (char *)buf, strlen(buf), (char **)&buf,
                     (long *)&rlen, TPNOFLAGS);
        if (ret == -1) { error processing}
        ...
    }
}
```

- 관련 함수

  tpsleep(), tp_sleep(), tp_usleep()

3.2.52. tpsendtocli

서버에서 사용되는 함수로 지정된 클라이언트에게 비요청 메시지를 송신한다. **tpbroadcast()**는 Tmax 시스템에 접속되어 있는 임의의 클라이언트에 비요청 메시지를 전송한다. tpsendtocli()는 서버 프로세스에서 해당 서버 프로세스가 제공하는 서비스를 요청한 클라이언트에게만 비요청 메시지를 보내기 위해 사용한다.

- 프로토타입

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

- 파라미터

파라미터	설명
clid	tpgetclid()로 얻은 클라이언트의 유일한 번호이다.
data	tpalloc()에 의해 할당된 버퍼로 data가 길이를 명시하지 않아도 되는 버퍼로 설정된 경우 len은 무시되고 보통 0이 사용된다. data가 길이를 반드시 명시해야 하는 버퍼로 설정된 경우 len은 0이 될 수 없다. data가 NULL인 경우 len은 무시된다.
len	송신하는 버퍼의 길이이다.

파라미터	설명
flags	TPNOFLAG, TPUDP, TPFLOWCONTROL이 있으며 해당 flags에 따라 동작 방식이 결정된다. flags로 사용 가능한 값은 다음과 같다. • TPNOFLAG(0) 메시지는 클라이언트에게 수신되어야 한다. 하지만 클라이언트가 수신된 메시지를 빠르게 처리하지 못한다면 요청한 결과를 수신할 때 오랜 시간이 걸릴 수 있다. • TPUDP TPUDP flags는 클라이언트와 데이터를 통신하는 방식이 UDP라는 의미가 아니다. 호출자가 데이터를 송신할 때 송신할 내부 버퍼에 전달될 메시지가 가득 차서 보내지 못할 경우가 있다. 이 경우 데이터는 버려도 된다는 의미이다. 통신의 UDP처럼 데이터가 도중에 분실될 수 있다는 것을 의미이다. • TPFLOWCONTROL 클라이언트의 상태를 점검하고 다른 메시지를 요청할 수 있는지 확인한다. 해당 클라이언트로의 송신 메시지가 너무 많이 쌓여 있다면 tpsendtocli()는 –1을 반환하고 tperrno를 TPEQFULL로 설정한다. 이 flags는 Tmax 시스템의 부하를 줄여준다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpsendtocli()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEBADDESC]	clid가 유효하지 않다.
[TPEPROTO]	tpsendtocli()가 부적절한 상황에서 호출되었다.
[TPESYSTEM]	Tmax 시스템 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.
[TPEQFULL]	송신할 메시지가 있으므로 같은 메시지일 경우에는 재전송할 필요가 없다.

- 예제

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

```
SERVICE(TPSVCINFO *msg)
{
    int ret, clid;
    char *buf;

    buf = (char *)tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }
    strcpy(buf, msg->data);
    data process….
    clid = tpgetclid();
    if (clid==-1) { error processing }

    ret=tpsendtocli(clid, (char *)buf, 0, 0);
    if (ret==-1) { error processing }
    tpreturn(TPSUCCESS, 0, 0, 0);
}
```

- 관련 함수

  tpbroadcast()

3.2.53. tpsetcliwatcher

clid에 대응하는 클라이언트가 종료될 때 콜백함수가 호출되도록 설정한다. 클라이언트가 접속한 clh에 등록하는 과정이 끝나야 리턴된다. 콜백 함수가 호출되는 시점은 tpschedule()이 호출되고 클라이언트의 종료 이벤트가 감지되는 시점이다.

- 프로토타입

```
#include <usrinc/ucs.h>
int tpsetcliwatcher(int clid, CliWatcherCallback callback, void *args, int flags)
```

- 파라미터

파라미터	설명
clid	tpgetclid() 등으로 확인한 클라이언트의 id를 입력한다. 클라이언트가 아닌 서버 등의 clid를 입력하면 TPEINVAL 에러가 발생한다.
callback	클라이언트가 종료될 때 통지받을 함수를 위의 CliWatcherCallback 타입으로 생성하여 입력한다.
args	클라이언트가 종료될 때 콜백함수가 호출 될 때 인자로 받을 변수를 입력한다. 사용자가 자유롭게 정의할 수 있다.
flags	현재 사용되지 않는다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpsetcliwatcher()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 clid 값이 클라이언트가 아니거나, 인자가 잘못되었다.
[TPENOTREADY]	clid에 해당하는 클라이언트가 접속되지 않았다.
[TPEOS]	메모리 할당이 실패하여 요청이 실패하였다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.

- 예제

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

void logout_callback(int clid, int reason, void *args) {
        printf("[%s] clid:%#x, args:%p, reason:%d\n", "CALLBACK", clid, args, reason);
}

LOGIN(TPSVCINFO *msg) {
        int clid, ret;
        time_t curtime = time(NULL);

        clid = tpgetclid();
        ret = tpsetcliwatcher(clid, logout_callback, (void *)curtime, TPNOFLAGS);

        printf("[%s] clid:%#x, curtime:%ld, set_watch:%d(%s), data:%s\n",
                        msg->name, clid, curtime, ret, tpstrerror(tperrno), msg->data);

        tpreturn(TPSUCCESS, 0, NULL, 0, 0);
}
```

- 관련함수

  CliWatcherCallback(), tpclrcliwatcher()

3.2.54. tpsetdbsessionid

RM의 세션 정보를 서비스에서 처리하는 경우 매번 처리해야 하는 불편함을 없애기 위하여 RM 세션 정보를 설정하는 콜백 함수이다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
int tpsetdbsessionid (char dbsessionid[MAX_DBSESSIONID], int flags);
```

- 파라미터

파라미터	설명
dbsessionid	개발자는 tpsetdbsessionid 루틴을 임의로 작성하고 얻어낸 ID 값을 dbsessionid 변수에 저장한다. Tmax 엔진에서는 RM과 연결을 새로 맺을 때마다 사용자가 작성한 tpsetdbsessionid 루틴를 호출해서 최신 값으로 갱신하고 내부적으로 관리한다.
flags	현재 버전에서는 지원하지 않으나 TPNOFLAGS로 설정한다.

- 예제

```
#include <usrinc/tmaxapi.h>
...
EXEC SQL include sqlca.h;
EXEC SQL begin declare section;
...
char  h_ssid[20];
EXEC SQL end declare section;

int tpsetdbsessionid(char dbsessionid[MAX_DBSESSIONID_SIZE], int flags)
{
    EXEC SQL SELECT TO_CHAR(USERENV('sessionid')) into :h_ssid FROM dual;
    if ( sqlca.sqlcode != 0 ){
        printf( "getting session id fail = %d\n",sqlca.sqlcode );
        return -1;
    }
    printf("RM session id = %s\n", h_ssid);

    memset(dbsessionid, 0x00, MAX_DBSESSIONID_SIZE);
    strcpy(dbsessionid, h_ssid);
    return 0;
}

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

3.2.55. tpsuspendtx

기존 전역 트랜잭션을 중지한다. 해당 함수를 사용하면 전역 트랜잭션이 실행 중인 상황에서 새로운 트랜잭션을 시작할 때 현재 실행 중인 전역 트랜잭션을 중지할 수 있다. 중지된 트랜잭션의 자원은 계속 ACTIVE 상태로 존재한다. 예를 들어 트랜잭션 타임아웃이 발생하게 되면 해당 트랜잭션은 rollback 처리된다. 중지된 트랜잭션은 tpresumetx(), tpsuspendxt()를 통하여 재개시킬 수 있다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
int tpsuspendtx(TRANID *tranid, int flags)
```

- 파라미터

파라미터	설명
tranid	개발자가 할당해야 하는 영역으로 반드시 TRANID 구조체의 포인터이어야 하며 현재의 트랜잭션 ID가 채워진다.
flags	현재 버전에서는 지원하지 않으나 TPNOFLAGS로 설정한다.

- 예제

```
SVC_TRAN( TPSVCINFO *msg )
{
    str  sndbuf;
    char *rcvbuf;
    char tmp[4096];
    int  cd, ret;
    long rlen;

    /*********************************************************
    TSR(Transaction Suspend and Resume) - TPTRANID Structure
    *********************************************************/
    TPTRANID *tranid = (TPTRANID *)malloc(sizeof(struct tptranid));
    sndbuf = (str)msg->data;
    rcvbuf=(char *)tpalloc("STRING", NULL, 4096);
    h_empno = sndbuf->empno;
    h_sal   = sndbuf->sal;
    strcpy( h_ename, sndbuf->ename );
    strcpy( h_job  , sndbuf->job   );
    strcpy( h_date , sndbuf->date  );

    ret = tx_begin();
    cd = tpcall("ORGTRAN1", (char *)msg->data, msg->len, (char **)&rcvbuf,
                (long *)&rlen, 0);
    strcpy(tmp, rcvbuf);

    /*********************************************************
    TSR(Transaction Suspend and Resume) - Suspend is Started
    *********************************************************/
    Ret = tpsuspendtx(tranid, 0)
    cd = tpcall("NEWTRAN", (char *)msg->data, msg->len, (char **)&rcvbuf,
               (long *)&rlen, 0);
    strcat(tmp, rcvbuf);

    /*********************************************************
    TSR(Transaction Suspend and Resume) - Resume is Started
    *********************************************************/
    ret = tpresumetx(tranid, 0);
    cd = tpcall("ORGTRAN2", (char *)msg->data, msg->len, (char **)&rcvbuf,
                (long *)&rlen, 0);
```

```
    ret = tx_commit();
    tpreturn( TPSUCCESS, 0, rcvbuf, strlen(rcvbuf), 0 );
}
```

3.2.56. tpsvctimeout

서비스 타임아웃이 발생했을 경우에 호출되는 루틴으로, 서비스 타임아웃이 발생했을 때 서버 프로그램은 자동적으로 tpsvctimeout()를 호출한다. 사용자가 함수를 재정의한 경우는 재정의한 함수를 호출한다.

만약 서비스 실행 중 tx_commit(), tx_rollback() 함수를 실행할 때 내부적으로 xa_commit(), xa_rollback() 단계에서 SVCTIMEOUT이 발생하면 이를 구별하기 위해서 tperrno에 TPETRAN을 설정하여 tpsvctimeout 함수 안에서 참조할 수 있도록 한다.

tpsvctimeout 함수는 libsvr.so 라이브러리가 내부적으로 가지고 있는 함수가 존재하며, 여기에서는 tpreturn(TPFAIL)로 리턴하고 있다. 사용자가 tpsvctimeout 함수를 재정의할 경우에는 재정의한 object 파일을 서버 컴파일하는 경우 libsvr 라이브러리 링크 위치보다 앞에 두어야 심볼을 적용할 때 재정의한 함수가 정상적으로 호출될 수 있다.

- 프로토타입

```
# include <tmaxapi.h>
void  tpsvctimeout(TPSVCINFO *msg)
```

- 파라미터

파라미터	설명
msg	타임아웃이 발생한 서비스를 호출할 때 사용한 메시지이다.

- 반환값

  tpsvctimeout()은 개발자가 서비스의 타임아웃이 발생하는 경우 필요한 작업을 수행하도록 작성하는 함수로 반환값은 없고 오류도 발생하지 않는다.

- 예제

```
#include <stdio.h>
#include <usrinc/atmi.h>
SERVICE(TPSVCINFO *msg)
{
    ...
    tpreturn(TPSUCCESS,0,(char *)msg->data, 0,TPNOFLAGS);
}

void tpsvctimeout(TPSVCINFO *msg)
{
    ...
    tpreturn(TPFAIL, TPETIME, (char *)buf, sizeof(buf), TPNOFLAGS);
}
```

3.2.57. tpsvrdone

서버 프로세스를 종료하는 함수로 Tmax 응용 서버 프로그램의 분리된 main은 서비스 요청 처리를 모두 마치고 프로세스를 종료하기 전에 tpsvrdone()을 호출한다. 루틴이 실행될 때 그 서버 프로세스는 여전히 시스템의 일부이기는 하지만 서비스는 지원하지 않는다. tpsvrdone() 루틴 내에서 Tmax 통신이 수행되거나 트랜잭션이 정의될 수도 있다.

tpsvrdone()이 대화형 연결을 유지하고 있는 경우는 Tmax는 대화형 연결을 종료한다. 비동기성 응답을 대기하고 있는 경우는 대기하고 있던 비동기성 응답들을 무시한다. 트랜잭션 모드에 있는 동안은 트랜잭션을 중지하고 서버는 바로 종료된다.

애플리케이션에서 tpsvrdone() 루틴을 제공하지 않는다면 Tmax가 제공하는 기본(default) 루틴이 대신 호출된다. 기본 tpsvrdone()은 트랜잭션을 처리하는 서버 그룹에 포함된 서버이면 **tx_close()**와 **userlog()**를 호출하여 서버가 곧 종료할 것임을 알린다. tpreturn()이나 tpforward() 중 하나가 tpsvrdone() 내에서 호출된다면 이러한 루틴들은 아무런 작동없이 단순히 반환만 한다.

- 프로토타입

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

- 반환값

  tpsvrdone()은 개발자가 서버 프로세스의 종료를 수행하기 전에 필요한 작업을 수행하도록 작성하는 함수로 반환값은 없고 오류도 발생하지 않는다.

- 예제

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

EXEC SQL INCLUDE sqlca.h;

SERVICE(TPSVCINFO *msg)
{
    int ret, cd;
    char *buf;

    EXEC SQL begin declare section;
    ….
    EXEC SQL end declare section;
    EXEC SQL CONNECT : scott IDENTIFIED BY : tiger;
    buf=tpalloc(“STRING”, NULL, 0);
    if (buf==NULL) { error processing }
    data process….

    cd=tpgetclid();
    if (cd==-1) { error processing }
    ret=tpsendtocli(cd, buf, 0, TPNOFLAGS);
    if (ret==-1) { error processing }
    data process....

    tpreturn(TPSUCCESS,0,buf, strlen(buf), 0);
```

```
}

int tpsvrdone()
{
    printf(“ Sevice end\n”);
}
```

- 관련 함수

  tx_close(), tpsvrinit()

3.2.58. tpsvrdown

UCS 방식 서버 프로세스를 정상적으로 종료시키는 함수이다. UCS 방식 서버 프로세스는 일반적으로 외부 시스템과의 통신에 많이 사용되며 대표적인 예로 은행 시스템 사이의 통신을 들 수 있다. 어떤 문제가 발생하여 외부 시스템이 서비스 요청을 처리하지 못하는 경우와 같은 상황을 대비하는 방안이 필요하다.

tpsvrdown()은 UCS 방식의 서버 프로세스를 정상적으로 종료시키므로 더 이상 사고가 발생한 외부 시스템에 서비스를 요청하지 않는다. 따라서 리소스를 절약할 수 있을 뿐만 아니라 외부 시스템의 부하도 줄일 수 있다.

- 프로토타입

```
#include <ucs.h>
int  tpsvrdown(void)
```

- 반환값

  tpsvrdown()은 개발자가 서버 프로세스를 종료하기 전에 필요한 작업을 수행하는 함수이므로 반환값이 없고 오류도 발생하지 않는다.

- 예제

```
...
#include <usrinc/atmi.h>
#include <usrinc/ucs.h>
int usermain(int argc, char *argv[])
{
    count=0;
    ...
    while(1)
    {
        ...
        tpschedule(3);
        cd = tpacall(“SERVICE”, buf, strlen(buf), TPNOREPLY);
        if (cd < 0) { error processing }
        ...
        if (count == 10) tpsvrdown();
    }
}

void reply_receive(UCSMSGINFO *reply)
```

```
{
    printf(“data....%s\n”, reply->data);
}
```

- 관련 함수

  tpsvrinit(), tpsvrdone()

3.2.59. tpsvrinit

개발자가 서비스를 수행하기 전에 Tmax 서버의 초기화 과정을 수행하는 작업을 처리하는 함수이다. Tmax 응용 서버 프로그램의 분리된 main의 초기화 과정에 tpsvrinit()을 호출한다. 이 루틴은 프로세스가 수행되고 난 후 아직 어떤 서비스 요청도 처리하기 전에 호출된다. 그러므로 tpsvrinit() 루틴 내에 Tmax 통신이 수행되거나 트랜잭션이 정의될 수도 있다.

애플리케이션에서 tpsvrinit() 루틴을 제공하지 않는다면 Tmax가 제공하는 기본(default) 루틴이 대신 호출된다. 기본 tpsvrinit()은 트랜잭션을 처리하는 서버 그룹에 포함된 서버이면 **tx_open()**과 **userlog()**를 호출하여 서버가 성공적으로 시작되었음을 알린다.

애플리케이션별 명령어 라인 옵션(CLOPT)은 서버에게 전달되어 tpsvrinit()에서 처리될 수 있다. 명령어 라인 옵션(CLOPT)에 대한 자세한 설명은 source config 파일의 SERVER 절 중 CLOPT 항목을 참고한다. 옵션은 argc와 argv를 통해 전달된다.

getopt()가 Tmax 서버 main()에서 사용되기 때문에 optarg, optind, opterr가 tpsvrinit() 내에서 옵션 parsing 및 에러 검출을 제어하는 데 사용된다.

- 프로토타입

```
# include <tmaxapi.h>
int  tpsvrinit (int  argc, char  **argv)
```

- 파라미터

파라미터	설명
argc	명령 라인 파라미터의 개수이다.
argv	명령 라인 파라미터이다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
음수	함수 호출에 실패한 경우이다. 어떤 서비스 요청도 받지 않고 서버 프로세스는 종료되고 에러는 발생하지 않는다.

- 예제

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

EXEC SQL INCLUDE sqlca.h;
tpsvrinit(int argc, char **argv)
{
    EXEC SQL begin declare section;
    char user_name[30];
    char user_passwd[30];
    EXEC SQL end declare section;
    EXEC SQL CONNECT scott IDENTIFIED BY tiger;
    return(0);
}

SERVICE(TPSVCINFO *msg)
{
    int ret, cd;
    char *buf;
    buf=tpalloc("STRING", NULL, 0);
    if (buf==NULL) { error processing }

    data process....
    cd=tpgetclid();
    if (cd==-1) { error processing }
    ret=tpsendtocli(cd, buf, 0, TPNOFLAGS);
    if (ret==-1) { error processing }

    data process....
    printf(" Sevice end\n");
    EXEC SQL COMMIT WORK RELEASE;
    tpreturn(TPSUCCESS, buf, strlen(buf), 0);
}
```

- 관련 함수

  tx_open(), tpsvrdone()

3.2.60. tpsvrthrdone

MultiThread 및 MultiContext 서버에서만 제공되는 함수이다. MultiThread 및 MultiContext 서버는 서버 프로세스가 종료될 경우 tpsvrdone 함수를 수행하기에 앞서 서비스 Thread들을 종료시킨다. 서비스 Thread는 자신이 종료할 때 이 함수가 정의되어 있다면 자동으로 호출해준다. 개발자는 Thread가 종료되기 전에 처리해야 할 필요한 작업을 수행하도록 루틴을 작성하면 된다. 이 함수 안에서 Tmax 통신이 수행되거나 트랜잭션이 수행될 수 있다. 만약 이러한 작업들이 모두 완료되지 않은 상태에서 그냥 반환하게 되면 Thread가 종료되면서 미완료된 작업들은 모두 무시하게 된다.

함수를 사용하여 하나의 클라이언트가 기존에 생성된 다른 context를 현재 클라이언트에 할당할 수 있다. 대부분의 ATMI 함수들은 per-context 기반으로 되어 있다. 클라이언트는 여러 개의 context를 사용할 수 있지만 해당하는 순간에는 단 하나의 context만을 갖게 된다. 예를 들면 context1에서 tpacall()을 한 경우 다른 context를 사용했다 하더라도 tpgetrply()를 정상적으로 하기 위해서는 tpgetrply()를 하는 시점에서는 context1을 현재 context로 설정해 주어야 한다. 자세한 내용은 tpsvrdone을 참고한다.

- 프로토타입

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

- 반환값

  tpsvrthrdone()은 개발자가 서버 프로세스의 종료를 수행하기 전에 필요한 작업을 수행하도록 작성하는 함수로 반환값은 없고 오류도 발생하지 않는다.

- 예제

  tpsvrthrinit() 함수의 예제를 참고한다.

- 관련 함수

  tpsvrthrinit()

3.2.61. tpsvrthrinit

Multithread 및 Multicontext 서버에서만 제공되는 함수이다. Tmax 서버는 서버 프로세스가 시작할 때 초기화 과정을 수행할 수 있는 tpsvrinit 함수를 제공한다. 마찬가지로 Multithread 및 Multicontext 서버에서는 tpsvrinit 함수가 호출된 이후에 Thread Pool에서 관리되는 서비스 Thread에 대해서도 Thread를 생성할 때 각각의 Thread마다 고유한 초기화 작업을 수행할 수 있도록 초기화 함수를 제공한다.

Thread Pool은 MINTHR, MAXTHR 항목에 따라 동작하므로 서버 프로세스가 처음 기동될 때에는 MINTHR 개수까지 서비스 Thread가 생성되면서 tpsvrthrinit()을 호출하게 되고, 이후 Thread Pool에 유휴 서비스 Thread가 없을 경우 최대 MAXTHR까지 필요한 개수만큼 서비스 Thread가 새롭게 생성되면서 이 함수를 호출하게 된다. 만약 MINTHR 항목이 0일 경우 프로세스 기동 초기에는 서비스 Thread를 생성하지 않기 때문에 tpsvrinit() 함수만 호출하고 서비스 요청이 들어오기를 기다린다.

tpsvrinit() 함수가 호출된 이후 그리고 각 Thread에서 서비스 요청을 처리하기 전에 수행되고 tpsvrinit() 함수에 전달된 것과 동일한 파라미터가 전달된다. 이 파라미터는 환경설정 SERVER 절의 CLOPT 항목에 설정한 내용들이다. 사용자는 tpsvrinit()과 tpsvrthrinit()으로 전달되는 파라미터가 동일하다는 것을 고려해서 작성해야 한다. 자세한 설명은 tpsvrinit을 참고한다.

- 프로토타입

```
# include <tmaxapi.h>
int  tpsvrthrinit (int  argc, char  **argv)
```

- 파라미터

파라미터	설명
argc	명령 라인 파라미터의 개수이다.
argv	명령 라인 파라미터이다.

- 반환값

  tpsvrthrinit() 함수를 통해 초기화 작업을 수행하는 과정에서 초기화가 실패할 경우 사용자는 -1을 리턴한다. 서버 프로세스는 tpsvrthrinit()을 호출한 후 -1이 리턴되면 프로세스 기동을 취소하고 종료된다.

반환값	설명
0	함수 호출에 성공한 경우이다.
음수	함수 호출에 실패한 경우이다. 서버 프로세스는 프로세스 기동을 취소하고 종료된다.

- 예제

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

void tpsvrthrinit(int argc, char **argv)
{
    param_t *param;
    param = get_threadspecificdata();
    if (pthread_create(&param->tid, NULL, THREAD_ROUTINE, (void *)param) != 0) {
        printf("user_create_thread failed\n");
        return -1;
    }
    pthread_mutex_init(&param->mutex, NULL);
    pthread_cond_init(&param->cond, NULL);
    return 0;
}

void tpsvrthrdone()
{
    void *ret;
    param_t *param;

    param = get_threadspecificdata();
    param->state = EXIT;
    pthread_cond_signal(&param->cond);
    pthread_join(param->tid, &ret);

    pthread_mutex_destroy(&param->mutex);
    pthread_cond_destroy(&param->cond);
    printf("user_create_thread destroyed\n");
}

SERVICE(TPSVCINFO *msg)
{
    param_t *param;
    param = get_threadspecificdata();
    ...
    ret = tpgetctxt(&param->ctxtid, TPNOFLAGS);
    ret = pthread_cond_signal(&param->cond);
    ...
}

void *THREAD_ROUTINE(void *arg)
{
```

```
    param_t *param;
    param = (param_t *)arg;

    while(1) {
        pthread_mutex_lock(&param->mutex); {
            pthread_cond_wait(&param->cond, &param->mutex);
            if (param->state == EXIT)
                break;
            if (tpsetctxt(param->ctxtid, TPNOFLAGS) == -1) {
                printf("tpsetctxt(%d) failed, [tperrno:%d]", param->ctxtid,
                        tperrno);
                return NULL;
            }

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

            if (tpsetctxt(TPNULLCONTEXT, TPNOFLAGS) == -1) {
                printf("tpsetctxt(TPNULLCONTEXT) failed, [tperrno:%d]", tperrno);
                return NULL;
            }
            ...
        } pthread_mutex_unlock(&param->mutex);
    }
    return NULL;
}
```

- 관련 함수

  tpsvrthrdone()

3.2.62. tptsleep

TMM으로부터 서버 프로세스 종료 이벤트를 대기하는 함수로 tpprechk() 콜백 함수에서 대기해야 하는 경우에 주기적으로 tptsleep()을 호출하여 정상적인 tmdown이 수행되도록 한다.

- 프로토타입

```
#include <usrinc/tmaxapi.h>
int tptsleep(struct timeval *timeout)
```

- 파라미터

파라미터	설명
timeout	종료 이벤트를 대기할 때의 타임아웃 시간을 초 단위로 지정한다. select() 시스템 함수와 동일하게 적용된다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다.

- 예제

```
int tpprechk(void)
{
     struct timeval timeout;
     int ret;
     timeout.tv_sec = 5;
     timeout.tv_usec = 0;
     while(1)
     {
          ret = tptsleep(&timeout);
          ...
     }
     return 0;
}
```

3.2.63. tpunadvertise

서버 프로세스가 제공하는 서비스를 서버에서 등록해제(unadvertise)하는 함수로 특정 서비스명을 등록(advertise) 또는 등록해제(unadvertise)할 수 있다.

특정 서비스를 등록하는 경우 TMM이 관리하는 서버별 서비스의 이름 테이블에 해당 서비스명을 등록한다.
등록해제하는 경우 서버별 서비스명을 테이블에서 삭제한다. 등록은 서버 프로세스가 서버가 제공하는 새로운 서비스를 등록하고, 등록해제는 서버가 제공하는 서비스를 등록해제한다. 등록해제된 서비스를 호출하고자 할 경우, TPENOENT 에러를 수신한다.

다음은 각각의 다양한 경우에 동작하는 방식에 대한 설명이다.

- **환경설정 파일에 등록된 서비스**

  해당 서비스의 상태가 UNADV로 변경되며, 서비스를 호출하는 경우 TPENOENT 에러를 수신한다.

- **mksvr로 등록된 서비스**

  해당 서비스의 상태가 UNADV로 변경되며, 서비스를 호출하는 경우 TPENOENT 에러를 수신한다. 해당 서버의 모든 프로세스가 종료하면 서비스 자체가 자동 삭제된다.

- **새롭게 등록된 서비스**

  해당 서비스의 상태가 UNADV로 변경되며, 서비스를 호출하는 경우 TPENOENT 에러를 수신한다. 해당 서버의 모든 프로세스가 종료하면 서비스 자체가 자동 삭제된다.

다음은 함수의 사용법에 대한 설명이다.

- 프로토타입

```
#include <atmi/usrinc.h>
int tpunadvertise(char *svcname);
```

- 파라미터

반환값	설명
svcname	등록해제할 서비스명을 지정한다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

에러 코드	설명
[TPEINVAL]	서비스명이 NULL일 경우이다.
[TPENOENT]	서비스가 존재하지 않을 경우이다.
[TPELIMIT]	서비스명이 지정된 길이를 초과하여 등록되었을 경우이다.
[TPEPROTO]	함수가 부적절한 상황에서 호출되었을 경우이다.
[TPESYSTEM]	Tmax 시스템에 에러가 발생하였다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

SVC2TN_2( TPSVCINFO *msg )
{
    int     ret;
    char    input[MAXLEN];

    memset(input, 0x00, MAXLEN);
    strncpy(input, msg->data, msg->len);

    ret = tpunadvertise(input);
    if (ret < 0) {
        tpreturn(TPFAIL, tperrno, (char*)msg->data, msg->len, 0);
    }
    tpreturn(TPSUCCESS, 0, (char*)msg->data, msg->len, 0);
}
```

3.2.64. tpunregancb

tmadmin으로부터 admntoi 명령으로 들어오는 요청에 대해서 처리하는 콜백 함수를 해제한다.

- 프로토타입

```
# include <tmaxapi.h>
int  tpunregancb ()
```

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpunregancb()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEPROTO]	등록된 콜백 함수가 없다.

- 예제

```
...
int admnoti(int reqno, int len, char *args);
void foo();
void bar();
TMAXSVRINFO svrinfo;

int tpsvrinit(int argc, char *argv[])
{
   tmax_my_svrinfo(&svrinfo);
   if (tpunregancb() < 0)
      error processing;
}

int admnoti(int reqno, int len, char *args)
{
   printf("svg[%s] svr[%s] spri[%d] reqno[%d] message[%s]",
      svrinfo.svgname, svrinfo.svrname, svrinfo.spri, reqno, args);
   if (strncmp(args, "foo", 3) == 0) {
      foo();
   } else if (strncmp(args, "bar", 3) == 0) {
      bar();
   } else {
      return -1;
   }
   return 0;
}
```

- 관련 함수

  TPADMNOTI, tpregancb

3.2.65. tpunregcb

UCS 방식의 서버 프로세스에서 비동기형 요청에 대한 응답을 받는 루틴을 재설정하는 함수로 서버 프로그램에서 응답을 받으면 수행되는 루틴을 재설정(reset)한다.

- 프로토타입

```
#include <ucs.h>
int  tpunregcb (void)
```

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpunregcb()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

```
...
#include <usrinc/atmi.h>
#include <usrinc/ucs.h>
void reply_receive(UCSMSGINFO *reply);

int usermain(int argc, char *argv[])
{
    ...
    ret = tpregcb(reply_receive);
    if (ret == -1){ error processing  }

    ret = tpunregcb();
    if (ret == -1){  error processing  }
    while(1)
    {
        ...
        tpschedule(3);
        cd = tpacall(“SERVICE”, buf, strlen(buf), TPNOFLAGS);
```

```
        if (cd < 0) { error processing  }
        ...
    }
}

void reply_receive(UCSMSGINFO *reply)
{
    printf("first reply receive\n");
    printf("data....%s\n", reply->data);
}
```

- 관련 함수

  tpregcb()

3.2.66. tpuschedule

UCS 서버 프로세스에서 데이터의 도착을 microsecond 단위로 입력한 시간 동안 기다리는 함수이다. tpuschedule()은 UCS 형태의 서버 프로세스에서만 사용 가능한 함수로 최대 타임아웃(timeout) 시간 동안 대기하다가 정해진 시간 안에 데이터가 도착하면 즉시 반환한다.

- 프로토타입

```
#include <ucs.h>
int tpuschedule (int timeout)
```

- 파라미터

파라미터	설명
timeout	대기할 시간을 시간을 microsecond 단위로 입력해야 한다. • –1 : 데이터가 도착했는지 체크만 하고 즉시 종료한다. • 0 : 데이터가 도착할 때까지 대기한다.

- 반환값

반환값	설명
0	타임아웃 시간까지 데이터가 도착하지 않는 경우이다.
양의 정수	타임아웃 시간까지 데이터가 도착한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpuschedule()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

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

- 관련 함수

  tpschedule()

3.2.67. tx_close

서버에서 사용하는 함수로 리소스 관리자들과의 연결을 종료한다.

tx_close() 함수는 호출자와 연결되어 있는 모든 리소스 관리자와의 연결을 종료한다. 리소스 관리자별 특정 연결 종료 호출 대신 사용되기 때문에 애플리케이션은 이식성에 문제를 일으킬 수 있는 리소스 관리자별 특정 연결 종료 호출을 사용하지 않아도 된다. 리소스 관리자별로 연결 종료 방식이 다르기 때문에 각 리소스 관리자를 종료하는 데 필요한 정보는 각 리소스 관리자에 의해 준비되어야 한다.

tx_close()는 내부적으로 호출되므로 사용자는 사용하지 않아도 된다. 리소스 관리자별 연결 종료 방식이 다르기 때문에 tx_close()는 트랜잭션 관리자가 리소스 관리자별 특정 방법에 관한 정보를 리소스 관리자에게 전달하도록 한다.

tx_close() 함수는 응용 프로세스가 더 이상 전역 트랜잭션에 참여하지 않을 때 호출되어야 한다. 호출자가 트랜잭션 모드에 있다면 tx_close() 함수 호출은 실패 처리되고, [TX_PROTOCOL_ERROR]를 반환하고 어떤 리소스 관리자와의 연결도 종료되지 않는다.

- 프로토타입

```
# include  <tx.h>
```

```
int  tx_close(void)
```

- 반환값

반환값	설명
TX_OK	함수 호출에 성공한 경우이다. 프로세스에 연결된 모든 자원 관리자와의 연결이 종료된다.
음수	함수 호출에 실패한 경우로 에러 코드를 반환한다.

- 오류

  tx_close()가 정상적으로 수행되지 않을 경우 다음의 음수값을 반환한다.

에러 코드	설명
[TX_PROTOCOL_ERROR]	함수가 부적절한 상황에서 호출되었다. 예를 들어, 호출자가 트랜잭션 모드에 있는 경우예 발생한다. 어떠한 자원 관리자와의 연결도 종료되지 않는다.
[TX_ERROR]	트랜잭션 관리자 또는 리소스 관리자가 일시적으로 에러를 만났다. 그 에러의 정확한 원인은 제품의 특성에 따라 다르다. 가능한 자원 관리자와의 연결이 모두 종료된다.
[TX_FAIL]	치명적인 에러가 발생하여 트랜잭션 관리자나 리소스 관리자가 더 이상 애플리케이션을 위하여 작업할 수 없다. 에러의 정확한 원인은 제품의 특성에 따라 다르다.

- 예제

```
….
int tpsvrinit(int argc, char *argv[])
{
    int ret;
    ret = tx_close();
    if (ret < 0){ error processing }
    ret = tx_open();
    if (ret < 0){ errror processing }
}

int tpsvrdone()
{
    int ret;
    ret = tx_close();
    if (ret < 0) { error processing }
}
```

- 관련 함수

  tx_open()

3.2.68. tx_open

관련된 리소스 관리자와 연결하는 함수로 tx_close()는 내부적으로 호출되므로 사용자는 사용하지 않아도 된다. 리소스 관리자별 연결 방식이 다르기 때문에 tx_open()은 트랜잭션 관리자가 리소스 관리자별 특정 연결 정보를 호출자와 관련된 리소스 관리자에게 전달하도록 한다.

tx_open() 함수는 애플리케이션과 관련된 모든 리소스 관리자와 연결한다. 이는 이식성에 문제를 일으킬 수 있는 리소스 관리자별 연결 함수 대신 사용되기 때문에 애플리케이션은 리소스 관리자별 연결 함수를 사용하지 않아도 된다. 리소스 관리자별 연결방식이 다르기 때문에 각 리소스 관리자별 연결 정보는 각 리소스 관리자에 의해 준비되어야 한다. 애플리케이션이 연결되지 않은 리소스 관리자에 접근하면 리소스별 특정 에러가 반환된다. tx_open() 함수는 프로세스가 전역 트랜잭션에 참여하기 이전에 성공적으로 반환되어야 한다.

한번 tx_open()이 성공적으로 반환하면 재호출된 tx_open() 함수는 tx_close() 호출 전까지는 아무 영향을 주지 않는다. 이미 리소스 관리자들과 연결된 후 tx_open()이 재호출되면 트랜잭션 관리자는 어떤 리소스 관리자와도 다시 연결하지 않고 성공으로만 반환한다. **tpopen()** 함수는 tx_open() 함수와 동일하다.

- 프로토타입

```
# include  <tx.h>
int  tx_open(void)
```

- 반환값

반환값	설명
TX_OK	함수 호출에 성공한 경우이다. 일부 리소스 관리자나 또는 모든 리소스 관리자와 연결된다.
음수	함수 호출에 실패한 경우로 에러 코드를 반환한다.

- 오류

  tx_open()이 정상적으로 수행되지 않을 경우 다음의 음수값을 반환한다.

에러 코드	설명
[TX_ERROR]	트랜잭션 관리자나 리소스 관리자가 일시적으로 에러를 만났다. 어떤 리소스 관리자와도 연결되지 않는다. 에러의 정확한 원인은 제품의 특성에 따라 다르다.
[TX_FAIL]	치명적인 에러가 발생하여 트랜잭션 관리자나 리소스 관리자는 더 이상 애플리케이션을 위한 작업을 수행할 수 없다. 에러의 정확한 원인은 제품의 특성에 따라 다르다.

- 예제

```
int tpsvrinit(int argc, char *argv[])
{
    int ret;
    ret = tx_close();
    if (ret < 0){
        error processing
    }
```

```
    ret = tx_open();
    if (ret < 0){
        errror processing
    }
}

int tpsvrdoen()
{
    int ret;
    ret = tx_close();
    if (ret < 0){
        error processing
    }
}
```

- 관련 함수

  tx_close()

3.3. 클라이언트 함수

3.3.1. gettpurcode

urcode 서비스에 설정된 urcode를 클라이언트에 반환하는 함수로 클라이언트가 호출한 **tpreturn()**의 두 번째 파라미터를 출력한다. 이 값은 클라이언트와 서비스 프로그램 사이의 추가된 통신 방식에 사용된다. 예를 들어 특정 오류 핸들링 루틴할 경우에 해당 에러 번호를 클라이언트에게 반환할 수 있다.

- 프로토타입

```
#include <atmi.h>
long gettpurcode(void)
```

- 반환값

  urcode를 반환한다.

- 예제

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

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

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

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

    ret=tpcall(“SERVICE”, buf, 0, &buf, &len, TPNOFLAGS);
    if (ret==-1) {error processing }
    printf(“urcode from SERVICE Return: %d”, gettpurcode());
    data process....

    tpend();
}
```

gettpurcode 함수의 자세한 내용은 "Tmax Programming Guide (Dynamic Library)"를 참고한다.

3.3.2. tpchkunsol

함수를 호출할 때 서버로부터 비요청 메시지가 있는 경우 해당 메시지를 처리하기 위한 함수를 호출한다.
클라이언트는 tpsetunsol()을 호출한 후 비요청 메시지를 처리하는 함수를 설정하거나 **tpcall()**이나 **tpacall()**을 호출한 후에 비요청 메시지를 처리하는 함수를 호출해야 한다.

tpchkunsol() 함수를 호출하면 tpcall()이나 tpacall()을 호출하기 전에도 비요청 메시지가 있는지 확인한 후 메시지를 가져올 수 있다.

- 프로토타입

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

- 반환값

반환값	설명
메시지 개수	함수 호출에 성공한 경우이다. 서버로부터 수신된 비요청 메시지의 개수를 반환한다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpchkunsol()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

에러 코드	설명
[TPETIME]	타임아웃이 발생하였다. 함수 호출자가 트랜잭션 모드에 있다면 트랜잭션 타임아웃이 발생한 것이며 트랜잭션은 rollback된다. 그렇지 않다면 TPNOTIME이나 TPNOBLOCK이 모두 설정되지 않은 상황에서 블록 타임아웃이 발생한다. 트랜잭션 타임아웃이 발생하는 경우 트랜잭션이 rollback될 때까지 새로운 서비스 요청을 송신한다거나 아직 수신되지 않은 응답을 기다리는 일은 모두 [TPETIME] 에러로 실패한다.
[TPEPROTO]	tpstart()를 호출할 때 tpsetunsol_flag()를 통해 비요청 메시지를 핸들러를 통해 수신하겠다는 TPUNSOL_HND flags가 설정되어 있어야 한다. TPUNSOL_HND flags가 설정되어 있지 않을 경우 [TPEPROTO] 에러로 실패한다.

- 예제

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

void get_unsol(char *data, long len, long flag)
{
    printf("unsolicited data = %s\n", data);
}

int main(int argc, char *argv[]){
    char *sndbuf;
    char *rcvbuf;
    long rcvlen;
    int RecvCnt = 0;
    TPSTART_T *tpinfo;

    tpinfo = (TPSTART_T *)tpalloc(« TPSTART », NULL, sizeof(TPSTART_T));
    if (tpinfo == NULL) {
        printf("tpalloc failed !<-tpinfo\n");
        exit(1);
    }
    strcpy(tpinfo->usrname, "starbj81");
    strcpy(tpinfo->cltname, "client");
    if(tpstart((TPSTART_T *)tpinfo) == -1)
    {
        fprintf(stderr, "tpstart error\n");
        exit(1);
    }
    tpsetunsol_flag(TPUNSOL_HND);
    tp_sleep(5);

    if(tpsetunsol(get_unsol) == TPUNSOLERR)
        printf("tpsetunsol failed..\n");

    RecvCnt = tpchkunsol();
    if(RecvCnt < 0)
       printf("tpchkunsol failed\n");
    else
       printf("Received Unsol Data Count : %d\n", RecvCnt);
```

```
    if((sndbuf = (char *)tpalloc("CARRAY", NULL, 1024)) == NULL)
        error processing
    if((rcvbuf = (char *)tpalloc("CARRAY", NULL, 1024)) == NULL)
        error processing
    if((cd = tpcall("LOGIN", sndbuf, 1024, (char **)&rcvbuf,
        (long *)&rcvbuf,  0)) == -1)
        error processing

    printf("After tpacall() received Message from server:%s\n", rcvbuf);
    tpfree((char *)sndbuf);
    tpfree((char *)rcvbuf);
    tpend();
}
```

- 관련 함수

  tpsetunsol(), tpsetunsol_flag()

3.3.3. tpend

클라이언트에서 Tmax 시스템과의 연결을 해제하는 함수로 클라이언트가 트랜잭션 모드에 있다면 트랜잭션은 자동으로 rollback된다. tpend()가 성공적으로 반환하면 호출자는 더 이상 어떤 프로그램과도 통신할 수 없고, 어떤 트랜잭션에도 참여할 수 없다. 또한 유지되고 있는 대화형 연결은 즉시 종료된다.

tpend()가 한 번 이상 호출된다면(이미 Tmax 시스템과 연결이 해제된 후에 또 다시 호출된다면) -1 값을 반환하지만 시스템에는 아무런 작동도 하지 않는다.

- 프로토타입

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

- 반환값

반환값	설명
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpend()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEPROTO]	tpend()가 부적절한 상황에서 호출되었다. 예를 들어 호출자가 서버인 경우나 또는 연결이 해제된 후에 다시 호출된 경우에 발생한다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

void main(int argc, char *argv[])
{
    char    *sndbuf, *rcvbuf;
    long    rcvlen, sndlen;
    int     ret;

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

    buf = (char *)tpalloc(“STRING”, NULL, 0)
    if (buf=NULL) { error processing }

    data process …

    ret=tpcall(“SERVICE”, buf, 0, &buf, &len, TPNOFLAGS);
    if (ret==-1) { error processing }

    data process …
    printf(“ data: %s\n”, buf);
    tpfree((char *)buf);
    tpend();
}
```

- 관련 함수

  tpstart()

3.3.4. tpgetclid

Tmax에 연결되어 있을 경우 Tmax에서 관리하는 자신의 clid 값을 얻어오는 함수이다. tpstart를 하지 않으면 의미가 없는 함수이다.

- 프로토타입

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

- 반환값

반환값	설명
0 이상의 정수값	함수 호출에 성공하는 경우이다. 클라이언트의 번호에 해당하는 0 이상의 정수값을 반환한다.
-1	tpend가 호출되어 있는 상태이다.

- 예제

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

    tpstart()...

    clid = tpgetclid();
    if (clid==-1) { error process }
```

3.3.5. tpgethostaddr

클라이언트와 Tmax 시스템에 접속 여부를 확인하거나, 접속된 Tmax 시스템의 IP 주소와 포트에 대한 정보를 얻을 때 사용하는 함수이다.

- 프로토타입

```
#include <tuxatmi.h>
int tpgethostaddr(int *ip, int *port, long flags);
```

- 파라미터

파라미터	설명
ip	해당 클라이언트의 IP 주소를 설정한다. IP 주소는 sockaddr_in 구조체에 있는 s_addr 필드이기 때문에 dot 형태로 바꾸기 위해서는 **inet_ntoa()**를 이용해야 한다.
port	해당 클라이언트의 포트 정보를 설정한다.
flags	flags에 설정 가능한 값은 다음과 같다. • GET_SVR_CON Tmax 접속 여부만 판단한다. • GET_CUR_IP 접속된 서버의 IP와 포트에 대한 정보도 함께 얻는다.

- 반환값

반환값	설명
- 1	Tmax 시스템에 접속되지 않은 경우이다.
1	Primary Tmax 시스템에 접속되어 있는 경우이다.
2	백업으로 설정된 Tmax 시스템에 접속되어 있는 경우이다.

- 예제

```
#include <usrinc/atmi.h>
#include <arpa/inet.h>
#include <usrinc/tmaxapi.h>

main(int argc, char *argv[])
{
        char    *sndbuf, *rcvbuf;
        long    rcvlen, sndlen;
        int     ret;
        int     i;
        TPSTART_T       *tpinfo;
        struct  sockaddr_in addr;
        char *cli_ip="192.168.1.86";
        int port;

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

        tpinfo = (TPSTART_T *)tpalloc("TPSTART",NULL,sizeof(TPSTART_T));
        if (tpinfo == -1){
                printf("tpinfo alloc failed\n");
                exit(1);
        }

        ret=tpstart(tpinfo);
        if (ret == -1){
                printf("tpstart failed\n");
                printf("[tperrno(%d) : %s]\n", tperrno,tpstrerror(tperrno));
                exit(1);
        }
        addr1.sin_addr.s_addr = inet_addr(cli_ip);
        port=8888;
        ret=tpgethostaddr(&addr.sin_addr.s_addr, &port, GET_CUR_IP);
        printf("ip = %s ,port = %d\n", inet_ntoa(addr.sin_addr),port);
```

3.3.6. tpgetunsol

클라이언트의 요청 없이 일방적으로 전달받은 메시지를 처리하는 함수이다. 메시지를 보내는 측에서 **tpbroadcast()** 또는 **tpsendtoci()**, **tppost()**를 통해 전달한다.

ttpgetunsol() 호출 전에 전달된 일방적인 메시지들은 무시된다. tpgetunsol()을 통해 비요청 메시지를 받으려면 tpstart()를 통해 Tmax 시스템에 연결할 때에 flags를 TPUNSOL_POLL이나 TPUNSOL_HND로 지정해야 한다. tpgetunsol()이 프로그램에 호출되면 tpstart()의 flags가 TPUNSOL_IGN로 설정되었다 하더라도 내부적으로 TPUNSOL_POLL로 전환되어 서버로부터 비요청 메시지를 받게 된다.

- 프로토타입

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

- 파라미터

파라미터	설명
type	서버에서 전달된 메시지의 형식으로 UNSOL_TPPOST, UNSOL_TPBROADCAST, UNSOL_TPSENDTOCLI 등이 있다.
data	전달된 메시지에 대한 포인터이다. 클라이언트가 알지 못하는 버퍼 유형 및 하위 유형(subtype)일 수 있는데 이 경우 data는 사용할 수 없다.
len	메시지의 총 길이이다.
flags	메시지의 블로킹(blocking) 처리 여부를 결정한다. flags로 사용 가능한 값은 다음과 같다. • TPBLOCK tpgetunsol()을 호출할 때에 블로킹 상태로 일방적인 메시지가 올 때까지 기다린다. • TPNOCHANGE 수신된 응답 버퍼와 *data가 가리키는 버퍼의 유형이 다르다면 *data의 버퍼 유형은 수신자가 인식할 수 있는 한도 내에서 수신된 응답 버퍼의 유형으로 변경된다. flags가 설정된 경우 *data가 가리키는 버퍼의 유형은 변경되지 못한다. 수신된 응답 버퍼의 유형 및 하위 유형은 *data가 가리키는 버퍼의 유형 및 하위 유형과 반드시 일치해야 한다. • TPNOTIME 함수 호출자가 블록 타임아웃을 무시하고 응답이 수신될 때까지 무한정 기다리겠다는 것을 의미한다. 트랜잭션 모드에서 tpgetrply()를 수행한 경우에는 여전히 트랜잭션 타임아웃이 적용된다. • TPSIGRSTRT 시그널(signal) 인터럽트를 수용하는 경우 사용하는 flags로 시스템 함수 호출이 방해될 때 시스템 함수 호출이 재실행된다. TPSIGRSTRT flag와 TPBLOCK flag가 설정되어 있으면 BLOCKTIME 시간동안 계속 응답을 기다린다. TPSIGRSTRT flags가 설정되지 않은 경우 시그널 인터럽트가 발생했다면, 함수는 실패하고 tperrno에 TPGOTSIG가 설정된다. • TPGETANY 입력값으로 구별자(cd)를 무시하고, 이에 상관없이 수신 가능한 응답을 반환하도록 한다. cd는 반환된 응답에 대한 호출 구별자가 된다. 응답이 없으면 일반적으로 tpgetrply()는 응답이 도착할 때까지 대기한다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.

반환값	설명
-1	함수 호출에 실패한 경우이다. tperrno에 상황에 해당하는 값이 설정된다.

- 오류

  tpgetunsol()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEPROTO]	tpgetunsol()가 부적절한 상황에서 호출되었다. tpstart()가 수행되지 않았으며 TMAX_ACTIVATE_AUTO_TPSTART=N 옵션이 설정되었을 경우 발생한다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

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

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

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

    while(1)
    {
        ret=tp_sleep(2);
        if (ret==-1) { error processing }

        if (ret==0)
            printf("nothing happened\n");
        else {
            ret=tpgetunsol(UNSOL_TPSENDTOCLI, (char **)&buf, &len, TPNOCHANGE);
            if (ret==-1) { error processing }
            printf("received data : %s\n", buf);
        }

        data process....
        if (strncmp(buf, "end", 3)==0) break;
    }

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

```
}
```

- 관련함수

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

3.3.7. tpinit

Tuxedo에서 사용한 함수를 Tmax 시스템에 그대로 적용하기 위해서 사용하는 함수로 Tuxedo로 개발된 프로그램을 변경없이 Tmax로 변환할 수 있도록 지원한다. 함수의 기능은 **tpstart()**와 동일하다.

- 프로토타입

```
#include <tuxatmi.h>
int tpinit (TPINIT *tpinfo)
```

- 파라미터

파라미터	설명
tpinfo	인증 정보 및 Multicontext를 사용하는지에 대해 설정한다.

- 반환값

  tpstart를 참고한다.

- 예제

```
#include <stdio.h>
#include <tuxinc/tuxatmi.h>
int main(int argc, char *argv[])
{
    char *buf;
    long len;
    int ret;

    ret = tpinit((TPINIT *)NULL);
    if (ret == -1){ error processing }
    buf = (char *)tpalloc("STRING", NULL, 0);
    if (buf == NULL){ error processing }

    strcpy(buf, argv[1]);
    ret = tpcall("SERVICE", buf, 0, &buf, &len, TPNOFLAGS);
    if (ret == -1){ error processing }

    printf("data: %s\n", buf);
    tpfree(buf);
    tpterm();
}
```

- 관련함수

  tpstart()

3.3.8. tpreset

클라이언트에서 사용하는 함수로 현재 접속된 클라이언트를 즉시 해제한다. 클라이언트 모듈에 TPESYSTEM 에러가 발생하는 경우는 대부분 네트워크 오류이므로 Tmax 시스템을 재접속한다.

tpreset()를 사용해서 접속을 해제하고 서비스를 재요청하거나 Tmax 시스템을 다시 연결한다.

- 프로토타입

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

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpreset()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEPROTO]	같은 context를 여러 스레드가 공유한 상태에서 한 스레드가 tpreset을 실행하는 경우 여러 스레드가 context를 공유한 상황이라면 TPEPROTO를 발생한다. 이것은 tpend에서 TPEPROTO가 발생하는 상황과 마찬가지로 해당 스레드의 context 할당은 해제되지만 다른 스레드에서는 context를 사용할 수 있다.
[TPESYSTEM]	Tmax 시스템 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

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

    ret=tpstart((TPSTART_T *)NULL);
    if (ret==-1){
```

```
        if (tperrno=TPESYSTEM)
        {
            printf ("system error \n");
            ret=tpreset();
            if (ret==-1) { error process }
            tpend();
            exit(1);
        }
        error processing....
     }
    data process..
    tpend();
}
```

3.3.9. tpsetunsol

클라이언트에서 사용되는 함수로 비요청 수신 메시지를 처리하는 루틴 설정한다. 시스템이 비요청 메시지를 통지받는 데 사용되는 방법은 애플리케이션에 임의로 결정되며, 이는 각 클라이언트별로 변경이 가능하다.

tpsetunsol()이 처음으로 호출되기 전에 Tmax 라이브러리에 의해 수신된 비요청 메시지는 무시된다. NULL 함수 포인터의 tpsetunsol() 호출도 마찬가지로 무시된다. 호출로 전달된 함수 포인터는 주어진 파라미터 정의에 적합해야 한다.

- 프로토타입

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

- 파라미터

파라미터	설명
data	수신된 유형 버퍼를 지시한다. 데이터도 수반되지 않았다면 data는 NULL이 될 수 있다. data가 클라이언트가 알지 못하는 버퍼 유형 및 하위 유형인 경우 데이터는 이해되지 못한다. 애플리케이션은 data를 제거할 수 없으며, 대신 시스템이 이를 제거하고 데이터 영역을 무효화하여 반환한다.
len	데이터의 길이를 나타낸다.
flags	현재 사용되지 않는다.

- 반환값

반환값	설명
포인터 / NULL	함수 호출에 성공한 경우이다. • 포인터: 이전에 설정된 비요청 메시지 처리 루틴의 포인터를 반환한다. • NULL: 이전에 어떤 메시지 처리 함수도 설정되지 않았음을 나타내는 것으로 성공적인 반환이다.

반환값	설명
TPUNSOLERR	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  tpsetunsol()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEPROTO]	tpsetunsol()이 부적절한 상황에서 호출되었다.
[TPESYSTEM]	Tmax 시스템 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

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

void get_unsol(char *data, long len, long flags)
{
    printf(“get unsolicited data = %s\n”, data);
    data process....
}

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

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

    ret=tpsetunsol(get_unsol);
    if (ret==TPUNSOLERR) { error processing }

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

    ret=tpcall(“SERVICE”, buf, 20, (char **)&buf, &len, TPNOFLAGS);
    if (ret==-1) { error processing }
    data process...

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

- 관련함수

tpstart(), tpend(), tpgetunsol()

3.3.10. tpsetunsol_flag

클라이언트에서 사용되는 함수로 비요청 메시지 수신 flags를 변경한다. **tpstart()** 함수를 이용하여 Tmax 시스템과 연결할 때 사용한 flags 값을 재설정한다.

- 프로토타입

```
# include <atmi.h>
int  tpsetunsol_flag  (int  flag)
```

- 파라미터

파라미터	설명
flag	비요청 메시지를 수신과 관련된 flags로 설정한다. 다음 중 하나를 설정한다. • TPUNSOL_IGN : 비요청 메시지를 받지 않는다. • TPUNSOL_HND, TPUNSOL_POLL : 비요청 메시지를 받는다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. flag가 설정 가능한 3가지 중 하나가 아닌 경우로 에러가 발생해도 tperrno에는 값이 설정되지 않는다.

- 예제

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

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

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

    ret=tpsetunsol(get_unsol);
    if (ret==TPUNSOLERR) { error processing }
    buf = (char *)tpalloc(“CARRAY”, NULL, 20);
```

```
    if (buf==NULL) {error processing };
    data process...

    ret=tpcall("SERVICE", buf, 20, &buf, &len, TPNOFLAGS);
    if (ret==-1) { error processing }
    data process...

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

void get_unsol(char *, long, long);
{
    printf("get unsolicited data.\n");
    data process....
}
```

- 관련 함수

  tpstart()

3.3.11. tpstart

클라이언트를 Tmax 시스템에 연결하는 함수로 서비스 요청이나 트랜잭션 처리 등의 ATMI 함수들을 사용하기 전에 tpstart()를 이용하여 클라이언트를 Tmax 시스템에 연결해야 한다.

tpstart()를 호출하기 전에 다른 ATMI 함수들(tpalloc()이나 tpcall() 등)이 먼저 호출되면 내부적으로 tpstart(NULL) 함수가 호출된다. 이와 같은 동작을 원하지 않을 경우에는 환경변수 TMAX_ACTIVATE_AUTO_TPSTART를 N으로 지정 할 수 있으며, 다른 ATMI 함수를 호출할 때 내부적으로 tpstart(NULL)을 호출하지 않고 TPEPROTO 에러가 발생한다. tpstart()가 성공적으로 반환되면 클라이언트는 최초 서비스 요청이나 트랜잭션 정의 등을 할 수 있다. 성공적으로 Tmax 시스템에 연결이 된 후에 다시 tpstart()가 호출되면 TPEPROTO 에러가 발생한다. tpstart()를 사용하여 Tmax 시스템과 연결하기 위해서는 Tmax 시스템이 설치된 서버의 IP와 포트 번호를 알아야한다.

다음은 서버의 정보를 알기 위한 환경변수에 대한 설명이다.

변수명	설명
TMAX_HOST_ADDR	클라이언트가 연결될 노드의 IP 주소로 클라이언트가 tpstart()를 호출할 경우에 내부적으로 서버 시스템에 연결되는 데 사용된다.
TMAX_HOST_PORT	클라이언트가 연결될 노드의 포트 번호를 지정한다. 클라이언트가 tpstart()를 호출할 때 내부적으로 서버 시스템과의 연결을 위해 TMAX_HOST_ADDR과 함께 사용된다. Tmax 환경 파일에 TPORTNO로 정의된 값이어야 한다. 클라이언트와 서버가 같은 노드에 존재하는 경우 TCP/IP소켓을 이용하지 않고 도메인 소켓을 이용하여 더 효과적인 처리를 할 수 있다. 이 경우에는 Tmax 환경 파일에 TPORTNO로 정의된 값 대신에 PATHDIR을 지정하면 된다. Tmax 환경 파일의 DOMAIN 절과 NODE 절의 TPORTNO 항목을 참고한다.

변수명	설명
TMAX_BACKUP_ADDR	TMAX_HOST_ADDR 주소의 노드에 장애가 생겼을 경우에 대비하여 또 다른 Tmax 시스템 노드를 지정한다. 클라이언트는 TMAX_HOST_ADDR 주소의 노드로 연결을 시도하고, 그 노드와의 연결에 실패하면 TMAX_BACKUP_ADDR 주소의 노드로 접속을 시도한다.
TMAX_BACKUP_PORT	TMAX_BACKUP_ADDR 주소의 노드에 대한 Tmax 시스템 포트 번호이다.
TMAX_CONNECT_TIMEOUT	Tmax 시스템에 연결될 때의 타임아웃 시간으로 micro-seconds resolution이 제공된다(예: 3.5).

- 프로토타입

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

- 파라미터

tpinfo는 TPSTART_T 라는 구조체에 대한 포인터(pointer)로 TPSTART라는 버퍼 타입을 사용하며, tpstart()를 호출하기 전에 반드시 tpalloc()에 의해 할당되어야 한다. 할당된 버퍼는 tpstart() 호출 후에 tpfree()를 통해 제거되어야 한다. 클라이언트는 시스템 연결할 때에 구조체 형식인 tpinfo 파라미터를 통해 필요한 정보를 넘긴다. tpinfo 파라미터는 클라이언트 정보, 비요청 메시지 처리 여부, 보안 정보 등을 포함한다.

tpinfo는 NULL 값을 사용할 수 있다. NULL인 경우 cltid, dompwd, usrname, usrpwd는 길이가 0인 string이 주어지고 Tmax 보안 특징을 사용하지 않으며 flags는 선택되지 않는다.

다음은 TPSTART_T 구조체의 구성이다.

```
struct TPSTART_T{
      char cltid[MAXTIDENT+2];  /*클라이언트명 (tpbroadcast())*/
      char dompwd[MAX_PASSWD_LENGTH+2];  /*시스템 접속 보안에 대한 암호*/
      char usrname[MAXTIDENT+2];   /*사용자 인증 보안에 대한 계정*/
      char usrpwd[MAX_PASSWD_LENGTH+2]; /*사용자 인증 보안에 대한 암호*/
      int flags; /*비요청 메시지 유형과 시스템 접근 방법결정*/
} ;
```

멤버	설명
cltid	최대 길이 30자까지 가능한 NULL로 끝나는 문자열로 애플리케이션에서 정의한 이름이다. tpbroadcast()에서 비요청 메시지를 보낼 클라이언트를 지정할 때 사용된다.
dompwd	Tmax에서 제공하는 보안 단계 중 시스템 접속 제어를 위해 사용된다. Tmax 환경 파일 중 DOMAIN 절의 OWNER 항목에 설정된 계정에 대한 암호를 등록한다.
usrname	Tmax에서 제공하는 보안 단계 중 사용자 인증을 위해 사용된다. usrname은 Tmax 시스템의 passwd 파일에 등록된 계정 이름이다.

멤버	설명
usrpwd	usrname에 해당하는 암호이다. 사용자 인증 보안이 설정된 경우 클라이언트는 usrname과 usrpwd를 등록해야만 Tmax 시스템에 연결되고 시스템에서 제공하는 서비스를 받을 수 있다. 보안 설정에 대한 내용은 source config 파일의 DOMAIN 절에 있는 SECURITY 항목을 참고한다.
flags	비요청(notification) 메시지 처리와 시스템 접근 방법을 결정하기 위해 사용된다. 비요청 메시지 처리를 위해 사용할 수 있는 flags이다. • TPUNSOL_POLL : 비요청 메시지를 수신한다. • TPUNSOL_HND : 비요청 메시지를 수신할 함수를 지정한다. 자세한 내용은 tpsetunsol을 참고한다. • TPUNSOL_IGN : 비요청 메시지를 무시한다. 기본값이 정의되지 않으면 TPUNSOL_IGN이 설정된다.

- 반환값

반환값	설명
0 또는 1	함수 호출에 성공한 경우이다. primary host에 0을 반환하고 backup host에 1을 반환한다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

tpstart()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 tpinfo가 NULL이나 TPSTART_T 구조체에 대한 포인터가 아닌 경우에 발생한다.
[TPEITYPE]	tpinfo가 TPSTART_T 구조체에 대한 포인터가 아니다.
[TPEPROTO]	tpstart()가 부적절한 곳에서 호출되었다. 예를 들어 서버 프로그램에서 사용되었거나 이미 연결된 상황에서 재호출된 경우에 발생한다.
[TPESYSTEM]	Tmax 시스템 에러가 발생한 경우로 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하거나 환경변수 설정이 잘못된 경우이다. 예를 들어 TMAX_HOST_ADDR나 TMAX_HOST_PORT가 잘못 설정되어 연결이 실패한 경우에 발생한다.

- 예제

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

```
    long len;
    TPSTART_T  *tpinfo;

    tpinfo = (TPSTART_T *)tpalloc("TPSTART", NULL, sizeof(TPSTART_T));
    if (tpinfo==NULL) { error processing }

    strcpy(tpinfo->cltname, "cli1");
    strcpy(tpinfo->usrname, "navis");
    strcpy(tpinfo->dompwd, "tmax");
    tpinfo->flags = TPUNSOL_HND;
    ret=tpstart(tpinfo);
    if (ret==-1) { error processing }

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

    ret=tpcall("SERVICE", buf, 20, &buf, &len, TPNOFLAGS);
    if (ret==-1) { error processing }

    data process....
    tpfree((char *) buf);
    tpend();
}
```

- 관련 함수

  tpend()

3.3.12. tpterm

클라이언트에서 Tmax 시스템과의 연결을 해제하는 함수이다. Tuxedo에서 사용한 함수를 Tmax 시스템에 그대로 적용하기 위해서 사용하는 함수로 Tuxedo로 개발된 프로그램을 변경없이 Tmax로 변환할 수 있도록 지원한다.
함수의 기능은 **tpend()** 함수와 동일하다.

- 프로토타입

```
#include <tuxatmi.h>
int  tpterm(void)
```

- 반환값

  tpend를 참고한다.

- 예제

```
...
#include <tuxinc/tuxatmi.h>
int main(int argc, char *argv[])
{
    int ret;
    long len;
```

```
    char *buf;

    ret = tpinit((TPINIT *)NULL);
    if (ret == -1){ error processing }

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

    ret = tpcall("SERVICE", buf, 0, &buf, &len, TPNOFLAGS);
    if (ret == -1){ error processing }

    data process...

    ret = tpterm();
    if (ret ==–1){ error processing }
}
```

- 관련 함수

  tpend()

3.3.13. tptobackup

클라이언트에서 Tmax 백업 시스템으로 연결하는 함수이다. **tpstart()**는 클라이언트가 Tmax 시스템에 연결하는 함수로 연결하려는 서버가 비정상적이어서 연결할 수 없을 경우 자동적으로 백업 시스템에 연결을 시도한다. 그러나 tptobackup()은 처음부터 백업 시스템에 연결하려고 할 때 사용하는 함수로 사용자가 임의로 백업 시스템에 연결할 때 사용한다.

tptobackup() 함수는 파라미터로 TPSTART_T를 받지 않으므로 보안이 필요한 시스템에는 접속할 수 없다. 함수로 연결될 백업 시스템은 보안에 관련된 항목이 config에 설정하면 접속할 수 없다. 또한 비요청 메시지를 받기 위해서는 **tpsetunsol_flag()** 함수를 이용하여 비 요청 메시지를 받을 수 있도록 flags를 변경해야 한다. tptobackup()을 사용하려면 **TMAX_BACKUP_ADDR**과 **TMAX_BACKUP_PORT**를 시스템 환경 파일에 등록해야 한다. (예: Korn 셸의 .profile)

- 프로토타입

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

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다(tperrno에 에러 코드가 설정된다).

- 오류

tptobackup()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEPROTO]	tptobackup()이 부적절한 상황에서 호출되었다. 예를 들어 서버 프로그램에서 사용되었거나 이미 연결된 상황에서 재호출된 경우에 발생한다.
[TPESYSTEM]	Tmax 시스템 에러가 발생한 경우로 자세한 정보는 로그파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생한 경우이거나 잘못된 환경변수가 설정된 경우이다. 예를 들어 TMAX_BACKUP_ADDR 또는 TMAX_BACKUP_PORT가 잘못 설정되어 접속에 실패한 경우에 발생한다.

- 예제

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

void main(int argc, char *argv[])
{
    tpputenv(“TMAX_BACKUP_ADDR=xxx.xxx.xxx.xxx”);
    tpputenv(“TMAX_BACKUP_PORT=xxxx”);
    tptobackup();
    data process....
    tpend();
}
```

3.4. TCP/IP 게이트웨이 함수

3.4.1. allow_connection

TCPGW로 리모트 노드가 연결을 시도하거나 또는 TCPGW가 리모트 노드로 연결을 시도해서 연결이 성공했을 때 remote_connected() Callback 함수가 호출되기 전에 호출된다. 해당 리모트와의 연결을 허용할 것인지 결정할 수 있다.

- 프로토타입

```
int allow_connection(int addr, int portno, int type, int fd)
```

- 파라미터

파라미터	설명
addr	리모트 노드의 주소이다.

파라미터	설명
portno	연결된 서버의 포트 번호이다. TCPGW가 클라이언트 모드로 동작한다면 리모트 노드의 포트 번호이고, TCPGW가 서버 모드로 동작한다면 Listen하고 있는 소켓의 포트 번호이다.
type	리모트 노드와 연결된 채널 타입이다. IN_CHANNEL 또는 OUT_CHANNEL인지를 나타낸다.
fd	리모트 노드와 연결된 소켓 번호이다.

- 반환값

반환값	설명
1	해당 리모트 노드와의 연결 요청을 허용한다.
음수	해당 리모트 노드와의 연결을 허용하지 않는다.

- 예제

```
int allow_connection(int addr, int portno, int type, int fd)
{
    int i, n, len;
    struct in_addr in;
    char *ipaddr, *endptr;
    char *deny[] = {"10.10.30.1",
                    "121.100.100.*",
                    NULL};

    if (addr == -1)
        return -1;
    in.s_addr = addr;
    ipaddr = inet_ntoa(in);

    for (i = 0; deny[i] != NULL; i++) {
        if ((endptr = strpbrk(deny[i], "*")) != NULL) {
            if (strncmp(deny[i], ipaddr, (endptr - deny[i])) == 0)
                return -1;
        } else {
            if (strcmp(deny[i], ipaddr) == 0)
                return -1;
        }
    }
    return 1;
}
```

3.4.2. allow_connection_ipv6

IPv6 프로토콜 환경에서 TCPGW로 리모트 노드가 연결을 시도하거나 또는 TCPGW가 리모트 노드로 연결을 시도해서 연결이 성공했을 때 remote_connected() Callback 함수가 호출되기 전에 호출된다. 해당 리모트와의 연결을 허용할 것인지 결정할 수 있다.

IPv6 환경에서는 반드시 allow_connection Callback 함수가 아닌 allow_connection_ipv6 Callback 함수를 설정해야 한다. 이 Callback 함수를 설정하지 않고 IPv6 환경에서 사용할 경우에는 allow_connection 함수가 호출되지만 ipaddr 값에 임의의 값이 설정되며 slog에 Warning 메시지가 기록된다.

- 프로토타입

```
#include <arpa/inet.h>

int allow_connection(struct sockaddr *saddr, int portno, int type, int fd)
```

- 파라미터

파라미터	설명
saddr	리모트 노드의 주소 및 포트 번호이다. IPv4 또는 IPv6 프로토콜 환경에 적합한 sockaddr 구조체 타입으로 캐스팅하여 사용한다.
portno	연결된 서버의 포트 번호이다. TCPGW가 클라이언트 모드로 동작한다면 리모트 노드의 포트 번호이고, TCPGW가 서버 모드로 동작한다면 Listen하고 있는 소켓의 포트 번호이다.
type	리모트 노드와 연결된 채널 타입이다. IN_CHANNEL 또는 OUT_CHANNEL인지를 나타낸다.
fd	리모트 노드와 연결된 소켓 번호이다.

- 반환값

반환값	설명
1	해당 리모트 노드와의 연결 요청을 허용한다.
음수	해당 리모트 노드와의 연결을 허용하지 않는다.

- 예제

```
int
allow_connection_ipv6(struct sockaddr *saddr, int svr_portno, int type, int fd)
{
    char buf[INET6_ADDRSTRLEN];
    char *ipaddr;
    int cli_portno;

    int i, n, len;
    char *endptr;
    char *allow[] = {"10.10.30.1",
                     "121.100.100.*",
                     NULL};

    if (saddr->sa_family == AF_INET6) {
        ipaddr = (char *)inet_ntop(AF_INET6, &(((struct sockaddr_in6 *)saddr)->sin6_addr), buf,
sizeof(buf));
        cli_portno = ntohs(((struct sockaddr_in6 *)saddr)->sin6_port);
    } else if (saddr->sa_family == AF_INET) {
```

```
        ipaddr = (char *)inet_ntop(AF_INET, &(((struct sockaddr_in *)saddr)->sin_addr), buf,
sizeof(buf));
        cli_portno = ntohs(((struct sockaddr_in *)saddr)->sin_port);
    }
    if (ipaddr == NULL)
        ipaddr = "unknown";

    for (i = 0; allow[i] != NULL; i++) {
        if ((endptr = strpbrk(allow[i], "*")) != NULL) {
            if (strncmp(allow[i], ipaddr, (endptr - deny[i])) == 0)
                return 1;
        } else {
            if (strcmp(allow[i], ipaddr) == 0)
                return 1;
        }
    }
    return -1;
}
```

3.4.3. chk_end_msg

리모트 노드로부터 데이터를 수신할 때 데이터의 끝을 나타내는 특정 문자 또는 비트 스트림이 존재할 경우 사용자가 호출할 수 있는 함수이다.

chk_end_msg()에서 수신 데이터보다 리턴한 데이터 크기가 작을 경우, 리턴한 데이터 부분을 처리한 후에 수신 데이터의 남겨진 뒷 부분에 대해서 다시 한 번 check_end_msg()를 호출한다. 사용자는 만약 남겨진 뒷 부분의 데이터가 미완성 상태라면 return -1을 호출하도록 작성하면 된다. 그러면 그 이후에 상대방 채널로부터 메시지가 수신될 때 이전 데이터와 새로 수신된 데이터를 하나로 합쳐서 chk_end_msg()를 호출해준다.

chk_end_msg()에서 수신 데이터보다 리턴한 데이터 크기가 클 경우에 체크로직이 없으므로, 에러 로그를 slog에 남기게 된다. 이때 tcpgw CLOPT 옵션에 "-XCHK_UID" 를 지정하면 해당 요청에 대해서 폐기하고 에러 처리한다. 혹은 만약 CLOPT 옵션 "-e"와 함께 사용하는 경우에는 동일한 상황에서 해당 채널을 강제로 종료시킨다.

- 프로토타입

```
int chk_end_msg(int len, char *data)
```

- 파라미터

파라미터	설명
len	리모트 노드로부터 읽은 데이터 길이이다.
data	리모트 노드로부터 읽은 데이터 부분이다.

3.4.4. chk_extpong_msg

리모트 서버로부터 메시지를 수신했을 때 해당 메시지가 채널 장애 감지를 위한 응답 메시지인지를 확인하는 함수이다.

- 프로토타입

```
int chk_extpong_msg(msg_header_t *hp, char *data, int len)
```

- 파라미터

파라미터	설명
hp	채널 장애가 감지되었을 때 리모트 서버로부터 수신된 메시지 헤더이다.
data	채널 장애가 감지되었을 때 리모트 서버로부터 수신된 메시지 바디이다.
len	수신된 메시지 바디의 길이이다.

- 반환값

반환값	설명
양수	리모트 서버로부터 수신한 메시지가 정상적인 채널 장애 감시 응답 메시지인 경우이다.
0	리모트 서버로부터 수신한 메시지는 채널 장애 감시 응답 메시지가 아닌 일반적인 메시지이다. 해당 메시지는 get_msg_info 함수에서 처리된다.
음수	리모트 서버로부터 수신한 메시지가 비정상적인 채널 장애 감시 응답 메시지인 경우이다. 해당 채널을 종료시킨다.

- 예제

```
int chk_extpong_msg(msg_header_t *hp, char *data, int len)
{
        msg_body_t *body;
        char data2[15];

        body = (msg_body_t *)data;
        printf("chk_extpong_msg : data = %s\n", body->data);

        if (strcmp(body->data, "ping_reply")
                return 1;
        return 0;
}
```

3.4.5. chk_pong_msg

리모트 서버로부터 메시지를 수신했을 때 해당 메시지가 채널 장애 감지를 위한 응답 메시지인지를 여부를 확인하는 함수이다.

- 프로토타입

```
int chk_pong_msg(msg_header_t *hp)
```

- 파라미터

파라미터	설명
hp	리모트로부터 수신된 메시지이다.

- 반환값

반환값	설명
양수	리모트 서버로부터 수신한 메시지가 정상적인 채널 장애 감시 응답 메시지인 경우이다.
0	리모트 서버로부터 수신한 메시지는 채널 장애 감시 응답 메시지가 아닌 일반적인 메시지인 경우이다. 해당 메시지는 get_msg_info 함수에서 처리된다.
음수	리모트 서버로부터 수신한 메시지가 비정상적인 채널 장애 감시 응답 메시지인 경우이다. 해당 채널을 종료시킨다.

3.4.6. get_channel_num

Tmax 서비스나 클라이언트로부터 요청한 데이터를 리모트 노드에 전송할 때 사용자가 채널을 선택할 수 있도록 하는 함수이다. 사용자는 주어진 데이터의 특성에 따라서 전송할 채널을 지정할 수 있다. 단, 여기서 지정하는 것은 리모트 노드와 연결된 소켓 번호가 아니라 단순한 채널 번호이다. TCPGW는 사용자가 지정한 채널을 사용할 수 없으면 오류를 반환한다.

- 프로토타입

```
int get_channel_num(char *data)
```

- 파라미터

파라미터	설명
data	리모트 노드로 보내기 위한 데이터이다.

- 반환값

  채널 번호를 반환한다.

3.4.7. get_extmsg_info

get_msg_info와 기본적으로 기능은 동일하다. 단, 데이터 버퍼를 더블 포인터형으로 넘겨서 사용자가 버퍼의 메모리를 재할당(realloc)할 수 있다. 그러나 이 함수는 get_msg_info와 동시에 사용될 수 없으며, 사용하려면 TCPGW 컴파일할 때 -D_TCPGW_USE_EXTMSG flags를 설정해야 한다.

- 프로토타입

```
int get_extmsg_info(msg_header_t *hp, char **data, int asize, msg_info_t *info)
```

- 파라미터

파라미터	설명
hp	리모트 노드로부터 읽어온 메시지 헤더 데이터 대한 포인터이다. get_msg_length()에서 사용한 구조체와 동일하다.
data	리모트 노드로부터 읽은 데이터 버퍼의 주소값이다.
asize	리모트 노드로부터 읽은 데이터 버퍼에 할당된 메모리 크기이다.
info	TCPGW 라이브러리(libtcpgw.a, libtcpgw.so)와 custom.c와 인터페이스 역할을 하는 구조체 구조이다. 사용자가 수신받은 데이터를 기초로 해서 info 구조체의 항목에 각종 정보를 설정한다.

- 반환값

  Tmax 서비스로 보낼 타입을 정의한다. TCPGW는 값을 바탕으로 Tmax로 어떤 처리를 할 것인지를 판단한다. 예를 들어 REMOTE_REQUEST는 리모트 노드로부터 Request가 발생한 것으로 판단하며, REMOTE_REPLY는 Tmax 서비스로부터 Request가 발생하여 리모트 노드로부터 Response가 오는 경우 반환하는 값이다. REMOTE_REPLY_CONT는 Response 메시지가 이어서 올 경우 반환하는 값이며, REMOTE_SENDTOCLI는 비요청 메시지일 경우 반환하는 값이다. 리모트 노드로부터 응답을 수신한 경우에는 반드시 UID 값을 info 구조체의 uid 항목에 지정해 주어야 한다. 그리고 기타 다른 값들 상황에 맞게 주어야 한다.

3.4.8. get_msg_info

리모트 노드으로부터 요청이나 응답이 도착하여 데이터를 읽은 후 데이터를 Tmax 서비스 프로그램으로 다시 요청이나 응답를 전송하기 전에 해당 데이터 값을 가공하거나, 정보 전송을 위한 여러 정보(uid, len, flags, 서비스명 등)들을 TCPGW 라이브러리와 custom.c와의 인터페이스 역할을 하는 info를 참조 또는 가공하는 함수이다.

- 프로토타입

```
int get_msg_info(msg_header_t *hp, char *data,msg_info_t *info)
```

- 파라미터

파라미터	설명
hp	리모트 노드로부터 읽어온 메시지 헤더 데이터 대한 포인터이다. get_msg_length에서 사용한 구조체와 동일하다.
data	리모트 노드로부터 읽은 데이터 부분이다.
info	TCPGW 라이브러리(libtcpgw.a, libtcpgw.so)와 custom.c와 인터페이스 역할을 해주는 구조체 구조이다. 사용자가 수신받은 데이터를 기초로 해서 info 구조체의 항목에 각종 정보를 함수에서 설정한다.

- 반환값

  Tmax 서비스로 보낼 타입을 정의한다. TCPGW는 값을 바탕으로 Tmax로 어떤 처리를 할 것인지를 판단한다. 예를 들어 REMOTE_REQUEST는 리모트 노드로부터 Request가 발생한 것으로 판단하며, REMOTE_REPLY는 Tmax 서비스로부터 Request가 발생하여 리모트 노드로부터 Response가 오는 경우 반환하는 값이다. REMOTE_REPLY_CONT는 Response 메시지가 이어서 올 경우 반환하는 값이며, REMOTE_SENDTOCLI는 비요청 메시지일 경우 반환하는 값이다.

  -1을 리턴 하고 환경설정파일의 CLOPT 절에 -e 옵션을 설정하면 해당 채널이 종료된다.

  리모트 노드로부터 응답을 수신한 경우에는 반드시 UID 값을 info 구조체의 uid 항목에 지정해 주어야 한다. 기타 다른 값들은 상황에 맞게 주어야 한다.

3.4.9. get_msg_length

리모트 노드로부터 요청이나 응답이 도착하여, 해당 채널에서 msg_header_t 부분만을 recv(msg_header_size 또는 comm_header_size에서 지정한 값만큼을 읽는다)한 후 호출하는 함수로 반환된 값만큼 실 데이터를 다시 read하게 된다.

- 프로토타입

```
int get_msg_length(msg_header_t *hp)
```

- 파라미터

파라미터	설명
hp	개발자가 custom.h에 정의할 수 있는 msg_header_t 구조체의 포인터이다. 일반적으로 해당 채널로부터 데이터를 읽는 경우 정해진 헤더의 크기만큼 데이터를 읽은 후에 헤더에 설정되어 있는 데이터 길이값을 얻어서 이를 바탕으로 다음 실 데이터를 읽을 수 있도록 되어 있다. hp는 TCPGW가 리모트 노드로부터 읽은 헤더 데이터를 넘겨준다.

- 반환값

  리모트 노드로부터 실 데이터의 길이를 반환한다. 함수의 반환값으로 TCPGW는 실 데이터를 리모트 노드에서 읽게 된다. 헤더를 확인하고 문제가 있을 경우에 -2를 리턴하면 해당 채널이 종료된다.

3.4.10. get_service_name

Tmax에서 리모트 노드로 요청을 보낼 때 요청을 보내는 서버와 결과를 받는 서버를 달리하는 NON 블록형이나 비동기형 TCPGW를 구성하는 경우, tpreply()나 tpacall()을 할 서비스의 이름을 오류 코드에 따라서 설정한다.

- 프로토타입

```
int get_service_name(char *header, int err, char *svc)
```

- 파라미터

파라미터	설명
header	[-H] 또는 [-h] 옵션으로 설정한 TCPGW에서 저장하고 있는 사용자 헤더의 포인터이다.
err	오류 코드이다.
svc	tpreply()나 tpacall()를 받는 서비스의 이름을 설정한다.

3.4.11. get_msg_security

get_msg_info() 또는 get_extmsg_info() 함수가 수행된 후에 호출되는 함수이다. 사용자 데이터를 가공할 필요가 있을 경우에 사용한다. get_extmsg_info() 함수가 이 함수를 대체할 수 있다.

- 프로토타입

```
int get_msg_security(char **data, int asize, int len)
```

- 파라미터

파라미터	설명
data	리모트 노드로부터 읽은 데이터 버퍼의 주소값이다. 버퍼의 크기가 부족한 경우에는 realloc()을 통해 버퍼를 확장하는 것을 허용한다. realloc()을 호출하여 버퍼의 포인터가 변경된 경우에는 이 주소값에 변경된 포인터 주소값을 저장해야 한다.
asize	리모트 노드로부터 읽은 데이터 버퍼에 할당된 메모리 크기이다.
len	메시지 바디의 길이이다.

- 반환값

반환값	설명
양수	가공된 사용자 데이터의 전체 길이를 리턴한다.
0	사용자 데이터의 변경사항을 반영하지 않는다.

- 예제

```
int get_msg_security(char **data, int asize, int len)
{
     ...
     new_size = len * 2;
     if (new_size > asize) {
         tmpbuf = (char *)realloc(*data, new_size);
         if (tmpbuf == NULL) {
```

```
            /* error processing */
        }
        *data = tmpbuf;
    }
    ...
    return new_size;
}
```

3.4.12. init_remote_info

리모트 노드와 연결을 맺기 전에 호출되는 함수이다. TCPGW가 자신의 초기화 작업을 종료한 후 즉시 호출되는 함수이며 한 번만 호출된다. Tmax 환경 파일의 CLOPT에 [?k] 옵션으로 공유 메모리 키를 설정한 경우 연결에 대한 정보 등을 저장하기 위해 공유 메모리를 생성하는 로직을 구현할 수 있다. 경우에 따라 내부 로직을 구현하지 않아도 된다.

- 프로토타입

```
int init_remote_info(char *myname, int mynumber, int num_channel, int key)
```

- 파라미터

파라미터	설명
myname	TCPGW 서버명이다.
mynumber	같은 TCPGW가 동시에 여러 개 실행될 경우에 각각의 프로세스를 구분할 수 있는 TCPGW 프로세스 번호로 0부터 시작한다.
num_channel	TCPGW가 연결하고 있는 max 채널 수이다. Tmax 환경 파일에 [-n], [-N]옵션으로 설정한 값의 합이다.
key	Tmax 환경 파일에서 [?k] 옵션으로 설정한 공유 메모리 키 값이다.

3.4.13. inmsg_recovery

노드로부터 요청을 tpacall(..., TPBLOCK)로 처리했을 때 서버가 떠 있지 않으면 에러가 돌아오게 되는데, 이러한 때에 호출된다. 사용자는 함수 내에서 적당히 새로운 데이터를 생성하고, 데이터의 크기를 반환한다.

- 프로토타입

```
int inmsg_recovery(char *data, msg_info_t *info)
```

- 파라미터

파라미터	설명
data	리모트 노드로부터 읽은 데이터 부분이다.

파라미터	설명
info	리모트 노드로부터 읽은 데이터의 정보로 의미있는 값은 다음과 같다. • info→svc : tpacall()한 서비스명 • info→len : tpacall()한 데이터 길이 • info→err : 에러가 발생한 경우 • tperrno info→uid : 이전 get_msg_info()할 때 생성되었던 UID 값

- 반환값

  사용자는 새로운 데이터의 길이를 반환해야 한다.

반환값	설명
양수	• info→svc에 값이 존재하는 경우 : 해당 서비스로 tpacall(..., TPNOREPLY)한다. • 그 외의 경우 : 리모트 노드로 응답을 보낸다.
음수	데이터를 버린다.

3.4.14. outmsg_recovery

리모트 노드로 요청을 보낼 때 에러가 발생할 경우 호출된다. 사용자는 함수 내에서 적당히 새로운 데이터를 생성하고 원하는 서비스명 및 UID 등을 설정하고, 데이터의 크기를 반환한다.

- 프로토타입

```
int inmsg_recovery(char *data, msg_info_t *info)
```

- 파라미터

파라미터	설명
data	put_msg_info() 콜백함수의 data 인자에서 사용된 사용자 데이터이다.
info	put_msg_info() 콜백 함수의 msg_info_t 인자에서 사용된 데이터의 정보로 의미있는 값은 다음과 같다. • info→svc : 콜백 함수를 호출하는 경우 비어있는 값으로 설정되어 있다. 콜백 함수 안에서 이 값에 서비스명을 지정하면, 지정한 서비스로 tpacall(TPNOREPLY) 형식으로 메시지를 전송해준다. • info→err : 콜백 함수를 호출하는 경우 TPESYSTEM 에러코드가 설정되어 있다. 콜백 함수 안에서 이 값을 0으로 지정하면 호출측에 정상 리턴을 한다. 그렇지 않으면 info→svc를 지정하지 않았을 경우에 호출측에 TPESVCFAIL 에러를 리턴한다.

- 반환값

반환값	설명
음수	info 구조체의 설정을 확인하지 않고 무조건 호출측에 TPESYSTEM 에러를 리턴한다.
양수	info 구조체의 설정을 확인하고 파라미터에 기술된 방식으로 메시지를 처리한다.

3.4.15. prepare_shutdown

TCPGW가 종료하기 직전에 call되는 함수로 일반적으로 init_remote_info() 함수에서 생성한 공유 메모리를 해제한다.

- 프로토타입

```
int prepare_shutdown(int code)
```

- 파라미터

파라미터	설명
code	shutdown code로 현재는 사용되지 않는다.

3.4.16. put_extmsg_info

put_msg_info와 기본적으로 기능은 동일하다. 단, 데이터 버퍼를 더블 포인터형으로 넘겨서 사용자가 버퍼의 메모리를 재할당(realloc)할 수 있다. put_msg_info와 동시에 사용될 수 없으며, 사용하려면 TCPGW 컴파일할 때 -D_TCPGW_USE_EXTMSG flags를 설정해야 한다.

- 프로토타입

```
int put_extmsg_info(msg_header_t *hp, char **data, int asize, msg_info_t *info)
```

- 파라미터

파라미터	설명
hp	리모트 노드로 보낼 메시지의 헤더이다.
data	리모트 노드로 보낼 데이터 버퍼의 주소값이다.
asize	리모트 노드로 보낼 데이터 버퍼에 할당된 메모리 크기이다.
info	리모트 노드로 보낼 데이터의 정보이다.

- 반환값

  사용자는 함수에서 실제적으로 리모트 노드에 전송할 데이터의 전체 길이를 반환해야 한다. 메시지 헤더와 실 데이터를 더한 길이를 반환한다.

3.4.17. put_msg_complete

리모트 노드로 메시지를 전송한 후에 호출되는 함수로 데이터가 완전히 리모트 노드에 전송되었다는 것을 알려주기 위해서 호출되는 함수이다.

- 프로토타입

```
int put_msg_complete(msg_header_t *hp, char *data, msg_info_t *info)
```

- 파라미터

파라미터	설명
hp	리모트 노드로 보낸 메시지의 헤더이다.
data	리모트 노드로 보낸 데이터이다.
info	리모트 노드로 보낸 데이터의 정보이다.

3.4.18. put_msg_info

리모트 노드로 메시지를 전송하고자 할 때 호출되는 함수이다. 동기형 통신인 경우에는 사용자가 함수에서 UID를 메시지에 저장(Save)해야 한다. UID는 info 구조체의 uid 항목의 값을 사용해도 되고 또는 사용자가 임의의 UID를 생성해서 사용한 후에 info의 uid 항목에 넣어주면 된다.

사용자는 msg_header_t 구조체의 각각의 항목에 적절한 값을 저장해야 한다. 구조체는 사용자가 임의로 설정할 수 있는 항목이기 때문에 TCPGW에서는 msg_header_t의 구조체 항목에 어떤 값도 저장하지 않는다.

msg_header_t 구조체 길이 이상으로 설정할 경우 data에 영향을 주므로 주의해서 값을 사용해야 한다.

- 프로토타입

```
int put_msg_info(msg_header_t *hp, char *data, msg_info_t *info)
```

- 파라미터

파라미터	설명
hp	리모트 노드로 보낼 메시지의 헤더이다.
data	리모트 노드로 보낼 데이터이다.
info	리모트 노드로 보낼 데이터의 정보이다.

- 반환값

  사용자는 실제적으로 리모트 노드에 전송할 데이터의 전체 길이를 반환해야 한다. 메시지 헤더와 실 데이터를 더한 길이를 반환한다.

3.4.19. put_msg_security

put_msg_info() 또는 put_extmsg_info() 함수가 수행되기 전에 호출되는 함수이다. 사용자 데이터를 가공할 필요가 있을 경우에 사용한다. put_extmsg_info() 함수가 이 함수를 대체할 수 있다.

- 프로토타입

```
int put_msg_security(char **data, int asize, int len)
```

- 파라미터

파라미터	설명
data	리모트 노드로 보낼 데이터 버퍼의 주소값이다. 버퍼의 크기가 부족한 경우에는 realloc()을 통해 버퍼를 확장하는 것을 허용한다. realloc()을 호출하여 버퍼의 포인터가 변경된 경우에는 이 주소값에 변경된 포인터 주소값을 저장해야 한다.
asize	리모트 노드로 보낼 데이터 버퍼에 할당된 메모리 크기이다.
len	메시지 바디의 길이이다.

- 반환값

반환값	설명
양수	가공된 사용자 데이터의 전체 길이를 리턴한다.
0	사용자 데이터의 변경사항을 반영하지 않는다.

- 예제

```
int put_msg_security(char **data, int asize, int len)
{
     ...
     new_size = len * 2;
     if (new_size > asize) {
         tmpbuf = (char *)realloc(*data, new_size);
         if (tmpbuf == NULL) {
             /* error processing */
         }
         *data = tmpbuf;
     }
     ...
     return new_size;
}
```

3.4.20. remote_closed

리모트 노드와 연결을 종료한 후 호출되는 함수로, 리모트 노드와 연결이 끊어진 후에 해야 할 작업이 있다면 함수에서 한다. init_remote_info 함수에서 공유 메모리를 생성하는 로직을 구현한 경우 이 함수에서 해제하는 작업

로직을 구현한다. 채널 수만큼 호출된다.

- 프로토타입

```
int remote_closed(int index, int type)
```

- 파라미터

파라미터	설명
index	TCPGW가 [-n] 또는 [-N] 옵션으로 여러 개의 채널을 연결하고 있는 경우 각각의 채널에 대한 자신의 index 값이다.
type	리모트 노드와 연결된 채널 타입이다. IN_CHANNEL 또는 OUT_CHANNEL인지를 나타낸다.

3.4.21. remote_connected

리모트 노드와 연결을 맺은 후 호출되는 함수로, 리모트 노드와 연결을 맺은 후 해야 할 작업이 있다면 함수에서 하도록 한다. 함수는 채널 수만큼 호출되고 도중에 채널이 해제되었다가 다시 연결될 때도 호출된다.

- 프로토타입

```
int remote_connected(int index, int addr, int portno, int type, int fd)
```

- 파라미터

파라미터	설명
index	TCPGW가 [-n] 또는 [-N] 옵션으로 여러 개의 채널을 연결하고 있는 경우 각각의 채널에 대한 자신의 index 값이다.
addr	리모트 노드의 주소이다.
portno	연결된 서버의 포트 번호이다. TCPGW가 클라이언트 모드로 동작한다면 리모트 노드의 포트 번호이고, TCPGW가 서버 모드로 동작한다면 Listen하고 있는 소켓의 포트 번호이다.
type	리모트 노드와 연결된 채널 타입이다. IN_CHANNEL 또는 OUT_CHANNEL인지를 나타낸다.
fd	리모트 노드와 연결된 소켓 번호이다.

3.4.22. remote_connected_ipv6

IPv6 프로토콜 환경에서 리모트 노드와 연결을 맺은 후 호출되는 함수로 리모트 노드와 연결을 맺은 후 해야 할 작업이 있다면 함수에서 하도록 한다. 함수는 채널 수만큼 호출되고 도중에 채널이 해제되었다가 다시 연결될 때도 호출된다.

IPv6 환경에서는 반드시 remote_connected Callback 함수가 아닌 remote_connected_ipv6 Callback 함수를 설정해야 한다. 이 Callback 함수를 설정하지 않고 IPv6 환경에서 사용할 경우에는 remote_connected 함수가 호출되지만 ipaddr 값에 임의의 값이 설정되며 slog에 Warning 메시지가 기록된다.

- 프로토타입

```
#include <arpa/inet.h>
int remote_connected_ipv6(int index, struct sockaddr *saddr, int portno, int type, int fd)
```

- 파라미터

파라미터	설명
index	TCPGW가 [-n] 또는 [-N] 옵션으로 여러 개의 채널을 연결하고 있는 경우 각각의 채널에 대한 자신의 index 값이다.
saddr	리모트 노드의 주소 및 포트 번호가 저장되어 있다. IPv4 또는 IPv6 프로토콜 환경에 적합한 sockaddr 구조체 타입으로 캐스팅하여 사용한다.
portno	연결된 서버의 포트 번호이다. TCPGW가 클라이언트 모드로 동작한다면 리모트 노드의 포트 번호이고, TCPGW가 서버 모드로 동작한다면 Listen하고 있는 소켓의 포트 번호이다.
type	리모트 노드와 연결된 채널 타입이다. IN_CHANNEL 또는 OUT_CHANNEL인지를 나타낸다.
fd	리모트 노드와 연결된 소켓 번호이다.

- 예제

```
int
remote_connected_ipv6(int index, struct sockaddr *saddr, int portno, int type, int fd)
{
    char buf[INET6_ADDRSTRLEN];
    char *ipaddr = NULL;
    int cli_portno;

    if (index < 0)
        return -1;

    if (saddr->sa_family == AF_INET6) {
        ipaddr = (char *)inet_ntop(AF_INET6, &(((struct sockaddr_in6 *)saddr)->sin6_addr), buf,
sizeof(buf));
        cli_portno = ntohs(((struct sockaddr_in6 *)saddr)->sin6_port);
    } else if (saddr->sa_family == AF_INET) {
        ipaddr = (char *)inet_ntop(AF_INET, &(((struct sockaddr_in *)saddr)->sin_addr), buf,
sizeof(buf));
        cli_portno = ntohs(((struct sockaddr_in *)saddr)->sin_port);
    }
    if (ipaddr == NULL)
        ipaddr = "unknown";

    printf("remote_connected_ipv6, REMOTE[%d] IPADDR[%s] CLIPORT[%d] SVRPORT[%d] TYPE[%d] fd[%d]
connected\n",
            index, ipaddr, cli_portno, portno, type, fd);
```

```
    return 1;
}
```

3.4.23. reset_extping_msg

채널 장애 감지(TCP/IP ping) 메시지의 전송 여부를 결정하고, 전송 메시지를 재설정하기 위해 주기적으로 호출되는 함수이다.

set_ping_msg 함수에서 설정한 interval 주기에 따라 ping 메시지를 리모트 서버로 전송하기 전에, 이 함수가 호출되어 전송 메시지를 확인하고 필요한 경우 수정할 수 있다. 또한 반환값에 따라 ping 메시지를 리모트 서버에 전송할지 여부를 결정한다.

- 프로토타입

```
int reset_extping_msg(msg_header_t *hp, char *data, int len)
```

- 파라미터

파라미터	설명
hp	채널 장애 감지를 위해 전송할 메시지의 헤더이다. 기본값으로 set_ping_msg 함수에서 사용자가 정의한 헤더가 설정된다.
data	재설정할 메시지 바디의 포인터이다. 버퍼의 최대 크기는 len 에 설정된 크기이다. 실제 전송할 버퍼의 크기는 반환값에 지정한다.
len	사용 가능한 메시지 바디의 최대 크기이다.

- 반환값

반환값	설명
양수	ping 메시지를 리모트 서버로 전송한다. 지정된 크기만큼의 메시지 바디를 전송한다.
0	ping 메시지를 리모트 서버로 전송한다. 메시지 바디를 전송하지 않는다.
음수	ping 메시지를 리모트 서버로 전송하지 않는다.

- 예제

```
int reset_extping_msg(msg_header_t *hp, char *data, int len)
{
        int body_len;
        char *message = "reset_msg";

        body_len = min(len, strlen(message));
        strncpy(data, message, (body_len - 1));
        data[(body_len - 1)] = '\0';
        hp->len = body_len;
        printf("reset_extping_msg : data = %s\n", data);
```

```
        return body_len;
}
```

3.4.24. reset_ping_msg

채널 장애 감지(TCP/IP ping) 메시지의 전송 여부를 결정하고, 전송 메시지를 재설정하기 위해 주기적으로 호출되는 함수이다.

set_ping_msg 함수에서 설정한 interval 주기에 따라 ping 메시지를 리모트 서버로 전송하기 전에, 이 함수가 호출되어 전송 메시지를 확인하고 필요한 경우 수정할 수 있다. 또한 반환값에 따라 ping 메시지를 리모트 서버에 전송할지 여부를 결정한다.

- 프로토타입

```
int reset_ping_msg(msg_header_t *hp)
```

- 파라미터

파라미터	설명
hp	채널 장애 감지를 위해 전송할 메시지의 헤더이다. 기본값으로 set_ping_msg 함수에서 사용자가 정의한 헤더가 설정된다.

- 반환값

반환값	설명
양수	ping 메시지를 리모트 서버로 전송한다.
음수	ping 메시지를 리모트 서버로 전송하지 않는다.

- 예제

```
int reset_ping_msg(msg_header_t *hp)
{
        hp->len = 0;
        hp->msgtype = HEALTH_CHECK;
        return 1;
}
```

3.4.25. set_service_timeout

서비스 타임아웃이 발생한 경우 사용자가 호출할 수 있는 함수이다.

- 프로토타입

```
int set_service_time_out(int uid, char *header)
```

- 파라미터

파라미터	설명
uid	서비스 타임아웃이 발생한 거래의 user ID이다.
header	서비스 타임아웃이 발생한 거래의 헤더이다.

3.4.26. set_ping_msg

채널 장애 감시를 위해서 보낼 메시지나 메시지의 주기, 타임아웃 등을 설정할 수 있는 사용자 함수로 [-x] 옵션을 지정할 경우 반드시 설정해야 한다. 이 함수는 TCPGW 프로그램이 기동할 때 한 번 호출된다.

- 프로토타입

```
int set_ping_msg(msg_header_t *hp, int *interval, int *binterval, int *timeout,
                 int *mode)
```

- 파라미터

파라미터	설명
hp	채널 장애 감시를 위해서 주기적으로 보내지는 메시지이다. 사용자는 이 메시지를 설정해야 한다.
interval	채널 장애 감시 주기이다. 0일 경우 채널 장애 감지 기능은 비활성화된다. (단위: 초) 반드시 interval이 timeout 보다 큰 값으로 설정해야 된다.
binterval	백업 채널 모드일 때 메인 채널의 상태를 체크하는 주기이다. 0일 경우 메인 채널의 상태 체크 기능은 비활성화된다. (단위: 초)
timeout	채널 장애 감시 타임아웃으로 시간 내에 응답이 오지 않으면 채널의 연결은 끊어진다. (단위: 초) 실제로 채널의 연결이 종료되는 시점은 interval 주기에 따라 ping 메시지를 전송할 때 reset_ping_msg()가 호출되기 전이다. 0일 경우에는 Ping 메시지만 보내고 Pong 메시지에 대해서는 신경쓰지 않는다. (Half Duplex 장애 감지) 반드시 interval이 timeout 보다 큰 값으로 설정해야 된다.

파라미터	설명
mode	장애 감시할 채널 종류를 지정한다. • 0 : OUT 채널 • 1 : IN 채널 • 2 : 모든 채널

- 반환값

  에러가 발생할 때 음수값을 반환해야 하며, 이 경우 장애 감시 기능은 비활성화된다.

3.4.27. set_extping_msg

채널 장애 감시를 위해 보낼 메시지나 메시지의 주기, 타임아웃 등을 설정할 수 있는 사용자 함수로 [?x] 옵션을 지정할 경우 반드시 설정해야 한다. 이 함수는 TCPGW 프로그램이 기동할 때 한 번 호출된다.

- 프로토타입

```
int set_extping_msg(msg_header_t *hp, char *data, int len, int *interval,
                    int *binterval, int *timeout, int *mode)
```

- 파라미터

파라미터	설명
hp	채널 장애 감시를 위해서 주기적으로 보내지는 메시지이다. 사용자는 이 메시지를 설정해야 한다.
data	채널 장애 감시를 위해서 주기적으로 보내지는 메시지 뒤에 함께 전송될 메시지 바디이다. 필요한 경우에 설정한다.
len	채널 장애 감지 메시지 바디의 최대 길이이다. [-b] 옵션으로 최대 길이를 설정하고, 지정하지 않으면 크기는 0이다.
interval	채널 장애 감시 주기이다. 0일 경우 채널 장애 감지 기능은 비활성화된다. (단위: 초)
binterval	백업 채널 모드일 때 메인 채널의 상태를 체크하는 주기이다. 0일 경우 메인 채널의 상태 체크 기능은 비활성화된다. (단위: 초)
timeout	채널 장애 감시 타임아웃으로 시간 내에 응답이 오지 않으면 채널의 연결은 끊어진다. (단위: 초) 0일 경우에는 Ping 메시지만 보내고 Pong 메시지에 대해서는 신경쓰지 않는다. (Half Duplex 장애 감지)

파라미터	설명
mod	• 0 : OUTBOUND 채널 감지 • 1 : INBOUND 채널 감지 • 2 : OUTBOUND / INBOUND 채널을 모두 감지

- 반환값

반환값	설명
0	함수 수행이 성공한 경우이다.
음수	함수 수행이 실패한 경우이다.

- 예제

```
int set_extping_msg(msg_header_t *hp, char *data, int len, int *interval,
                    int *binterval, int *timeout, int *mode)
{
        msg_body_t *body;

        body = (msg_body_t *)data;
        memset(body->data, 0x00, 52);

        memcpy(body->data, "tmax50", 7);
        body->data[51] = 0;
        printf("set_extping_msg : data = %s\n", body->data);

        hp->len = 7;
        *interval = 10;
        *binterval = 20;
        *timeout = 100;
        *mode = 0; /* OUTBOUND CHANNEL */

        return 1;
}
```

3.4.28. set_error_msg

TCPGW를 통한 리모트 노드와의 거래 도중 타임아웃 혹은 네트워크 단절과 같은 에러가 발생했을 경우 자동으로 호출되는 함수로, 해당 함수 내에서 사용자는 사용자 헤더 또는 사용자 데이터를 수정할 수 있다.

- 프로토타입

```
int set_error_msg(msg_header_t *hp, int err, char *data, int len)
```

- 파라미터

파라미터	설명
hp	에러가 발생했을 경우 전송된 메세지의 헤더로 사용자가 수정할 수 있다.
data	에러가 발생했을 경우 전송된 데이터로 사용자가 수정할 수 있다.
len	메시지 바디의 길이이다.

- 반환값

반환값	설명
양수	리턴한 길이만큼의 데이터를 전송한다. 단, 이 경우 CLOPT="-I" 옵션을 설정해야만 적용된다.
0	사용자 헤더 부분까지만 전송한다.
음수	해당 메세지를 CLH로 전송하지 않는다.

- 오류

에러 코드	설명
[TPECLOSE]	리모트 노드로 서비스를 요청한 이후에 리모트 노드와 연결이 끊어진 경우에 발생한다.
[TPENOENT]	[-E] 옵션을 사용할 경우 서비스 요청마다 리모트 노드와 연결을 하고 데이터를 전송하는데, 동시에 호출한 수가 [-n] 옵션으로 지정한 채널의 수를 초과했을 때 발생한다.
[TPENOREADY]	리모트 노드와의 연결이 끊어져 사용할 수 있는 채널이 없는 경우에 발생한다.
[TPEOS]	TCPGW 내부에서 메모리를 확보하는데, 확보되지 않을 경우에 발생한다.
[TPEPROTO]	tpforward로 TCPGW를 호출할 때 TCPGW가 동기형 모드가 아닌 비동기형 모드일 경우 발생한다. tpforward & tprelay 방식은 반드시 동기형 방식이어야 한다.
[TPESVCERR]	put_msg_info 에서 0 혹은 음수를 리턴하는 경우 발생한다.
[TPESYSTEM]	TCPGW에서 코드 변환을 사용할 때 요청한 데이터에 대한 map 파일이 로드되지 않을 경우에 발생한다. 또는 코드 변환오류가 발생하여 사용자 함수 put_msg_info에서 음수를 반환한 경우 리모트 노드로 데이터를 전송할 때 오류가 발생한 경우이다.
[TPETIME]	리모트 노드로 서비스를 요청하고 지정된 시간 내에 응답이 없을 경우에 발생한다.

- 예제

```
int set_error_msg(msg_header_t *hp, int err, char *data, int len)
{
     msg_body_t * body;
     body = (msg_body_t *)data;
     strcpy(body->data, "changed hello data");
     /* 에러 메시지에는 데이터가 포함되지 않으므로
       데이터 부분까지 전달되기 위해서는 -I 옵션을 사용해야 한다.
       사용하지 않을 시 사용자 헤더까지만 CLH로 전달된다.   */
     /* 사용자헤더가 없는 경우에  hp와 data의 값이 같을 수 있다. */
     strcpy(hp->retsvcname, "RECVSVC_CHANGE");
     return len;
```

```
}
```

3.5. TDL 함수

TDL 함수의 자세한 내용은 "Tmax Programming Guide (Dynamic Library)"를 참고한다.

3.5.1. tdlcall

최신 버전의 동적 모듈 함수를 호출하는 함수로 동적 모듈 함수는 반드시 **long funcname(void *args)** 형태의 원형을 가져야 한다. TDL 환경 파일(tdl.cfg)의 VERSION이 1 또는 2로 설정되었을 때 사용 가능하다. 동적 모듈은 최초 tdlcall()될 때 로드되며 특별히 업데이트(tdlupdate)가 되지 않을 경우 재사용하여 성능 저하를 해소한다. 버전 정합성(Version Consistency) 유지 상황에서는 정합성을 반영한 버전이 사용된다.

- 프로토타입

```
#include <tdlcall.h>
int tdlcall(char *funcname, void *args, long *urcode, int flags)
```

- 파라미터

파라미터	설명
funcname	동적 모듈의 함수명이다. (최대 크기: TDL_FUNCNAME_SIZE – 1)
args	호출될 동적 모듈 함수의 파라미터이다.
urcode	호출된 동적 모듈 함수의 반환값이다.
flags	TDLTRAN으로 설정이 가능하며 이 경우에 urcode로 반드시 Global Sequence 번호를 전달해야 한다. 사용하지 않을 경우 TDL_NOFLAGS로 설정한다.

- 반환값

반환값	설명
TDL_OK	성공적으로 종료한 경우이다.
TDL_OPEN_ERROR	dlopen()을 사용할 때 에러가 발생한 경우이다.
TDL_SYM_ERROR	dlsym()을 사용할 때 에러가 발생한 경우이다.
TDL_CLOSE_ERROR	dlclose()를 사용할 때 에러가 발생한 경우이다.
TDL_SYSTEM_ERROR	기타 시스템 콜을 사용하다가 에러가 발생한 경우이다.
TDL_INT _ERROR	라이브러리 내부에서 오류가 발생한 경우이다.
TDL_ENOFUNC	해당 모듈이나 함수를 찾을 수 없다.
TDL_ENV_ERROR	환경변수 설정에 오류가 있다.
TDL_VER_ERROR	TDL 공유 메모리 버전과 환경 파일(tdl.cfg) 버전이 일치하지 않는다.

반환값	설명
TDL_ARG _ERROR	잘못된 파라미터가 설정되었다.
TDL_ENOLIB	지정한 라이브러리를 찾을 수 없다.
TDL_TRAN_ERROR	버전 정합성 처리 중 에러가 발생했다.
TDL_EINACTIVE	모듈이 일시적으로 사용 중지된 상태이다.

3.5.2. tdlcall2

최신 버전의 동적 모듈 함수를 호출하는 함수로 라이브러리명과 함수명을 키로 사용한다. 동적 모듈 함수는 반드시 **long funcname(void *args)** 형태의 원형을 가져야 한다. 라이브러리명과 함수명을 키로 사용하는 것을 제외하고는 tdlcall() 함수와 동일한 기능을 제공한다.

동적 모듈 함수는 반드시 long funcname(void *args) 형태의 원형을 가져야 한다. TDL 환경 파일(tdl.cfg)에 VERSION=3으로 설정되었을 때 사용 가능하다.

- 프로토타입

```
#include <tdlcall.h>
int tdlcall2(char *libname, char *funcname, void *args, long *urcode, int flags)
```

- 파라미터

파라미터	설명
libname	동적 모듈의 라이브러리명이다. (최대 크기: TDL_FUNCNAME_SIZE – 1)
funcname	동적 모듈의 함수명이다. (최대 크기: TDL_FUNCNAME_SIZE – 1)
args	호출될 동적 모듈 함수의 파라미터이다.
urcode	호출된 동적 모듈 함수의 반환값이다.
flags	TDLTRAN으로 설정이 가능하며 이 경우에 urcode로 반드시 Global Sequence 번호를 전달해야 한다. 사용하지 않을 경우 TDL_NOFLAGS로 설정한다.

- 반환값

반환값	설명
TDL_OK	성공적으로 종료한 경우이다.
TDL_OPEN_ERROR	dlopen()을 사용할 때 에러가 발생한 경우이다.
TDL_SYM_ERROR	dlsym()을 사용할 때 에러가 발생한 경우이다.
TDL_CLOSE_ERROR	dlclose()를 사용할 때 에러가 발생한 경우이다.
TDL_SYSTEM_ERROR	기타 시스템 콜을 사용하다가 에러가 발생한 경우이다.
TDL_INT _ERROR	라이브러리 내부에서 오류가 발생한 경우이다.
TDL_ENOFUNC	해당 모듈이나 함수를 찾을 수 없다.

반환값	설명
TDL_ENV_ERROR	환경변수 설정에 오류가 있다.
TDL_VER_ERROR	TDL 공유 메모리 버전과 환경 파일(tdl.cfg) 버전이 일치하지 않는다.
TDL_ARG _ERROR	잘못된 파라미터가 설정되었다.
TDL_ENOLIB	지정한 라이브러리를 찾을 수 없다.
TDL_TRAN_ERROR	버전 정합성 처리 중 에러가 발생했다.
TDL_EINACTIVE	모듈이 일시적으로 사용 중지된 상태이다.

3.5.3. tdlcall2s

최신 버전의 동적 모듈 함수를 호출하는 함수로 라이브러리명과 함수명을 키로 사용한다. 동적 모듈 함수는 반드시 **long funcname(void *input, void *output)** 형태의 원형을 가져야 한다.

라이브러리명과 함수명을 키로 사용하는 것을 제외하고는 tdlcall() 함수와 동일한 기능을 제공한다. TDL 환경 파일(tdl.cfg)에 VERSION=3으로 설정되었을 때 사용 가능하다.

- 프로토타입

```
#include <tdlcall.h>
int tdlcall2s(char *libname, char *funcname, void *input, void *output,
              long *urcode, int flags)
```

- 파라미터

파라미터	설명
libname	동적 모듈의 라이브러리명이다. (최대 크기: TDL_FUNCNAME_SIZE – 1)
funcname	동적 모듈의 함수명이다. (최대 크기: TDL_FUNCNAME_SIZE – 1)
input	호출될 동적 모듈 함수의 입력 버퍼 포인터이다.
output	호출될 동적 모듈 함수의 출력 버퍼 포인터이다.
urcode	호출된 동적 모듈 함수의 반환값이다.
flags	TDLTRAN으로 설정이 가능하며 이 경우에 urcode로 반드시 Global Sequence 번호를 전달해야 한다. 사용하지 않을 경우 TDL_NOFLAGS로 설정한다.

- 반환값

반환값	설명
TDL_OK	성공적으로 종료한 경우이다.
TDL_OPEN_ERROR	dlopen()을 사용할 때 에러가 발생한 경우이다.
TDL_SYM_ERROR	dlsym()을 사용할 때 에러가 발생한 경우이다.
TDL_CLOSE_ERROR	dlclose()를 사용할 때 에러가 발생한 경우이다.

반환값	설명
TDL_SYSTEM_ERROR	기타 시스템 콜을 사용하다가 에러가 발생한 경우이다.
TDL_INT _ERROR	라이브러리 내부에서 오류가 발생한 경우이다.
TDL_ENOFUNC	해당 모듈이나 함수를 찾을 수 없다.
TDL_ENV_ERROR	환경변수 설정에 오류가 있다.
TDL_VER_ERROR	TDL 공유 메모리 버전과 환경 파일(tdl.cfg) 버전이 일치하지 않는다.
TDL_ARG _ERROR	잘못된 파라미터가 설정되었다.
TDL_ENOLIB	지정한 라이브러리를 찾을 수 없다.
TDL_TRAN_ERROR	버전 정합성 처리 중 에러가 발생했다.
TDL_EINACTIVE	모듈이 일시적으로 사용 중지된 상태이다.

3.5.4. tdlcall2v

최신 버전의 동적 모듈 함수를 호출하는 함수로 라이브러리명과 함수명을 키로 사용한다. 동적 모듈 함수는 반드시 **long funcname(int argc, char *argv[])** 형태의 원형을 가져야 한다.

라이브러리명과 함수명을 키로 사용하는 것을 제외하고는 tdlcall() 함수와 동일한 기능을 제공한다. TDL 환경 파일(tdl.cfg)에 VERSION=3으로 설정되었을 때 사용 가능하다.

- 프로토타입

```
#include <tdlcall.h>
int tdlcall2v(char *libname, char *funcname, int argc, char *argv[],
              long *urcode, int flags)
```

- 파라미터

파라미터	설명
libname	동적 모듈의 라이브러리명이다. (최대 크기: TDL_FUNCNAME_SIZE – 1)
funcname	동적 모듈의 함수명이다. (최대 크기: TDL_FUNCNAME_SIZE – 1)
argc	호출될 동적 모듈 함수의 파라미터 개수이다.
argv	호출될 동적 모듈 함수의 파라미터 벡터이다.
urcode	호출된 동적 모듈 함수의 반환값이다.
flags	TDLTRAN으로 설정이 가능하며 이 경우에 urcode로 반드시 Global Sequence 번호를 전달해야 한다. 사용하지 않을 경우 TDL_NOFLAGS로 설정한다.

- 반환값

반환값	설명
TDL_OK	성공적으로 종료한 경우이다.

반환값	설명
TDL_OPEN_ERROR	dlopen()을 사용할 때 에러가 발생한 경우이다.
TDL_SYM_ERROR	dlsym()을 사용할 때 에러가 발생한 경우이다.
TDL_CLOSE_ERROR	dlclose()를 사용할 때 에러가 발생한 경우이다.
TDL_SYSTEM_ERROR	기타 시스템 콜을 사용하다가 에러가 발생한 경우이다.
TDL_INT _ERROR	라이브러리 내부에서 오류가 발생한 경우이다.
TDL_ENOFUNC	해당 모듈이나 함수를 찾을 수 없다.
TDL_ENV_ERROR	환경변수 설정에 오류가 있다.
TDL_VER_ERROR	TDL 공유 메모리 버전과 환경 파일(tdl.cfg) 버전이 일치하지 않는다.
TDL_ARG _ERROR	잘못된 파라미터가 설정되었다.
TDL_ENOLIB	지정한 라이브러리를 찾을 수 없다.
TDL_TRAN_ERROR	버전 정합성 처리 중 에러가 발생했다.
TDL_EINACTIVE	모듈이 일시적으로 사용 중지된 상태이다.

3.5.5. tdlcallva

최신 버전의 동적 모듈 함수를 호출하는 함수로 함수명을 키로 사용하는 함수이다. 동적 모듈 함수의 프로토 타입이 고정되지 않은 경우에 대해서 파라미터를 그대로 전달하여 함수를 호출한다. 단, 전달되는 인자들은 모두 (void *) 포인터 타입을 가져야 한다. 동적 모듈 함수에서도 전달받는 파라미터가 모두 (void *) 포인터 타입으로 작성되어야 한다.

TDL 환경 파일(tdl.cfg)에 VERSION=1 또는 VERSION=2로 설정된 경우 사용 가능하다.

- 프로토타입

```
#include <tdlcall.h>
int tdlcallva(char *funcname, long urcode, int flags, int rettype, void *retval,
              int argc, ⋯);
```

- 파라미터

파라미터	설명
funcname	동적 모듈 함수명이다. (최대 크기: TDL_FUNCNAME_SIZE – 1)
urcode	다른 tdlcall() 계열의 함수와 달리 Global Sequence 번호를 전달할 때만 사용한다. 사용하지 않을 경우에는 0을 입력한다.
flags	TDLTRAN으로 설정이 가능하며 이 경우에 urcode로 반드시 Global Sequence 번호를 전달해야 한다. 사용하지 않을 경우 TDL_NOFLAGS로 설정한다.

파라미터	설명
rettype	호출될 동적 모듈 함수의 리턴 타입을 정의한다. 다음의 값을 설정할 수 있다. • TDL_VA_CHAR : 호출될 동적 모듈 함수의 리턴 타입이 char 형이다. • TDL_VA_SHORT : 호출될 동적 모듈 함수의 리턴 타입이 short 형이다. • TDL_VA_INT : 호출될 동적 모듈 함수의 리턴 타입이 int 형이다. • TDL_VA_LONG : 호출될 동적 모듈 함수의 리턴 타입이 long 형이다. • TDL_VA_FLOAT : 호출될 동적 모듈 함수의 리턴 타입이 float 형이다. • TDL_VA_DOUBLE : 호출될 동적 모듈 함수의 리턴 타입이 double 형이다. • TDL_VA_PVOID : 호출될 동적 모듈 함수의 리턴 타입이 void * 형이다. • TDL_VA_VOID : 호출될 동적 모듈 함수의 리턴 타입이 void 형이다.
retval	호출된 동적 모듈 함수의 리턴값을 저장할 버퍼의 포인터를 전달한다.
argc	호출될 동적 모듈 함수로 전달될 argument의 수를 입력한다. 현재 최대 10개까지 가능하다.
...	호출될 동적 모듈 함수로 전달할 실제 파라미터들을 가변인자형으로 입력한다. 반드시 (void *) 포인터 타입의 파라미터를 전달해야 한다.

- 반환값

반환값	설명
TDL_OK	성공적으로 종료한 경우이다.
TDL_OPEN_ERROR	dlopen()을 사용할 때 에러가 발생한 경우이다.
TDL_SYM_ERROR	dlsym()을 사용할 때 에러가 발생한 경우이다.
TDL_CLOSE_ERROR	dlclose()를 사용할 때 에러가 발생한 경우이다.
TDL_SYSTEM_ERROR	기타 시스템 콜을 사용하다가 에러가 발생한 경우이다.
TDL_INT _ERROR	라이브러리 내부에서 오류가 발생한 경우이다.
TDL_ENOFUNC	해당 모듈이나 함수를 찾을 수 없다.
TDL_ENV_ERROR	환경변수 설정에 오류가 있다.
TDL_VER_ERROR	TDL 공유 메모리 버전과 환경 파일(tdl.cfg) 버전이 일치하지 않는다.
TDL_ARG _ERROR	잘못된 인자가 설정되었다.
TDL_ENOLIB	지정한 라이브러리를 찾을 수 없다.
TDL_TRAN_ERROR	버전 정합성 처리 중 에러가 발생했다.
TDL_EINACTIVE	모듈이 일시적으로 사용 중지된 상태이다.

- 예제
  - 동적 모듈 함수

```
int myfunc(double *a, double *b, double *c) {
    double sum;
    return (int)(((double)(*a) + (double)(*b) + (double)(*c))/3);
}

char * myfunc2(int *type) {
    char *msg;
    switch (*type) {
        case 1: msg = “foo”; break;
        case 2: msg = “bar”;break;
    }
    return msg;
}
```

◦ 호출 프로그램

```
#include <tdlcall.h>
int main(int argc, char *argv[]) {
    int n;
    long urcode;
    int retval;
    double a, b, c;
    int type;
    char *retmsg;

    urcode = tdlgetseqno();
    a = 2.5;
    b = 3.0;
    c = 3.5;
    if ((n = tdlcallva2(“mylib001”, “myfunc”, urcode, TDLTRAN,
             TDL_VA_INT, &retval, 3, &a, &b, &c)) != TDL_OK) {
        error processing;
    }

    type = 1;
    if ((n = tdlcallva2(“mylib001”, “myfunc2”, urcode, TDLTRAN,
             TDL_VA_PVOID, &retmsg, 1, &type)) != TDL_OK) {
        error processing;
    }
}
```

3.5.6. tdlcallva2

최신 버전의 동적 모듈 함수를 호출하는 함수로 라이브러리명과 함수명을 키로 사용하는 함수이다. 동적 모듈 함수의 프로토타입이 고정되지 않은 경우에 대해서 파라미터를 그대로 전달하여 함수를 호출한다. 단, 전달되는 인자들은 모두 (void *) 포인터 타입을 가져야 한다. 동적 모듈 함수에서도 전달받는 파라미터가 모두 (void *) 포인터 타입으로 작성되어야 한다.

라이브러리명과 함수명을 키로 사용하는 것을 제외하고는 tdlcallva() 함수와 동일한 기능을 제공한다. TDL 환경 파일(tdl.cfg)에 VERSION=3으로 설정된 경우 사용 가능하다.

- 프로토타입

```
#include <tdlcall.h>
int tdlcallva2(char *libname, char *funcname, long urcode, int flags,
               int rettype, void *retval, int argc, ⋯);
```

- 파라미터

파라미터	설명
libname	동적 모듈 라이브러리명이다. (최대 크기: TDL_FUNCNAME_SIZE – 1)
funcname	동적 모듈 함수명이다. (최대 크기: TDL_FUNCNAME_SIZE – 1)
urcode	다른 tdlcall() 계열의 함수와 달리 Global Sequence 번호를 전달할 때만 사용한다. 사용하지 않을 경우에는 0을 입력한다.
flags	TDLTRAN으로 설정이 가능하며 이 경우에 urcode로 반드시 Global Sequence 번호를 전달해야 한다. 사용하지 않을 경우 TDL_NOFLAGS로 설정한다.
rettype	호출될 동적 모듈 함수의 리턴 타입을 정의한다. 다음의 값을 설정할 수 있다. • TDL_VA_CHAR : 호출될 동적 모듈 함수의 리턴 타입이 char 형이다. • TDL_VA_SHORT : 호출될 동적 모듈 함수의 리턴 타입이 short 형이다. • TDL_VA_INT : 호출될 동적 모듈 함수의 리턴 타입이 int 형이다. • TDL_VA_LONG : 호출될 동적 모듈 함수의 리턴 타입이 long 형이다. • TDL_VA_FLOAT : 호출될 동적 모듈 함수의 리턴 타입이 float 형이다. • TDL_VA_DOUBLE : 호출될 동적 모듈 함수의 리턴 타입이 double 형이다. • TDL_VA_PVOID : 호출될 동적 모듈 함수의 리턴 타입이 void * 형이다. • TDL_VA_VOID : 호출될 동적 모듈 함수의 리턴 타입이 void 형이다.
retval	호출된 동적 모듈 함수의 리턴값을 저장할 버퍼의 포인터를 전달한다.
argc	호출될 동적 모듈 함수로 전달될 argument의 수를 입력한다. 현재 최대 10개까지 가능하다.
...	호출될 동적 모듈 함수로 전달할 실제 파라미터들을 가변인자형으로 입력한다. 반드시 (void *) 포인터 타입의 파라미터를 전달해야 한다.

- 반환값

반환값	설명
TDL_OK	성공적으로 종료한 경우이다.
TDL_OPEN_ERROR	dlopen()을 사용할 때 에러가 발생한 경우이다.
TDL_SYM_ERROR	dlsym()을 사용할 때 에러가 발생한 경우이다.
TDL_CLOSE_ERROR	dlclose()를 사용할 때 에러가 발생한 경우이다.
TDL_SYSTEM_ERROR	기타 시스템 콜을 사용하다가 에러가 발생한 경우이다.

반환값	설명
TDL_INT _ERROR	라이브러리 내부에서 오류가 발생한 경우이다.
TDL_ENOFUNC	해당 모듈이나 함수를 찾을 수 없다.
TDL_ENV_ERROR	환경변수 설정에 오류가 있다.
TDL_VER_ERROR	TDL 공유 메모리 버전과 환경 파일(tdl.cfg) 버전이 일치하지 않는다.
TDL_ARG _ERROR	잘못된 인자가 설정되었다.
TDL_ENOLIB	지정한 라이브러리를 찾을 수 없다.
TDL_TRAN_ERROR	버전 정합성처리 중 에러가 발생했다.
TDL_EINACTIVE	모듈이 일시적으로 사용 중지된 상태이다.

3.5.7. tdlclose

해당 모듈의 레퍼런스 카운트를 0으로 초기화하거나, 모듈을 직접 메모리에서 해제한다.

- 프로토타입

```
#include <tdlcall.h>
int tdlclose(char *name, int flags)
```

- 파라미터

파라미터	설명
name	해당하는 모듈명이다.
flags	0으로 설정할 경우 해당하는 모듈의 레퍼런스 카운트만 0으로 초기화하고, 메모리에서 해제하지 않는다. TDLCLOSE_HARD로 설정할 경우 dlclose하여 해당 모듈을 메모리에서 해제한다.

- 반환값

반환값	설명
TDL_OK	함수 호출에 성공한 경우이다.
TDL_ENOLIB	해당하는 이름의 모듈이 존재하지 않는 경우이다.

3.5.8. tdlcreate

최신 버전의 동적 모듈에서 Class Factory를 사용하여 클래스 인스턴스를 생성하는 함수로 라이브러리명과 namespace, 클래스명을 키로 사용하고, TDL 환경 파일(tdl.cfg)에 VERSION=4로 설정되었을 때 사용 가능하다.

- 프로토타입

```
#include <tdlcall.h>
int tdlcreate(char *libname, char *namespace, char *classname, void *args, void *object, long
*urcode, int flags)
```

- 파라미터

파라미터	설명
libname	동적 모듈의 라이브러리명이다. (최대 크기: TDL_FUNCNAME_SIZE – 1)
namespace	namespace를 지정한다. (최대 크기: TDL_FUNCNAME_SIZE – 1)
classname	인스턴스를 생성할 클래스명이다. (최대 크기: TDL_FUNCNAME_SIZE – 1)
args	tdlcreate_cb() 콜백 함수로 사용자 정의 데이터를 전달할 파라미터이다.
object	tdlcreate_cb() 콜백 함수에서 생성된 클래스 인스턴스의 레퍼런스를 저장할 변수의 포인터이다. 사용자는 함수 호출이 성공하면 object 파라미터를 적절한 타입으로 변환하여 사용한다.
urcode	tdlcreate_cb() 콜백 함수에서 수행된 결과의 반환값이다.
flags	TDLTRAN으로 설정이 가능하며, 이 경우에 urcode로 반드시 Global Sequence 번호를 전달해야 한다. 사용하지 않을 경우 TDL_NOFLAGS로 설정한다.

- 반환값

반환값	설명
TDL_OK	성공적으로 종료한 경우이다.
TDL_OPEN_ERROR	dlopen()을 사용할 때 에러가 발생한 경우이다.
TDL_SYM_ERROR	dlsym()을 사용할 때 에러가 발생한 경우이다.
TDL_CLOSE_ERROR	dlclose()를 사용할 때 에러가 발생한 경우이다.
TDL_SYSTEM_ERROR	기타 시스템 콜을 사용하다가 에러가 발생한 경우이다.
TDL_INT _ERROR	라이브러리 내부에서 오류가 발생한 경우이다.
TDL_ENOFUNC	해당 모듈이나 함수를 찾을 수 없다.
TDL_ENV_ERROR	환경변수 설정에 오류가 있다.
TDL_VER_ERROR	TDL 공유 메모리 버전과 환경 파일(tdl.cfg) 버전이 일치하지 않는다.
TDL_ARG _ERROR	잘못된 파라미터가 설정되었다.
TDL_ENOLIB	지정한 라이브러리를 찾을 수 없다.
TDL_TRAN_ERROR	버전 정합성 처리 중 에러가 발생했다.
TDL_EINACTIVE	모듈이 일시적으로 사용 중지된 상태이다.

동적 모듈에서 Class Factory를 사용하기 위해서는 반드시 **long tdlcreate_cb(char *namespace, char *classname, void *args, void *object)** 형태의 콜백 함수를 구현해야 한다. 콜백 함수에서는 tdlcreate()에서

전달한 파라미터 정보를 이용해서 실제 클래스 인스턴스를 생성하고, 생성된 인스턴스의 레퍼런스를 object로 전달해주도록 사용자가 구현한다.

사용자는 이 함수로 생성한 인스턴스의 사용이 완료되면 반드시 tdldestroy() 함수를 통해서 인스턴스를 파괴해야 한다. tdlcreate()로 생성한 이후 tdldestroy()로 인스턴스를 삭제하기 전에는 tdlupdate가 중간에 호출되어 동적 모듈이 새로운 버전으로 변경되어도 현재 생성되어 사용 중인 인스턴스는 기존의 버전으로 동작한다. 이 경우에는 tdldestroy()를 호출한 뒤에 다시 tdlcreate()를 호출하면 변경된 동적 모듈의 인스턴스가 생성된다.

Class Factory를 사용하기 위해서 동적 모듈은 tdlcreate_cb()와 tdldestroy_cb() 콜백 함수를 구현해야 한다.

- 콜백 함수 프로토타입

```
long tdlcreate_cb(char *namespace, char *classname, void *args, void *object)
```

- 파라미터

파라미터	설명
namespace	tdlcreate() 함수로부터 전달받은 namespace이다.
classname	tdlcreate() 함수로부터 전달받은 인스턴스를 생성할 클래스명이다.
args	사용자 정의 데이터이다.
object	콜백 함수에서 생성된 클래스 인스턴스의 레퍼런스를 저장할 변수의 포인터이다.

3.5.9. tdldestroy

최신 버전의 동적 모듈에 Class Factory를 사용해서 생성된 클래스 인스턴스를 파괴하는 함수로 라이브러리명과 namespace, 클래스명을 키로 사용하고, TDL 환경 파일(tdl.cfg)에 VERSION=4로 설정되었을 때 사용 가능하다.

- 프로토타입

```
#include <tdlcall.h>
int tdldestroy(char *libname, char *namespcae, char *classname, void *args, void *object, long
*urcode, int flags)
```

- 파라미터

파라미터	설명
libname	동적 모듈의 라이브러리명이다. (최대 크기: TDL_FUNCNAME_SIZE – 1)
namespace	namespace를 지정한다. (최대 크기: TDL_FUNCNAME_SIZE – 1)
classname	인스턴스를 생성할 클래스명이다. (최대 크기: TDL_FUNCNAME_SIZE – 1)
args	tdldestroy_cb() 콜백 함수로 사용자 정의 데이터를 전달할 파라미터이다.

파라미터	설명
object	파괴할 클래스 인스턴스의 레퍼런스를 전달한다. tdldestroy_cb() 콜백 함수에서 해당 레퍼런스를 파괴한다. 반드시 tdlcreate() 함수로 생성한 object만을 사용해야 한다.
urcode	tdldestroy_cb() 콜백 함수에서 수행된 결과의 반환값이다.
flags	TDLTRAN으로 설정이 가능하며 이 경우에 urcode로 반드시 Global Sequence 번호를 전달해야 한다. 사용하지 않을 경우 TDL_NOFLAGS로 설정한다.

- 반환값

반환값	설명
TDL_OK	성공적으로 종료한 경우이다.
TDL_OPEN_ERROR	dlopen()을 사용할 때 에러가 발생한 경우이다.
TDL_SYM_ERROR	dlsym()을 사용할 때 에러가 발생한 경우이다.
TDL_CLOSE_ERROR	dlclose()를 사용할 때 에러가 발생한 경우이다.
TDL_SYSTEM_ERROR	기타 시스템 콜을 사용하다가 에러가 발생한 경우이다.
TDL_INT _ERROR	라이브러리 내부에서 오류가 발생한 경우이다.
TDL_ENOFUNC	해당 모듈이나 함수를 찾을 수 없다.
TDL_ENV_ERROR	환경변수 설정에 오류가 있다.
TDL_VER_ERROR	TDL 공유 메모리 버전과 환경 파일(tdl.cfg) 버전이 일치하지 않는다.
TDL_ARG _ERROR	잘못된 파라미터가 설정되었다.
TDL_ENOLIB	지정한 라이브러리를 찾을 수 없다.
TDL_TRAN_ERROR	버전 정합성 처리 중 에러가 발생했다.
TDL_EINACTIVE	모듈이 일시적으로 사용 중지된 상태이다.

동적 모듈에서 Class Factory를 사용하기 위해서는 반드시 **long tdldestroy_cb(char *namespace, char *classname, void *args, void *object)** 형태의 콜백 함수를 구현해야 한다. 콜백 함수에서는 tdldestroy()에서 전달받은 파라미터 정보를 이용해서 클래스 인스턴스를 파괴하도록 사용자가 구현한다.

사용자는 tdlcreate()로 생성한 인스턴스의 사용이 완료되면 반드시 이 함수를 통해서 인스턴스를 파괴해야 한다. tdlcreate()로 생성한 이후 tdldestroy()로 인스턴스를 삭제하기 전에는 tdlupdate가 중간에 호출되어 동적 모듈이 새로운 버전으로 변경되어도 현재 생성되어 사용 중인 인스턴스는 기존의 버전으로 동작한다. 이 경우에는 tdldestroy()를 호출한 뒤에 다시 tdlcreate()를 호출하면 변경된 동적 모듈의 인스턴스가 생성된다.

- 콜백 함수 프로토타입

```
long tdldestroy_cb(char *namespace, char *classname, void *args, void *object)
```

- 파라미터

파라미터	설명
namespace	tdlcreate() 함수로부터 전달받은 namespace이다.
classname	tdlcreate() 함수로부터 전달받은 인스턴스를 생성할 클래스명이다.
args	사용자 정의 데이터이다.
object	콜백 함수에서 파괴할 클래스 인스턴스의 레퍼런스이다.

3.5.10. tdldone

공유 메모리를 초기화하는 함수이다.

- 프로토타입

```
#include <tdlcall.h>
int tdldone(int flags)
```

- 파라미터

파라미터	설명
flags	현재 사용되지 않는다.

- 반환값

반환값	설명
TDL_OK	함수 호출에 성공한 경우이다.
0보다 작은 값	함수 호출에 실패한 경우이다. tdlerror로 확인할 수 있다.

3.5.11. tdlend

명시적 버전 정합성(Explicit Version Consistency) 유지가 종료된다.

- 프로토타입

```
#include <tdlcall.h>
int tdlend(void)
```

- 반환값

반환값	설명
TDL_OK	함수가 성공적으로 수행된 경우이다.
TDL_TRAN _ERROR	함수 호출 이전에 tdlstart()가 수행된 경우이다. 자세한 내용은 tdlcall을 참고한다.

3.5.12. tdlerror

tdlcall()에 대한 에러가 발생했을 때 문자열 형태로 변환하는 함수이다. 직전의 tdlcall() 함수에서 에러가 발생한 경우 에러에 대한 자세한 정보를 문자열 형태로 전달한다. 주의할 점은 반환값으로 전달되는 포인터는 내부의 정적변수에 대한 포인터이므로 다음 tdlcall()을 호출할 경우에 다른 내용으로 채워져 버릴 수 있다는 것이다.

따라서 이 내용을 보관하거나 수정하고 싶으면 다른 변수로 복사를 하여 사용해야 한다.

- 프로토타입

```
#include <tdlcall.h>
char* tdlerror(int retval)
```

- 파라미터

파라미터	설명
retval	직전 tdlcall() 함수의 반환값이다.

- 반환값

반환값	설명
dlopen fail	tdlcall()의 반환값으로 TDL_OPEN_ERROR를 받을 경우이다.
dlsym fai	tdlcall()의 반환값으로 TDL_SYM_ERROR를 받을 경우이다.
dlclose fail	tdlcall()의 반환값으로 TDL_CLOSE_ERROR를 받을 경우이다.
etc system call error	tdlcall()의 반환값으로 TDL_SYSTEM_ERROR를 받을 경우이다.
TDL lib internal error	tdlcall()의 반환값으로 TDL_INT_ERROR를 받을 경우이다.
funcname not found	tdlcall()의 반환값으로 TDL_ENOFUNC를 받을 경우이다.
environment not found	tdlcall()의 반환값으로 TDL_ENV_ERROR를 받을 경우이다.
shared version mismatch	tdlcall()의 반환값으로 TDL_VER_ERROR를 받을 경우이다.
invalid arguments	tdlcall()의 반환값으로 TDL_ARG_ERROR를 받을 경우이다.
library not found	tdlcall()의 반환값으로 TDL_ENOLIB를 받을 경우이다.
transaction error	tdlcall()의 반환값으로 TDL_TRAN_ERROR를 받을 경우이다.
inactive funcation	tdlcall()의 반환값으로 TDL_EINACTIVE를 받을 경우이다.

3.5.13. tdlfind

모듈의 인덱스를 찾는 함수이다. TDL 환경 파일(tdl.cfg)에 VERSION=1 또는 VERSION=2로 설정된 경우 사용한다.

- 프로토타입

```
#include <tdlcall.h>
int tdlfind(char *funcname, int flags)
```

- 파라미터

파라미터	설명
funcname	찾으려는 함수명이다.
flags	현재 사용되지 않는다.

- 반환값

반환값	설명
0이나 0보다 큰 값	함수 호출에 성공한 경우이다.
0보다 작은 값	함수 호출에 실패한 경우이다. tdlerror로 확인할 수 있다.

3.5.14. tdlfind2

모듈의 인덱스를 찾는 함수이다. TDL 환경 파일(tdl.cfg)의 VERSION=3으로 설정된 경우 사용한다.

- 프로토타입

```
#include <tdlcall.h>
int tdlfind2(char *libname, char *funcname, int flags)
```

- 파라미터

파라미터	설명
libname	찾으려는 라이브러리명이다.
funcname	찾으려는 함수명이다.
flags	현재 사용되지 않는다.

- 반환값

반환값	설명
0이나 0보다 큰 값	함수 호출에 성공한 경우이다.
0보다 작은 값	함수 호출에 실패한 경우이다. tdlerror로 확인할 수 있다.

3.5.15. tdlgetseqno

Global sequence 번호를 가져오는 함수로 이를 반환값으로 반환한다.

- 프로토타입

```
#include <tdlcall.h>
unsigned int tdlgetseqno(void)
```

- 반환값

반환값	설명
0보다 큰 값	함수 호출에 성공한 경우이다.
0	함수 호출에 실패한 경우이다. tdlerror로 확인할 수 있다.

3.5.16. tdlinit

공유 메모리를 초기 설정하는 함수이다.

- 프로토타입

```
#include <tdlcall.h>
int tdlinit(int flags)
```

- 파라미터

파라미터	설명
flags	현재 사용되지 않는다.

- 반환값

반환값	설명
TDL_OK	함수 호출에 성공한 경우이다.
0보다 작은 값	함수 호출에 실패한 경우이다. tdlerror로 확인할 수 있다.

3.5.17. tdlload

tdlcall()을 통해 특정 모듈을 최초로 호출할 경우 공유 메모리의 Hashtable 검색 및 라이브러리의 메모리 적재로 인해 약간의 시간이 소모된다. 이로 인해 최초 호출이 약간 지연되는 현상을 막기 위한 방법으로 Hashtable 검색 및 라이브러리 적재를 tdlcall()을 하기 전에 미리 수행하여 로컬 캐시에 해당 모듈의 정보를 저장하는 함수이다.

TDL 환경 파일(tdl.cfg)에 VERSION=1 또는 VERSION=2로 설정된 경우 tdlload()를 사용하고, VERSION=3일 경우에는 tdlload2(), VERSION=4일 경우에는 tdlload3()을 사용한다.

- 프로토타입

```
#include <tdlcall.h>
```

```
int tdlload(char *funcname, int flags)
```

- 파라미터

파라미터	설명
funcname	로드를 수행할 함수명이다.
flags	현재 사용하지 않는다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
TDL_ENOFUNC	함수 인자로 libname이나 funcname이 NULL이 전달되거나 Hashtable에 존재하지 않는 값을 전달했을 경우이다.
TDL_OPEN_ERROR	Hashtable에 존재하는 동적 라이브러리(Dynamic Library)를 메모리로 적재하는 것이 실패한 경우이다.
TDL_SYSTEM_ERROR	로컬 캐시를 생성하기 위한 시스템 자원 할당이 실패한 경우이다.

3.5.18. tdlload2

tdlload와 동일하므로 자세한 내용은 tdlload를 참고한다.

- 프로토타입

```
#include <tdlcall.h>
int tdlload2(char *libname, char *funcname, int flags)
```

- 파라미터

파라미터	설명
libname	로드를 수행할 라이브러리명이다.
funcname	로드를 수행할 함수명이다.
flags	현재 사용하지 않는다.

- 반환값

  tdlload와 동일하므로 자세한 내용은 tdlload를 참고한다.

3.5.19. tdlresume

일시적으로 중지된 버전 정합성 유지를 재개하는 함수이다.

- 프로토타입

```
#include <tdlcall.h>
int tdlresume(int sd)
```

- 파라미터

파라미터	설명
sd	tdlsuspend() 함수에서 전달받은 Descriptor이다.

- 반환값

반환값	설명
TDL_OK	성공적으로 종료된 경우이다.
TDL_TRAN _ERROR	sd가 유효한 값이 아닌 경우이다. 자세한 내용은 tdlcall을 참고한다.

3.5.20. tdlstart

명시적 버전 정합성(Explicit Version Consistency) 유지가 시작된다.

- 프로토타입

```
#include <tdlcall.h>
int tdlstart(void)
```

- 반환값

반환값	설명
TDL_OK	함수 호출에 성공한 경우이다.
TDL_TRAN _ERROR	함수 호출 이전에 tdlstart()가 수행된 경우이다. 자세한 내용은 tdlcall을 참고한다.

3.5.21. tdlstat

TDL 통계 정보를 출력하는 함수이다.

TDL 환경 파일(tdl.cfg)에 VERSION=1 또는 VERSION=2로 설정된 경우 사용하고, 환경설정에서 MONITOR=Y로 설정해야 한다.

- 프로토타입

```
#include <tdlcall.h>
int tdlstat(char *funcname, struct timeval svc_time, struct timeval cpu_time)
```

- 파라미터

파라미터	설명
funcname	통계 정보를 수집할 함수명이다.
svc_time	통계 정보 중 서비스 타임이 기록되는 변수이다.
cpu_time	통계 정보 중 CPU 타임이 기록되는 변수이다.

- 반환값

반환값	설명
TDL_OK	함수 호출에 성공한 경우이다.
0보다 작은 값	함수 호출에 실패한 경우이다. tdlerror로 확인할 수 있다.

3.5.22. tdlstat2

TDL 통계 정보를 출력하는 함수이다.

TDL 환경 파일(tdl.cfg)에 VERSION=3으로 설정된 경우 사용하고, 환경설정에서 MONITOR=Y로 설정해야 한다.

- 프로토타입

```
#include <tdlcall.h>
int tdlstat2(char *libname, char *funcname, struct timeval svc_time, struct timeval cpu_usrtime,
struct timeval cpu_systime)
```

- 파라미터

파라미터	설명
libname	통계 정보를 수집할 라이브러리명이다.
funcname	통계 정보를 수집할 함수명이다.
svc_time	통계 정보 중 서비스 타임이 기록되는 변수이다.
cpu_usrtime	통계 정보 중 유저 CPU 타임이 기록되는 변수이다.
cpu_systime	통계 정보 중 시스템 CPU 타임이 기록되는 변수이다.

- 반환값

반환값	설명
TDL_OK	함수 호출에 성공한 경우이다.
0보다 작은 값	함수 호출에 실패한 경우이다. tdlerror로 확인할 수 있다.

3.5.23. tdlsuspend

일시적으로 버전 정합성 유지를 중지한다. sd(suspend descriptor)를 반환값으로 반환하며, 최대 동시 sd 개수는 8이다.

- 프로토타입

```
#include <tdlcall.h>
int tdlsuspend(void)
```

- 반환값

반환값	설명
0보다 크거나 같은 값	함수 호출에 성공한 경우이다.
TDL_TRAN _ERROR	현재 버전 정합성 유지 모드가 아닌 경우이다. 자세한 내용은 tdlcall을 참고한다.

3.6. 보안 함수

3.6.1. tmax_accept_crypt

tmax_init_crypt와 마찬가지로 tpstart() 내부에서 클라이언트와 CAS 둘 다 호출된다. 클라이언트는 CAS로부터 tmax_init_crypt에 대한 응답이 오면 이 함수가 호출된다. CAS는 클라이언트의 tmax_accept_crypt가 성공하면 이 함수가 호출된다.

- 프로토타입

```
#include <tmaxapi.h>
int tmax_accept_crypt(const TMAX_SEC_T *ctoken_from_peer, TMAX_SEC_T *ctoken_to_peer, TMAX_SEC_T
*own_ctoken)
```

- 파라미터

파라미터	설명
ctoken_from_peer	클라이언트는 CAS의 tmax_init_crypt 함수의 ctoken_to_peer 토큰을 이 곳에 받는다. CAS는 클라이언트의 tmax_accept_crypt의 ctoken_to_peer 토큰을 이곳에 받는다.
ctoken_to_peer	클라이언트는 CAS에게 전송할 토큰이 있다면 ctoken_to_peer에 할당한다. CAS의 tmax_accept_crypt는 이 파라미터를 사용하지 않는다.
own_ctoken	own_ctoken은 tmax_init_crypt 함수를 통해 생성된 암호화를 위한 정보가 담긴 토큰이다. 만약 재가공할 것이 있다면 own_ctoken에 할당한다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다.

- 관련 함수

  tmax_init_crypt(), tmax_fini_crypt(), tmax_wrap(), tmax_unwrap()

3.6.2. tmax_auth_plugin_init

Tmax 환경설정의 CASOPT에 설정된 값들을 전달한다. CAS의 기동 시에 인증, 인가, 감사 라이브러리와 관련된 초기화 과정이 필요하다면 이 함수에서 구현한다.

- 프로토타입

```
#include <tmaxapi.h>
int tmax_auth_plugin_init(int argc, char *argv[])
```

- 파라미터

파라미터	설명
argc	CASOPT에 설정된 옵션들의 갯수이다.
argv	CASOPT에 설정된 옵션들의 값이다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다.

- 관련함수

  tmax_auth_plugin_fini()

3.6.3. tmax_chk_authen

tpstart() 내부에서 클라이언트의 tmax_init_auth() 호출에 대한 CAS의 인증 검사 함수이다. 또한 해당 클랑이언트에 대한 인증, 인가, 감사를 위한 보안 토큰 정보를 생성한다.

- 프로토타입

```
#include <tmaxapi.h>
```

```
int tmax_chk_authen(const TMAX_SEC_T *atoken_from_peer, TMAX_SEC_T *own_atoken)
```

- 파라미터

파라미터	설명
atoken_from_peer	클라이언트의 tmax_init_auth함수의 atoken_to_peer 토큰을 이 파라미터에 받는다.
own_atoken	해당 클라이언트에 대한 보안 토큰 정보를 이 파라미터에 할당한다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다.

- 관련 함수

  tmax_init_auth()

3.6.4. tmax_chk_author

CAS에서 인가 검사시에 호출되는 함수이다.

- 프로토타입

```
#include <tmaxapi.h>
int tmax_chk_author(int desc, const char *svc_name, const TMAX_SEC_T *atoken_from_peer,
TMAX_SEC_T *own_atoken, int *flag)
```

- 파라미터

파라미터	설명
desc	클라이언트로 부터의 인가 요청이라면 1, 그 외에는 2로 설정된다.
svc_name	tpcall 호출하고자 하는 서비스 이름이다.
atoken_from_peer	인증을 요청해온 상대방의 토큰 정보를 이 파라미터에 받는다.
own_atoken	tmax_chk_authen 함수에서 생성된 해당 클라이언트에 대한 보안 토큰 정보를 이 파라미터에 받는다.
flag	인가 요청한 서비스가 자기 노드의 서비스가 아닌 경우 그 서비스가 있는 노드로 인가 요청을 할 것인지 여부를 결정하는 flag이다. 0으로 설정하면 안하고 그 외에는 다른 노드로 요청한다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다.

- 관련 함수

  tmax_req_auth()

3.6.5. tmax_crypt_plugin_init

Tmax 환경설정의 CASOPT에 설정된 값들을 전달한다. CAS의 기동 시에 암호화 라이브러리와 관련된 초기화 과정이 필요하다면 이 함수에서 구현한다.

- 프로토타입

```
#include <tmaxapi.h>
int tmax_crypt_plugin_init(int argc, char *argv[])
```

- 파라미터

파라미터	설명
argc	CASOPT에 설정된 옵션들의 갯수이다.
argv	CASOPT에 설정된 옵션들의 값이다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다.

- 관련함수

  tmax_crypt_plugin_fini()

3.6.6. tmax_crypt_plugin_fini

Tmax 환경설정의 CASOPT에 설정된 값들을 전달한다. CAS의 종료하는 경우 암호화 라이브러리와 관련된 후처리 작업이 필요하다면 이 함수에서 구현한다.

- 프로토타입

```
#include <tmaxapi.h>
```

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

- 파라미터

파라미터	설명
argc	CASOPT에 설정된 옵션들의 갯수이다.
argv	CASOPT에 설정된 옵션들의 값이다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다.

- 관련함수

  tmax_crypt_plugin_init()

3.6.7. tmax_fini_auth

tmax_init_auth 함수를 통해 생성된 해당 클라이언트의 보안 토큰 정보를 삭제하는 함수이다.

- 프로토타입

```
# include  <atmi.h>
# include  <tmaxapi.h>
int tmax_fini_auth(TMAX_SEC_T *own_atoken)
```

- 파라미터

파라미터	설명
own_atoken	tmax_init_auth 함수를 통해 생성된 보안 토큰이다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다.

- 관련 함수

  tmax_fini_auth(), tmax_req_auth()

3.6.8. tmax_fini_crypt

tmax_init_crypt 함수를 통해 생성된 own_ctoken을 정리해주는 함수다. 클라이언트나 CAS에서 암호화 과정이 실패하거나 정상 종료됐을 때 호출한다.

- 프로토타입

```
# include  <tmaxapi.h>
int tmax_fini_crypt(TMAX_SEC_T *own_ctoken)
```

- 파라미터

파라미터	설명
own_atoken	tmax_init_crypt 함수를 통해 생성된 보안 토큰이다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다.

- 관련 함수

  tmax_init_crypt(), tmax_accept_crypt(), tmax_wrap(), tmax_unwrap()

3.6.9. tmax_init_auth

클라이언트가 tpstart() 내부에서 호출하는 함수다. 인증 및 인가를 위한 보안 토큰을 생성한다.

- 프로토타입

```
# include  <atmi.h>
# include  <tmaxapi.h>
int tmax_init_auth(const TPSTART_T *info, TMAX_SEC_T *atoken_to_peer, TMAX_SEC_T *own_atoken)
```

- 파라미터

파라미터	설명
info	tpstart()서의 TPSTART_T *tpinfo 인자 그대로 받는다.
atoken_to_peer	클라이언트가 CAS에게 인증 요청를 위해 건네줄 정보가 있다면 이 곳에 할당한다.

파라미터	설명
own_atoken	info를 바탕으로 클라이언트가 유지하고 있어야 될 보안 토큰을 생성한다. 여기서 생성된 보안 정보는 인증 뿐만 아니라 해당 클라이언트의 인가 검사를 위해서도 쓰인다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다.

- 관련 함수

  tpstart(), tmax_fini_auth(), tmax_req_auth()

3.6.10. tmax_init_crypt

암호화 작업을 위한 초기화 과정을 수행하는 함수다. 이 함수를 통해 클라이언트와 CAS 프로세스간의 암호화 통신할때 사용되는 토큰을 생성한다. 이 함수는 실질적으로 **tpstart()**의 내부에서 클라이언트와 CAS 양단에서 모두 호출된다.

- 프로토타입

```
#include <tmaxapi.h>
int tmax_init_crypt(const TMAX_SEC_T *ctoken_from_peer, TMAX_SEC_T *ctoken_to_peer, TMAX_SEC_T
*own_ctoken)
```

- 파라미터

파라미터	설명
ctoken_from_peer	클라이언트는 ctoken_from_peer를 사용하지 않는다. CAS는 클라이언트가 전송한 토큰을 ctoken_from_peer에서 받는다.
ctoken_to_peer	클라이언트에서 CAS에 전송할 토큰이 있다면 ctoken_to_peer에 할당한다. CAS는 클라이언트에게 다시 건네줄 토큰이 있다면 ctoken_to_peer에 할당한다. 이 ctoken_to_peer는 클라이언트의 tmax_accept_crypt 함수의 ctoken_from_peer 파라미터에서 받는다.
own_ctoken	클라이언트와 CAS 모두 내부에서 유지하고 있어야 할 암호화 관련 토큰을 own_ctoken에 할당한다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다.

- 관련 함수

  tmax_fini_crypt(), tmax_accept_crypt(), tmax_wrap(), tmax_unwrap()

3.6.11. tmax_req_auth

클라이언트의 tpcall 호출 내부에서 요청하는 서비스에 대한 인가를 위해 호출되는 함수이다.

- 프로토타입

```
#include <tmaxapi.h>
int tmax_req_auth(const char *svc_name, TMAX_SEC_T *atoken_to_peer, TMAX_SEC_T *own_atoken)
```

- 파라미터

파라미터	설명
svc_name	tpcall()할 때 호출한 서비스 이름이다.
atoken_to_peer	클라이언트가 CAS에게 인가 요청를 위해 건네줄 정보가 있다면 이 곳에 할당한다.
own_atoken	tmax_init_auth 함수를 통해 생성된 토큰이다. 이 토큰 정보는 다시 클라이언트에서 유지하므로 필요한 정보가 있으면 재할당한다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다.

- 관련 함수

  tpcall(), tpacall(), tmax_init_auth(), tmax_fini_auth()

3.6.12. tmax_tencrypt

Tmax 암/복호화 API를 사용하는 경우 tencerrno 변수는 전역변수로 thread-safe하지 않는 것에 주의한다.

- 프로토타입

```
#include <usrinc/tencrypt.h>
```

```
int tmax_tencrypt(char *src, int srclen, char **dest, int *destlen, int mode)
```

- 파라미터

파라미터	설명
src	암호화 또는 복호화하려는 경우 입력 데이터의 주소값을 지정한다.
srclen	입력 데이터의 길이를 지정한다. 0을 지정하면 내부적으로 strlen()을 호출하여 사용한다.
dest	암호화 또는 복호화가 수행된 후 출력 데이터의 버퍼 주소를 넘겨받을 주소값을 지정한다. 내부에서 버퍼를 메모리 할당하여 리턴하며, 함수 호출 이후 해당 주소에 대해서 free()를 호출해줘야 한다.
destlen	암호화 또는 복호화가 수행된 후 출력 데이터의 길이이다.
mode	src를 암호화하려는 경우 TENC_ENCRYPT를 src를 복호화하려는 경우 TENC_DECRYPT를 지정한다. TENC_ENCRYPT일 경우 src의 평문 최대 길이는 255자로 제한한다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tencerrno에 에러 코드가 설정된다. 설정가능한 값은 tperrno와 동일하다.

- 오류

tmax_tencrypt()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEPROTO]	암호화를 수행하는 경우 비밀 키 파일을 로드하지 못했다.
[TPEINVAL]	암호화를 수행하는 경우 입력 데이터가 올바르지 않거나 NULL이다.
[TPEOS]	메모리 할당이 실패했다.
[TPELIMIT]	암호화를 수행하는 경우 생성된 암호문이 허용 가능한 크기를 초과했다.

- 예제

```
#include <stdio.h>
#include <string.h>
#include <usrinc/tencrypt.h>
int main(int argc, char **argv)
{
    int n, len;
    char *cipher, *plain;
    if (argc < 2)
        return 1;
```

```
    cipher = plain = NULL;
    n = tmax_tencrypt(argv[1], strlen(argv[1]), &plain, &len, TENC_ENCRYPT);
    if (n < 0)
        printf("ENC failed. error = %d\n", tencerrno);
    else
        printf("ENC [%s] ==> [%s][%d]\n", argv[1], plain, len);

    n = tmax_tencrypt(plain, len, &cipher, &len, TENC_DECRYPT);
    if (n < 0)
        printf("DEC failed. error = %d\n", tencerrno);
    else
        printf("DEC [%s] ==> [%s][%d]\n", plain, cipher, len);

    free(cipher);
    free(plain);
    return 0;
}
```

- 빌드

```
cc test.c -o test -I$TMAXDIR -L $TMAXDIR/lib64 -ltencrypt
```

3.6.13. tmax_unwrap

클라이언트와 Tmax 간의 암호화된 메시지를 복호화하는 함수다.

- 프로토타입

```
#include <tmaxapi.h>
int tmax_unwrap(const TMAX_SEC_T *own_ctoken, const TMAX_SEC_T *cmsg, TMAX_SEC_T *pmsg)
```

- 파라미터

파라미터	설명
own_ctoken	클라이언트와 CAS 모두 tmax_init_crypt와 tmax_accept_crypt를 통해 생성된 own_ctoken이다.
cmsg	암호화된 전문이다.
pmsg	복호화 과정이 끝난 평문이 여기에 할당된다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다.

- 관련 함수

  tmax_init_crypt(), tmax_fini_crypt(), tmax_accept_crypt(), tmax_unwrap()

3.6.14. tmax_wrap

클라이언트 와 Tmax간의 메시지를 암호화하는 함수다.

- 프로토타입

```
#include <tmaxapi.h>
int tmax_wrap(const TMAX_SEC_T *own_ctoken, const TMAX_SEC_T *pmsg, TMAX_SEC_T *cmsg)
```

- 파라미터

파라미터	설명
own_ctoken	클라이언트와 CAS 모두 tmax_init_crypt 와 tmax_accept_crypt를 통해 생성된 own_ctoken이다.
pmsg	아직 암호화되지 않은 전문이다.
cmsg	암호화 과정이 끝난 전문이 여기에 할당된다.

- 반환값

반환값	설명
0	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다.

- 관련 함수

  tmax_init_crypt(), tmax_fini_crypt(), tmax_accept_crypt(), tmax_unwrap()

3.7. Windows 관련 함수

3.7.1. WinTmaxAcall

Windows 시스템 환경에서 클라이언트를 사용하는 함수로 MultiThread 환경에서의 비동기 서비스의 송신을 요청한다. MultiThread 환경에서 **tpacall()**과 같은 기능하는 함수로 새로운 Thread를 만들고, Thread 내에서 **tpstart()** → **tpacall()** → **tpgetrply()**를 수행한다. tpgetrply()를 호출한 후에는 SendMessage(wHandle, msgType, (UINT) &msg, tperrno)를 호출하게 된다.

- 프로토타입

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

- 파라미터

파라미터	설명
sinfo	Tmax 시스템에 클라이언트의 정보를 넘길 필요가 있을 경우 사용하는 구조체로 tpstart()의 파라미터와 동일하다.
wHandle	메시지를 받을 Windows의 핸들러를 지정한다.
msgtype	도착 메시지를 지정한다. 일반적으로 WM_USER를 개발자 임의로 define하여 사용한다. svc Tmax 환경 파일에 등록된 서비스명을 지정한다.
svc	송신을 요청할 서비스를 지정한다.
sndbuf	서비스를 호출할 때 전달되는 데이터로 NULL이 아닌 경우는 반드시 tpalloc()으로 할당된 버퍼를 사용해야 한다.
len	보내는 데이터의 길이를 지정한다. CARRAY, X_OCTET, 구조체 배열 타입일 경우에는 반드시 설정해야 한다.

파라미터	설명
flags	tpacall()의 flags를 그대로 사용한다. flags로 사용 가능한 값은 다음과 같다. • TPBLOCK flags 없이 tpacall()이 사용되었다면 svc에 호출된 서비스가 없거나 잘못된 결과가 반환되었어도 정상적인 결과가 반환된다. tpgetrply()를 호출할 때 에러가 반환된다. TPBLOCK flags를 이용해 tpacall()을 호출할 경우 서비스 상태의 정상 여부를 확인할 수 있다. • TPNOTRAN 트랜잭션 모드에서 svc가 트랜잭션을 지원하지 않는 서비스라면 tpacall()이 트랜잭션 모드에서 호출되는 경우 flags는 반드시 TPNOTRAN으로 설정해야 한다. tpacall() 호출자가 트랜잭션 모드 상태에서 TPNOTRAN을 설정하여 svc 서비스를 요청했다면 svc 서비스는 트랜잭션 모드에서 제외되어 수행된다. 트랜잭션 모드 내에서의 tpacall()을 호출할 때 TPNOTRAN로 설정되었어도 여전히 트랜잭션 타임아웃(timeout)에 영향을 받는다. TPNOTRAN으로 호출된 서비스가 실패했을 경우 호출자의 트랜잭션에는 영향을 미치지 않는다. • TPNOREPLY tpacall()로 송신한 서비스 요청은 응답을 기다리지 않고 즉시 반환한다. 결과는 나중에 tpacall()에서 반환한 구별자를 이용하여 tpgetrply()로 결과를 수신한다. flags가 TPNOREPLY로 설정하면 서비스에 대한 응답을 받지 않는겠다고 설정된다. TPNOREPLY로 설정하면 tpacall()은 서비스가 정상적으로 호출되면 0을 반환한다. 함수 호출자가 트랜잭션 모드에 있을 경우에는 반드시 TPNOTRAN flags와 함께 설정해야만 TPNOREPLY flags를 사용할 수 있다. TPNOREPLY flags인 경우 서비스 상태의 정상 여부를 체크하기 위해서는 TPBLOCK flags를 함께 설정해야 한다. TPBLOCK flags를 설정하지 않으면 서비스가 NRDY인 경우에도 에러를 반환하지 않는다. • TPNOBLOCK 내부 버퍼가 송신할 메시지들로 가득 찬 경우와 같은 블로킹(blocking) 상황을 만나면 서비스 요청은 실패하도록 설정하는 flags이다. TPNOBLOCK flags 설정 없이 tpacall()이 호출된 경우 블로킹(blocking) 상황이 발생하면 함수 호출자는 블로킹(blocking) 상황이 풀리거나 타임아웃(트랜잭션 타임아웃 또는 블록 타임아웃)이 발생할 때까지 대기한다. • TPNOTIME 함수 호출자가 블록 타임아웃을 무시하고 응답이 수신될 때까지 무한정 대기하도록 설정하는 flags이다. 트랜잭션 타임아웃 내에서 tpacall()을 한 경우에는 여전히 트랜잭션 타임아웃이 적용된다. • TPSIGRSTRT 시그널(signal) 인터럽트를 수용하는 경우 사용하는 flags로 시스템 함수 호출이 방해될 때 함수 호출이 재실행된다. TPSIGRSTRT가 설정되지 않은 상태에서 시그널 인터럽트가 발생한 경우 함수는 실패하고 tperrno에 TPGOTSIG가 설정된다.

- 반환값

반환값	설명
1	함수 호출이 성공한 경우이다. 구별자(descriptor)를 반환하고 구별자는 송신된 서비스 요청에 대한 응답을 수신하는 데 사용된다.
-1	함수 호출이 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  WinTmaxAcall()이 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 svc가 NULL이거나 데이터가 tpalloc()으로 할당되지 않은 버퍼를 가리키거나, 또는 flags가 유효하지 않은 경우에 발생한다.
[TPENOENT]	svc라는 서비스가 존재하지 않아서 서비스를 요청할 수 없다.
[TPEITYPE]	data의 유형 및 하위 유형이 svc가 지원하지 않는 유형이다. 구조체인 경우 사용된 구조체가 SDLFILE 파일에 선언되어 있지 않다.
[TPELIMIT]	처리되지 않은 비동기성 서비스 요청 수가 최대 한계에 도달했기 때문에 호출자의 서비스 요청이 송신되지 않았다.
[TPETRAN]	트랜잭션 서비스를 호출할 때 데이터베이스에 문제가 발생하여 xa_start가 실패하였다.
[TPETIME]	타임아웃이 발생하였다. 함수 호출자가 트랜잭션 모드에 있다면 트랜잭션 타임아웃이 발생한 것이며 트랜잭션은 rollback된다. 그렇지 않다면 TPNOTIME이나 TPNOBLOCK이 모두 설정되지 않은 상황에서 블록 타임아웃이 발생한 것이다. 트랜잭션 타임아웃이 발생하는 경우 트랜잭션이 rollback될 때까지 새로운 서비스 요청을 송신한다거나 아직 수신되지 않은 응답을 기다리는 일은 모두 [TPETIME] 에러로 실패한다.
[TPEBLOCK]	TPNOBLOCK이 설정된 상태에서 블로킹 상황이 발생하였다.
[TPGOTSIG]	TPSIGRSTRT가 설정되지 않은 상태에서 시그널이 수신되었다.
[TPEPROTO]	트랜잭션 모드에서의 TPNOREPLY 서비스를 호출하는 경우 TPNOTRAN flags를 설정하지 않는 경우 등 부적절한 상황에서 발생한다.
[TPESYSTEM]	Tmax 시스템에 에러가 발생하였다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

```
...
#include <usrinc/tmaxapi.h>
#define WM_WINTMAX_RECV WM_USER + 1
...

BEGIN_MESSAGE_MAP(CWinTmaxAcall2TestDlg, CDialog)
...
ON_MESSAGE(WM_WINTMAX_RECV, OnWinTmaxAcall)
...
END_MESSAGE_MAP()
```

```
BOOL CWinTmaxAcall2TestDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    ...
    ret = tmaxreadenv("tmax.env", "TMAX");
    if (ret == -1){
        AfxMessageBox("tmaxreadenv fail...");
        return FALSE;
    }
    return TRUE;  // return TRUE  unless you set the focus to a control
}

void CWinTmaxAcall2TestDlg::OnOK()
{
    ...
    GetDlgItemText(IDC_EDIT1, m_Input);
    lstrcpy((LPTSTR)buf, (LPCTSTR)m_Input.operator const char *());
    ...

    buf = tpalloc("STRING", NULL, 0);
    if (buf == NULL){
        AfxMessageBox("buf alloc fail...");
        return FALSE;
    }

    ret = WinTmaxAcall((TPSTART_T *)NULL, m_hWnd, WM_WINTMAX_RECV,
           "TOUPPER", buf, 0, TPNOFLAGS);
    if (ret == -1){
        error processing
    }
}

LRESULT CWinTmaxAcall2TestDlg::OnWinTmaxAcall(WPARAM wp, LPARAM lp)
{
    char msg[100];
    memset(msg, 0x00, 100);
    TPSVCINFO *get = (TPSVCINFO *)wp;
    if (lp < 0){
        error processing
    }
    ...
    SetDlgItemText(IDC_EDIT2, get->data);
    return 0;
}
```

- 관련함수

  tpacall(), WinTmaxAcall2()

3.7.2. WinTmaxAcall2

Windows 시스템 환경에서 클라이언트사용하는 함수로 Multi Thread 환경에서의 비동기 서비스의 송신을 요청한다. MultiThread 환경에서 **tpacall()**과 같은 기능하는 함수로 새로운 Thread를 생성하고, Thread 내에서 **tpstart()** → **tpacall()** → **tpgetrply()**를 수행한다. tpgetrply()를 호출한 후에는 지정된 Callback 함수로 수신된

데이터를 전달한다.

- 프로토타입

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

- 파라미터

파라미터	설명
sinfo	Tmax 시스템에 클라이언트의 정보를 전달할 필요가 있을 경우 사용하는 구조체로, tpstart()의 파라미터와 동일하다.
fn	서비스 요청에 대한 응답을 받을 콜백 함수를 지정한다.
svc	Tmax 환경 파일에 등록된 서비스명을 지정한다.
sndbuf	서비스를 호출할 때 전달되는 데이터로 NULL이 아닌 경우는 반드시 tpalloc() 으로 할당된 버퍼를 사용해야 한다.
len	보내는 데이터의 길이를 지정한다. CARRAY, X_OCTET, 구조체 배열 타입일 경우에는 반드시 설정해야 한다.

파라미터	설명
flags	tpacall()의 flags를 그대로 사용한다. flags로 사용 가능한 값은 다음과 같다. • TPBLOCK flags 없이 tpacall() 이 사용되었다면 svc에 호출된 서비스가 없거나 잘못된 결과가 반환되었어도 정상적인 결과가 반환된다. tpgetrply()를 호출할 때 에러가 반환된다. TPBLOCK flags를 이용해 tpacall()을 호출할 경우 서비스 상태의 정상 여부를 확인할 수 있다. • TPNOTRAN 트랜잭션 모드에서 svc가 트랜잭션을 지원하지 않는 서비스라면 tpacall()이 트랜잭션 모드에서 호출되는 경우 flags는 반드시 TPNOTRAN으로 설정해야 한다. tpacall() 호출자가 트랜잭션 모드 상태에서 TPNOTRAN을 설정하여 svc 서비스를 요청하였다면, svc 서비스는 트랜잭션 모드에서 제외되어 수행된다. 트랜잭션 모드 내에서의 tpacall()함수를 호출할 때 TPNOTRAN로 설정되었어도 여전히 트랜잭션 타임아웃(timeout)에 영향을 받는다. TPNOTRAN으로 호출된 서비스가 실패했을 경우 호출자의 트랜잭션에는 영향을 미치지 않는다. • TPNOREPLY tpacall()로 송신한 서비스 요청은 응답을 기다리지 않고 즉시 반환하고 결과는 나중에 tpacall()에서 반환한 구별자를 이용하여 tpgetrply()로 결과를 수신한다. flags를 TPNOREPLY로 설정하면 서비스에 대한 응답을 받지 않는겠다고 설정된다. TPNOREPLY로 설정하면 tpacall()은 서비스가 정상적으로 호출되면 0을 반환한다. 함수 호출자가 트랜잭션 모드에 있을 경우에는 반드시 TPNOTRAN flags와 함께 설정해야만 TPNOREPLY flags를 사용할 수 있다. TPNOREPLY flags인 경우 서비스 상태의 정상 여부를 체크하기 위해서는 TPBLOCK flags를 함께 설정해야 한다. TPBLOCK flags를 설정하지 않으면 서비스가 NRDY인 경우에도 에러를 반환하지 않는다. • TPNOBLOCK 내부 버퍼가 송신할 메시지들로 가득 찬 경우와 같은 블로킹(blocking) 상황을 만나면 서비스 요청은 실패하도록 설정하는 flags이다. TPNOBLOCK flags 설정 없이 tpacall()이 호출된 경우 블로킹 상황이 발생하면 함수 호출자는 블로킹 상황이 풀리거나 타임아웃(트랜잭션 타임아웃 또는 블록 타임아웃)이 발생할 때까지 대기한다. • TPNOTIME 함수 호출자가 블록 타임아웃을 무시하고 응답이 수신될 때까지 무한정 대기하도록 설정하는 flags이다. 트랜잭션 타임아웃 내에서 tpacall()을 한 경우에는 여전히 트랜잭션 타임아웃이 적용된다. • TPSIGRSTRT 시그널(signal) 인터럽트를 수용하는 경우 사용하는 flags로 시스템 함수 호출이 방해될 때 함수 호출이 재실행된다. TPSIGRSTRT가 설정되지 않은 상태에서 시그널 인터럽트가 발생했다면, 함수는 실패하고 tperrno에 TPGOTSIG가 설정된다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.

반환값	설명
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

WinTmaxAcall2()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어 svc가 NULL이거나 data가 tpalloc()으로 할당되지 않은 버퍼를 가리키거나, 또는 flags가 유효하지 않은 경우에 발생한다.
[TPENOENT]	svc라는 서비스가 존재하지 않아서 서비스를 요청할 수 없다.
[TPEITYPE]	data의 유형 및 하위 유형이 svc가 지원하지 않는 유형이다. 구조체인 경우 사용된 구조체가 SDLFILE 파일에 선언되어 있지 않다.
[TPELIMIT]	처리되지 않은 비동기성 서비스 요청 수가 최대 한계에 도달했기 때문에, 호출자의 서비스 요청이 송신되지 않았다.
[TPETRAN]	svc는 트랜잭션을 지원하지 않는 서비스이고, TPNOTRAN이 설정되지 않았다.
[TPETIME]	타임아웃이 발생한 경우로, 함수 호출자가 트랜잭션 모드에 있다면 트랜잭션 타임아웃이 발생한 것이며 트랜잭션은 rollback된다. 그렇지 않다면 TPNOTIME이나 TPNOBLOCK이 모두 설정되지 않은 상황에서 블록 타임아웃이 발생한 것이다. 트랜잭션 타임아웃이 발생한 경우 트랜잭션이 rollback될 때까지 새로운 서비스 요청을 송신한다거나 아직 수신되지 않은 응답을 기다리는 일은 모두 [TPETIME] 에러로 실패한다.
[TPEBLOCK]	TPNOBLOCK이 설정된 상태에서 블로킹 상황이 발생하였다.
[TPGOTSIG]	TPSIGRSTRT가 설정되지 않은 상태에서 시그널이 수신되었다.
[TPEPROTO]	트랜잭션 모드에서의 TPNOREPLY 서비스 호출할 때 TPNOTRAN flags를 설정하지 않는 경우 등 부적절한 상황에서 발생한다.
[TPESYSTEM]	Tmax 시스템에 에러가 발생하였다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

```
#include <usrinc/tmaxapi.h>
#define WM_WINTMAX_RECV WM_USER + 1
int mycallfn(unsigned int, long);
...

BEGIN_MESSAGE_MAP(CWinTmaxAcall2TestDlg, CDialog)
...
END_MESSAGE_MAP()

BOOL CWinTmaxAcall2TestDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    ...
    ret = tmaxreadenv(“tmax.env”, “TMAX”);
```

```
    if (ret == -1){
        AfxMessageBox("tmaxreadenv fail...");
        return FALSE;
    }
    return TRUE;  // return TRUE  unless you set the focus to a control
}

void CWinTmaxAcall2TestDlg::OnOK()
{
    ...
    GetDlgItemText(IDC_EDIT1, m_Input);
    lstrcpy((LPTSTR)buf, (LPCTSTR)m_Input.operator const char *());
    ...
    buf = tpalloc("STRING", NULL, 0);
    if (buf == NULL){
        AfxMessageBox("buf alloc fail...");
        return FALSE;
    }
    ret = WinTmaxAcall2((TPSTART_T *)NULL, (WinTmaxCallback)mycallfn,
                        "TOUPPER", (char *)buf, 0, TPNOFLAGS);
    if (ret == -1){
        error processing
    }
}

int mycallfn(unsigned int msg, long retval)
{
    TPSVCINFO *svcinfo;
    char infomsg[30];
    memset(infomsg, 0x00, sizeof(infomsg));
    svcinfo = (TPSVCINFO *)msg;
    strncpy(infomsg, svcinfo->data, svcinfo->len);
    if (retval != 0){
        strcpy(infomsg,tpstrerror(retval));
        AfxMessageBox(infomsg);
        return -1;
    } else {
        strncpy(infomsg, svcinfo->data, sizeof(infomsg) - 1);
        AfxMessageBox(infomsg);
        return 1;
    }
}
```

- 관련함수

  tpacall(),WinTmaxAcall()

3.7.3. WinTmaxEnd

Windows 시스템 환경에서 클라이언트에서 사용되는 함수로 Tmax 시스템과 연결을 종료한다. 기능상으로는 **tpend()**와 같다. 멀티 Thread 환경에서 사용하는 경우는 Thread별로 **WinTmaxStart()**를 사용하여 연결을 맺게 됨으로써 WinTmaxEnd()도 Thread별로 사용한다.

- 프로토타입

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

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  WinTmaxEnd()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPETIME]	임계 영역으로 들어갈 수 없는 경우에 발생한다. 시스템 내부 오류로서 시스템 상태를 확인한다.
[TPEPROTO]	tpend()가 부적절한 상황에서 호출되었다. 예를 들어, 호출자가 서버이다. 또는 연결이 해제된 후에 다시 호출된 경우에 발생한다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 관련함수

  tpend(), WinTmaxStart()

3.7.4. WinTmaxSend

Windows 시스템 환경에서 클라이언트에서 사용되는 함수로 데이터를 송신한다. 멀티 Windows 환경에서 사용되며 서비스를 요청하는 함수이다.

- 프로토타입

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

- 파라미터

파라미터	설명
recvContext	Tmax 시스템으로부터 응답을 수신하는 경우에 수신 대상 Windows를 지정한다. WinTmaxSetContext()의 반환값이다.
svc	서비스명이다.
data	tpalloc()으로 할당된 버퍼로 서비스 요청할 때에 전달할 데이터가 저장된다.

파라미터	설명
len	데이터의 길이 값으로 CARRAY 버퍼 또는 다중 구조체 버퍼를 사용하는 경우에는 정확한 길이 값을 명시해야 한다.
flags	WinTmaxSend()에서 사용할 수 있는 flags 값은 표 이후에 설명한다.

WinTmaxSend()는 tpacall() 함수와 비슷하게 동작하며, 서비스를 요청하고 응답이 도착할 때까지 기다리지 않고 즉시 반환한다. 이후 응답은 WinTmaxStart()를 호출할 때 생성되었던 Thread에서 처리하여 WinTmaxSetContext()에서 설정된 윈도우로 응답 결과를 알려준다. 따라서 응답을 받기 위한 별도의 API는 제공되지 않는다.

다음은 비어있는 slot 하나를 찾아서 (hwd, 0x300)을 저장하고 index를 반환하는 예이다. WinTmaxSetContext()의 반환값 rc를 WinTmaxSend()의 recvContext 파라미터로 사용하며, WinTmaxSend(rc, svc, data, len, 0) 호출 이후에 응답이 오면 hwd windows에 0 x 300번 메시지 및 응답 결과를 전송한다.

```
rc = WinTmaxSetContext(hwd, 0x300, -1);
int nSendResult = WindTmaxSend(rc, (LPSTR)(LPCSTR)strService, (Char*)tpbuf, nLen, TPNOFLAGS);
/*
  이후 내부 Thread에서 응답 메시지를 처리하여 해당 윈도우로 다음과 같이 메시지를 전달한다.
  SendMessage(hwd, 0x300, (UINT) &msg, callRet);
*/
```

메시지를 받은 윈도우에서는 WPARAM에 응답 데이터(msg), LPARAM에 응답 결과(callRet)를 전달받는다. callRet가 0일 경우 정상적인 응답이고, -1일 경우 오류가 발생했음을 알려준다. 이때 tperrno 값을 통해 오류의 원인을 파악할 수 있다.

WinTmaxSend()는 구조상 트랜잭션을 지원하지 않으며, Tmax 시스템의 환경설정 BLOCKTIME 항목 또는 tpsettimeout()의 영향을 받지 않는다. 다만 WinTmaxSend() 호출할 때 TPBLOCK flag를 주었을 경우에만 해당 flag의 동작 방식에 대해서 BLOCKTIME이 적용된다.

WinTmaxSend()에서 사용할 수 있는 flags 값은 다음과 같다.

flags 값	설명
TPBLOCK	TPBLOCK flags 없이 WinTmaxSend() 함수가 사용되었다면 svc가 등록되어 있지 않거나 서비스 수행에 실패해도 정상적인 결과가 반환된다. 에러는 응답을 받는 시점에 확인할 수 있다. TPBLOCK flags를 이용하면 서비스 호출 시점에 정상 여부를 확인할 수 있다. 즉, WinTmaxSend()는 BLOCKTIME 이내에 요청한 서비스가 정상적으로 수행가능한지 여부를 확인하고 리턴한다. 만약 요청한 서비스가 수행 불가능할 경우 이에 대한 오류를 tperrno에 저장하고 -1을 리턴한다. 만약 BLOCKTIME 이내에 서비스 수행 여부에 대한 확인이 불가능할 경우 TPETIME 오류를 반환한다. 이 경우 클라이언트에서는 요청한 서비스가 수행되었는지 여부를 알 수 없으므로 에러 루틴을 작성할 때 주의해야 한다. 만약 재요청 처리를 해야 할 경우 기존의 요청이 처리되었는지 여부를 확인하는 방법으로 주의 깊게 처리해야 한다.

flags 값	설명
TPNOREPLY	WinTmaxSend()로 송신한 서비스 요청은 응답을 기다리지 않고 즉시 반환한다. 결과는 작업 Thread에 의해서 수신되어 등록된 Windows에 전달된다. 그러나 TPNOREPLY flags는 서비스에 대한 응답을 받지 않겠다고 설정하는 것이다.
TPNOTRAN	WinTmaxSend() 함수 호출자가 트랜잭션 모드 상태에서 flags를 설정하여 svc 서비스를 요청했다면 svc 서비스는 트랜잭션 모드에서 제외되어 수행된다. 트랜잭션 모드에서 svc가 트랜잭션을 지원하지 않는 서비스라면 WinTmaxSend() 함수가 트랜잭션 모드에서 호출되는 경우 flags는 반드시 TPNOTRAN으로 설정해야 한다. 트랜잭션 모드 내에서의 WinTmaxSend() 함수를 호출하는 경우 TPNOTRAN로 설정되었어도 여전히 트랜잭션 타임아웃(timeout)에 영향을 받는다. TPNOTRAN으로 호출된 서비스가 실패했을 경우 호출자의 트랜잭션에는 영향을 미치지 않는다.
TPNOBLOCK	TPNOBLOCK flags가 설정한 상태에서, 내부 버퍼가 송신할 메시지들로 가득 찬 경우와 같은 블로킹(blocking) 상황을 만나면 서비스 요청은 실패한다. TPNOBLOCK flags 설정 없이 WinTmaxSend()를 호출하는 경우 블로킹 상황이 발생하면 함수 호출자는 블로킹 상황이 풀리거나 타임아웃(트랜잭션 타임아웃 또는 블록 타임아웃)이 발생할 때까지 기다리게 된다.
TPNOTIME	함수 호출자가 블록 타임아웃을 무시하고 응답이 수신될 때까지 무한정 기다리겠다는 것을 의미한다. 그러나 트랜잭션 타임아웃 내에서 WinTmaxSend()을 한 경우에는 여전히 트랜잭션 타임아웃이 적용된다.
TPSIGRSTRT	시그널(signal) 인터럽트를 수용하고자 할 때 사용한다. 시스템 함수 호출이 방해될 때 시스템 함수 호출이 재실행된다. flags가 설정되지 않은 상태에서 시그널 인터럽트가 발생했다면 함수는 실패하고 tperrno에 TPGOTSIG가 설정된다.

- 반환값

반환값	설명
구별자(descriptor)	함수 호출에 성공한 경우이다. 구별자(descriptor)를 반환한다. 현재 이 구분자는 사용되지 않는다.
-1	함수 호출이 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  WinTmaxSend()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPENOENT]	svc라는 서비스가 존재하지 않아서 서비스를 요청할 수 없다.
[TPEITYPE]	data의 유형 및 하위 유형이 svc가 지원하지 않는 유형이다. 구조체인 경우 사용된 구조체가 SDLFILE 파일에 선언되어 있지 않다.
[TPELIMIT]	처리되지 않은 비동기성 서비스 요청 수가 최대 한계에 도달했기 때문에, 호출자의 서비스 요청이 송신되지 않았다.
[TPETIME]	타임아웃이 발생한 경우로 TPNOTIME이나 TPNOBLOCK이 모두 설정되지 않은 상황에서 블록 타임아웃이 발생한다.

에러 코드	설명
[TPEBLOCK]	TPNOBLOCK이 설정된 상태에서 블로킹 상황이 발생하였다.
[TPGOTSIG]	TPSIGRSTRT가 설정되지 않은 상태에서 시그널이 수신되었다.
[TPEPROTO]	트랜잭션 모드에서 TPNOREPLY 서비스가 호출될 경우 TPNOTRAN flags를 설정하지 않는 경우 등 부적절한 상황에서 발생한다.
[TPESYSTEM]	Tmax 시스템에 에러가 발생하였다.
[TPECLOSE]	네트워크 문제나 기타 여러 가지 원인으로 인해 Tmax 시스템과의 연결이 해제되었다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

Windows로 전달된 메시지의 LPARAM이 -1인 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEBADDESC]	응답 메시지를 수신했으나 올바른 구별자(descriptor)를 찾을 수 없는 경우에 발생한다.
[TPEOTYPE]	응답 메시지를 수신했으나 클라이언트와 서버 프로그램의 버퍼 타입이 일치하지 않는 경우에 발생한다.
[TPEPROTO]	WinTmaxStart(), WinTmaxEnd() 등을 부적절한 상황에서 호출할 경우에 발생한다.
[TPESYSTEM]	Tmax 시스템에 에러가 발생하였다.
[TPECLOSE]	네트워크 문제나 기타 여러 가지 원인으로 Tmax 시스템과의 연결이 해제되었다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

```
...
int CTMaxGwView::SendData2(char *data, long nLen)
{
    CString szTemp;
    m_send_length.Format(“%d”,nLen);
    UpdateData(FALSE);

    char *tpbuf = tpalloc(“STRING”, NULL, nLen);
    if (tpbuf == NULL) {
        szTemp.Format(“tpalloc Error [%s]”, tpstrerror(tperrno));
        LogDisplay2(2, (char *)(const char *)szTemp, szTemp.GetLength());
        return -1;
    }

   memcpy(tpbuf, data, nLen);

   CString strService;
   strService.Format(“TOUPPER_STRING”);

   int nSendResult=WinTmaxSend(2,(LPSTR)(LPCSTR)strService, (char*)tpbuf, nLen, 0);
   tpfree(tpbuf);
    ...
}
```

- 관련 함수

  WinTmaxStart(),WinTmaxEnd()

3.7.5. WinTmaxSetContext

Windows 시스템 환경에서 클라이언트에서 사용되는 함수로 Windows 핸들과 메시지 타입을 설정한다.

멀티 Windows 환경에서 사용되며 지정된 Windows 핸들과 메시지 타입을 라이브러리의 빈 slot에 저장하여 관리하도록 한다. 이렇게 관리되는 정보는 작업 Thread가 응답을 수신한 후 지정되어 있는 Windows에게 지정된 메시지 타입으로 데이터를 전송하게 된다. 다음은 데이터 전송 포맷이다.

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

응답을 받게되면 작업 Thread는 SendMessage(winhandle, msgType, (UINT) &msg, callRet)로 데이터를 전송한다. 이 경우 지정된 Windows에서는 WPARAM에는 msg가 대응되며 LPARAM에는 callRet가 대응된다. msg는 TPSVCINFO 구조체이며 callRet는 서비스 처리 결과값으로서 적절한 메시지가 수신된 경우에는 0이며 잘못된 메시지가 수신된 경우에는 –1이다.

예를 들어 정상적으로 tpreturn(TPSUCCESS, ...)으로 서비스가 처리되거나 또는 실패한 경우로서 tpreturn(TPFAIL, ...)으로 처리되거나 또는 비요청 메시지가 수신된 경우에는 적절한 메시지로 간주하고 callRet 값은 0이 된다. 하지만 동기형 결과값이나 대화형 메시지가 전달되는 경우 이는 멀티 Windows 환경에서 사용할 수 없는 형태이기 때문에 callRet 값은 –1이 된다. callRet 값이 -1일 경우 tperrno 값을 확인하여 오류의 원인을 파악할 수 있다.

- 프로토타입

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

- 파라미터

파라미터	설명
winhandle	수신 데이터를 처리할 Windows 핸들이다.
msgType	메시지 타입이다.

파라미터	설명
slot	지정된 Windows 핸들과 메시지 타입을 할당할 slot을 의미한다. 할당 가능한 slot의 최대 개수는 256이며 시스템 내부적으로 0과 1은 각각 기본 출력과 에러를 위한 것으로 설정된다. 데이터 수신 과정에서 에러가 발생한다든가 출력용 Windows가 지정되어 있지 않은 경우에는 기본 Windows가 사용된다. slot은 사용자에 의해서 재정의되어 사용될 수 있다. 비요청 메시지의 경우 사용자가 지정한 윈도우가 없기 때문에 0번 기본 출력 윈도우로 메시지가 전달된다. 다음은 slot에 설정할 수 있는 값이다. • –1 : 시스템 내부적으로 비어있는 slot을 자동으로 검색하여 지정된 Windows 핸들과 메시지 타입을 할당한다. • 0 이상의 값 : 주어진 index의 slot에 지정된 Windows와 메시지 타입을 할당한다.

- 반환값

반환값	설명
index	함수 호출에 성공한 경우이다. slot에 대한 index를 반환하고 index는 WinTmaxSend()에서 첫 번째 파라미터로 사용된다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류WinTmaxSetContext()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPESYSTEM]	Tmax 시스템에 에러가 발생하였다.
[TPEOS]	운영 시스템에 에러가 발생하였다.

- 예제

```
...
int CTMaxGwView::Connect()
{
    CString szTemp, Fname;
    WinTmaxEnd();
    int Ret = tmaxreadenv(TMAXINI, “TMAX117”);
    if(Ret<0)
    {
        szTemp.Format(“tmaxreadenv error”);
        LogDisplay2(2, (char *)(const char *)szTemp, szTemp.GetLength());
        return FALSE;
    }
    if (WinTmaxStart((TPSTART_T *)NULL) == -1) {
        szTemp.Format(“WinTmaxStart 에러 = [%s]”, tpstrerror(tperrno));
        LogDisplay2(2, (char *)(const char *)szTemp, szTemp.GetLength());
        return FALSE;
    }

    WinTmaxSetContext(m_hWnd, WM_TMAX_RECV_RDP, 0);
```

```
    WinTmaxSetContext(m_hWnd, WM_TMAX_RECV_ERR, 1);
    WinTmaxSetContext(m_hWnd, WM_TMAX_RECV, 2);

    return TRUE;
}
```

- 관련함수

  WinTmaxStart(),WinTmaxEnd(), WinTmaxSend()

3.7.6. WinTmaxStart

Windows 시스템 환경에서 클라이언트에서 사용되는 함수로 Multi Windows 환경에서 Tmax 시스템과 연결하는 데 사용된다. 기능상으로는 **tpstart()**와 동일하다. 다중 Thread를 사용하는 경우에는 각각의 Thread별로 WinTmaxStart()를 이용하여 별도의 연결을 맺고 서비스를 호출해야 한다.

- 프로토타입

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

- 파라미터

  TPSTART_T에 대해서는 tpstart를 참고한다.

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. tperrno에 에러 코드가 설정된다.

- 오류

  WinTmaxStart()가 정상적으로 수행되지 않을 경우 tperrno에 다음 값 중 하나가 설정된다.

에러 코드	설명
[TPEINVAL]	파라미터가 유효하지 않다. 예를 들어, tpinfo가 NULL이나 TPSTART_T 구조체에 대한 포인터가 아니다.
[TPEITYPE]	tpinfo가 TPSTART_T 구조체에 대한 포인터가 아니다.
[TPEPROTO]	WinTmaxStart()가 부적절한 곳에서 호출되었다. 예를 들어, 서버 프로그램에서 사용되었거나 이미 연결된 상황에서 재호출된 경우에 발생한다.
[TPESYSTEM]	Tmax 시스템 에러가 발생하였다. 자세한 정보는 로그 파일에 기록된다.

에러 코드	설명
[TPEOS]	운영 시스템에 에러가 발생했다. 또는 환경변수 설정이 잘못된 경우이다. 예를 들어 TMAX_HOST_ADDR나 TMAX_HOST_PORT가 잘못 설정되어 연결이 실패한 경우에 발생한다.

- 관련함수

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

3.8. 기타 함수

3.8.1. tlog_close

트랜잭션 로그를 분석하기 위한 함수 중 하나로 해당 로그 파일을 닫는 함수이다. Tmax 엔진과는 별개로 사용이 가능하며, **<usring/tlog.h>**와 **<libtlog.so>** 라이브러리만 있으면 사용이 가능하다.

- 프로토타입

```
#include <usring/tlog.h>
int tlog_close (tlog_file_t *log);
```

- 파라미터

파라미터	설명
log	tlog_open()를 호출할 때 해당 로그 파일에 대한 내부 정보가 채워졌으며, tlog_find()에도 사용되었던 구조체에 대한 포인터이다.

- 반환값

반환값	설명
TLOG_OK	성공적으로 로그 파일을 close한 경우이다.
음수	함수 호출 중 에러가 발생한 경우이다.

- 예제

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

int main(int argc, char *argv[])
{
   int n;
   tlog_entry_t entry;
   tlog_file_t log;
```

```
   printf("\n====== TXLOG =====\n");
   n = tlog_open(NULL, &log, TLOG_LOCAL);
   printf("n = %d\n", n);
   if (n < 0) {
      printf("tlog_open = %d, errno = %d\n", n, errno);
      exit(1);
   }
   ...
   tlog_close(&log);
}
```

- 관련 함수

  tlog_close(), tlog_find(), tlog_nodeno()

3.8.2. tlog_find

해당 트랜잭션 로그 파일에서 entry에 지정된 정보와 matching되는 entry를 찾는 함수로 함수의 결과는 파라미터로 전달된 entry에 저장되어 반환된다. Tmax 엔진과는 별개로 사용이 가능하며, **<usring/tlog.h>**와 **<libtlog.so>** 라이브러리만 있으면 사용이 가능하다.

- 프로토타입

```
#include <usring/tlog.h>
int tlog_find (tlog_file_t *log, tlog_entry_t *entry, int flags);
```

- 파라미터

파라미터	설명
log	tlog_open으로 생성한 tlog 정보 구조체이다.

파라미터	설명

entry	조회할 entry를 반환한다.
	
	tlog_entry_t는 해당 트랜잭션 로그 파일에 들어있는 내용을 보여 준다.
	
	`typedef struct {`
	`  int decision;`
	`  TXID xid;`
	`  time_t ltime;`
	`  /* following fields have meaning only for TLOG_REMOTE & TLOG_NONTMAX */`
	`  TXID remote_xid;`
	`  char gateway_name[GATEWAY_NAME_SIZE];`
	`} tlog_entry_t;`
	
	다음은 멤버에 대한 설명이다.
	
	• decision : 해당 트랜잭션에 대해 어떤 동작이 이루어졌나를 기록하고 있다.
	
	다음은 decision에 사용 가능한 값에 대한 정의이다.
	
	`/* invalid entry */`
	`#define TXDEC_INVALID  -1`
	`/* commit */`
	`#define TXDEC_COMMIT  0`
	`/* rollback */`
	`#define TXDEC_ROLLBACK  1`
	`/* rollback due to svr/cli down or failure during the prepare phase */`
	`#define TXDEC_ABNORMAL_ROLLBACK 2`
	`/* tx initiated from a remote domain */`
	`#define TXDEC_REMOTE  3`
	`/* prepare phase */`
	`#define TXDEC_PREPARE  4`
	
	• xid : 로그를 남긴 XID로 게이트웨이에서 남긴 트랜잭션의 로그에는 remote_xid와 gateway_name의 2가지 필드가 추가된다.
	• ltime : 로그를 남긴 시간이다.

파라미터	설명

flags	다음은 flags에 사용할 수 있는 6가지 매크로의 bit-wise OR의 정의이다.
	
	`/* get next log entry */`
	`#define TLOG_NEXT 0x0001`
	`/* find an entry logged later than or equal to the ltime field */`
	`#define TLOG_TIME 0x0002`
	`/* find an entry with matching xid */`
	`#define TLOG_XID 0x0004`
	`/* find an entry with matching remote_xid */`
	`#define TLOG_REMOTE_XID 0x0008`
	`/* find an entry from the matching gateway_name field */`
	`#define TLOG_GATEWAY 0x0010`
	`/* find an entry with same decision */`
	`#define TLOG_DECISION 0x0020`
	
	• TLOG_NEXT : bit가 ON된 경우 로그 파일의 현재 위치부터 찾으며, 그렇지 않으면 파일의 처음부터 찾는다.
	• TLOG_TIME : bit가 ON된 경우 로그 기록 시간이 entry→ltime 이후인 entry들만 찾는다.
	• TLOG_XID : entry→xid가 matching되는 entry만을 찾으며, XID 비교하는 경우 branch qualifier 부분은 무시되고 전역 트랜잭션 ID 부분만 참고한다.
	• TLOG_REMOTE_XID : entry-→remote_xid가 같은 entry를 찾는다.
	• TLOG_GATEWAY : entry→gateway_name이 같은 entry를 찾는다.
	• TLOG_DECISION : entry→decision이 같은 entry를 찾는다.

- 반환값

반환값	설명
TLOG_OK	함수가 성공적으로 수행한 경우이다.
TLOG_INVAL	파라미터가 유효하지 않은 경우이다.
TLOG_NOTFOUND	파일의 끝까지 찾았으므로 더 이상 찾을 수 없는 경우이다.
TLOG_ESYSTEM	Tmax 시스템에 에러가 발생하였다. 자세한 정보는 errno를 참고한다.

- 예제

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

int main(int argc, char *argv[])
{
   int n;
   tlog_entry_t entry;
   tlog_file_t log;

   printf("\n====== TXLOG =====\n");
```

```
    n = tlog_open(NULL, &log, TLOG_LOCAL);
    printf("n = %d\n", n);
    if (n < 0) {
       printf("tlog_open = %d, errno = %d\n", n, errno);
       exit(1);
    }
    printf("tlog_open success\n");
    while(1) {
    n = tlog_find(&log, &entry, TLOG_NEXT);
    if (n == TLOG_NOTFOUND)
    {
       printf("Not found any more\n");
       break;
     }

    else if (n < 0) {
       printf("tlog_find = %d, errno = %d\n", n,errno);
       tlog_close(&log);
       exit(1);
    }
    tlog_close(&log);
}
```

- 관련 함수

  tlog_open(), tlog_close(), tlog_nodeno()

3.8.3. tlog_nodeno

XID를 이용하여 트랜잭션이 시작된 노드 번호를 찾는 함수이다. 멀티 노드 구성의 경우 트랜잭션 로그는 트랜잭션이 시작된 노드에 남겨지기 때문에 XID로부터 해당 노드를 알아낼 필요가 있다. tlog_nodeno()는 Tmax 엔진과는 별개로 사용이 가능하며, **<usring/tlog.h>**와 **<libtlog.so>** 라이브러리만 있으면 사용이 가능하다.

- 프로토타입

```
#include <usring/tlog.h>
int tlog_nodeno (TXID *xid);
```

- 파라미터

파라미터	설명
xid	노드 번호를 알아낼 XID이다.

- 반환값

반환값	설명
노드번호	함수 호출에 성공한 경우이다. 트랜잭션을 시작한 노드의 번호를 반환한다.
음수	함수 호출에 실패한 경우이다.

- 예제

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

int main(int argc, char *argv[])
{
   int n, nodeno;
   tlog_entry_t entry;
   tlog_file_t log;

   printf("\n====== TXLOG =====\n");
   n = tlog_open(NULL, &log, TLOG_LOCAL);
   printf("n = %d\n", n);
   if (n < 0) {
      printf("tlog_open = %d, errno = %d\n", n, errno);
      exit(1);
   }
   printf("tlog_open success\n");
   while(1) {
      n = tlog_find(&log, &entry, TLOG_NEXT);
      ...
   }

   nodeno = tlog_nodeno(&(entry.xid));
   printf("nodeno of txlog = %d\n", nodeno);
   tlog_close(&log);
}
```

- 관련 함수log_open(), tlog_close(), tlog_find()

3.8.4. tlog_open

트랜잭션 로그를 분석하기 위한 함수로 해당 로그 파일을 open하는 함수이다. Tmax 엔진과는 별개로 사용이 가능하며, **<usring/tlog.h>**와 **<libtlog.so>** 라이브러리만 있으면 사용이 가능하다.

- 프로토타입

```
#include <usring/tlog.h>
int tlog_open (char *name, tlog_file_t *log, int flags);
```

- 파라미터

파라미터	설명
name	로그 파일의 이름이다.
log	해당 로그 파일에 대한 내부 정보로 tlog_close()나 tlog_find()에 사용된다.

파라미터	설명

flags	로그와 관련된 설정이다.
	
	다음은 flags에 설정되는 값에 대한 설명이다.
	
	• TLOG_LOCAL
	
	로컬 노드의 TX의 로그를 조회한다.
	
	• TLOG_REMOTE
	
	도메인 게이트웨이를 통해 발생된 TX의 로그를 조회한다.
	
	• TXLOG
	
	해당 노드에서 2PC 동작이 발생한 경우 기록을 남긴다.
	
	• GWTXLOG
	
	해당 노드에서 실행 중인 게이트웨이를 통해서 트랜잭션과 관련된 요청이 들어온 경우 remote_xid와 local_xid mapping 정보를 기록한다.
	
	다음은 각 파라미터의 설정에 따른 동작에 대한 설명이다.
	
	• (TMAXDIR)/log/tlog/TXLOG 파일을 기본으로 open한다.
	
	`name = NULL, flags = TLOG_LOCAL`
	
	• $(TMAXDIR)/log/tlog/GWTXLOG 파일을 open한다.
	
	`name = NULL, flags = TLOG_REMOTE`
	
	• name != NULL
	
	`name = 입력된이름, flags = TLOG_LOCAL`

- 반환값

반환값	설명
TLOG_OK	성공적으로 로그 파일을 open한 경우이다.
음수	성공적으로 로그 파일을 open할 때 에러가 발생한 경우이다.

- 예제

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

```
int main(int argc, char *argv[])
{
   int n;
   tlog_entry_t entry;
   tlog_file_t log;

   printf(“\n====== TXLOG =====\n”);
   n = tlog_open(NULL, &log, TLOG_LOCAL);
   printf(“n = %d\n”, n);

   if (n < 0) {
      printf(“tlog_open = %d, errno = %d\n”, n, errno);
      exit(1);
   }

   printf(“tlog_open success\n”);
   ...
}
```

- 관련 함수

  tlog_close(), tlog_find(), tlog_nodeno()

3.8.5. tmaxoserrno

시스템 호출 도중 에러가 발생할 경우 통합된 에러 번호가 설정되는 변수이다. 에러 발생 이후 여러 시스템 콜 호출시 에러 번호가 최초 에러 발생 이후에 변경되어 원인 파악이 어려워 질 수 있다. 이 때 tmaxoserrno 변수를 확인하면 tperrno 가 TPEOS, TPESYSTEM 발생 시점의 시스템 에러 번호를 알 수 있다. Windows 는 GetLastError() 값을 , Unix에서는 errno 의 값을 저장하고 있다.

3.8.6. Usiginit

사용자 시그널 핸들링에 필요한 매크로 설정에 사용되는 함수로 Windows 시스템 환경에는 사용되지 않는다. Usiginit()는 Tmax 시그널 핸들러를 초기화한다.

- 프로토타입

```
# include <usignal.h>
int  Usiginit(void)
```

- 반환값

반환값	설명
1	함수 호출에 성공한 경우이다.
-1	함수 호출에 실패한 경우이다. 에러가 발생한 경우에도 tperrno에는 값이 설정되지 않는다.

- 예제

```
...
#include <signal.h>
#include <usrinc/usignal.h>

void sig_alrm(int);
SERVICE(TPSVCINFO *msg)
{
    int  i, ret;

    data process...

    ret=Usiginit();
    if (ret==-1) { error processing }

    Usignal(SIGALRM, &sig_alrm);
    alarm(1);

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

void sig_alrm(int signo)
{
    if (signo == SIGALRM) printf("SIGALRM recieve\n");
}
```

3.8.7. Usignal

사용자 시그널 핸들링에 필요한 매크로 설정에 사용되는 함수로 Windows 시스템 환경에는 사용되지 않는다.
기본적인 Tmax 시그널 핸들러는 내부 수행에 필요한 SIGALRM, SIGPIPE, SIGTERM을 핸들링한다.

signal()을 이용하여 핸들링할 경우에는 시그널에 의한 Tmax 내부 장치가 작동하지 않는데 Tmax 라이브러리는 Usignal()를 제공하여 이를 방지한다. Usignal()은 사용자 시그널 핸들러에 대한 기록이 저장된 테이블을 관리하며 시그널이 발생한 경우에 Tmax 시그널 핸들러는 이 테이블을 참고한다. Tmax에서는 사용자 시그널 핸들러가 Tmax 시그널 핸들러에 우선되므로 사용자 시그널 핸들링을 마친 뒤에 Tmax 시그널 핸들러가 작동한다.

- 프로토타입

```
# include <usignal.h>
Sigfunc *Usignal(int sig, Sigfunc *func)
```

- 파라미터

파라미터	설명
sig	숫자 또는 상수 값이다. (예: 9 또는 SIGKILL)
func	호출할 함수 값이다.

- 반환값

반환값	설명
사용자 시그널 핸들러	예전 사용자 시그널 핸들러가 존재할 경우이다.
SIG_DFL	예전 사용자 시그널 핸들러가 존재하지 않는 경우이다. 오류가 발생한 경우에도 tperrno에는 값이 설정되지 않는다.

- 예제

```
...
#include <signal.h>
#include <usrinc/usignal.h>

void sig_alrm(int);
SERVICE(TPSVCINFO *msg)
{
    int             i, ret;
    data process...
    ret=Usiginit();
    if (ret==-1) { error processing }
    Usignal(SIGALRM, &sig_alrm);
    alarm(1);
    tpreturn(TPSUCCESS,0,(char *)msg->data, 0,0);
}

void sig_alrm(int signo)
{
    if (signo == SIGALRM) printf("SIGALRM recieve\n");
}
```

3.8.8. Uunixerr

시스템 호출 도중 에러가 발생할 경우 통합된 에러 번호가 설정되는 변수이다.

```
int Uunixerr
```

3.8.9. Uunix_err

ATMI API 호출이 실패하고 tperrno가 TPEOS로 설정된 경우 시스템 에러의 종류를 stderr로 출력하는 함수이다.

- 프로토타입

```
# include <Uunix.h>
void Uunix_err (char *msg)
```

- 파라미터

파라미터	설명
msg	에러가 발생한 시스템 호출에 앞서 추가하고 싶은 메시지이다. 일반적으로 프로그램명을 기록한다. 다음 중에 하나가 출력된다. `UCLOSE, UCREAT, UEXEC, UFCTNL, UFORK, ULSEEK, UMSGCTL, UMSGGET, UMSGSND, UMSGRCV, UOPEN, UPLOCK, UREAD, USEMCTL, USEMGET, USEMOP, USHMCTL, USHMGET, USHMAT, USHMDT, USTAT, UWRITE, USBRK, USYSMUL, UWAIT, UKILL, UTIME, UMKDIR, ULINK, UUNLINK, UUNAME, UNLIST`

- 예제

```
ret=tmaxreadenv("NO THAT FILE", "TMAX");
if (ret<0) {
   Uunix_err("myprog");
   exit(1);
}
```

다음은 앞의 예제를 수행한 결과이다.

```
mypog: UOPEN
```

3.8.10. Ustrerror

시스템 에러 코드(errno)에 대한 통합 에러 메시지를 반환하는 함수이다.

- 프로토타입

```
# include <Uunix.h>
char * Ustrerror(int err);
```

- 파라미터

파라미터	설명
err	알고 싶은 에러 메시지의 통합 에러 번호이다.

- 반환값

함수 호출에 성공하는 경우 errno에 대한 통합 에러 메시지의 포인터 주소가 반환된다.

다음 목록 중 하나가 chr * 타입 포인터로 반환된다.

```
UCLOSE, UCREAT, UEXEC, UFCTNL, UFORK, ULSEEK, UMSGCTL, UMSGGET, UMSGSND,
UMSGRCV, UOPEN, UPLOCK, UREAD, USEMCTL, USEMGET, USEMOP, USHMCTL, USHMGET,
USHMAT, USHMDT, USTAT, UWRITE, USBRK, USYSMUL, UWAIT, UKILL, UTIME, UMKDIR,
ULINK, UUNLINK, UUNAME, UNLIST
```

- 예제

```
ret=tmaxreadenv("NO THAT FILE", "TMAX");
if (ret<0)
{
    printf("%d->%s\n", Uunixerr, Ustrerror(Uunixerr));
    exit(1);
}
```

다음은 앞의 예제를 수행한 결과이다.

```
11->UOPEN
```

Appendix A: 에러별 대응 방안

2 : TPEBADDESC

설명	cd가 유효하지 않은 구별자이다. 함수를 사용할 때 반환된 구별자가 유효하지 않은 경우이다.
대응 방법	tpcall()을 통해 할당된 cd를 확인하거나 tpstart()를 다시 호출한다.

3 : TPEBLOCK

설명	요청한 서비스가 블로킹된 상태이다. TPNOBLOCK 설정된 상태에서 블로킹이 발생하였다. 서비스를 다시 요청하거나 블로킹이 해제된 후 다시 시도해야 한다. tpgetrply나 tpgetunsol를 NonBlock Mode로 호출했을 때 발생되었다면 호출한 시점에 소켓에 데이커가 없는 상태이므로 해당 에러를 무시하고 일정시간 sleep 후 재시도하는 Logic 추가가 필요하다.
대응 방법	서비스를 다시 요청하거나 블로킹이 해제된 후 다시 시도한다.

5 : TPELIMIT

설명	시스템 자원 또는 Tmax에서 제공하는 자원의 부족한 경우이다.
대응 방법	Tmax에서는 자원이 충분하지 않은 경우 운영체제에서 제공하는 자원을 할당받아 이를 이용한다.

6 : TPENOENT

설명	서비스 테이블에 서비스가 존재하지 않는 경우이거나 Tmax 엔진에서 서비스를 인식하지 못하는 경우이다.
대응 방법	환경파일이 수정되었다면 gst로 서비스 테이블을 새로 만들어 서버 애플리케이션 컴파일할 때 함께 컴파일한다.

7 : TPEOS

설명	운영체제 오류이다.
대응 방법	운영체제를 비롯하여 네트워크 및 기존 운영되던 환경이 변경되었는지 제반환경을 점검한다.

9 : TPEPROTO

설명	부적절한 상황에서 함수가 호출된 경우이다.
대응 방법	단계별 함수 호출이 잘못 사용된 경우로 버퍼를 설정하고 서비스를 요청하거나 대화형 통신하는 경우 send와 recv 등의 단계를 준수한다.

10 : TPESVCERR

설명	서비스 수행 중 서버 프로세스에서 에러가 발생한 경우이다.
대응 방법	서버 프로세스를 확인해야 한다.

11 : TPESVCFAIL

설명	서비스 수행 중 애플리케이션 레벨에서 에러가 발생한 경우이다. 서비스 요청에 대한 응답을 송신하는 서비스 루틴이 애플리케이션에서 에러가 발생한 경우이다.
대응 방법	애플리케이션을 확인해야 한다.

12 : TPESYSTEM

설명	Tmax 시스템에 이상이 발견된 경우이다.
대응 방법	일반적인 모든 에러를 포함하므로 여러가지를 점검한다. 네트워크에 대한 장애 및 서버 프로세스등 관련 제반 환경을 점검한다.

13 : TPETIME

설명	블로킹되어 있거나 어떤 원인에 의해 지정된 시간을 초과한 경우에 발생한다.
대응 방법	애플리케이션 레벨에서 시간 초과의 원인을 파악하고 정상적인 경우라면 타임아웃 시간을 조정한다. 애플리케이션에서 타임아웃 시간을 지정할 수 있다. 기본적으로는 환경파일의 BLOCKTIME 값을 적용한다.

14 : TPETRAN

설명	트랜잭션을 지원하지 않는 서비스인 경우 발생한다.
대응 방법	tpcall를 호출할 때 flags를 TPNOTRAN으로 설정한다.

15 : TPGOTSIG

설명	TPSIGRSTRT가 설정되지 않은 상태에서 시그널이 수신된 경우에 발생한다. COUSIN 항목을 이용하여 DDR을 하는 경우 CLH가 어느 노드에 있는 서비스를 요청해야 될지 모르는 경우에도 발생한다.

17 : TPEITYPE

설명	입력된 버퍼의 유형을 알 수 없는 경우에 발생한다. COUSIN 항목을 이용하여 DDR을 하는 경우 CLH가 어느 노드에 있는 서비스를 요청해야 될지 모르는 경우에도 발생한다.
대응 방법	데이터 값을 검사한다. 필드키 버퍼를 사용하는 경우에는 해당 필드키의 정의 여부를 확인하고, 구조체 버퍼를 사용하는 경우에는 구조체의 선언 여부를 확인한다.

18 : TPEOTYPE

설명	입력된 버퍼의 유형을 호출자가 알지 못하는 것으로 데이터의 유형 및 하위 유형과 입력된 버퍼의 유형이 일치하지 않은 경우에 발생한다.
대응 방법	데이터 값을 검사한다.

22 : TPEEVENT

설명	대화형 모드에서 발생하는 에러이다. 이벤트가 발생하였고 데이터는 전달되지 않은 경우로 이벤트는 revent 값에 반환된다.
대응 방법	이벤트 (revent) 값을 확인한다.

23 : TPEMATCH

설명	해당 조건을 만족하는 제거할 데이터를 찾지 못한 경우에 발생한다.
대응 방법	조건이 올바른지 검토한다.

24 : TPENOREADY

설명	서비스가 준비되지 않은 것이거나 구동은 되어있으나 활성화가 안 된 경우에 발생한다. 예를 들어 tpcall()할 때 이미 서비스가 NRDY 상태인 경우에 발생한다.
대응 방법	tmadmin에서 st –s로 서비스에 대한 상태를 확인하고 NRDY로 나타난다면 제대로 구동되어 있지 않은 것이다. 서버 프로세스를 재확인하고 다시 구동해야 하고 tpstart()를 다시 호출한다.

25 : TPESECURITY

설명	보안 설정을 사용하는 경우 허용되지 않은 사용자의 접근한 경우 발생한다.
대응 방법	사용자에 대한 권한을 확인한다.

26 : TPEQFULL

설명	요청된 서비스가 지정한 Max 큐에 도달한 경우에 발생한다.

대응 방법	Tmax 환경파일에서 또는 동적으로 tmadmin 툴에서 Max 큐 값을 조절할 수 있다.

27 : TPEQPURGE

설명	관리자가 강제로 큐를 Purge한 경우에 발생한다. 요청된 서비스가 큐에 대기 중 관리자에 의해 Purge된 경우에 발생한다.

28 : TPECLOSE

설명	Tmax가 구동되지 않았거나 접속을 할 수 없는 경우에 발생한다.
대응 방법	Tmax의 정상 구동 여부를 확인한 후 tpstart()를 다시 호출한다.

29 : TPESVRDOWN

설명	tpcall()한 서비스 때문에 서버가 다운된 상황이다.
대응 방법	서버 프로세스가 정상 작동 중인지를 확인한다. 또는 애플리케이션 로직의 문제로서 프로그램이 수행 중에 비정상적으로 종료될 수도 있으므로 애플리케이션을 확인한다.

30 : TPEPRESVC

설명	RQ 절에 등록하여 사용한 PRESVC에 대한 에러로서 현재는 사용하지 않으므로 발생하지 않는 에러이다.

31 : TPERDOWN

설명	Rolling Down이 발생한 경우 기존 서버 큐에 쌓여 있는 요청들에 대하여 tpgetrply를 호출할 때 TPERDOWN 에러가 설정되며 tpacall를 사용할 때 송신되었던 데이터가 tpgetrply의 수신 버퍼에 다시 담겨진다. Rolling Down이 발생한 이후에 호출되는 tpacall에 대해서도 TPERDOWN 에러가 설정된다.
대응 방법	릴리즈 노트를 참조한다.

32 : TPERDCLOSE

설명	더이상 서버 큐에 적체된 요청이 없는 상황에서 tpgetrply가 호출된 경우 TPERDCLOSE 에러가 설정된다.
대응 방법	릴리즈 노트를 참조한다.

33 : TPEMAXNO

설명	동시에 접속할 수 있는 유저가 한정되어 있는데 사용자 수가 최대 사용자 수에 도달한 경우 발생하는 에러이다.
대응 방법	tmadmin에서 ci(client Information)로 확인한다.

Appendix B: 헤더 파일 예제

B.1. <atmi.h>

Tmax 시스템을 사용하여 프로그램을 개발하기 위해 본 파일을 반드시 첨부해야 한다.

```
/* --------------------- usrinc/atmi.h ------------------------ */
/*                                */
/*              Copyright (c) 2000 - 2008 Tmax Soft Co., Ltd    */
/*                   All Rights Reserved            */
/*                                */
/* ------------------------------------------------------------ */


#ifndef _TMAX_ATMI_H
#define _TMAX_ATMI_H

#ifdef __EXPORT
#undef __EXPORT
#endif
#ifdef _WIN32
#ifdef _TMAX4GL
#define __EXPORT __stdcall
#else
#define __EXPORT __cdecl
#endif
#else
#define __EXPORT
#endif
#ifndef THLVAR
#define THLVAR
#endif

/* Flags to tpinit() for Tuxeedo compatability */
#define TPU_MASK    0x00000007
#define TPU_SIG     0x00000001
#define TPU_DIP     0x00000002
#define TPU_IGN     0x00000004
#define TPSA_FASTPATH   0x00000008
#define TPSA_PROTECTED  0x00000010

#if defined(_TMAX_MTLIB) || defined(_MCONTEXT)
#define TPSINGLECONTEXT  0
#define TPINVALIDCONTEXT -1
#define TPNULLCONTEXT    -2
#endif

/* ---------- flags from API ----- */
/* Most Significant Two Bytes are reserved for internal use */
#ifndef TPNOFLAGS
#define TPNOFLAGS       0x00000000
#endif
#define TPNOBLOCK       0x00000001
#define TPSIGRSTRT      0x00000002
#define TPNOREPLY         0x00000004
```

```
#define TPNOTRAN           0x00000008
#define TPTRAN              0x00000010
#define TPNOTIME           0x00000020
#define TPNOGETANY       0x00000040
#define TPGETANY           0x00000080
#define TPNOCHANGE       0x00000100
#define TPBLOCK         0x00000200
#define TPCONV              0x00000400
#define TPFLOWCONTROL     (TPCONV)
#define TPSENDONLY       0x00000800
#define TPRECVONLY       0x00001000
#define TPUDP           0x00002000
#define TPRQS                0x00004000
#define TPFUNC              0x00008000  /* API dependent functional flag */
#define TPABSOLUTE       (TPFUNC)
#define TPACK               (TPFUNC)
#define TPURGENT           (TPCONV)

/* --- flags used in tpstart() --- */
#define TPUNSOL_MASK       0x00000007
#define TPUNSOL_HND     0x00000001
#define TPUNSOL_IGN     0x00000002
#define TPUNSOL_POLL    0x00000004
#define TPUNIQUE           0x00000010
#define TPONLYONE         0x00000020
#if defined(_TMAX_MTLIB) || defined(_MCONTEXT)
#define TPMULTICONTEXTS 0x00000040
#endif


/* --- flags used in tpreturn() --- */
#define TPFAIL             0x0001
#define TPSUCCESS        0x0002
#define TPEXIT          0x0004
#define TPDOWN             0x0008
#define TP_FORWARD      0x0010         /* Internal use only */

/* --- return code used in tpreturn() for Messaging System --- */
#define TPFAIL_ACK      -1

/* ------ flags for reply type check ----- */
#define TPREQ           0
#define TPERR           -1

/* -------- for Tuxedo Compatability ------- */
/* Flags to tpscmt() - Valid TP_COMMIT_CONTROL characteristic values */
#define TP_CMT_LOGGED   0x01       /* return after commit decision is logged */
#define TP_CMT_COMPLETE 0x02       /* return after commit has completed */

/* Return values to tpchkauth() */
#define TPNOAUTH          0       /* no authentication */
#define TPSYSAUTH        1        /* system authentication */
#define TPAPPAUTH        2        /* system and application authentication */

#define XATMI_SERVICE_NAME_LENGTH 32       /* where x must be > 15 */

struct clid_t {
    long clientdata[4];
};
```

```
typedef struct clid_t CLIENTID;

struct tpsvcinfo {
  char name[XATMI_SERVICE_NAME_LENGTH];
  char *data;
  long len;
  long flags;
  int    cd;
  CLIENTID cltid;
};
typedef struct tpsvcinfo TPSVCINFO;


#if defined(_WIN32) || defined(_TMAX_MTLIB) || defined(_MCONTEXT)
#if defined(__cplusplus)
extern "C" {
#endif
/*
   Internal functions: ONLY BE CALLED FROM AUTOMATICALLY
   GENERATED STUB FILES. DO NOT DIRECTLY CALL THESE FUNCTIONS.
 */
int *__EXPORT _tmget_tperrno_addr(void);
long *__EXPORT _tmget_tpurcode_addr(void);

#ifdef _TMAX_KERNEL
extern THLVAR int tperrno;
extern THLVAR long tpurcode;
#else
#define tperrno  (*_tmget_tperrno_addr())
#define tpurcode (*_tmget_tpurcode_addr())
#endif
#if defined(__cplusplus)
}
#endif
#else
extern int tperrno;
extern long tpurcode;
#endif

#define TPEMINNO          1
#define TPEBADDESC      2
#define TPEBLOCK          3
#define TPEINVAL          4
#define TPELIMIT          5
#define TPENOENT          6
#define TPEOS            7
#define TPEPROTO          9
#define TPESVCERR        10
#define TPESVCFAIL      11
#define TPESYSTEM        12
#define TPETIME           13
#define TPETRAN           14
#define TPGOTSIG          15
#define TPEITYPE          17
#define TPEOTYPE          18
#define TPEEVENT          22
#define TPEMATCH          23
#define TPENOREADY      24
#define TPESECURITY    25
```

```
#define TPEQFULL          26
#define TPEQPURGE        27
#define TPECLOSE          28
#define TPESVRDOWN      29
#define TPEPRESVC        30
#define TPERDOWN     31
#define TPERDCLOSE   32
#define TPEMAXNO     33

/* ---- flags used in conv[]: don't use dummy flags ----*/
#define TPEV_DISCONIMM   0x00000001
#define TPEV_SVCERR     0x00000002
#define TPEV_SVCFAIL       0x00000004
#define TPEV_SVCSUCC       0x00000008
#define TPEV_CONVCLOSE   0x00000010  /* don't use this flag */
#define TPEV_SENDONLY     0x00000020
#define TPCONV_DUMMY1     0x00000800  /* don't use this flag: TPSENDONLY */
#define TPCONV_DUMMY2     0x00001000  /* don't use this flag: TPRECVONLY */
#define TPCONV_OUT       0x00010000
#define TPCONV_IN         0x00020000

#define X_OCTET           "X_OCTET"
#define X_C_TYPE          "X_C_TYPE"
#define X_COMMON          "X_COMMON"

#define TMTYPEFAIL       -1
#define TMTYPESUCC      0


#ifndef MAXTIDENT
#define MAXTIDENT        16 /* max len of identifier */
#endif

#ifndef MAX_PASSWD_LENGTH
#define MAX_PASSWD_LENGTH  16
#endif
#ifndef MAX_MNAME_LENGTH
#define MAX_MNAME_LENGTH  16
#endif


struct  tpstart_t {
  char  usrname[MAXTIDENT+2];   /* usr name */
  char  cltname[MAXTIDENT+2];   /* application client name */
  char  dompwd[MAX_PASSWD_LENGTH+2]; /* domain password */
  char  usrpwd[MAX_PASSWD_LENGTH+2]; /* passwd for usrid */
  int   flags;
};
typedef struct  tpstart_t TPSTART_T;


/* X/Open Typed Buffer related Function */

#if defined (__cplusplus)
extern "C" {
#endif

/* ----- client API ----- */
int __EXPORT tpstart(TPSTART_T *);
```

```
int __EXPORT tpend(void);
char *__EXPORT  tpalloc(char *type,char *subtype, long size);
char *__EXPORT  tprealloc(char *ptr, long size);
long __EXPORT tptypes(char *ptr,char *type,char *subtype);
void __EXPORT tpfree(char *ptr);
int __EXPORT tpcall(char *svc, char *idata, long ilen, char **odata,
              long *olen, long flags);
int __EXPORT tpacall(char *svc, char *data, long len, long flags);
int __EXPORT tpgetrply(int *cd, char **data, long *len, long flags);
int __EXPORT tpcancel(int cd);
char *__EXPORT  tpstrerror(int err_no);

/* ----- conversational API ----- */
int __EXPORT tpconnect(char *svc, char *data, long lenl, long flagsl);
int __EXPORT tpdiscon(int cd);
int __EXPORT tpsend(int cd, char *data, long lenl, long flagsl, long *revent);
int __EXPORT tprecv(int cd, char **data, long *len, long flagsl, long *revent);

/* ----- server API -------- */
void __EXPORT tpreturn(int rval, long rcode, char *data, long len, long flags);
void __EXPORT tpforward(char *svc, char *data, long len, long flags);
int __EXPORT tpadvertise(char *svcname, void (* func) (TPSVCINFO *));
int __EXPORT tpunadvertise(char *svcname);

/* ----- etc API ----- */
int __EXPORT tpnotify(CLIENTID *id, char *data, long len, long flags);
int __EXPORT gettperrno(void);
long __EXPORT gettpurcode(void);

/* ----- multithread/multicontext API ----- */
int __EXPORT tpsetctxt(int ctxtid, long flags);
int __EXPORT tpgetctxt(int *ctxtid, long flags);


#if defined (__cplusplus)
}
#endif

#endif       /* end of _TMAX_ATMI_H  */
```

B.2. <tmaxapi.h>

이 헤더 파일은 Tmax 시스템에 의해 제공되는 비표준 함수를 포함한다.

Tmax 시스템을 사용하여 프로그램을 개발하기 위해 본 파일을 반드시 첨부해야 한다.

```
/* ------------------------ usrinc/tmaxapi.h ------------------ */
/*                              */
/*              Copyright (c) 2000 - 2008 Tmax Soft Co., Ltd    */
/*                   All Rights Reserved            */
/*                              */
/* ------------------------------------------------------------ */

#ifndef _TMAXAPI_H
#define _TMAXAPI_H
```

```
#ifndef _CE_MODULE
#include <sys/types.h>
#endif
#include <usrinc/atmi.h>
#ifdef _WIN32
#ifdef _CE_MODULE
#include <winsock.h>
#else
#include <winsock2.h>
#endif  /* _CE_MODULE */
#include <usrinc/svct.h>
#include <usrinc/sdl.h>
#else
#ifndef ORA_PROC
#include <sys/socket.h>
#endif
#endif

/* client logout type */
#define CLIENT_CLOSE_NORMAL     0
#define CLIENT_CLOSE_ABNORMAL     1
#define CLIENT_PRUNED                2

/* RQ Sub-queue type */
#define TMAX_ANY_QUEUE              0
#define TMAX_FAIL_QUEUE            1
#define TMAX_REQ_QUEUE              2
#define TMAX_RPLY_QUEUE            3
#define TMAX_MAX_QUEUE          4

extern char _rq_sub_queue_name[TMAX_MAX_QUEUE][XATMI_SERVICE_NAME_LENGTH];

/* RQ related macros */
#define RQ_NAME_LENGTH             16

/* unsolicited msg type */
#define UNSOL_TPPOST                   1
#define UNSOL_TPBROADCAST         2
#define UNSOL_TPNOTIFY              3
#define UNSOL_TPSENDTOCLI         4
#define UNSOL_ANY                     5

/* RM type */
#define ORACLE_TYPE                1
#define SYBASE_TYPE                2
#define INFORMIX_TYPE              3
#define DB2_TYPE                   4

#define NONXA_MODE                  0
#define XA_MODE                        1

/* Check SVCINFO cmds */
#define ISSVC_FORWARDED 0x00000001
#define ISSVC_NOREPLY     0x00000002

/* TPEVCTL ctl_flags */
#define TPEV_SVC           0x00000001
#define TPEV_PROC         0x00000002
```

```
/* tpgethostaddr flags */
#define GET_SVR_CON     0x00000000
#define GET_CUR_IP       0x00000001

/* tmax_sq_get/tmax_sq_put flags */
#define TPSQ_PEEK         0x00001000
#define TPSQ_UPDATE     0x00002000
#define TPSQ_SYSKEY     0x00004000
#define TPSQ_KEYGEN     0x00008000

#define SQ_KEYLIST_T       void*
#define SQ_KEYINFO_T       sq_keyinfo_t
#define SQ_SYSKEY_SIZE   16

struct sq_keyinfo_s {
    long keylen;
    long datalen;
    time_t starttime;
    char *key;
};

typedef struct sq_keyinfo_s sq_keyinfo_t;

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

typedef struct tpevctl TPEVCTL;
typedef void __EXPORT Unsolfunc(char *, long, long);
#define TPUNSOLERR      ((Unsolfunc *) -1)

struct tptranid {
    int  info[3];
    int  flags;
};

typedef struct tptranid TPTRANID;

/* Multicast call related structures */
struct svglist {
    int ns_entry;     /* number of entries of s_list */
    int nf_entry;     /* number of entries of f_list */
    int *s_list;       /* list of server group numbers */
    int *f_list;       /* list of server group numbers */
};

/* Jun/23/2008 KANAKO TPMCALL_UPDATE <<start>> */
struct svglistx {
    int ns_entry;    /* number of entries of s_list */
    int nf_entry;    /* number of entries of f_list */
    int nr_entry;    /* number of entries of r_list */
    int *s_list;      /* list of server group numbers */
    int *f_list;      /* list of server group numbers */
    int *r_list;      /* list of tpurcode of each server group */
};
```

```
/* Jun/23/2008 KANAKO TPMCALL_UPDATE <<end>> */

/* My svrinfo */
#ifndef TMAX_NAME_SIZE
#define TMAX_NAME_SIZE          16
#endif

#ifndef MAX_DBSESSIONID_SIZE
#define MAX_DBSESSIONID_SIZE       128
#endif

typedef struct {
    int nodeno; /* node index */
    int svgi;     /* server group index; unique in the node */
    int svri;     /* server index; unique in the node */
    int spri;     /* server process index; unique in the node */
    int spr_seqno;  /* server process seqno ; unique in the server */
    int min, max;   /* min/max server process number */
    int clhi;     /* for RDP only, corresponding CLH id */
    char nodename[TMAX_NAME_SIZE];
    char svgname[TMAX_NAME_SIZE];
    char svrname[TMAX_NAME_SIZE];
    char reserved_char[TMAX_NAME_SIZE];
    /* for more detail use tmadmin API */
} TMAXSVRINFO;

#define MSGIDLEN        32
#define CORRIDLEN       32

typedef struct {               /* control parameters to queue primitives */
        int flags;             /* indicates which of the values are set */
        int deq_time;          /* absolute/relative  time for dequeuing */
        int priority;          /* enqueue priority */
        int diagnostic;        /* indicates reason for failure */
        char msgid[MSGIDLEN]; /* id of message before which to queue */
        char corrid[CORRIDLEN];/* correlation id used to identify message */
        char replyqueue[TMAX_NAME_SIZE];  /* queue name for reply message */
        char failurequeue[TMAX_NAME_SIZE];/* queue name for failure message */
        CLIENTID cltid;         /* client identifier for originating client */
        int urcode;            /* application user-return code */
        int appkey;            /* application authentication client key */
        int delivery_qos;
        int reply_qos;
        int exp_time;
} TMQCTL;

#ifdef _WIN32
typedef int (__EXPORT *WinTmaxCallback)(WPARAM, LPARAM);
#endif

/* Macro functions */
#define tpalivechk()    tmax_chk_conn(0)

/* mode for IP-based ACL */
#define TMAX_ACL_ALLOW      0
#define TMAX_ACL_DENY       1

/* reserved mask value for IP-based ACL */
#define TMAX_ACL_IPADDR     32
```

```
#if defined (__cplusplus)
extern "C" {
#endif

/* ----- unsolicited messaging API ----- */
long __EXPORT tpsubscribe(char *eventexpr, char *filter, TPEVCTL *ctl, long flags);
long __EXPORT tpsubscribe2(char *eventexpr, char *svcname, long flags);
int __EXPORT tpunsubscribe(long sd, long flags);
int __EXPORT tppost(char *eventname, char *data, long len, long flags);
int __EXPORT tpbroadcast(char *lnid, char *usrname, char *cltname, char *data,
        long len, long flags);
Unsolfunc *__EXPORT tpsetunsol(Unsolfunc *func);
int __EXPORT tpsetunsol_flag(int flag);
int __EXPORT tpgetunsol(int type, char **data, long *len, long flags);
int __EXPORT tpclearunsol(void);
int __EXPORT tpchkunsol(void);

/* ----- RQS API -------- */
int __EXPORT tpenq(char *qname, char *svc, char *data, long len, long flags);
int __EXPORT tpdeq(char *qname, char *svc, char **data, long *len, long flags);
int __EXPORT tpenq_ctl(char *qname, char *svc, TMQCTL *ctl, char *data,
                        long len, long flags);
int __EXPORT tpdeq_ctl(char *qname, char *svc, TMQCTL *ctl, char **data,
                        long *len, long flags);
int __EXPORT tpqstat(char *qname, long type);
int __EXPORT tpqsvcstat(char *qname, char *svc, long type);
int __EXPORT tpextsvcname(char *data, char *svc);
int __EXPORT tpextsvcinfo(char *data, char *svc, int *type, int *errcode);
int __EXPORT tpreissue(char *qname, char *filter, long flags);
char *__EXPORT tpsubqname(int type);

/* ----- server API -------- */
int __EXPORT tpgetminsvr(void);
int __EXPORT tpgetmaxsvr(void);
int __EXPORT tpgetmaxuser(void);
int __EXPORT tpgetsvrseqno(void);
int __EXPORT tpgetmysvrid(void);
int __EXPORT tpgetmysvrno(void);
int __EXPORT tpgetmaxuser(void);
int __EXPORT tpsendtocli(int clid, char *data, long len, long flags);
int __EXPORT tpgetclid(void);
int __EXPORT tpgetpeer_ipaddr(struct sockaddr *name, int *namelen);
int __EXPORT tpchkclid(int clid);
int __EXPORT tmax_clh_maxuser(void);
int __EXPORT tmax_my_svrinfo(TMAXSVRINFO*);
int __EXPORT tmax_cind2clid(int cind);
char *__EXPORT tpgetmynode(int *nodeno);
char *__EXPORT tpgetmysvg(void);
int __EXPORT tpgetmysvgno(void);
int __EXPORT tmax_is_restarted(void);
char *__EXPORT tpgetsvcname(int svci);
int __EXPORT tpsuspendtx(TPTRANID *tranid, long flags);
int __EXPORT tpresumetx(TPTRANID *tranid, long flags);

/* ----- etc API ----------- */
int __EXPORT tp_sleep(int sec);
int __EXPORT tptsleep(struct timeval *timeout);
int __EXPORT tp_usleep(int usec);
```

```
int __EXPORT tpset_timeout(int sec);
int __EXPORT tpget_timeout(void);
int __EXPORT tmaxreadenv(char *file, char *label);
char *__EXPORT tpgetenv(char* str);
int __EXPORT tpputenv(char* str);
int __EXPORT tpgetsockname(struct sockaddr *name, int *namelen);
int __EXPORT tpgetpeername(struct sockaddr *name, int *namelen);
int __EXPORT tpgetactivesvr(char *nodename, char **outbufp);
int __EXPORT tperrordetail(int i);
int __EXPORT tpreset(void);
int __EXPORT tptobackup(void);
struct svglist *__EXPORT
    tpmcall(char *qname, char *svc, char *data, long len, long flags);
struct svglistx *__EXPORT
    tpmcallx(char *svc, char *data, long len, long flags);
struct svglist *__EXPORT tpgetsvglist(char *svc, long flags);
int __EXPORT tpsvgcall(int svgno, char *qname,
    char *svc, char *data, long len, long flags);
int __EXPORT tpflush(void);
char *__EXPORT tmaxlastsvc(char *idata, char *odata, long flags);
int __EXPORT tpgetorgnode(int clid);
int __EXPORT tpgetorgclh(int clid);
char *__EXPORT tpgetnodename(int nodeno);
int __EXPORT tpgetnodeno(char *nodename);
int __EXPORT tpgetasize(char *data);
int __EXPORT tpgettype(char *data);
char * __EXPORT tpgetsubtype(char *data);
int __EXPORT tpgetcliaddr(int clid, int *ip, int *port, long flags);
int __EXPORT tmax_chk_conn(int timeout);
int __EXPORT tpgethostaddr(int *ip, int *port, long flags);
char * __EXPORT tpgetdbsessionid(int flags);
int __EXPORT tpcallsvg(int svgno, char *svc, char *idata, long ilen,
    char **odata, long *olen, long flags);
int __EXPORT tpacallsvg(int svgno, char *svc, char *data, long len, long flags);

#if defined(_WIN32)
int __EXPORT WinTmaxAcall(TPSTART_T *sinfo, HANDLE wHandle, unsigned int msgType,
    char *svc, char *sndbuf, int len, int flags);
int __EXPORT WinTmaxAcall2(TPSTART_T *sinfo, WinTmaxCallback fn,
    char *svc, char *sndbuf, int len, int flags);
#endif

#if !defined(_TMAX_KERNEL) && !defined(_TMAX_RCA_H)
/* ------- User supplied routines ---------- */
int __EXPORT tpsvrinit(int argc, char *argv[]);
int __EXPORT tpsvrdone(void);
void __EXPORT tpsvctimeout(TPSVCINFO *msg);
int __EXPORT _tmax_event_handler(char *progname, int pid, int tid, char *msg,
                                 int flags);
int __EXPORT tpsetdbsessionid(char dbsessionid[MAX_DBSESSIONID_SIZE], int flags);
int __EXPORT tpprechk(void);
#endif

/*
   Internal functions: ONLY BE CALLED FROM AUTOMATICALLY
   GENERATED STUB FILES. DO NOT DIRECTLY CALL THESE FUNCTIONS.
 */
int __EXPORT get_clhfd(void);
int __EXPORT tmax_chk_svcinfo(int cmd);
```

```
int __EXPORT _tmax_main(int argc, char *argv[]);
int __EXPORT _tmax_cob_main(int argc, char *argv[]);
#if defined(_WIN32)
int __EXPORT _tmax_regfn(void *initFn, void *doneFn, void *timeoutFn,
                         void *userMainFn);
int __EXPORT _tmax_regtab(int svcTabSz, _svc_t *svcTab, int funcTabSz,
                          void *funcTab);
int __EXPORT _tmax_regsdl(int _sdl_table_size2, struct _sdl_struct_s *_sdl_table2,
    int _sdl_field_table_size2, struct _sdl_field_s *_sdl_field_table2);
int __EXPORT _tmax_regevthnd(void *evthndFn);
#endif
int __EXPORT _double_encode(char *in, char *out);
int __EXPORT _double_decode(char *in, char *out);
/* --- power builder interface API --- */
int __EXPORT _make_struct_from_pbindata(char *subtype, char *tpidata, int ilen,
                                        char *indata);
int __EXPORT _make_field_from_pbindata(char **tpidata, char *indata);
int __EXPORT _make_pbodata_from_struct(char *subtype, char *odata, int olen,
                                       char *tpodata);
int __EXPORT _make_pbodata_from_field(char *form, char *odata, char *tpodata);
int __EXPORT _make_pbfodata_from_field(char *fform, char *fodata, char *tpodata);
int __EXPORT _get_value_from_pbsdata(char *cur, char *vald);
int __EXPORT _get_name_value_from_pbfdata(char *cur, int *n2);
int __EXPORT _get_name_from_form(char *cur);
int __EXPORT _insert_value_to_pbodata(int type, char *out, char *in, int asize,
                                      int asize2);
int __EXPORT tmax_get_db_usrname(char *svgname, char *usrname, int type);
int __EXPORT tmax_get_db_passwd(char *svgname, char *passwd, int type);
int __EXPORT tmax_get_db_tnsname(char *svgname, char *tnsname, int type);

int __EXPORT tmax_is_xa();

/* 4.0 Sp2 Fix1 added */
int __EXPORT tmax_get_svccnt();
int __EXPORT tmax_get_svclist(char *buf, int bufsize);

/* ----- SessionQ API ----------- */
int __EXPORT tmax_sq_put(char *key, long keylenl, char *data, long lenl,
                         long flagsl);
int __EXPORT tmax_sq_get(char *key, long keylenl, char **data, long *lenl,
                         long flagsl);
int __EXPORT tmax_sq_count(void);
int __EXPORT tmax_sq_purge(char *key, long keylenl);
int __EXPORT tmax_sq_keygen(char *key, long keylenl);
SQ_KEYLIST_T __EXPORT tmax_sq_getkeylist(char *key, long keylenl);

int __EXPORT tmax_gq_put(char *key, long keylenl, char *data, long lenl,
                         long flagsl);
int __EXPORT tmax_gq_get(char *key, long keylenl, char **data, long *lenl,
                         long flagsl);
int __EXPORT tmax_gq_count();
int __EXPORT tmax_gq_purge(char *key, long keylenl);
int __EXPORT tmax_gq_keygen(char *key, long keylenl);
SQ_KEYLIST_T __EXPORT tmax_gq_getkeylist(char *key, long keylenl);

int __EXPORT tmax_get_sessionid(void);
int __EXPORT tmax_keylist_count(SQ_KEYLIST_T keylist);
int __EXPORT tmax_keylist_getakey(SQ_KEYLIST_T keylist,int nth,SQ_KEYINFO_T *keyinfo);
int __EXPORT tmax_keylist_free(SQ_KEYLIST_T keylist);
```

```
int __EXPORT tpgetsprlist(char *svc,int svgno, int *starti, int *endi, long flagsl);
int __EXPORT tpspracall(char *svcname,int spri, char *data, long lenl, long flagsl);
int __EXPORT tpspracall2(char *svcname, int startspri, int nth, char *data, long lenl,
                         long flagsl);

/* API for IP-based ACL */
int __EXPORT tmax_add_acl(int nodeno, char *ip, int mask, int mode, int flags);

#if defined (__cplusplus)
}
#endif

#endif       /* end of _TMAXAPI_H  */
```