



BEATuxedo®

ATMI COBOL Function Reference

Copyright

Copyright © 2005 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, BEA Liquid Data for WebLogic, BEA WebLogic Server, Built on BEA, Jolt, JoltBeans, SteelThread, Top End, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Manager, BEA MessageQ, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Enterprise Security, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Java Adapter for Mainframe, BEA WebLogic JDriver, BEA WebLogic JRockit, BEA WebLogic Log Central, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server Process Edition, BEA WebLogic WorkGroup Edition, BEA WebLogic Workshop, and Liquid Computing are trademarks of BEA Systems, Inc. BEA Mission Critical Support is a service mark of BEA Systems, Inc. All other company and product names may be the subject of intellectual property rights reserved by third parties.

All other trademarks are the property of their respective companies.

Contents

About This Document

What You Need to Know	vii
e-docs Web Site	vii
How to Print the Document	viii
Related Information	viii
Contact Us!	viii
Documentation Conventions	ix

Section 3(cbl) - COBOL Functions

Introduction to the COBOL Application-Transaction Monitor Interface	7
FINIT, FINIT32(3cbl)	40
FVFTOS, FVFTOS32(3cbl)	41
FVSTOF(3cbl).	43
TPABORT(3cbl)	45
TPACALL(3cbl)	46
TPADVERTISE(3cbl)	50
TPBEGIN(3cbl).	52
TPBROADCAST(3cbl).	54
TPCALL(3cbl).	57
TPCANCEL(3cbl)	62
TPCHKAUTH(3cbl)	63
TPCHKUNSOL(3cbl)	64

TPCLOSE(3cbl)	66
TPCOMMIT(3cbl)	67
TPCONNECT(3cbl)	70
TPDEQUEUE(3cbl)	73
TPDISCON(3cbl)	83
TPENQUEUE(3cbl)	85
TPFORWAR(3cbl)	96
TPGBLKTIME(3cbl)	98
TPGETCTXT(3cbl)	100
TPGETLEV(3cbl)	101
TPGETRPLY(3cbl)	102
TPGETUNSOL(3cbl)	107
TPGPRIOR(3cbl)	108
TPINITIALIZE(3cbl)	109
TPKEYCLOSE(3cbl)	117
TPKEYGETINFO(3cbl)	118
TPKEYOPEN(3cbl)	121
TPKEYSETINFO(3cbl)	124
TPNOTIFY(3cbl)	126
TPOPEN(3cbl)	128
TPPOST(3cbl)	130
TPPRECV(3cbl)	134
TPRESUME(3cbl)	139
TPRETURN(3cbl)	140
TPSBLKTIME(3cbl)	144
TPSCMT(3cbl)	146
TPSEND(3cbl)	149
TPSETCTXT(3cbl)	152

TPSETUNSOL(3cbl)	154
TPSPRIO(3cbl)	156
TPSUBSCRIBE(3cbl)	157
TPSUSPEND(3cbl)	163
TPSVCSTART(3cbl)	165
TPSVRDONE(3cbl)	168
TPSVRINIT(3cbl)	169
TPTERM(3cbl)	170
TPUNADVERTISE(3cbl)	172
TPUNSUBSCRIBE(3cbl)	173
TXBEGIN(3cbl)	176
TXCLOSE(3cbl)	177
TXCOMMIT(3cbl)	179
TXINFORM(3cbl)	181
TXOPEN(3cbl)	182
TXROLLBACK(3cbl)	184
TXSETCOMMITRET(3cbl)	186
TXSETTRANCTL(3cbl)	187
TXSETTIMEOUT(3cbl)	189
USERLOG(3cbl)	190

About This Document

This document provides reference information on COBOL bindings used in the BEA Tuxedo ATMI environment. The reference pages are arranged in alphabetical order by function name.

What You Need to Know

This document is intended for the following audiences:

- administrators who are interested in configuring and managing applications in a BEA Tuxedo environment
- application developers who are interested in programming applications in a BEA Tuxedo environment

This document assumes a familiarity with the BEA Tuxedo platform and COBOL programming.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the “e-docs” Product Documentation page at <http://e-docs.bea.com>.

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser.

A PDF version of this document is available on the BEA Tuxedo documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the BEA Tuxedo documentation Home page, click the PDF files button and select the document you want to print.

If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe Web site at <http://www.adobe.com/>.

Related Information

Related documents are listed in the See Also section of each reference page.

Contact Us!

Your feedback on the BEA Tuxedo documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the BEA Tuxedo documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA Tuxedo 9.0 release.

If you have any questions about this version of BEA Tuxedo, or if you have problems installing and running BEA Tuxedo, contact BEA Customer Support through BEA WebSupport at **www.bea.com**. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
<i>italics</i>	Indicates emphasis or book titles.
monospace text	Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard. <i>Examples:</i> <pre>#include <iostream.h> void main () the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float</pre>
monospace boldface text	Identifies significant words in code. <i>Example:</i> <pre>void commit ()</pre>
<i>monospace italic text</i>	Identifies variables in code. <i>Example:</i> <pre>String <i>expr</i></pre>
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. <i>Examples:</i> <pre>LPT1 SIGNON OR</pre>

Convention	Item
{ }	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
[]	Indicates optional items in a syntax line. The brackets themselves should never be typed. <i>Example:</i> <pre>buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...</pre>
	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.
...	Indicates one of the following in a command line: <ul style="list-style-type: none"> • That an argument can be repeated several times in a command line • That the statement omits additional optional arguments • That you can enter additional parameters, values, or other information The ellipsis itself should never be typed. <i>Example:</i> <pre>buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...</pre>
.	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

Section 3(cbl) - COBOL Functions

Table 1 BEA Tuxedo ATMI COBOL Functions

Name	Description
Introduction to the COBOL Application-Transaction Monitor Interface	Provides an introduction to the COBOL ATMI
FINIT, FINIT32 (3cbl)	Initializes fielded buffer
FVFTOS, FVFTOS32 (3cbl)	Copies from fielded buffer to COBOL structure
FVSTOF (3cbl)	Copies from C structure to fielded buffer
TPABORT (3cbl)	Abort current BEA Tuxedo ATMI transaction
TPACALL (3cbl)	Routine to send a message to a service asynchronously
TPADVERTISE (3cbl)	Routine for advertising service names
TPBEGIN (3cbl)	Routine to begin a BEA Tuxedo ATMI transaction
TPBROADCAST (3cbl)	Broadcasts notification by name
TPCALL (3cbl)	Routine to send a message to a service synchronously
TPCANCEL (3cbl)	Cancels a communication handle for an outstanding reply
TPCHKAUTH (3cbl)	Checks if authentication required to join a BEA Tuxedo ATMI application

Table 1 BEA Tuxedo ATMI COBOL Functions

Name	Description
TPCHKUNSOL (3cbl)	Checks for unsolicited message
TPCLOSE (3cbl)	Closes the BEA Tuxedo ATMI resource manager
TPCOMMIT (3cbl)	Commits current BEA Tuxedo ATMI transaction
TPCONNECT (3cbl)	Establishes a conversational connection
TPDEQUEUE (3cbl)	Routine to dequeue a message from a queue
TPDISCON (3cbl)	Takes down a conversational connection
TPENQUEUE (3cbl)	Routine to enqueue a message
TPFORWAR (3cbl)	Forwards a BEA Tuxedo ATMI service request to another routine
TPGBLKTIME (3cbl)	Routine for retrieving a previously set, per second, blocktime value
TPGETCTXT (3cbl)	Retrieves a context identifier for the current application association
TPGETLEV (3cbl)	Checks if a BEA Tuxedo ATMI transaction is in progress
TPGETRPLY (3cbl)	Gets reply from asynchronous message
TPGETUNSOL (3cbl)	Gets unsolicited message
TPGPRIOR (3cbl)	Gets service request priority
TPINITIALIZE (3cbl)	Joins a BEA Tuxedo ATMI application
TPKEYCLOSE (3cbl)	Closes a previously opened key handle
TPKEYGETINFO (3cbl)	Gets information associated with a key handle
TPKEYOPEN (3cbl)	Opens a key handle for digital signature generation, message encryption, or message decryption
TPKEYSETINFO (3cbl)	Sets optional attribute parameters associated with a key handle
TPNOTIFY (3cbl)	Sends notification by client identifier

Table 1 BEA Tuxedo ATMI COBOL Functions

Name	Description
TPOPEN (3cbl)	Opens the BEA Tuxedo ATMI resource manager
TPPOST (3cbl)	Posts an event
TPRECV (3cbl)	Receives a message in a conversational connection

Table 1 BEA Tuxedo ATMI COBOL Functions

Name	Description
TPRESUME (3cbl)	Resumes a global transaction
TPRETURN (3cbl)	Returns from a BEA Tuxedo ATMI service routine
TPSBLKTIME (3cbl)	Routine for setting the blocktime value, in seconds, of a potential blocking API.
TPSCMT (3cbl)	Sets when TPCOMMIT should return
TPSEND (3cbl)	Routine to send a message in a conversational connection
TPSETCTXT (3cbl)	Sets a context identifier for the current application association
TPSETUNSOL (3cbl)	Sets method for handling unsolicited messages
TPSPRIO (3cbl)	Sets service request priority
TPSUBSCRIBE (3cbl)	Subscribes to an event
TPSUSPEND (3cbl)	Suspends a global transaction
TPSVCSTART (3cbl)	Starts a BEA Tuxedo ATMI service
TPSVRDONE (3cbl)	Routine to terminate a BEA Tuxedo ATMI server
TPSVRINIT (3cbl)	Routine to initialize a BEA Tuxedo ATMI server
TPTERM (3cbl)	Leaves an application
TPUNADVERTISE (3cbl)	Routine for unadvertising service names
TPUNSUBSCRIBE (3cbl)	Unsubscribes to an event
TXBEGIN (3cbl)	Begins a global transaction
TXCLOSE (3cbl)	Closes a set of resource managers
TXCOMMIT (3cbl)	Commits a transaction
TXINFORM (3cbl)	Returns global transaction information
TXOPEN (3cbl)	Opens a set of resource managers
TXROLLBACK (3cbl)	Rolls back a transaction

Table 1 BEA Tuxedo ATMI COBOL Functions

Name	Description
TXSETCOMMITRET (3cbl)	Sets <i>commit_return</i> characteristic

Table 1 BEA Tuxedo ATMI COBOL Functions

Name	Description
TXSETTRANCTL (3cbl)	Sets <i>transaction_control</i> characteristic
TXSETTIMEOUT (3cbl)	Sets <i>transaction_timeout</i> characteristic
USERLOG (3cbl)	Writes a message to the BEA Tuxedo ATMI central event log

Introduction to the COBOL Application-Transaction Monitor Interface

Description

The Application-Transaction Monitor Interface (ATMI) provides the interface between the COBOL application and the transaction processing system. This interface is known as ATMI and these pages specify its COBOL language binding. It provides routines to open and close resources, manage transactions, manage record types, and invoke request/response and conversational service calls.

Communication Paradigms

The routines described in the ATMI reference pages imply a particular model of communication. This model is expressed in terms of how client and server programs can communicate using request and reply messages.

There are two basic communication paradigms: request/response and conversational. Request/response services are invoked by service requests along with their associated data. Request/response services can receive exactly one request (upon entering the service routine) and send at most one reply (upon returning from the service routine). Conversational services, on the other hand, are invoked by connection requests along with a means of referring to the open connection (that is, a handle used in calling subsequent connection routines). Once the connection has been established and the service routine invoked, either the connecting program or the conversational service can send and receive data as defined by the application until the connection is torn down.

Note that a program can initiate both request/response and conversational communication, but cannot accept both request/response and conversational service requests. The following sections describe the two communication paradigms in greater detail.

Note: In various parts of the BEA Tuxedo ATMI documentation we refer to *threads*. Because the BEA Tuxedo system does not support multithreading in COBOL, COBOL programmers may assume that the term *thread* refers to an entire process or context, depending on the circumstances. For example:

- A multithreaded/multicontexted C client with three threads associated with three contexts maps to a multicontexted COBOL client with three contexts.
- A multithreaded/single-context C client with three threads associated with a single context maps to a non-threaded, single-context COBOL client.

BEA Tuxedo Request/ Response Paradigm for Client/Server

With regard to request/response communication, a client is defined as a program that can send requests and receive replies. By definition, clients cannot receive requests nor send replies. A client can send any number of requests, and can wait for the replies synchronously or receive (some limited number of) the replies at its convenience. In certain cases, a client can send a request that has no reply. `TPINITIALIZE()` and `TPTERM()` allow a client to join and leave a BEA Tuxedo ATMI application.

A request/response server is a program that can receive one (and only one) service request at a time and send at most one reply to that request. While a server is working on a particular request, it can act like a client by initiating request/response or conversational requests and receiving their replies. In such a capacity, a server is called a requester. Note that both client and server programs can be requesters (in fact, a client can be nothing but a requester).

A request/response server can forward a request to another request/response server. Here, the server passes along the request it received to another server and does not expect a reply. It is the responsibility of the last server in the chain to send the reply to the original requester. Use of the forwarding routine ensures that the original requester ultimately receives its reply.

Servers and service routines offer a structured approach to writing BEA Tuxedo ATMI applications. In a server, the application writer can concentrate on the work performed by the service rather than communications details such as receiving requests and sending replies. Because many of the communication details are handled by the BEA Tuxedo system, the application must adhere to certain conventions when writing a service routine. At the time a server finishes its service routine, it can send a reply using `TPRETURN()` or forward the request using `TPFORWAR()`. A service is not allowed to perform any other work nor is it allowed to communicate with any other program after this point. Thus, a service performed by a server is started when a request is received and ended when either a reply is sent or the request is forwarded.

Concerning request and reply messages, there is an inherent difference between the two: a request has no associated context before it is sent, but a reply does. For example, when sending a request, the caller must supply addressing information, whereas a reply is always returned to the program that originated the request, that is, addressing context is maintained for a reply and the sender of the reply can exert no control over its destination. The differences between the two message types manifest themselves in the parameters and descriptions of the routines described in `TPCALL()`.

When a request message is sent, it is sent at a particular priority. The priority affects how a request is dequeued: when a server dequeues requests, it dequeues the one with the highest priority. To

prevent starvation, the oldest request is dequeued every so often regardless of priority. By default, a request's priority is associated with the service name to which the request is being sent. Service names can be given priorities at configuration time (see [UBBCONFIG\(5\)](#)). A default priority is used if none is defined. In addition, the priority can be set at run time using a routine (`TPSPRIO()`) described in `TPCALL()`. By doing so, the caller can override the configuration or default priority when the message is sent.

BEA Tuxedo System Conversational Paradigm for Client/Server

With regard to conversational communication, a client is defined as a program that can initiate a conversation but cannot accept a connection request.

A conversational server is a program that can receive connection requests. Once the connection has been established and the service routine invoked, either the connecting program or the conversational service can send and receive data as defined by the application until the connection is torn down. The conversation is half-duplex in nature such that one side of the connection has control and can send data until it gives up control to the other side. While the connection is established, the server is "reserved" such that no other program can establish a connection with the server.

As with a request/response server, the conversational server can act as a requester by initiating other requests or connections with other servers. Unlike a request/response server, a conversational server can not forward a request to another server. Thus, a conversational service performed by a server is started when a request is received and ended when the final reply is sent via `TPRETURN()`.

Once the connection is established, the communications handle implies any context needed regarding addressing information for the participants. Messages can be sent and received as needed by the application. There is no inherent difference between the request and reply messages and no notion of priority of messages.

BEA Tuxedo System Queued Message Model

The BEA Tuxedo ATMI queued message model allows for enqueueing a request message to stable storage for subsequent processing without waiting for its completion, and optionally getting a reply via a queued response message. The ATMI functions that queue messages and dequeue responses are `TPENQUEUE()` and `TPDEQUEUE()`. They can be called from any type of BEA Tuxedo ATMI application processes: client, server, or conversational.

The queued message facility is an XA-compliant resource manager. Persistent messages are enqueued and dequeued within transactions to ensure reliable one-time-only processing.

ATMI Transactions

The BEA Tuxedo system supports two sets of mutually exclusive functions for defining and managing transactions: the BEA Tuxedo system's ATMI transaction demarcation functions (the names of which include the prefix `TP`) and X/Open's TX Interface functions (the names of which include the prefix `TX_`). Because X/Open used ATMI's transaction demarcation functions as the base for the TX Interface, the syntax and semantics of the TX Interface are quite similar to those of the ATMI. This section is an overview of ATMI transaction concepts. The next section introduces additional concepts about the TX Interface.

In the BEA Tuxedo system, a *transaction* is used to define a single logical unit of work that either wholly succeeds or has no effect whatsoever. A transaction allows work performed in many processes, possibly at different sites, to be treated as an atomic unit of work. The initiator of a transaction normally uses `TPBEGIN()` and either `TPCOMMIT()` or `TPABORT()` to delineate the operations within a transaction.

The initiator may also suspend its work on the current transaction by issuing `TPSUSPEND()`. Another process may take over the role of the initiator of a suspended transaction by issuing `TPRESUME()`. As a transaction initiator, a program must call one of the following: `TPSUSPEND()`, `TPCOMMIT()`, or `TPABORT()`. Thus, one program can start a transaction that another may finish.

If a program calling a service is in transaction mode, then the called service routine is also placed in transaction mode on behalf of the same transaction. Otherwise, whether the service is invoked in transaction mode or not depends on options specified for the service in the configuration file. A service that is not invoked in transaction mode can define multiple transactions between the time it is invoked and the time it ends. On the other hand, a service routine invoked in transaction mode can participate in only one transaction, and work on that transaction is completed upon termination of the service routine. Note that a connection cannot be upgraded to transaction mode: if `TPBEGIN()` is called while a conversation exists, the conversation remains outside of the transaction (as if `TPCONNECT()` had been called with the `TPNOTRAN` setting).

A service routine joining a transaction that was started by another program is called a *participant*. A transaction can have several participants. A service can be invoked to do work on the same transaction more than once. Only the initiator of a transaction (that is, a program calling either `TPBEGIN()` or `TPRESUME()`) can call `TPCOMMIT()` or `TPABORT()`. Participants influence the outcome of a transaction by using `TPRETURN()` or `TPFORWAR()`. These two calls signify the end of a service routine and indicate that the routine has finished its part of the transaction.

TX Transactions

Transactions defined by the TX Interface are practically identical with those defined by the ATMI functions. An application writer may use either set of functions when writing clients and service

routes. In fact, the BEA Tuxedo system does not require all client and server programs within a single application to use one set of functions or the other. However, the two function sets may not be used together within a single program (that is, a program cannot call `TPBEGIN()` and later call `TXCOMMIT()`).

The TX Interface has two calls for opening and closing resource managers in a portable manner, `TXOPEN()` and `TXCLOSE()`, respectively. Transactions are started with `TXBEGIN()` and completed with either `TXCOMMIT()` or `TXROLLBACK()`. `TXINFORM()` is used to retrieve transaction information, and there are three calls to set options for transactions: `TXSETCOMMITRET()`, `TXSETTRANCTL()`, and `TXSETTIMEOUT()`. The TX Interface has no equivalents to ATMI's `TPSUSPEND()` and `TPRESUME()`.

In addition to the semantics and rules defined for ATMI transactions, the TX Interface has some additional semantics that are worth introducing here. First, service routine writers wanting to use the TX Interface must supply their own `TPSVRINIT()` routine that calls `TXOPEN()`. The default BEA Tuxedo system-supplied `TPSVRINIT()` calls `TPOpen()`. The same rule applies for `TPSVRDone()`: if the TX Interface is being used, then service routine writers must supply their own `TPSVRDone()` that calls `TXCLOSE()`.

Second, the TX Interface has two additional semantics not found in ATMI. These are chained and unchained transactions, and transaction characteristics.

Chained and Unchained Transactions

The TX Interface supports chained and unchained modes of transaction execution. By default, clients and service routines execute in the unchained mode; when an active transaction is completed, a new transaction does not begin until `TXBEGIN()` is called.

In the chained mode, a new transaction starts implicitly when the current transaction completes. That is, when `TXCOMMIT()` or `TXROLLBACK()` is called, the BEA Tuxedo system coordinates the completion of the current transaction and initiates a new transaction before returning control to the caller. (Certain failure conditions may prevent a new transaction from starting.)

Clients and service routines enable or disable the chained mode by calling `TXSETTRANCTL()`. Transitions between the chained and unchained mode affect the behavior of the next `TXCOMMIT()` or `TXROLLBACK()` call. The call to `TXSETTRANCTL()` does not put the caller into or take it out of transaction mode.

Since `TXCLOSE()` cannot be called when the caller is in transaction mode, a caller executing in chained mode must switch to unchained mode and complete the current transaction before calling `TXCLOSE()`.

Transaction Characteristics

A client or a service routine may call `TXINFORM()` to obtain the current values of their transaction characteristics and to determine whether they are executing in transaction mode.

The state of an application program includes several transaction characteristics. The caller specifies these by calling `TXSET*` functions. When a client or a service routine sets the value of a characteristic, it remains in effect until the caller specifies a different value. When the caller obtains the value of a characteristic via `TXINFORM()`, it does not change the value.

Timeouts

There are three types of timeouts in the BEA Tuxedo ATMI system: one is associated with the duration of a transaction from start to finish. A second is associated with the maximum length of time a blocking call will remain blocked before the caller regains control. The third is a service timeout and occurs when a call exceeds the number of seconds specified in the `SVCTIMEOUT` parameter in the `SERVICES` section of the configuration file.

The first kind of timeout is specified when a transaction is started with `TPBEGIN()` (see `TPBEGIN()` for details). The second kind of timeout can occur when using the BEA Tuxedo ATMI communication routines defined in `TPCALL()`. Callers of these routines typically block when awaiting a reply that has yet to arrive, although they can also block trying to send data (for example, if request queues are full). The maximum amount of time a caller remains blocked is determined by a BEA Tuxedo ATMI configuration file parameter. (See the `BLOCKTIME` parameter in [UBBCONFIG\(5\)](#) for details.)

Blocking timeouts are performed by default when the caller is not in transaction mode. When a client or server is in transaction mode, it is subject to the timeout value with which the transaction was started and is not subject to the blocking timeout value specified in the `UBBCONFIG` file.

When a transaction timeout occurs, replies to asynchronous requests made in transaction mode become invalid. That is, if a program is waiting for a particular asynchronous reply for a request sent in transaction mode and a transaction timeout occurs, the handle for that reply becomes invalid. Similarly, if a transaction timeout occurs, an event is generated on the connection handle associated with the transaction and that handle becomes invalid. On the other hand, if a blocking timeout occurs, the handle is still valid and the waiting program can reissue the call to await the reply.

The service timeout mechanism provides a way for the system to kill processes that may be frozen by some unknown or unexpected system error. When a service timeout occurs in a request/response service, the BEA Tuxedo system kills the server process that is executing the

frozen service and returns error code `TPESVCERR`. If a service timeout occurs in a conversational service, the `TPEV_SVCERR` event is returned.

If a transaction has timed out, the only valid communications before the transaction is aborted are calls to `TPACALL()` with `TPNOREPLY`, `TPNOTRAN`, and `TPNOBLOCK` set.

Dynamic Service Advertisements

By default, a server's services are advertised when it is booted and unadvertised when it is shut down. If a server needs to control the set of services that it offers at run time, it can do so by calling `TPADVERTISE()` and `TPUNADVERTISE()`. These routines affect only the services offered by the calling server unless that server belongs to a multiple server, single queue (MSSQ) set. Because all servers in an MSSQ set must offer the same set of services, these routines also affect the advertisements of all servers sharing the caller's MSSQ set.

Typed Records

In order to send data to another application program, the sending application program first places the data in a record. The ATMI interface supports the notion of a typed record. A typed record is really a pair of COBOL records. The data record is defined in static storage and contains application data to be passed to another application program. An auxiliary type record accompanies the data record and it identifies to the BEA Tuxedo system the interpretation and translation rules of the data record as it passes across heterogeneous machine boundaries. The auxiliary type record contains the data record's type, its optional subtype, and its optional length. Some record types require further specification via a subtype (for example, a particular record layout) and those of variable length require a length to be specified.

The application programmer may choose one of the six supported typed records. Note, the BEA Tuxedo system provides a method for adding user-specific typed records. For details, refer to the "Introduction to the C Language Application-Transaction Monitor Interface" in the *BEA Tuxedo ATMI C Function Reference*. `REC-TYPE` in `TPTYPE-REC` selects which record type the application wishes to send or receive. `SUB-TYPE` in `TPTYPE-REC` must also be given when further classification is required (for example, a view record). When sending, `LEN` in `TPTYPE-REC` indicates the number of bytes to be sent and when receiving the number of bytes to move into the user's record. The following are the supported `REC-TYPES`.

CARRAY

The `CARRAY` record type allows an arbitrary number of characters which may contain `LOW-VALUE` characters anywhere in the record. When sending data, `LEN` must contain the number of bytes to be transferred.

STRING

The `STRING` record type allows an arbitrary number of characters which may not contain `LOW-VALUE` characters within the record but may be at the end of the record. When sending data, `LEN` must contain the number of bytes to be transferred.

VIEW

This record type describes a COBOL record that was generated using the `viewc()` compiler. When using a `VIEW`, `SUB-TYPE` must contain the name of the view. When sending a `VIEW` type, `LEN` must contain the number of bytes to be transferred or set `NO-LENGTH` which will send the length of the view.

Two of the above record types have synonyms: `X_OCTET` is a synonym for `CARRAY`, and `X_COMMON` is a synonym for `VIEW`. `X_COMMON` supports a subset of the data types supported by `VIEW`: longs (`PIC S9(9) COMP-5`), shorts (`PIC S9(4) COMP-5`), and characters (`PIC X(n)`). `X_COMMON` should be used when both C and COBOL programs are communicating.

In all three cases, after a successful transfer, `LEN` contains the number of bytes transferred. When receiving data, `LEN` must contain the maximum number of bytes the data area contains. After a successful call, `LEN` contains the number of bytes moved into the data area. If the size of the incoming message is larger than the size specified in `LEN`, only `LEN` amount of data is moved into the data area; the remaining data is discarded.

Buffer Type Switch

The BEA Tuxedo system provides a method for adding user specific record types. For details, see the “Buffer Type Switch” section in [Introduction to the C Language Application-to-Transaction Monitor Interface](#).

Single or Multiple Application Context per Process

The BEA Tuxedo system allows client programs to create an association with one or more applications per process. If `TPINITIALIZE()` is called with the `TP-MULTI-CONTEXTS` setting of `CONTEXTS-FLAG` in `TPINFDEF-REC`, then multiple client contexts are allowed. If `TPINITIALIZE()` is called implicitly or the `CONTEXTS-FLAG` is not set to `TP-MULTI-CONTEXTS`, then only a single application association is allowed.

In single-context mode, if `TPINITIALIZE()` is called more than once (that is, if it is called after the client has already joined the application), no action is taken and success is returned.

In multi-context mode, each call to `TPINITIALIZE()` creates a new application association. The program can obtain a handle representing this application association by calling `TPGETCTXT()` and it can call `TPSETCTXT()` to set its context.

Once an application has chosen single-context mode, all calls to `TPINITIALIZE()` must specify single-context mode until all application associations are terminated. Similarly, once an application has chosen multi-context mode, all calls to `TPINITIALIZE()` must specify multi-context mode until all application associations are terminated.

Server programs can be associated with only a single application and cannot act as clients.

Note: In addition to allowing multiple application contexts per process, the BEA Tuxedo system allows multiple application threads per process. Multithreading is supported, however, only in the C language interface.

The following state table shows the transitions that may occur, within a client process, among the following states: the uninitialized state, the initialized in single-context mode state, and the initialized in multi-context mode state.

Table 2 Per-Process Context Modes

Function	States		
	Uninitialized S_0	Initialized Single-context Mode S_1	Initialized Multi-context Mode S_2
<code>TPINITIALIZE()</code> without <code>TP-MULTI-CONTEXTS</code>	S_1	S_1	S_2 (<i>error</i>)
<code>TPINITIALIZE()</code> with <code>TP-MULTI-CONTEXTS</code>	S_2	S_1 (<i>error</i>)	S_2
Implicit <code>TPINITIALIZE()</code>	S_1	S_1	S_2 (<i>error</i>)
<code>TPTERM()</code> - not last association			S_2
<code>TPTERM()</code> - last association		S_0	S_0
<code>TPTERM()</code> - no association	S_0		

Unsolicited Notification

There are two methods for sending messages to application clients outside the boundaries of the client/server interaction defined above. The first is the broadcast mechanism supported by `TPBROADCAST()`. This function allows application clients, servers, and administrators to broadcast typed record messages to a set of clients selected on the basis of the names assigned to them. The names assigned to clients are determined in part by the application (specifically, by the information passed in the *TPINFDEF-REC* data structure at `TPINITIALIZE` time) and in part by the system (based on the processor through which the client accesses the application).

The second is the notification of a particular client as identified from an earlier or current service request. Each service request contains a unique client identifier that identifies the originating client for the service request. Calls to the `TPCALL()` and `TPFORWAR()` functions from within a service routine do not change the originating client for that chain of service requests. Client identifiers can be saved and passed between application servers. The `TPNOTIFY()` function is used to notify clients identified in this manner.

COBOL Language ATMI Return Codes and Other Definitions

The following return code and setting definitions are used by the ATMI routines:

```
*
* TPSTATUS.cbl
*
05 TP-STATUS          PIC S9(9) COMP-5.
   88 TPOK             VALUE 0.
   88 TPEABORT         VALUE 1.
   88 TPEBADDESC       VALUE 2.
   88 TPEBLOCK        VALUE 3.
   88 TPEINVAL        VALUE 4.
   88 TPELIMIT        VALUE 5.
   88 TPENOENT        VALUE 6.
   88 TPEOS           VALUE 7.
   88 TPEPERM         VALUE 8.
   88 TPEPROTO        VALUE 9.
   88 TPESVCERR       VALUE 10.
   88 TPESVCFAIL      VALUE 11.
   88 TPESYSTEM       VALUE 12.
   88 TPETIME         VALUE 13.
   88 TPETRAN         VALUE 14.
   88 TPEGOTSIG       VALUE 15.
```

```

88 TPERMERR          VALUE 16.
88 TPEITYPE          VALUE 17.
88 TPEOTYPE          VALUE 18.
88 TPERELEASE        VALUE 19.
88 TPEHAZARD         VALUE 20.
88 TPEHEURISTIC      VALUE 21.
88 TPEEVENT          VALUE 22.
88 TPEMATCH          VALUE 23.
88 TPEDIAGNOSTIC     VALUE 24.
88 TPEMIB            VALUE 25.
88 TPEMAXVAL         VALUE 26.
05 TPEVENT           PIC S9(9) COMP-5.
88 TPEV-NOEVENT      VALUE 0.
88 TPEV-DISCONIMM    VALUE 1.
88 TPEV-SENDONLY     VALUE 2.
88 TPEV-SVCERR       VALUE 3.
88 TPEV-SVCFAIL      VALUE 4.
88 TPEV-SVCSUCC      VALUE 5.
05 TPSVCTIMOUT       PIC S9(9) COMP-5.
88 TPED-NOEVENT      VALUE 0.
88 TPEV-SVCTIMEOUT   VALUE 1.
88 TPEV-TERM         VALUE 2.
05 APPL-RETURN-CODE  PIC S9(9) COMP-5.

```

The *TPTYPE* COBOL structure is used whenever sending or receiving application data. *REC-TYPE* indicates the type of data record that is to be sent. *SUB-TYPE* indicates the name of the view if a *VIEW REC-TYPE* is specified. *LEN* indicates the amount of data to send and the amount received.

```

*
* TPTYPE.cbl
*
05 REC-TYPE          PIC X(8) .
88 X-OCTET           VALUE "X_OCTET" .
88 X-COMMON          VALUE "X_COMMON" .
05 SUB-TYPE          PIC X(16) .
05 LEN               PIC S9(9) COMP-5.
88 NO-LENGTH         VALUE 0.
05 TPTYPE-STATUS     PIC S9(9) COMP-5.

```

```

88 TPTYPEOK          VALUE 0.
88 TPTRUNCATE        VALUE 1.

```

The TPSVCDEF data structure is used by functions to pass settings to and from the BEA Tuxedo system:

```

*
* TPSVCDEF.cbl
*
05 COMM-HANDLE        PIC S9(9) COMP-5.
05 TPBLOCK-FLAG       PIC S9(9) COMP-5.
    88 TPBLOCK         VALUE 0.
    88 TPNOBLOCK       VALUE 1.
05 TPTRAN-FLAG        PIC S9(9) COMP-5.
    88 TPTRAN          VALUE 0.
    88 TPNOTRAN        VALUE 1.
05 TPREPLY-FLAG       PIC S9(9) COMP-5.
    88 TPREPLY         VALUE 0.
    88 TPNOREPLY       VALUE 1.
05 TPACK-FLAG         PIC S9(9) COMP-5 REDEFINES TPREPLY-FLAG.
    88 TPNOACK         VALUE 0.
    88 TPACK           VALUE 1.
05 TPTIME-FLAG        PIC S9(9) COMP-5.
    88 TPTIME          VALUE 0.
    88 TPNOTIME        VALUE 1.
05 TPSIGRSTRT-FLAG    PIC S9(9) COMP-5.
    88 TPNOSIGRSTRT    VALUE 0.
    88 TPSIGRSTRT      VALUE 1.
05 TPGETANY-FLAG      PIC S9(9) COMP-5.
    88 TPGETHANDLE     VALUE 0.
    88 TPGETANY        VALUE 1.
05 TPSENDRECV-FLAG    PIC S9(9) COMP-5.
    88 TSENDONLY       VALUE 0.
    88 TPRECVONLY      VALUE 1.
05 TPNOCHANGE-FLAG    PIC S9(9) COMP-5.
    88 TPCHANGE        VALUE 0.
    88 TPNOCHANGE      VALUE 1.
05 TPSERVICETYPE-FLAG PIC S9(9) COMP-5.
    88 TPREQRSP        VALUE IS 0.

```

```

      88 TPCONV                VALUE IS 1.
*
05 APPKEY                     PIC S9(9) COMP-5.
05 CLIENTID OCCURS 4 TIMES PIC S9(9) COMP-5.
05 SERVICE-NAME               PIC X(15).

```

The *TPINFDEF* data structure is used by `TPINITIALIZE()` to join the application:

```

*
* TPINFDEF.cbl
*
05 USERNAME                   PIC X(30).
05 CLTNAME                    PIC X(30).
05 PASSWD                     PIC X(30).
05 GRPNAME                    PIC X(30).
05 NOTIFICATION-FLAG          PIC S9(9) COMP-5.
    88 TPU-SIG                 VALUE 1.
    88 TPU-DIP                 VALUE 2.
    88 TPU-IGN                 VALUE 3.
05 ACCESS-FLAG                PIC S9(9) COMP-5.
    88 TPSA-FASTPATH           VALUE 1.
    88 TPSA-PROTECTED           VALUE 2.
05 CONTEXTS-FLAG              PIC S9(9) COMP-5.
    88 TP-SINGLE-CONTEXT         VALUE 0.
    88 TP-MULTI-CONTEXTS        VALUE 1.
05 DATALEN                   PIC S9(9) COMP-5.

```

The *TPCONTEXTDEF* data structure is used by `TPGETCTXT()` and `TPSETCTXT()` to manipulate program contexts:

```

*
* TPCONTEXTDEF.cbl
*
05 CONTEXT                    PIC S9(9) COMP-5.

```

The *TPQUEDEF* data structure is used to pass and retrieve information associated with enqueueing the message:

```

*
* TPQUEDEF.cbl

```

*

05 TPBLOCK-FLAG	PIC S9(9) COMP-5.
88 TPNOBLOCK	VALUE 0.
88 TPBLOCK	VALUE 1.
05 TPTRAN-FLAG	PIC S9(9) COMP-5.
88 TPNOTRAN	VALUE 0.
88 TPTRAN	VALUE 1.
05 TPTIME-FLAG	PIC S9(9) COMP-5.
88 TPNOTIME	VALUE 0.
88 TPTIME	VALUE 1.
05 TPSIGRSTRT-FLAG	PIC S9(9) COMP-5.
88 TPNOSIGRSTRT	VALUE 0.
88 TPSIGRSTRT	VALUE 1.
05 TPNOCHANGE-FLAG	PIC S9(9) COMP-5.
88 TPNOCHANGE	VALUE 0.
88 TPCHANGE	VALUE 1.
05 TPQUE-ORDER-FLAG	PIC S9(9) COMP-5.
88 TPQDEFAULT	VALUE 0.
88 TPQTOP	VALUE 1.
88 TPQBEFOREMSGID	VALUE 2.
05 TPQUE-TIME-FLAG	PIC S9(9) COMP-5.
88 TPQNOTIME	VALUE 0.
88 TPQTIME-ABS	VALUE 1.
88 TPQTIME-REL	VALUE 2.
05 TPQUE-PRIORITY-FLAG	PIC S9(9) COMP-5.
88 TPQNOPRIORITY	VALUE 0.
88 TPQPRIORITY	VALUE 1.
05 TPQUE-CORRID-FLAG	PIC S9(9) COMP-5.
88 TPQNOCORRID	VALUE 0.
88 TPQCORRID	VALUE 1.
05 TPQUE-REPLYQ-FLAG	PIC S9(9) COMP-5.
88 TPQNOREPLYQ	VALUE 0.
88 TPQREPLYQ	VALUE 1.
05 TPQUE-FAILQ-FLAG	PIC S9(9) COMP-5.
88 TPQNOFAILUREQ	VALUE 0.
88 TPQFAILUREQ	VALUE 1.
05 TPQUE-MSGID-FLAG	PIC S9(9) COMP-5.
88 TPQNOMSGID	VALUE 0.

88 TPQMSGID	VALUE 1.
05 TPQUE-GETBY-FLAG	PIC S9(9) COMP-5.
88 TPQGETNEXT	VALUE 0.
88 TPQGETBYMSGIDOLD	VALUE 1.
88 TPQGETBYCORRIDOLD	VALUE 2.
88 TPQGETBYMSGID	VALUE 3.
88 TPQGETBYCORRID	VALUE 4.
05 TPQUE-WAIT-FLAG	PIC S9(9) COMP-5.
88 TPQNOWAIT	VALUE 0.
88 TPQWAIT	VALUE 1.
05 TPQUE-DELIVERY-FLAG	PIC S9(9) COMP-5.
88 TPQNODELIVERYQOS	VALUE 0.
88 TPQDELIVERYQOS	VALUE 1.
05 TPQUEQOS-DELIVERY-FLAG	PIC S9(9) COMP-5.
88 TPQQOSDELIVERYDEFAULTPERSIST	VALUE 0.
88 TPQQOSDELIVERYPERSISTENT	VALUE 1.
88 TPQQOSDELIVERYNONPERSISTENT	VALUE 2.
05 TPQUE-REPLY-FLAG	PIC S9(9) COMP-5.
88 TPQNOREPLYQOS	VALUE 0.
88 TPQREPLYQOS	VALUE 1.
05 TPQUEQOS-REPLY-FLAG	PIC S9(9) COMP-5.
88 TPQQOSREPLYDEFAULTPERSIST	VALUE 0.
88 TPQQOSREPLYPERSISTENT	VALUE 1.
88 TPQQOSREPLYNONPERSISTENT	VALUE 2.
05 TPQUE-EXPTIME-FLAG	PIC S9(9) COMP-5.
88 TPQNOEXPTIME	VALUE 0.
88 TPQEXPTIME-ABS	VALUE 1.
88 TPQEXPTIME-REL	VALUE 2.
88 TPQEXPTIME-NONE	VALUE 3.
05 TPQUE-PEEK-FLAG	PIC S9(9) COMP-5.
88 TPQNOPEEK	VALUE 0.
88 TPQPEEK	VALUE 1.
05 DIAGNOSTIC	PIC S9(9) COMP-5.
88 QMEINVAL	VALUE -1.
88 QMEBADRMID	VALUE -2.
88 QMENOTOPEN	VALUE -3.
88 QMETRAN	VALUE -4.
88 QMEBADMSGID	VALUE -5.

88 QMESYSTEM	VALUE -6.
88 QMEOS	VALUE -7.
88 QMEABORTED	VALUE -8.
88 QMEPROTO	VALUE -9.
88 QMEBADQUEUE	VALUE -10.
88 QMENOMSG	VALUE -11.
88 QMEINUSE	VALUE -12.
88 QMENOSPACE	VALUE -13.
88 QMERELASE	VALUE -14.
88 QMEINVHANDLE	VALUE -15.
88 QMESHARE	VALUE -16.
05 DEQ-TIME	PIC S9(9) COMP-5.
05 EXP-TIME	PIC S9(9) COMP-5.
05 PRIORITY	PIC S9(9) COMP-5.
05 MSGID	PIC X(32).
05 CORRID	PIC X(32).
05 QNAME	PIC X(15).
05 QSPACE-NAME	PIC X(15).
05 REPLYQUEUE	PIC X(15).
05 FAILUREQUEUE	PIC X(15).
05 CLIENTID OCCURS4 TIMES	PIC S9(9) COMP-5.
05 APPL-RETURN-CODE	PIC S9(9) COMP-5.
05 APPKEY	PIC S9(9) COMP-5.

The *TPSVCRET* data structure is used by `TPRETURN()` to indicate the status of the transaction:

```
*
* TPSVCRET.cbl
*
05 TP-RETURN-VAL          PIC S9(9) COMP-5.
    88 TPSUCCESS          VALUE 0.
    88 TPFAIL              VALUE 1.
    88 TPEXIT              VALUE 2.
05 APPL-CODE              PIC S9(9) COMP-5.
```

The *TPTRXDEF* data structure is used by `TPBEGIN()` to set transaction timeouts, and by `TPSUSPEND()` and `TPRESUME()` to get and set, respectively, transaction identifiers:

```
*
* TPTRXDEF.cbl
```



```

*
05 T-OUT                                PIC S9(9) COMP-5 VALUE IS 0.
05 TRANID                                OCCURS 6 TIMES    PIC S9(9) COMP-5.

```

The *TPCMTDEF* data structure is used by *TPSCMT()* to set the commit level characteristics:

```

*
* TPCMTDEF.cbl
*
05 CMT-FLAG                                PIC S9(9) COMP-5.
   88 TP-CMT-LOGGED                        VALUE 1.
   88 TP-CMT-COMPLETE                      VALUE 2.
05 PREV-CMT-FLAG                          PIC S9(9) COMP-5.
   88 PREV-TP-CMT-LOGGED                  VALUE 1.
   88 PREV-TP-CMT-COMPLETE                VALUE 2.

```

The *TPAUTDEF* data structure is used by *TPCHKAUTH()* to check if authentication is required:

```

* TPAUTDEF.cbl
*
05 AUTH-FLAG                                PIC S9(9) COMP-5.
   88 TPNOAUTH                            VALUE 0.
   88 TPSYSAUTH                           VALUE 1.
   88 TPAPPAUTH                           VALUE 2.

```

The *TPPRIDEF* data structure is used by *TPSPRIO()* and *TPGPRIOR()* to manipulate message priorities:

```

*
* TPPRIDEF.cbl
*
05 PRIORITY                                PIC S9(9) COMP-5.
05 PRIO-FLAG                              PIC S9(9) COMP-5.
   88 TPABSOLUTE                          VALUE 0.
   88 TPRELATIVE                          VALUE 1.

```

The *TPTRXLEV* data structure is used by *TPGETLEV()* to receive transaction level setting:

```

*
* TPTRXLEV.cbl
*
05 TPTRXLEV-FLAG                          PIC S9(9) COMP-5.

```

```

88 TP-NOT-IN-TRAN      VALUE 0.
88 TP-IN-TRAN          VALUE 1.

```

The *TPBCTDEF* data structure is used by `TPNOTIFY()` and `TPBROADCAST()` to send notifications:

```

*
* TPBCTDEF.cbl
*
05 TPBLOCK-FLAG        PIC S9(9) COMP-5.
   88 TPBLOCK           VALUE 0.
   88 TPNOBLOCK         VALUE 1.
05 TPTIME-FLAG         PIC S9(9) COMP-5.
   88 TPTIME            VALUE 0.
   88 TPNOTIME          VALUE 1.
05 TPSIGRSTRT-FLAG     PIC S9(9) COMP-5.
   88 TPNOSIGRSTRT      VALUE 0.
   88 TPSIGRSTRT        VALUE 1.
05 LMID                PIC X(30).
05 USERNAME            PIC X(30).
05 CLTNAME             PIC X(30).

```

The *FML-INFO* data structure is used by `FINIT()`, `FVSTOF()`, and `FVFTOS()` to deal with FML buffers:

```

*
* FMLINFO.cbl
*
05 FML-STATUS          PIC S9(9) COMP-5.
   88 FOK               VALUE 0.
   88 FALIGNERR         VALUE 1.
   88 FNOTFLD           VALUE 2.
   88 FNOSPACE          VALUE 3.
   88 FNOTPRES          VALUE 4.
   88 FBADFLD           VALUE 5.
   88 FTYPERR           VALUE 6.
   88 FEUNIX            VALUE 7.
   88 FBADNAME          VALUE 8.
   88 FMALLOC           VALUE 9.
   88 FSYNTAX           VALUE 10.
   88 FFTOPEN           VALUE 11.

```

```

88 FFTSYNTAX          VALUE 12.
88 FEINVAL            VALUE 13.
88 FBADTBL            VALUE 14.
88 FBADVIEW           VALUE 15.
88 FVFSYNTAX          VALUE 16.
88 FVFOPEN            VALUE 17.
88 FBADACM            VALUE 18.
88 FNOCNAME           VALUE 19.
88 FEBADOP            VALUE 20.
*
05 FML-LENGTH          PIC S9(9) COMP-5.
*
05 FML-MODE            PIC S9(9) COMP-5.
88 FUPDATE            VALUE 1.
88 FCONCAT            VALUE 2.
88 FJOIN              VALUE 3.
88 FOJOIN             VALUE 4.
*
05 VIEWNAME            PIC X(33).

```

The *TPEVTDEF* data structure is used by *TPPOST()*, *TPSUBSCRIBE()*, and *TPUNSUBSCRIBE()* to handle event postings and subscriptions:

```

*
* TPEVTDEF.cbl
*
05 TPBLOCK-FLAG        PIC S9(9) COMP-5.
88 TPBLOCK            VALUE 0.
88 TPNOBLOCK          VALUE 1.
05 TPTRAN-FLAG         PIC S9(9) COMP-5.
88 TPTRAN             VALUE 0.
88 TPNOTRAN           VALUE 1.
05 TPREPLY-FLAG        PIC S9(9) COMP-5.
88 TPREPLY            VALUE 0.
88 TPNOREPLY          VALUE 1.
05 TPTIME-FLAG         PIC S9(9) COMP-5.
88 TPTIME             VALUE 0.
88 TPNOTIME           VALUE 1.
05 TPSIGRSTRT-FLAG     PIC S9(9) COMP-5.

```

```

      88 TPNOSIGRSTRT          VALUE 0.
      88 TPSIGRSTRT           VALUE 1.
05  TPEV-METHOD-FLAG        PIC S9(9) COMP-5.
      88 TPEVNOTIFY           VALUE 0.
      88 TPEVSERVICE         VALUE 1.
      88 TPEVQUEUE            VALUE 2.
05  TPEV-PERSIST-FLAG        PIC S9(9) COMP-5.
      88 TPEVNOPERSIST        VALUE 0.
      88 TPEVPERSIST          VALUE 1.
05  TPEV-TRAN-FLAG           PIC S9(9) COMP-5.
      88 TPEVNOTTRAN          VALUE 0.
      88 TPEVTRAN             VALUE 1.
*
05  EVENT-COUNT              PIC S9(9) COMP-5.
05  SUBSCRIPTION-HANDLE      PIC S9(9) COMP-5.
05  NAME-1                   PIC X(31).
05  NAME-2                   PIC X(31).
05  EVENT-NAME               PIC X(31).
05  EVENT-EXPR               PIC X(255).
05  EVENT-FILTER             PIC X(255).

```

The *TPKEYDEF* data structure is used by *TPKEYCLOSE()*, *TPKEYGETINFO()*, *TPKEYOPEN()*, and *TPKEYSETINFO()* to manage public-private keys for performing message-based digital signature and encryption operations:

```

*
*      TPKEYDEF.cbl
*
      05      KEY-HANDLE          PIC S9(9) COMP-5.
      05      PRINCIPAL-NAME      PIC X(512).
      05      LOCATION           PIC X(1024).
      05      IDENTITY-PROOF      PIC X(2048).
      05      PROOF-LEN           PIC S9(9) COMP-5.
      05      CRYPTO-PROVIDER     PIC X(128).
      05      SIGNATURE-FLAG      PIC S9(9) COMP-5.
           88 TPKEY-NOSIGNATURE    VALUE 0.
           88 TPKEY-SIGNATURE      VALUE 1.
      05      DECRYPT-FLAG         PIC S9(9) COMP-5.
           88 TPKEY-NODECRYPT       VALUE 0.

```

```

            88 TPKEY-DECRYPT          VALUE 1.
05    ENCRYPT-FLAG                    PIC S9(9) COMP-5.
            88 TPKEY-NOENCRYPT        VALUE 0.
            88 TPKEY-ENCRYPT          VALUE 1.
05    AUTOSIGN-FLAG                  PIC S9(9) COMP-5.
            88 TPKEY-NOAUTOSIGN      VALUE 0.
            88 TPKEY-AUTOSIGN        VALUE 1.
05    AUTOENCRYPT-FLAG                PIC S9(9) COMP-5.
            88 TPKEY-NOAUTOENCRYPT    VALUE 0.
            88 TPKEY-AUTOENCRYPT      VALUE 1.
05    ATTRIBUTE-NAME                 PIC X(64).
05    ATTRIBUTE-VALUE-LEN            PIC S9(9) COMP-5.

```

COBOL Language TX Return Codes and Other Definitions

The following return code and setting definitions are used by the TX routines:

```

*
* TXSTATUS.cbl
*
05 TX-STATUS          PIC S9(9) COMP-5.
    88 TX-NOT-SUPPORTED    VALUE 1.
*   Normal execution
    88 TX-OK                VALUE 0.
*   Normal execution
    88 TX-OUTSIDE           VALUE -1.
*   Application is in an RM local transaction
    88 TX-ROLLBACK          VALUE -2.
*   Transaction was rolled back
    88 TX-MIXED             VALUE -3.
*   Transaction was partially committed and partially
*   rolled back
    88 TX-HAZARD            VALUE -4.
*   Transaction may have been partially committed and
*   partially rolled back
    88 TX-PROTOCOL-ERROR    VALUE -5.
*   Routine invoked in an improper context
    88 TX-ERROR             VALUE -6.
*   Transient error

```

```

88 TX-FAIL                VALUE -7.
*   Fatal error
88 TX-EINVAL              VALUE -8.
*   Invalid arguments were given
88 TX-COMMITTED           VALUE -9.
*   The transaction was heuristically committed
88 TX-NO-BEGIN            VALUE -100.
*   Transaction committed plus new transaction could not
*   be started
88 TX-ROLLBACK-NO-BEGIN   VALUE -102.
*   Transaction rollback plus new transaction could not
*   be started
88 TX-MIXED-NO-BEGIN      VALUE -103.
*   Mixed plus new transaction could not be started
88 TX-HAZARD-NO-BEGIN     VALUE -104.
*   Hazard plus new transaction could not be started
88 TX-COMMITTED-NO-BEGIN  VALUE -109.
*   Heuristically committed plus transaction could not
*   be started

```

The *TXINFDEF* record defines a data structure where the result of the *TXINFORM()* call will be stored:

```

*
* TXINFDEF.cbl
*
05 XID-REC.
*   XID record
10 FORMAT-ID              PIC S9(9) COMP-5.
*   A value of -1 in FORMAT-ID means that the XID is NULL
10 GTRID-LENGTH           PIC S9(9) COMP-5.
10 BRANCH-LENGTH          PIC S9(9) COMP-5.
10 XID-DATA               PIC X(128).
05 TRANSACTION-MODE       PIC S9(9) COMP-5.
*   Transaction mode settings
88 TX-NOT-IN-TRAN          VALUE 0.
88 TX-IN-TRAN             VALUE 1.
05 COMMIT-RETURN          PIC S9(9) COMP-5.
*   Commit_return settings

```

```

      88 TX-COMMIT-COMPLETED          VALUE 0.
      88 TX-COMMIT-DECISION-LOGGED     VALUE 1.
05 TRANSACTION-CONTROL  PIC S9(9) COMP-5.
*   Transaction_control settings
      88 TX-UNCHAINED                  VALUE 0.
      88 TX-CHAINED                    VALUE 1.
05 TRANSACTION-TIMEOUT  PIC S9(9) COMP-5.
*   Transaction_timeout value
      88 NO-TIMEOUT                    VALUE 0.
05 TRANSACTION-STATE    PIC S9(9) COMP-5.
*   Transaction_state information
      88 TX-ACTIVE                      VALUE 0.
      88 TX-TIMEOUT-ROLLBACK-ONLY      VALUE 1.
      88 TX-ROLLBACK-ONLY              VALUE 2.

```

ATMI State Transitions

The BEA Tuxedo system keeps track of the state for each program and verifies that legal state transitions occur for the various function calls and options. The state information includes the program type (request/response server, conversational server, or client), the initialization state (uninitialized or initialized), the resource management state (closed or open), the transaction state of the program, and the state of all asynchronous request/response and connection handles. When an illegal state transition is attempted, the called function fails, setting *TPSTATUS-REC* to *TPEPROTO()*. The legal states and transitions for this information are described in the following tables.

The table below indicates which functions may be called by request/response servers, conversational servers, and clients. Note that *TPSVRINIT()* and *TPSVRDONE()* are not included in this table because they are not called by applications (that is, they are application-supplied functions that are invoked by the BEA Tuxedo system).

Table 3 Available Functions

Function	Process Type		
	Request/Response Server	Conversational Server	Client
TPABORT()	Y	Y	Y
TPACALL()	Y	Y	Y

Table 3 Available Functions

Function	Process Type		
	Request/Response Server	Conversational Server	Client
TPADVERTISE ()	Y	Y	N
TPBEGIN ()	Y	Y	Y
TPBROADCAST ()	Y	Y	Y
TPCALL ()	Y	Y	Y
TPCANCEL ()	Y	Y	Y
TPCHKAUTH ()	Y	Y	Y
TPCHKUNSOL ()	N	N	Y
TPCLOSE ()	Y	Y	Y
TPCOMMIT ()	Y	Y	Y
TPCONNECT ()	Y	Y	Y
TPDEQUEUE ()	Y	Y	Y
TPDISCON ()	Y	Y	Y
TPENQUEUE ()	Y	Y	Y
TPFORWAR ()	Y	N	N
TPGBLKTIME ()	Y	Y	Y
TPGETCTXT ()	Y	Y	Y
TPGETLEV ()	Y	Y	Y
TPGETRPLY ()	Y	Y	Y
TPGPRIOR ()	Y	Y	Y
TPINITIALIZE ()	N	N	Y
TPNOTIFY ()	Y	Y	Y

Table 3 Available Functions

Function	Process Type		
	Request/Response Server	Conversational Server	Client
TPOPEN ()	Y	Y	Y
TPPOST ()	Y	Y	Y
TPRECV ()	Y	Y	Y
TPRESUME ()	Y	Y	Y
TPRETURN ()	Y	Y	N
TPSBLKTIME ()	Y	Y	Y
TPSCMT ()	Y	Y	Y
TPSEND ()	Y	Y	Y
TPSETCTXT ()	N	N	Y
TPSETUNSOL ()	N	N	Y
TPSPRIO ()	Y	Y	Y
TPSUBSCRIBE ()	Y	Y	Y
TPSUSPEND ()	Y	Y	Y
TPTERM ()	N	N	Y
TPUNADVERTISE ()	Y	Y	N
TPUNSUBSCRIBE ()	Y	Y	Y

The remaining state tables are for both clients and servers, unless otherwise noted. Keep in mind that because some functions cannot be called by both clients and servers (for example, `TPINITIALIZE ()`), certain state transitions shown below may not be possible for both program types. The above table should be consulted to determine whether the program in question is allowed to call a particular function.

The following state table indicates whether or not a client program has been initialized and registered with the transaction manager. Note that this table assumes the use of `TPINITIALIZE()`, which is optional in single-context mode. That is, a single-context client may implicitly join an application by issuing one of many ATMI functions (for example, `TPACALL()` or `TPCALL()`). A client must use `TPINITIALIZE()` when one of the following is true:

- Application authentication is required. (See `TPINITIALIZE()` and the description of the `SECURITY` keyword in [UBBCONFIG\(5\)](#) for details.)
- The client wants to access an XA-compliant resource manager directly. (See [TPINITIALIZE\(3cb1\)](#) for details.)
- The client wants to create multiple application associations.

A server is placed in the initialized state by the BEA Tuxedo dispatcher before its `TPSVRINIT()` function is invoked, and it is placed in the uninitialized state by the BEA Tuxedo dispatcher after its `TPSVRDONE()` function has returned. Note that in all of the state tables shown below, an error return from a function causes the program to remain in the same state, unless otherwise noted.

Table 4 Initialization States

Function	States	
	Uninitialized I_0	Initialized I_1
<code>TPCHKAUTH()</code>	I_0	I_1
<code>TPGETCTXT()</code>	I_0	I_1
<code>TPINITIALIZE()</code>	I_1	I_1
<code>TPSETCTXT()</code> set to a non-NULL context	I_1	I_1
<code>TPSETCTXT()</code> with <code>TPNULLCONTEXT</code> set	I_0	I_0
<code>TPSETUNSOL()</code>	I_0	I_1

Table 4 Initialization States

Function	States	
	Uninitialized I_0	Initialized I_1
TPTERM ()	I_0	I_0
All other ATMI functions		I_1

The remaining state tables assume a precondition of state I (regardless of whether a process arrived in this state via TPINITIALIZE (), TPSETCTXT (), or the BEA Tuxedo service dispatcher).

The following table indicates the state of a client or server with respect to whether or not a resource manager associated with the process has been initialized:

Table 5 Resource Management States

Function	States	
	Closed R_0	Open R_1
TPOPEN ()	R_1	R_1
TPCLOSE ()	R_0	R_0
TPBEGIN ()		R_1
TPCOMMIT ()		R_1
TPABORT ()		R_1
TPSUSPEND ()		R_1
TPRESUME ()		R_1

Table 5 Resource Management States

Function	States	
	Closed R ₀	Open R ₁
TPSVCSTART () with TPTRAN		R ₁
All other ATMI functions	R ₀	R ₁

The following state table indicates the state of a process with respect to whether or not the process is associated with a transaction. For servers, transitions to states T₁ and T₂ assume a precondition of state R₁ (for example, TPOPEN () has been called with no subsequent call to TPCLOSE () or TPTERM ()).

Table 6 Transaction State of Application Association

Function	State		
	Not in Transaction T ₀	Initiator T ₁	Participant T ₂
TPBEGIN ()			
TPABORT ()		T ₀	
TPCOMMIT ()		T ₀	
SPSUSPEND ()		T ₀	
TPRESUME ()		T ₀	
TPSVCSTART () with TPTRAN	T ₂		
TPSVCSTART () (not in transaction mode)	T ₀		
TPRETURN ()	T ₀		T ₀
TPFORWAR ()	T ₀		T ₀

Table 6 Transaction State of Application Association

Function	State		
	Not in Transaction T_0	Initiator T_1	Participant T_2
TPCLOSE ()	R_0		
TPTERM ()	I_0	T_0	
All other ATMI functions	T_0	T_1	T_2

The following state table indicates the state of a single request handle returned by TPACALL () :

Table 7 Asynchronous Request Descriptor States

Function	States	
	No Descriptor A_0	Valid Descriptor A_1
TPACALL ()	A_1	
TPGETRPLY ()		A_0
TPCANCEL ()		A_0^a
TPABORT ()	A_0	A_0^b
TPCOMMIT ()	A_0	A_0^b
TPSUSPEND ()	A_0	A^c
TPRETURN ()	A_0	A_0
TPFORWAR ()	A_0	A_0
TPTERM ()	I_0	I_0
All other ATMI functions	A_0	A_1

Note: ^a This state change occurs only if the descriptor is not associated with the caller's transaction.

^b This state change occurs only if the descriptor is associated with the caller's transaction.

^c If the descriptor is associated with the caller's transaction, then `TPSUSPEND()` returns a protocol error.

The following state table indicates the state of a connection descriptor returned by `TPCONNECT()` or provided by a service invocation in the `TPSVCINFO` structure. For primitives that do not take a connection descriptor, the state changes apply to all connection descriptors, unless otherwise noted.

The states are as follows:

`C0` - No handle

`C1` - `TPCONNECT` handle send-only

`C2` - `TPCONNECT` handle receive-only

`C3` - `TPSVCDEF` handle send-only

`C4` - `TPSVCDEF` handle receive-only

Table 8 Connection Request Handle States

Function/Event	States				
	<code>C₀</code>	<code>C₁</code>	<code>C₂</code>	<code>C₃</code>	<code>C₄</code>
<code>TPCONNECT()</code> with <code>TPSENDONLY</code>	<code>C₁^a</code>				
<code>TPCONNECT()</code> with <code>TPRECVONLY</code>	<code>C₂^a</code>				
<code>TPSVCSTART()</code> with flag <code>TPSENDONLY</code>	<code>C₃^b</code>				
<code>TPSVCSTART()</code> with flag <code>TPRECVONLY</code>	<code>C₄^b</code>				
<code>TPRECV()/no event</code>			<code>C₂</code>		<code>C₄</code>
<code>TPRECV()/TPEV_SENDOONLY</code>		<code>C₁</code>			<code>C₃</code>
<code>TPRECV()/TPEV_DISCONIMM</code>			<code>C₀</code>		<code>C₀</code>
<code>TPRECV()/TPEV_SVCERR</code>			<code>C₀</code>		

Table 8 Connection Request Handle States

Function/Event	States				
	C ₀	C ₁	C ₂	C ₃	C ₄
TPRECV () /TPEV_SVCFAIL			C ₀		
TPRECV () /TPEV_SVCSUCC			C ₀		
TPSEND () /no event		C ₁		C ₃	
TPSEND () with flag TPRECVONLY		C ₂		C ₄	
TPSEND () /TPEV_DISCONIMM		C ₀		C ₀	
TPSEND () /TPEV_SVCERR		C ₀			
TPSEND () /TPEV_SVCFAIL		C ₀			
TPTERM() (client only)	C ₀	C ₀			
TPCOMMIT() (originator only)	C ₀	C ₀ ^c	C ₀ ^c		
TPSUSPEND () (originator only)	C ₀	C ₀ ^d	C ₀ ^d		
TPABORT () (originator only)	C ₀	C ₀ ^c	C ₀ ^c		
TPDISCON ()		C ₀	C ₀		
TPRETURN () (CONV server)		C ₀	C ₀	C ₀	C ₀
TPFORWAR () (CONV server)		C ₀	C ₀	C ₀	C ₀
All other ATMI functions	C ₀	C ₁	C ₂	C ₃	C ₄

Note: ^a If the program is in transaction mode and TPNOTRAN is not specified, the connection is in transaction mode.

^b If the TPTRAN flag is set, the connection is in transaction mode.

^c If the connection is not in transaction mode, no state change.

^d If the connection is in transaction mode, then TPSUSPEND () returns a protocol error.

TX State Transitions

BEA Tuxedo ensures that a process calls the TX functions in a legal sequence. When an illegal state transition is attempted (that is, a call from a state with a blank transition entry), the called function returns `TX_PROTOCOL_ERROR`. The legal states and transitions for the TX functions are shown in the table below. Calls that return failure do not make state transitions, except where described by specific state table entries. Any BEA Tuxedo client or server is allowed to use the TX functions.

The states are defined below:

- S₀

No RMs have been opened or initialized. A process cannot start a global transaction until it has successfully called `TXOPEN()`.
- S₁

A process has opened its RM but is not in a transaction. Its `transaction_control` characteristic is `TX-UNCHAINED`.
- S₂

A process has opened its RM but is not in a transaction. Its `transaction_control` characteristic is `TX-CHAINED`.
- S₃

A process has opened its RM and is in a transaction. Its `transaction_control` characteristic is `TX-UNCHAINED`.
- S₄

A process has opened its RM and is in a transaction. Its `transaction_control` characteristic is `TX-CHAINED`.

Table 9 TX State Transitions

Function	States				
	S ₀	S ₁	S ₂	S ₃	S ₄
TXBEGIN()		S ₃	S ₄		
TXCLOSE()	S ₀	S ₀	S ₀		
TXCOMMIT() -> TX_SET1				S ₁	S ₄

Table 9 TX State Transitions

Function	States				
	S ₀	S ₁	S ₂	S ₃	S ₄
TXCOMMIT() -> TX_SET2					S ₂
TXINFORM()		S ₁	S ₂	S ₃	S ₄
TXOPEN()	S ₁	S ₁	S ₂	S ₃	S ₄
TXROLLBACK() -> TX_SET1				S ₁	S ₄
TXROLLBACK() -> TX_SET2					S ₂
TXSETCOMMITRET()		S ₁	S ₂	S ₃	S ₄
TXSETTRANCTL() control = TX-CHAINED		S ₂	S ₂	S ₄	S ₄
TXSETTRANCTL() control = TX-UNCHAINED		S ₁	S ₁	S ₃	S ₃
TXSETTIMEOUT()		S ₁	S ₂	S ₃	S ₄

- TX_SET1 denotes any of the following: TX_OK, TX_ROLLBACK, TX_MIXED, TX_HAZARD, or TX_COMMITTED. TX_ROLLBACK is not returned by tx_rollback() and TX_COMMITTED is not returned by tx_commit().
- TX_SET2 denotes any of the following: TX_NO_BEGIN, TX_ROLLBACK_NO_BEGIN, TX_MIXED_NO_BEGIN, TX_HAZARD_NO_BEGIN, or TX_COMMITTED_NO_BEGIN. TX_ROLLBACK_NO_BEGIN is not returned by tx_rollback() and TX_COMMITTED_NO_BEGIN is not returned by tx_commit().
- If TX_FAIL is returned on any call, the application process is in an undefined state with respect to the above table.
- When tx_info() returns either TX_ROLLBACK_ONLY or TX_TIMEOUT_ROLLBACK_ONLY in the transaction state information, the transaction is marked rollback-only and is rolled back, regardless of whether the application program calls tx_commit() or tx_rollback().

See Also

`buffer(3c)`, `TPINITIALIZE(3cbl)`, `TPADVERTISE(3cbl)`, `TPBEGIN(3cbl)`,
`TPCALL(3cbl)`, `TPCONNECT(3cbl)`, `TPGETCTXT(3cbl)`, `TPKEYCLOSE(3cbl)`,
`TPKEYGETINFO(3cbl)`, `TPKEYOPEN(3cbl)`, `TPKEYSETINFO(3cbl)`, `TPOPEN(3cbl)`,
`TPSETCTXT(3cbl)`, `TPSVCSTART(3cbl)`, `tuxtypes(5)`, `typesw(5)`

FINIT, FINIT32(3cbl)

Name

`FINIT()`, `FINIT32()` - initialize fielded buffer

Synopsis

```
01 FML-BUFFER.  
   05 FML-ALIGN      PIC S9(9) USAGE IS COMP.  
   05 FML-DATA       PIC X(applen).  
  
01 FML-REC  
   COPY FMLINFO.  
  
CALL "FINIT" USING FML-BUFFER FML-REC.  
  
CALL "FINIT32" USING FML-BUFFER FML-REC.
```

Description

`FINIT()` can be called to initialize a fielded buffer. *FML-BUFFER* is the record to be used for the fielded buffer; it should be aligned on a 4-byte boundary to work with both FML16 and FML32. This can be accomplished by defining two record elements as shown in the synopsis above.

FML-LENGTH IN *FML-REC* is the length of the record. The internal structure is set up for a fielded buffer with no fields; the application program should not interpret the record, other than to pass it to `FINIT()`, `FVFTOS()`, or `FVSTOF()`, or an ATMI call that takes a typed record (in this case, the type is “FML” and there is no subtype).

`FINIT32()` is used with 32-bit FML.

Return Values

Upon successful completion, `FINIT()` sets *FML-STATUS* in *FML-REC* to FOK.

On error, *FML-STATUS* is set to a non-zero value.

Errors

Under the following conditions, `FINIT()` fails and sets `FML-STATUS` in `FML-REC` to:

[`FALIGNERR`]

“fielded buffer not aligned”

The buffer does not begin on the proper boundary.

[`FNOSPACE`]

“no space in fielded buffer”

The buffer size specified is too small for a fielded buffer.

Example

The correct way to reinitialize a buffer to have no fields is: `Finit(frfr, (FLDLLEN)Fsizeof(fbfr));`

See Also

[Introduction to FML Functions](#)

FVFTOS, FVFTOS32(3cbl)

Name

`FVFTOS()`, `FVFTOS32()` - copy from fielded buffer to COBOL structure

Synopsis

```
01 DATA-REC.
COPY User data.

01 FML-BUFFER.
05 FML-ALIGN      PIC S9(9) USAGE IS COMP.
05 FML-DATA       PIC X(applen).

01 FML-REC COPY FMLINFO.

CALL "FVFTOS" USING FML-BUFFER DATA-REC FML-REC.

CALL "FVFTOS32" USING FML-BUFFER DATA-REC FML-REC.
```

Description

The `FVFTOS()` function transfers data from a fielded buffer to a COBOL record. `FML-BUFFER` is a pointer to a fielded buffer initialized with `FINIT()`. `DATA-REC` is a pointer to a C structure. `VIEWNAME` IN `FML-REC` is the name of the view describing the COBOL record.

Fields are copied from the fielded buffer into the structure based on the element descriptions in `VIEWNAME`. If a field in the fielded buffer has no corresponding element in the COBOL record, it is ignored. If an element specified in the COBOL record has no corresponding field in the fielded buffer, a NULL value is copied into the element. The NULL value used is definable for each element in the view description.

To store multiple occurrences in the COBOL record, the record element should be defined with `OCCURS`. If the buffer has fewer occurrences of the field than there are occurrences of the element, the extra element slots are assigned NULL values. On the other hand, if the buffer has more occurrences of the field than there are occurrences of the element, the surplus occurrences are ignored.

`FVFTOS32()` is used for views defined with `view32()` typed buffers for larger views with more fields.

Return Values

Upon successful completion, `FVFTOS32()` sets `FML-STATUS` IN `FML-REC` to `FOK`.

On error, `FML-STATUS` is set to a non-zero value.

Errors

Under the following conditions, `FVFTOS()` fails and sets `FML-STATUS` to:

[`FALIGNERR`]

“fielded buffer not aligned”

The buffer does not begin on the proper boundary.

[`FNOTFLD`]

“buffer not fielded”

The buffer is not a fielded buffer or has not been initialized by `FINIT()`.

[`FEINVAL`]

“invalid argument to function”

One of the arguments to the function invoked was invalid.

[FBADACM]

“ACM contains negative value”

An Associated Count Member should not be a negative value while transferring data from a COBOL record to a fielded buffer.

[FBADVIEW]

“cannot find or get view”

The view description *VIEWNAME* was not found in the files specified by *VIEWDIR* or *VIEWFILES*.

See Also

[Introduction to FML Functions](#), [viewfile\(5\)](#)

FVSTOF(3cbl)

Name

FVSTOF() - copy from C structure to fielded buffer

Synopsis

```
01 DATA-REC.
   COPY User data.

01 FML-BUFFER.
   05 FML-ALIGN      PIC S9(9) USAGE IS COMP.
   05 FML-DATA       PIC X(applen) .

01 FML-REC
   COPY FMLINFO.

CALL "FVSTOF" USING FML-BUFFER DATA-REC FML-REC.

CALL "FVSTOF32" USING FML-BUFFER DATA-REC FML-REC.
```

Description

FVSTOF() transfers data from a C structure to a fielded buffer. *FML-BUFFER* is a record containing the fielded buffer. *DATA-REC* is the COBOL record. *VIEWNAME* IN *FML-REC* is the name of the view describing the COBOL record. *FML-MODE* IN *FML-REC* specifies the manner in which the transfer is made. *FML-MODE* has four possible values:

FUPDATE
FOJOIN
FJOIN
FCONCAT

The action of these modes are the same as that described in [Fupdate](#), [Fupdate32\(3fml\)](#), [Fojoin](#), [Fojoin32\(3fml\)](#), [Fjoin](#), [Fjoin32\(3fml\)](#), and [Fconcat](#), [Fconcat32\(3fml\)](#). One can even think of `FVSTOF()` as the same as these functions, except that where they specify a source buffer, `FVSTOF()` specifies a COBOL record. Bear in mind that `FUPDATE` does not move record elements that have NULL values.

`FVSTOF32()` is used for views defined with `view32()` typed buffers for larger views with more fields.

Return Values

Upon successful completion, `FVSTOF32()` sets `FML-STATUS` IN `FML-REC` to FOK.

On error, `FML-STATUS` is set to a non-zero value.

Errors

Under the following conditions, `FVSTOF()` fails and sets `FML-STATUS` to:

[FALIGNERR]

“fielded buffer not aligned”

The buffer does not begin on the proper boundary.

[FNOTFLD]

“buffer not fielded”

The buffer is not a fielded buffer or has not been initialized by `FINIT()`.

[FEINVAL]

“invalid argument to function”

One of the arguments to the function invoked was invalid.

[FBADACM]

“ACM contains negative value”

An Associated Count Member should not be a negative value while transferring data from a COBOL record to a fielded buffer.

[FBADVIEW]

“cannot find or get view”

The view description `VIEWNAME` was not found in the files specified by `VIEWDIR` or `VIEWFILES`.

See Also

[Introduction to FML Functions, viewfile\(5\)](#)

TPABORT(3cbl)

Name

TPABORT () - abort current BEA Tuxedo ATMI transaction

Synopsis

```

01  TPTRXDEF-REC.
    COPY TPTRXDEF.

01  TPSTATUS-REC.
    COPY TPSTATUS.

CALL "TPABORT" USING TPTRXDEF-REC TPSTATUS-REC.
```

Description

TPABORT () signifies the abnormal end of a transaction. When this call returns, all changes made to resources during the transaction are undone. Like TPCOMMIT () , this routine can be called only by the initiator of a transaction. Participants (that is, service routines) can express their desire to have a transaction aborted by calling TPRETURN () with TPFALL () .

If TPABORT () is called while communication handles exist for outstanding replies, then upon return from the routine, the transaction is aborted and those communications handles associated with the caller's transaction are no longer valid. Communications handles not associated with the caller's transaction remain valid.

For each open connection to a conversational server in transaction mode, TPABORT () will send a TPEV-DISCONIMM event to the server, whether or not the server has control of a connection. Connections opened before TPBEGIN () or with the TPNOTRAN setting (that is, not in transaction mode) are not affected.

The TPABORT () argument, TPTRXDEF-REC, is reserved for future use.

Return Values

Upon successful completion, TPABORT () sets TP-STATUS to [TPOK].

Errors

Under the following conditions, `TPABORT()` fails and sets `TP-STATUS` to:

[TPEINVAL]

Invalid arguments were given. The caller's transaction is not affected.

[TPEHEURISTIC]

Due to a heuristic decision, the work done on behalf of the transaction was partially committed and partially aborted.

[TPEHAZARD]

Due to some failure, the work done on behalf of the transaction could have been heuristically completed.

[TPEPROTO]

`TPABORT()` was called in an improper context (for example, by a participant).

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

Notices

When using `TPBEGIN()`, `TPCOMMIT()` and `TPABORT()` to delineate a BEA Tuxedo ATMI transaction, it is important to remember that only the work done by a resource manager that meets the XA interface (and is linked to the caller appropriately) has transactional properties. All other operations performed in a transaction are not affected by either `TPCOMMIT()` or `TPABORT`.

See Also

[TPBEGIN\(3cbl\)](#), [TPCOMMIT\(3cbl\)](#), [TPGETLEV\(3cbl\)](#)

TPACALL(3cbl)

Name

`TPACALL()` - routine to send a message to a service asynchronously

Synopsis

```

01  TPSVCDEF-REC.
    COPY TPSVCDEF.

01  TPTYPE-REC.
    COPY TPTYPE.

01  DATA-REC.
    COPY User data.

01  TPSTATUS-REC.
    COPY TPSTATUS.

CALL "TPACALL" USING TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.

```

Description

TPACALL () sends a request message to the service named by SERVICE-NAME in TPSVCDEF-REC. The request is sent out at the priority defined for SERVICE-NAME unless overridden by a previous call to TPSPRIO (). DATA-REC is a message to be sent and LEN in TPTYPE-REC specifies the amount of data in DATA-REC that should be sent. Note that if DATA-REC is a record of a type that does not require a length to be specified, then LEN is ignored (and may be 0). If REC-TYPE in TPTYPE-REC is SPACES, DATA-REC and LEN are ignored and a request is sent with no data portion. If REC-TYPE is STRING and LEN is 0, then the request is sent with no data portion. The REC-TYPE and SUB-TYPE of DATA-REC must match one of the REC-TYPE and SUB-TYPES recognized by SERVICE-NAME. Note that for each request sent while in transaction mode, a corresponding reply must ultimately be received.

The following is a list of valid settings in TPSVCDEF-REC.

TPNOTRAN

If the caller is in transaction mode and this setting is used, then when SERVICE-NAME is invoked, it is not performed on behalf of the caller's transaction. If SERVICE-NAME belongs to a server that does not support transactions, then this setting must be used when the caller is in transaction mode. A caller in transaction mode that uses this setting is still subject to the transaction timeout (and no other). If a service fails that was invoked with this setting, the caller's transaction is not affected. Either TPNOTRAN or TPTRAN must be set.

TPTRAN

If the caller is in transaction mode and this setting is used, then when `SERVICE-NAME` is invoked, it is performed on behalf of the caller's transaction. This setting is ignored if the caller is not in transaction mode. Either `TPNOTRAN` or `TPTRAN` must be set.

TPNOREPLY

Informs `TPACALL()` that a reply is not expected. When `TPNOREPLY` is set, the routine returns `[TPOK]` on success and sets `COMM-HANDLE` in `TPSVCDEF-REC` to 0, an invalid communications handle. When the caller is in transaction mode, this setting cannot be used when `TPTRAN` is also set. Either `TPNOREPLY` or `TPREPLY` must be set.

TPREPLY

Informs `TPACALL()` that a reply is expected. When `TPREPLY` is set, the routine returns `[TPOK]` on success and sets `COMM-HANDLE` to a valid communications handle. When the caller is in transaction mode, this setting must be used when `TPTRAN` is also set. Either `TPNOREPLY` or `TPREPLY` must be set.

TPNOBLOCK

The request is not sent if a blocking condition exists (for example, the internal buffers into which the message is transferred are full). Either `TPNOBLOCK` or `TPBLOCK` must be set.

TPBLOCK

When `TPBLOCK` is specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). Either `TPNOBLOCK` or `TPBLOCK` must be set.

TPNOTIME

This setting signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur. Either `TPNOTIME` or `TPTIME` must be set.

TPTIME

This setting signifies that the caller will receive blocking timeouts if a blocking condition exists and the blocking time is reached. Either `TPNOTIME` or `TPTIME` must be set.

TPSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is reissued. Either `TPNOSIGRSTRT` or `TPSIGRSTRT` must be set.

TPNOSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is not restarted and the call fails. Either `TPNOSIGRSTRT` or `TPSIGRSTRT` must be set.

Return Values

Upon successful completion, `TPACALL()` sets `TP-STATUS` to `[TPOK]`. In addition, if `TPREPLY` was set in `TPSVCDEF-REC`, then `TPCALL()` returns a valid communications handle in `COMM-HANDLE` that can be used to receive the reply of the request sent.

Errors

Under the following conditions, `TPACALL()` fails and sets `TP-STATUS` to (unless otherwise noted, failure does not affect the caller's transaction, if one exists):

[TPEINVAL]

Invalid arguments were given (for example, settings in `TPSVCDEF-REC` are invalid).

[TPENOENT]

Cannot send to `SERVICE-NAME` because it does not exist or is not a request/response service (that is, it is a conversational service).

[TPEITYPE]

The pair `REC-TYPE` and `SUB-TYPE` is not one of the allowed types and subtypes that `SERVICE-NAME` accepts.

[TPELIMIT]

The caller's request was not sent because the maximum number of outstanding asynchronous requests has been reached.

[TPETRAN]

`SERVICE-NAME` belongs to a server that does not support transactions and `TPTRAN` was set.

[TPETIME]

This error code indicates that either a timeout has occurred or `TPACALL()` has been attempted, in spite of the fact that the current transaction is already marked rollback only.

If the caller is in transaction mode, then either the transaction is already rollback only or a transaction timeout has occurred. The transaction is marked abort-only. If the caller is not in transaction mode, a blocking timeout has occurred. (A blocking timeout can occur only if both `TPBLOCK` and `TPTIME` are specified.)

If a transaction timeout has occurred, then, with one exception, any attempts to send new requests or receive outstanding replies will fail with `TPETIME` until the transaction has been aborted. The exception is a request that does not block, expects no reply, and is not sent on behalf of the caller's transaction (that is, `TPACALL()` with `TPNOTRAN`, `TPNOBLOCK`, and `TPNOREPLY` set).

When a service fails inside a transaction, the transaction is put into the `TX_ROLLBACK_ONLY` state. This state is treated, for most purposes, as though it were equivalent to a timeout. All further ATMI calls for this transaction (with the exception of those issued in the circumstances described in the previous paragraph) will fail with `TPETIME`.

[TPEBLOCK]

A blocking condition exists and `TPNOBLOCK` was specified.

[TPGOTSIG]

A signal was received and `TPNOSIGRSTRT` was specified.

[TPEPROTO]

`TPACALL()` was called improperly.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

See Also

[TPCALL\(3cbl\)](#), [TPCANCEL\(3cbl\)](#), [TPGETRPLY\(3cbl\)](#), [TPGPRIOR\(3cbl\)](#), [TPSPRIOR\(3cbl\)](#)

TPADVERTISE(3cbl)

Name

`TPADVERTISE()` - routine for advertising service names

Synopsis

```
01 SVC-NAME          PIC X(15) .
01 PROGRAM-NAME     PIC X(32) .
01 TPSTATUS-REC .
   COPY TPSTATUS.
```

```
CALL "TPADVERTISE" USING SVC-NAME PROGRAM-NAME TPSTATUS-REC.
```

Description

`TPADVERTISE()` allows a server to advertise the services that it offers. By default, a server's services are advertised when it is booted and unadvertised when it is shut down.

All servers belonging to a multiple server, single queue (MSSQ) set must offer the same set of services. These routines enforce this rule by affecting the advertisements of all servers sharing an MSSQ set.

`TPADVERTISE()` advertises *SVC-NAME* for the server (or the set of servers sharing the caller's MSSQ set). *SVC-NAME* should be 15 characters or less, but cannot be `SPACES`. (See `SERVICES` section of [UBBCONFIG\(5\)](#).) Longer names are truncated to 15 characters. Users should make sure that truncated names do not match other service names. *PROGRAM-NAME* is the name of a BEA Tuxedo ATMI service program. This program will be invoked whenever a request for *SVC-NAME* is received by the server. *PROGRAM-NAME* cannot be `SPACES`.

If *SVC-NAME* is already advertised for the server and *PROGRAM-NAME* matches its current program, then `TPADVERTISE()` returns success (this includes truncated names that match already advertised names). However, if *SVC-NAME* is already advertised for the server but *PROGRAM-NAME* does not match its current program, then an error is returned (this can happen if truncated names match already advertised names).

Return Values

`TPADVERTISE()` Upon successful completion, `TPADVERTISE()` sets `TP-STATUS` to `[TPOK]`.

Errors

Under the following conditions, `TPADVERTISE()` fails and sets `TP-STATUS` to:

[TPEINVAL]

Either *SVC-NAME* or *PROGRAM-NAME* is `SPACES`, or *PROGRAM-NAME* is not a name of a valid program.

[TPELIMIT]

SVC-NAME cannot be advertised because of space limitations. (See `MAXSERVICES` in the `RESOURCES` section of [UBBCONFIG\(5\)](#).)

[TPEMATCH]

SVC-NAME is already advertised for the server but with a program other than *PROGRAM-NAME*. Although `TPADVERTISE()` fails, *SVC-NAME* remains advertised with its current program (that is, *PROGRAM-NAME* does not replace the current program).

[TPEPROTO]

`TPADVERTISE()` was called improperly.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

Portability

On AIX on the RS6000, any services provided in the first COBOL object file are not available in the symbol table; their names must be specified using the `-s` option on the `buildserver` command so that they can be advertised at run time using `TPADVERTISE()`.

See Also

[TPUNADVERTISE\(3cbl\)](#)

TPBEGIN(3cbl)

Name

`TPBEGIN()` - routine to begin a BEA Tuxedo ATMI transaction

Synopsis

```
01  TPTRXDEF-REC.  
    COPY TPTRXDEF.
```

```
01  TPSTATUS-REC.  
    COPY TPSTATUS.
```

```
CALL "TPBEGIN" USING TPTRXDEF-REC TPSTATUS-REC.
```

Description

A transaction in the BEA Tuxedo system is used to define a single logical unit of work that either wholly succeeds or has no effect whatsoever. A transaction allows work being performed in many processes, at possibly different sites, to be treated as an atomic unit of work. The initiator of a transaction uses `TPBEGIN()` and either `TPCOMMIT()` or `TPABORT()` to delineate the operations within a transaction. Once `TPBEGIN()` is called, communication with any other program can place the latter (of necessity, a server) in “transaction mode” (that is, the server’s work becomes part of the transaction). Threads of control that join a transaction are called participants. A transaction always has one initiator and can have several participants. Only the initiator of a transaction can call `TPCOMMIT()` or `TPABORT()`. Participants can influence the outcome of a transaction by the settings in `TPSVCDEF-REC` they use when they call `TPRETURN()`. Once in

transaction mode, any service requests made to servers are processed on behalf of the transaction (unless the requester explicitly specifies otherwise).

Note that if a program starts a transaction while it has any open connections that it initiated to conversational servers, these connections will not be upgraded to transaction mode. It is as if the `TPNOTRAN` setting had been specified on the `TPCONNECT()` call.

`T-OUT` specifies that the transaction should be allowed at least `T-OUT` seconds before timing out. Once a transaction times out it must be aborted. If `T-OUT` is 0, then the transaction is given the maximum number of seconds allowed by the system before timing out (that is, the timeout value equals the maximum value for an unsigned long as defined by the system).

Return Values

Upon successful completion, `TPBEGIN()` sets `TP-STATUS` to `[TPOK]`.

Errors

Under the following conditions, `TPBEGIN()` fails and sets `TP-STATUS` to:

`[TPEINVAL]`

Invalid arguments were given.

`[TPETRAN]`

The caller cannot be placed in transaction mode because an error occurred starting the transaction.

`[TPEPROTO]`

`TPBEGIN()` was called in an improper context (for example, the caller is already in transaction mode).

`[TPESYSTEM]`

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

`[TPEOS]`

An operating system error has occurred.

Notices

When using `TPBEGIN()`, `TPCOMMIT()` and `TPABORT()` to delineate a BEA Tuxedo ATMI transaction, it is important to remember that only the work done by a resource manager that meets the XA0 interface (and is linked to the caller appropriately) has transactional properties. All other operations performed in a transaction are not affected by either `TPCOMMIT()` or `TPABORT()`. See [buildserver\(1\)](#) for details on linking resource managers that meet the XA interface into a

server such that operations performed by that resource manager are part of a BEA Tuxedo ATMI transaction.

See Also

[TPABORT\(3cbl\)](#), [TPCOMMIT\(3cbl\)](#), [TPGETLEV\(3cbl\)](#), [TPSCMT\(3cbl\)](#)

TPBROADCAST(3cbl)

Name

TPBROADCAST() - broadcast notification by name

Synopsis

```
01 TPBCTDEF-REC.  
   COPY TPBCTDEF.
```

```
01 TPTYPE-REC.  
   COPY TPTYPE.
```

```
01 DATA-REC.  
   COPY User data.
```

```
01 TPSTATUS-REC.  
   COPY TPSTATUS.
```

```
CALL "TPBROADCAST" USING TPBCTDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

Description

TPBROADCAST() allows a client or server to send unsolicited messages to registered clients within the system. The target client set consists of those clients matching identifiers passed to TPBROADCAST(). Wildcards can be used in specifying identifiers.

LMID, USERNAME and CLTNAME, all in TPBCTDEF-REC, are logical identifiers used to select the target client set. A SPACES value for any logical identifiers constitutes a wildcard for that argument. A wildcard argument matches all client identifiers for that field. Each identifier must meet the size restrictions defined for the system to be considered valid, that is, each identifier must be between 0 and 30 characters in length.

The data portion of the request is identified by *DATA-REC* and *LEN* in *TPTYPE-REC* specifies how much of *DATA-REC* to send. Note that if *DATA-REC* is a record of a type that does not require a length to be specified, then *LEN* is ignored (and may be 0). If *REC-TYPE* in *TPTYPE-REC* is *SPACES*, in which case *DATA-REC* and *LEN* are ignored and a request is sent with no data portion.

The following is a list of valid settings in *TPBCTDEF-REC*.

TPNOBLOCK

The request is not sent if a blocking condition exists (for example, the internal buffers into which the message is transferred are full). Either *TPNOBLOCK* or *TPBLOCK* must be set.

TPBLOCK

If a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). Either *TPNOBLOCK* or *TPBLOCK* must be set.

TPNOTIME

This setting signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur. Either *TPNOTIME* or *TPTIME* must be set.

TPTIME

This setting signifies that the caller will receive blocking timeouts if a blocking condition exists and the blocking time is reached. Either *TPNOTIME* or *TPTIME* must be set.

TPSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is reissued. Upon successful return from *TPBROADCAST()*, the message has been delivered to the system for forwarding to the selected clients. *TPBROADCAST()* does not wait for the message to be delivered to each selected client. Either *TPNOSIGRSTRT* or *TPSIGRSTRT* must be set.

TPNOSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is not restarted and the call fails. Either *TPNOSIGRSTRT* or *TPSIGRSTRT* must be set.

Return Values

Upon successful completion, *TPBROADCAST()* sets *TP-STATUS* to [TPOK].

Errors

Under the following conditions, *TPBROADCAST()* sends no broadcast messages to application clients and sets *TP-STATUS* to:

[TPEINVAL]

Invalid arguments were given. Note that use of an illegal LMID will cause TPBROADCAST () to fail and return TPEINVAL () . However, non-existent user or client names will simply successfully broadcast to no one.

[TPETIME]

A blocking timeout occurred. (A blocking timeout can occur only if both TPBLOCK and TPTIME are specified.)

[TPEBLOCK]

A blocking condition was found on the call and TPNOBLOCK was specified.

[TPGOTSIG]

A signal was received and TPSIGRSTRT was not specified.

[TPEPROTO]

TPBROADCAST () was called improperly.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

Portability

The interfaces described in TPNOTIFY () are supported on native site UNIX-based processors. In addition, the routines TPBROADCAST () and TPCHKUNSOL () as well as the routine TPSETUNSOL () are supported on UNIX and MS-DOS workstation processors.

Usage

Clients that select signal-based notification may not be signal-able by the system due to signal restrictions. When this occurs, the system generates a log message that it is switching notification for the selected client to dip-in and the client is notified then and thereafter via dip-in notification. (See [UBBCONFIG \(5\)](#) description of the RESOURCES NOTIFY parameter for a detailed discussion of notification methods.)

Because signaling of clients is always done by the system, the behavior of notification is always consistent, regardless of where the originating notification call is made. Therefore to use signal-based notification:

- A native client must be running as an application administrator.

- A Workstation client is not required to be running as the application administrator.

The ID for the application administrator is identified in the configuration file for the application.

If signal-based notification is selected for a client, then certain ATMI calls can fail, returning `TPGOTSIG()` due to receipt of an unsolicited message if `TPSIGRSTRT` is not specified. See [UBBCONFIG\(5\)](#) and [TPINITIALIZE\(3cbl\)](#) for more information on notification method selection.

See Also

[TPINITIALIZE\(3cbl\)](#), [TPNOTIFY\(3cbl\)](#), [TPTERM\(3cbl\)](#), [UBBCONFIG\(5\)](#)

TPCALL(3cbl)

Name

`TPCALL()` - routine to send a message to a service synchronously

Synopsis

```
01 TPSVCDEF-REC.
   COPY TPSVCDEF.
```

```
01 ITPTYPE-REC.
   COPY TPTYPE.
```

```
01 IDATA-REC.
   COPY User data.
```

```
01 OTPTYPE-REC.
   COPY TPTYPE.
```

```
01 ODATA-REC.
   COPY User data.
```

```
01 TPSTATUS-REC.
   COPY TPSTATUS.
```

```
CALL "TPCALL" USING TPSVCDEF-REC ITPTYPE-REC IDATA-REC OTPTYPE-REC
ODATA-REC TPSTATUS-REC.
```

Description

`TPCALL()` sends a request and synchronously awaits its reply. A call to this routine is the same as calling `TPACALL()` immediately followed by `TPGETRPLY()`. `TPCALL()` sends a request to the request/response service named by `SERVICE-NAME` in `TPSVCDEF-REC`. The request is sent out at the priority defined for `SERVICE-NAME` unless overridden by a previous call to `TPSPRIO()`. The data portion of a request is specified by `IDATA-REC` and `LEN` in `ITPTYPE-REC` specifies how much of `IDATA-REC` to send. Note that if `IDATA-REC` is a record of a type that does not require a length to be specified, then `LEN` in `ITPTYPE-REC` is ignored (and may be 0). If `REC-TYPE` in `ITPTYPE-REC` is `SPACES`, `IDATA-REC` and `LEN` in `ITPTYPE-REC` are ignored and a request is sent with no data portion. If `REC-TYPE` in `ITPTYPE-REC` is `STRING` and `LEN` in `ITPTYPE-REC` is 0, then the request is sent with no data portion. The `REC-TYPE` in `ITPTYPE-REC` and `SUB-TYPE` in `ITPTYPE-REC` must match one of the `REC-TYPES` and `SUB-TYPES` recognized by `SERVICE-NAME`.

`ODATA-REC` specifies where a reply is read into, and, on input `LEN` in `OTPTYPE-REC` indicates the maximum number of bytes that should be moved into `ODATA-REC`. If the same record is to be used for both sending and receiving, `ODATA-REC` should be REDEFINED to `IDATA-REC`. Upon successful return from `TPCALL()`, `LEN` in `OTPTYPE-REC` contains the actual number of bytes moved into `ODATA-REC`. `REC-TYPE` and `SUB-TYPE` in `OTPTYPE-REC` contain the replies type and subtype respectively. If the reply is larger than `ODATA-REC`, then `ODATA-REC` will contain only as many bytes as will fit in the record. The remainder of the reply is discarded and `TPCALL()` sets `TPTRUNCATE()`.

If `LEN` in `OTPTYPE-REC` is 0 upon successful return, then the reply has no data portion and `ODATA-REC` was not modified. It is an error for `LEN` in `OTPTYPE-REC` to be 0 on input.

The following is a list of valid settings in `TPSVCDEF-REC`.

TPNOTRAN

If the caller is in transaction mode and this setting is used, then when `SERVICE-NAME` is invoked, it is not performed on behalf of the caller's transaction. If the `SERVICE-NAME` belongs to a server that does not support transactions then this setting must be used when the caller is in transaction mode. A caller in transaction mode that sets this to `true` is still subject to the transaction timeout (and no other). If a service fails that was invoked with this setting, the caller's transaction is not affected. Either `TPNOTRAN` or `TPTRAN` must be set.

TPTRAN

If the caller is in transaction mode and this setting is used, then when `SERVICE-NAME` is invoked, it is performed on behalf of the caller's transaction. The setting is ignored if the caller is not in transaction mode. Either `TPNOTRAN` or `TPTRAN` must be set.

TPNOCHANGE

When this setting is used, the type of *ODATA-REC* is not allowed to change. That is, the type and subtype of the replied record must match *REC-TYPE* IN *OTPTYPE-REC* and *SUB-TYPE* IN *OTPTYPE-REC*, respectively, so long as the receiver recognizes the incoming record type. Either *TPNOCHANGE* or *TPCHANGE* must be set.

TPCHANGE

The type and/or subtype of the reply record is allowed to differ from those specified in *REC-TYPE* IN *OTPTYPE-REC* and *SUB-TYPE* IN *OTPTYPE-REC*, respectively, so long as the receiver recognizes the incoming record type. Either *TPNOCHANGE* or *TPCHANGE* must be set.

TPNOBLOCK

The request is not sent if a blocking condition exists (for example, the internal buffers into which the message is transferred are full). Note that this setting applies only to the send portion of *TPCALL()*: the routine may block waiting for the reply. Either *TPNOBLOCK* or *TPBLOCK* must be set.

TPBLOCK

When *TPBLOCK* is specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). Either *TPNOBLOCK* or *TPBLOCK* must be set.

TPNOTIME

This setting signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur. Either *TPNOTIME* or *TPTIME* must be set.

TPTIME

This setting signifies that the caller will receive blocking timeouts if a blocking condition exists and the blocking time is reached. Either *TPNOTIME* or *TPTIME* must be set.

TPSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is reissued. Either *TPNOSIGRSTRT* or *TPSIGRSTRT* must be set.

TPNOSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is not restarted and the routine fails. Either *TPNOSIGRSTRT* or *TPSIGRSTRT* must be set.

Return Values

Upon successful completion, `TPCALL()` sets `TP-STATUS` to `[TPOK]`. When `TP-STATUS` is set to `TPOK` or `TPESVCFail`, `APPL-RETURN-CODE` in `TPSTATUS-REC` contains an application-defined value that was sent as part of `TPRETURN()`.

If the size of the incoming message was larger than the size specified in `LEN` on input, `TPTRUNCATE()` is set and only `LEN` amount of data was moved to `ODATA-REC`, the remaining data is discarded.

Errors

Under the following conditions, `TPCALL()` fails and sets `TP-STATUS` to (unless otherwise noted, failure does not affect the caller's transaction, if one exists):

[TPEINVAL]

Invalid arguments were given (for example, `SERVICE-NAME` is `SPACES` or settings in `TPSVCDEF-REC` are invalid).

[TPENOENT]

Cannot send to `SERVICE-NAME` because it does not exist or is not a request/response service (that is, it is a conversational service).

[TPEITYPE]

The pair `REC-TYPE` and `SUB-TYPE` is not one of the allowed types and subtypes that `SERVICE-NAME` accepts.

[TPEOTYPE]

Either the type and subtype of the reply are not known to the caller; or, `TPNOCHANGE` was set and the `REC-TYPE` and `SUB-TYPE` in `ODATA-REC` do not match the type and subtype of the reply sent by the service. Neither `ODATA-REC` nor `LEN` in `OTPTYPE-REC` are changed. If the service request was made on behalf of the caller's current transaction, then the transaction is marked abort-only since the reply is discarded.

[TPETRAN]

`SERVICE-NAME` belongs to a server that does not support transactions and `TPTRAN` was set.

[TPETIME]

This error code indicates that either a timeout has occurred or `TPCALL()` has been attempted, in spite of the fact that the current transaction is already marked rollback only.

If the caller is in transaction mode, then either the transaction is already rollback only or a transaction timeout has occurred. The transaction is marked abort-only. If the caller is not in transaction mode, a blocking timeout has occurred. (A blocking timeout can occur

only if both `TPBLOCK` and `TPTIME` are specified.) In either case, no changes are made to `ODATA-REC` or `OTPTYPE-REC`.

If a transaction timeout has occurred, then, with one exception, any attempts to send new requests or receive outstanding replies will fail with `TPETIME` until the transaction has been aborted. The exception is a request that does not block, expects no reply, and is not sent on behalf of the caller's transaction (that is, `TPACALL()` with `TPNOTRAN`, `TPNOBLOCK`, and `TPNOREPLY` set).

When a service fails inside a transaction, the transaction is put into the `TX_ROLLBACK_ONLY` state. This state is treated, for most purposes, as though it were equivalent to a timeout. All further ATMI calls for this transaction (with the exception of those issued in the circumstances described in the previous paragraph) will fail with `TPETIME`.

[TPESVCFAIL]

The service routine sending the caller's reply called `TPRETURN()` with `TPFAIL()`. This is an application-level failure. The contents of the service's reply, if one was sent, is available in `ODATA-REC`. If the service request was made on behalf of the caller's current transaction, then the transaction is marked abort-only. Note that regardless of whether the transaction has timed out, the only valid communications before the transaction is aborted are calls to `TPACALL()` with `TPNOREPLY`, `TPNOTRAN`, and `TPNOBLOCK` set.

[TPESVCERR]

An error was encountered either in invoking a service routine or during its completion in `TPRETURN()` (for example, bad arguments were passed). No reply data is returned when this error occurs (that is, neither `ODATA-REC` nor `OTPTYPE-REC` are changed). If the service request was made on behalf of the caller's transaction (that is, `TPNOTRAN` was not set), then the transaction is marked abort-only. Note that regardless of whether the transaction has timed out, the only valid communications before the transaction is aborted are calls to `TPACALL()` with `TPNOREPLY`, `TPNOTRAN`, and `TPNOBLOCK` set.

[TPBLOCK]

A blocking condition was found on the send portion of `TPCALL()` and `TPNOBLOCK` was specified.

[TPGOTSIG]

A signal was received and `TPSIGRSTRT` was not specified.

[TPEPROTO]

`TPCALL()` was called improperly.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

See Also

[TPACALL\(3cbl\)](#), [TPFORWAR\(3cbl\)](#), [TPGPRIOR\(3cbl\)](#), [TPRETURN\(3cbl\)](#), [TPSPRIO\(3cbl\)](#)

TPCANCEL(3cbl)

Name

TPCANCEL() - cancel a communication handle for an outstanding reply

Synopsis

```
01 TPSVCDEF-REC.  
   COPY TPSVCDEF.
```

```
01 TPSTATUS-REC.  
   COPY TPSTATUS.
```

```
CALL "TPCANCEL" USING TPSVCDEF-REC TPSTATUS-REC.
```

Description

TPCANCEL() cancels a communication handle, COMM-HANDLE IN *TPSVCDEF-REC*, returned by *TPACALL()*. It is an error to attempt to cancel a communication handle associated with a transaction.

Upon success, COMM-HANDLE is no longer valid and any reply received on behalf of COMM-HANDLE will be silently discarded.

Return Values

Upon successful completion, TPCANCEL() sets TP-STATUS to [TPOK].

Errors

Under the following conditions, TPCANCEL() fails and sets TP-STATUS to:

[TPEBADDESC]

COMM-HANDLE is an invalid communication handle.

[TPETRAN]

COMM-HANDLE is associated with the caller's transaction. COMM-HANDLE remains valid and the caller's current transaction is not affected.

[TPEPROTO]

TPCANCEL () was called improperly.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

See Also

[TPACALL \(3cbl\)](#)

TPCHKAUTH(3cbl)**Name**

TPCHKAUTH ()—check if authentication required to join a BEA Tuxedo ATMI application

Synopsis

```
01  TPAUTDEF-REC.
   COPY TPAUTDEF.
```

```
01  TPSTATUS-REC.
   COPY TPSTATUS.
```

```
CALL "TPCHKAUTH" USING TPAUTDEF-REC TPSTATUS-REC.
```

Description

TPCHKAUTH () checks if authentication is required by the application configuration. This is typically used by application clients prior to calling TPINITIALIZE () to determine if a password should be obtained from the user.

Return Values

Upon successful completion, `TPCHKAUTH()` sets `TP-STATUS` to `[TPOK]` and sets one of the following values in `TPAUTDEF-REC`.

`TPNOAUTH`

Indicates that no authentication is required.

`TPSYSAUTH`

Indicates that only system authentication is required.

`TPAPPAUTH`

Indicates that both system and application specific authentication are required.

Errors

Under the following conditions, `TPCHKAUTH()` fails and sets `TP-STATUS` to:

`[TPESYSTEM]`

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

`[TPEOS]`

An operating system error has occurred.

Portability

The interfaces described in `TPCHKAUTH()` are supported on UNIX system and MS-DOS operating systems.

See Also

[`TPINITIALIZE\(3cbl\)`](#)

TPCHKUNSOL(3cbl)

Name

`TPCHKUNSOL()` - check for unsolicited message

Synopsis

```
01 MSG-NUM PIC S9(9) COMP-5.
```

```
01 TPSTATUS-REC.
```

```
    COPY TPSTATUS.
```

```
CALL "TPCHKUNSOL" USING MSG-NUM TPSTATUS-REC.
```

Description

TPCHKUNSOL () is used by a client to trigger checking for unsolicited messages. Calls to this routine in a client using signal-based notification do nothing and return immediately. Calls to this routine can result in calls to an application-defined unsolicited message handling routine by the BEA Tuxedo ATMI libraries.

Return Values

Upon successful completion, TPCHKUNSOL () sets TP-STATUS to [TPOK] and returns the number of unsolicited messages dispatched in MSG-NUM.

Errors

Under the following conditions, TPCHKUNSOL () fails and sets TP-STATUS to:

[TPEPROTO]

TPCHKUNSOL () was called in an improper context (for example, from within a server).

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

Portability

The interfaces described in TPNOTIFY () are supported on native site UNIX-based processors. In addition, the routines TPBROADCAST () and TPCHKUNSOL () as well as the routine TPSETUNSOL () are supported on UNIX and MS-DOS workstation processors.

Clients that select signal-based notification may not be signal-able by the system due to signal restrictions. When this occurs, the system generates a log message that it is switching notification for the selected client to dip-in and the client is notified then and thereafter via dip-in notification. (See [UBBCONFIG \(5\)](#) description of the RESOURCES NOTIFY parameter for a detailed discussion of notification methods.)

Because signaling of clients is always done by the system, the behavior of notification is always consistent, regardless of where the originating notification call is made. Therefore to use signal-based notification:

- A native client must be running as an application administrator.
- A Workstation client is not required to be running as the application administrator.

The ID for the application administrator is identified as part of the configuration for the application.

If signal-based notification is selected for a client, then certain ATMI calls can fail, returning `TPGOTSIG()` due to receipt of an unsolicited message if `TPSIGRSTRT` is not specified. See [UBBCONFIG\(5\)](#) and [TPINITIALIZE\(3cbl\)](#) for more information on notification method selection.

See Also

[TPBROADCAST\(3cbl\)](#), [TPINITIALIZE\(3cbl\)](#), [TPNOTIFY\(3cbl\)](#), [TPSETUNSOL\(3cbl\)](#)

TPCLOSE(3cbl)

Name

`TPCLOSE()` - close the BEA Tuxedo ATMI resource manager

Synopsis

```
01  TPSTATUS-REC.
    COPY  TPSTATUS.
```

```
CALL "TPCLOSE" USING  TPSTATUS-REC.
```

Description

`TPCLOSE()` tears down the association between the caller and the resource manager to which it is linked. Since resource managers differ in their `close` semantics, the specific information needed to close a particular resource manager is placed in a configuration file.

If a resource manager is already closed (that is, `TPCLOSE()` is called more than once), no action is taken and success is returned.

Return Values

Upon successful completion, `TPCLOSE()` sets `TP-STATUS` to `[TPOK]`.

Errors

Under the following conditions, `TPCLOSE()` fails and sets `TP-STATUS` to:

[TPERMERR]

A resource manager failed to close correctly. More information concerning the reason a resource manager failed to close can be obtained by interrogating a resource manager in its own specific manner. Note that any calls to determine the exact nature of the error hinder portability.

[TPEPROTO]

TPCLOSE() was called in an improper context (for example, while the caller is in transaction mode).

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

See Also

[TPOPEN\(3cbl\)](#)

TPCOMMIT(3cbl)**Name**

TPCOMMIT() - commit current BEA Tuxedo ATMI transaction

Synopsis

```
01 TPTRXDEF-REC.
   COPY TPTRXDEF.
```

```
01 TPSTATUS-REC.
   COPY TPSTATUS.
```

```
CALL "TPCOMMIT" USING TPTRXDEF-REC TPSTATUS-REC
```

Description

TPCOMMIT() signifies the end of a transaction, using a two-phase commit protocol to coordinate participants. TPCOMMIT() can be called only by the initiator of a transaction. If any of the participants cannot commit the transaction (for example, they call TPRETURN() with TPFail()), then the entire transaction is aborted and TPCOMMIT() fails. That is, all of the work involved in

the transaction is undone. If all participants agree to commit their portion of the transaction, then this decision is logged to stable storage and all participants are asked to commit their work.

Depending on the setting of the `TP-COMMIT-CONTROL` characteristic (see `TPSCMT()`), `TPCOMMIT()` can return successfully either after the commit decision has been logged or after the two-phase commit protocol has completed. If `TPCOMMIT()` returns after the commit decision has been logged but before the second phase has completed (`TP-CMT-LOGGED`), then all participants have agreed to commit the work they did on behalf of the transaction and should fulfill their promise to commit the transaction during the second phase. However, because `TPCOMMIT()` is returning before the second phase has completed, there is a hazard that one or more of the participants can heuristically complete their portion of the transaction (in a manner that is not consistent with the commit decision) even though the routine has returned success.

If the `TP-COMMIT-CONTROL` characteristic is set such that `TPCOMMIT()` returns after the two-phase commit protocol has completed (`TP-CMT-COMPLETE`), then its return value reflects the exact status of the transaction (that is, whether the transaction heuristically completed or not).

Note that if only a single resource manager is involved in a transaction, then a one-phase commit is performed (that is, the resource manager is not asked whether or not it can commit; it is simply told to commit). In this case, the `TP-COMMIT-CONTROL` characteristic has no bearing and `TPCOMMIT()` will return heuristic outcomes if present.

If `TPCOMMIT()` is called while communication handles exist for outstanding replies, then upon return from `TPCOMMIT()`, the transaction is aborted and those handles associated with the caller's transaction are no longer valid. Communication handles not associated with the caller's transaction remain valid.

`TPCOMMIT()` must be called after all connections associated with the caller's transaction are closed (otherwise `[TPEABORT]` is returned, the transaction is aborted and these connections are disconnected in a disorderly fashion with a `TPEV-DISCONIMM` event). Connections opened before `TPBEGIN()` or with the `TPNOTRAN` setting (that is, connections not in transaction mode) are not affected by calls to `TPCOMMIT()` or `TPABORT()`.

Currently, `TPCOMMIT()`'s argument, `TPTRXDEF-REC`, is reserved for future use.

Return Values

Upon successful completion, `TPCOMMIT()` sets `TP-STATUS` to `[TPOK]`.

Errors

Under the following conditions, `TPCOMMIT()` fails and sets `TP-STATUS` to:

[TPEINVAL]

TPTRXDEF-REC is not equal to 0. The caller's transaction is not affected.

[TPETIME]

The transaction has timed out and its status is unknown: it may have been either committed or aborted. If a transaction has timed out and its status is known to be aborted, than **TPEABORT** is returned.

[TPEABORT]

The transaction could not commit because either the work performed by the initiator or by one or more of its participants could not commit. This error is also returned if **TPCOMMIT()** is called with outstanding replies or open conversational connections.

[TPEHEURISTIC]

Due to a heuristic decision, the work done on behalf of the transaction was partially committed and partially aborted.

[TPEHAZARD]

Due to some failure, the work done on behalf of the transaction could have been heuristically completed.

[TPEPROTO]

TPCOMMIT() was called in an improper context (for example, by a participant).

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

Notices

When using **TPBEGIN()**, **TPCOMMIT()**, and **TPABORT()** to delineate a BEA Tuxedo ATMI transaction, it is important to remember that only the work done by a resource manager that meets the XA interface (and is linked to the caller appropriately) has transactional properties. All other operations performed in a transaction are not affected by either **TPCOMMIT()** or **TPABORT()**. See [buildserver\(1\)](#) for details on linking resource managers that meet the XA interface into a server such that operations performed by that resource manager are part of a BEA Tuxedo ATMI transaction.

See Also

[TPABORT\(3cbl\)](#), [TPBEGIN\(3cbl\)](#), [TPCONNECT\(3cbl\)](#), [TPGETLEV\(3cbl\)](#), [TPRETURN\(3cbl\)](#),
[TPSCMT\(3cbl\)](#)

TPCONNECT(3cbl)

Name

TPCONNECT() - establish a conversational connection

Synopsis

```
01 TPSVCDEF-REC.  
   COPY TPSVCDEF.
```

```
01 TPTYPE-REC.  
   COPY TPTYPE.
```

```
01 DATA-REC.  
   COPY User data.
```

```
01 TPSTATUS-REC.  
   COPY TPSTATUS.
```

```
CALL "TPCONNECT" USING TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

Description

TPCONNECT() allows a program to set up a half-duplex connection to a conversational service, *SERVICE-NAME* in *TPSVCDEF-REC*. The name must be one of the conversational service names posted by a conversational server.

As part of setting up a connection, the caller can pass application-defined data to the receiving service routine. If the caller chooses to pass data, then *DATA-REC* contains the data and *LEN* in *TPTYPE-REC* specifies how much of the record to send. Note that if *DATA-REC* is a record of a type that does not require a length to be specified, then *LEN* is ignored (and may be 0). If *REC-TYPE* in *TPTYPE-REC* is *SPACES*, *DATA-REC* and *LEN* are ignored (no application data is passed to the conversational service). *REC-TYPE* and *SUB-TYPE* in *TPTYPE-REC* must match one of the types and subtypes recognized by *SERVICE-NAME*.

Because the conversational service receives *DATA-REC* and *LEN* upon successful return from *TPSVCSTART()*, the service does not call *TPRECV()* to get the data sent by *TPCONNECT()*.

The following is a list of valid settings in *TPSVCDEF-REC*.

TPNOTRAN

If the caller is in transaction mode and this setting is used, then when *SERVICE-NAME* is invoked, it is not performed on behalf of the caller's transaction. If *SERVICE-NAME* belongs to a server that does not support transactions, then this setting must be used when the caller is in transaction mode. A caller in transaction mode that uses this setting is still subject to the transaction timeout (and no other). If a service fails that was invoked with this setting, the caller's transaction is not affected. Either *TPNOTRAN* or *TPTRAN* must be set.

TPTRAN

If the caller is in transaction mode and this setting is used, then when *SERVICE-NAME* is invoked, it is performed on behalf of the caller's transaction. This setting is ignored if the caller is not in transaction mode. Either *TPNOTRAN* or *TPTRAN* must be set.

TPSENDONLY

The caller wants the connection to be set up initially such that it can only send data and the called service can only receive data (that is, the caller initially has control of the connection). Either *TPSENDONLY* or *TPRECVONLY* must be specified.

TPRECVONLY

The caller wants the connection to be set up initially such that it can only receive data and the called service can only send data (that is, the service being called initially has control of the connection). Either *TPSENDONLY* or *TPRECVONLY* must be specified.

TPNOBLOCK

The connection is not established and the data is not sent if a blocking condition exists (for example, the data buffers through which the message is sent are full). Either *TPNOBLOCK* or *TPBLOCK* must be set.

TPBLOCK

When *TPBLOCK* is specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). Either *TPNOBLOCK* or *TPBLOCK* must be set.

TPNOTIME

This setting signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts will still affect the program. Either *TPNOTIME* or *TPTIME* must be set.

TPTIME

This setting signifies that the caller will receive blocking timeouts if a blocking condition exists and the blocking time is reached. Either `TPNOTIME` or `TPTIME` must be set.

TPSIGRSTRT

If a signal interrupts any underlying system calls, the interrupted call is reissued. Either `TPNOSIGRSTRT` or `TPSIGRSTRT` must be set.

TPNOSIGRSTRT

When `TPNOSIGRSTRT` is specified and a signal interrupts a system call, the call fails and `TP-STATUS` is set to `TPGOTSIG()`. Either `TPNOSIGRSTRT` or `TPSIGRSTRT` must be set.

Return Values

Upon successful completion, `TPCONNECT()` sets `TP-STATUS` to `[TPOK]` and returns a communications handle in `COMM-HANDLE` in *TPSVCDEF-REC* that is used to refer to the connection in subsequent calls.

Errors

Under the following conditions, `TPCONNECT()` fails and sets `TP-STATUS` to (unless otherwise noted, failure does not affect the caller's transaction, if one exists).

[TPEINVAL]

Invalid arguments were given (for example, settings in *TPSVCDEF-REC* are invalid).

[TPENOENT]

Can not initiate a connection to `SERVICE-NAME` because it does not exist or is not a conversational service.

[TPEITYPE]

The pair `REC-TYPE` and `SUB-TYPE` is not one of the allowed types and subtypes that `SERVICE-NAME` accepts.

[TPELIMIT]

The connection was not sent because the maximum number of outstanding connections has been reached.

[TPETRAN]

`SERVICE-NAME` belongs to a program that does not support transactions and `TPNOTRAN` was not set.

[TPETIME]

This error code indicates that either a timeout has occurred or `TPCONNECT()` has been attempted, in spite of the fact that the current transaction is already marked rollback only.

If the caller is in transaction mode, then either the transaction is already rollback only or a transaction timeout has occurred. The transaction is marked abort-only. If the caller is not in transaction mode, a blocking timeout has occurred. (A blocking timeout can occur only if both `TPBLOCK` and `TPTIME` are specified.)

If a transaction timeout has occurred, then, with one exception, any attempts to send new requests or receive outstanding replies will fail with `TPETIME` until the transaction has been aborted. The exception is a request that does not block, expects no reply, and is not sent on behalf of the caller's transaction (that is, `TPACALL()` with `TPNOTRAN`, `TPNOBLOCK`, and `TPNOREPLY` set).

When a service fails inside a transaction, the transaction is put into the `TX_ROLLBACK_ONLY` state. This state is treated, for most purposes, as though it were equivalent to a timeout. All further ATMI calls for this transaction (with the exception of those issued in the circumstances described in the previous paragraph) will fail with `TPETIME`.

[TPGOTSIG]

A signal was received and `TPSIGRSTRT` was not specified.

[TPEPROTO]

`TPCONNECT()` was called improperly.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

See Also

`TPDISCON(3cbl)`, `TPRECV(3cbl)`, `TPSEND(3cbl)`

TPDEQUEUE(3cbl)

Name

`TPDEQUEUE()` - routine to dequeue a message from a queue

Synopsis

```
01 TPQUEDEF-REC.
   COPY TPQUEDEF.
```

```

01 TPTYPE-REC.
   COPY TPTYPE.

01 DATA-REC.
   COPY User data.

01 TPSTATUS-REC.
   COPY STATDEF.

CALL "TPDEQUEUE" USING TPQUEDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.

```

Description

`TPDEQUEUE()` takes a message for processing from the queue named by `QNAME` in the `QSPACE-NAME` queue space.

By default, the message at the top of the queue is dequeued. The order of messages on the queue is defined when the queue is created. The application can request a particular message for dequeuing by specifying its message identifier using `MSGID` or correlation identifier using `CORRID`. *TPQUEDEF-REC* settings can also be used to indicate that the application wants to wait for a message, in the case when a message is not currently available. It is possible to use the *TPQUEDEF-REC* structure to look at a message without removing it from the queue or changing its relative position on the queue. See the section below describing this record.

DATA-REC specifies where a dequeued message is to be read into, and, on input `LEN` indicates the maximum number of bytes that should be moved into *DATA-REC*. Upon successful return, `LEN` contains the actual number of bytes moved into *DATA-REC*. `REC-TYPE` and `SUB-TYPE` contain the replies type and subtype respectively. If the reply is larger than *DATA-REC*, then *DATA-REC* will contain only as many bytes as will fit in the record. The remainder of the reply is discarded and `TPDEQUEUE()` fails returning `[TPTRUNCATE]`.

If `LEN` is 0 upon successful return, then the reply has no data portion and *DATA-REC* was not modified. It is an error for `LEN` to be 0 on input.

The message is dequeued in transaction mode if the caller is in transaction mode and `TPTRAN` is set. This has the effect that if *TPDEQUEUE* returns successfully and the caller's transaction is committed successfully, then the message is removed from the queue. If the caller's transaction is rolled back either explicitly or as the result of a transaction timeout or some communication error, then the message will be left on the queue (that is, the removal of the message from the

queue is also rolled back). It is not possible to enqueue and dequeue the same message within the same transaction.

The message is not dequeued in transaction mode if either the caller is not in transaction mode, or `TPNOTRAN` is set. When not in transaction mode, if a communication error or a timeout occurs, the application will not know whether or not the message was successfully dequeued and the message may be lost.

The following is a list of valid settings in `TPQUEDEF-REC`.

`TPNOTRAN`

If the caller is in transaction mode and this setting is used, the message is not dequeued within the caller's transaction. A caller in transaction mode that sets this to `true` is still subject to the transaction timeout (and no other). If message dequeuing fails that was invoked with this setting, the caller's transaction is not affected. Either `TPNOTRAN` or `TPTRAN` must be set.

`TPTRAN`

If the caller is in transaction mode and this setting is used, the message is dequeued within the same transaction as the caller. The setting is ignored if the caller is not in transaction mode. Either `TPNOTRAN` or `TPTRAN` must be set.

`TPNOBLOCK`

The message is not dequeued if a blocking condition exists. If `TPNOBLOCK` is set and a blocking condition exists such as the internal buffers into which the message is transferred are full, the call fails and `TP-STATUS` is set to `TPEBLOCK`. If `TPNOBLOCK` is set and a blocking condition exists because the target queue is opened *exclusively* by another application, the call fails, `TP-STATUS` is set to `TPEDIANOSTIC`, and the `DIAGNOSTIC` field of the `TPQUEDEF` record is set to `QMESHARE`. In the latter case, the other application, which is based on a BEA product other than the BEA Tuxedo system, opened the queue for exclusive read and/or write using the Queuing Services API (QSAPI). Either `TPNOBLOCK` or `TPBLOCK` must be set.

`TPBLOCK`

When `TPBLOCK` is set and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). This blocking condition does not include blocking on the queue itself if the `TPQWAIT` setting is specified. Either `TPNOBLOCK` or `TPBLOCK` must be set.

`TPNOTIME`

This setting signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur. Either `TPNOTIME` or `TPTIME` must be set.

TPTIME

This setting signifies that the caller will receive blocking timeouts if a blocking condition exists and the blocking time is reached. Either `TPNOTIME` or `TPTIME` must be set.

TPNOCHANGE

When this setting is used, the type of `DATA-REC` is not allowed to change. That is, the type and subtype of the dequeued message must match `REC-TYPE IN TPTYPE-REC` and `SUB-TYPE IN TPTYPE-REC`, respectively, so long as the receiver recognizes the incoming record type. Either `TPNOCHANGE` or `TPCHANGE` must be set.

TPCHANGE

The type and/or subtype of the dequeued message is allowed to differ from those specified in `REC-TYPE IN TPTYPE-REC` and `SUB-TYPE IN TPTYPE-REC`, respectively, so long as the receiver recognizes the incoming record type. Either `TPNOCHANGE` or `TPCHANGE` must be set.

TPSIGRSTRT

If a signal interrupts any underlying system calls, the interrupted system call is reissued. Either `TPNOSIGRSTRT` or `TPSIGRSTRT` must be set.

TPNOSIGRSTRT

If a signal interrupts any underlying system calls, the interrupted system call is not restarted and the routine fails. Either `TPNOSIGRSTRT` or `TPSIGRSTRT` must be set.

If `TPDEQUEUE()` returns successfully, the application can retrieve additional information about the message using the `TPQUEDEF-REC` structure. The information may include the message identifier for the dequeued message; a correlation identifier that should accompany any reply or failure message so that the originator can correlate the message with the original request; the quality of service the message was delivered with; the quality of service any replies to the message should be delivered with; the name of a reply queue if a reply is desired; and the name of the failure queue on which the application can queue information regarding failure to dequeue the message. These are described below.

Control Structure

`TPQUEDEF-REC` is used by the application program to pass and retrieve information associated with dequeuing the message. The settings in `TPQUEDEF-REC` are used to indicate what other elements in the structure are valid.

On input to `TPDEQUEUE()`, the following elements may be set in the `TPQUEDEF-REC`:

```
05 MSGID      PIC X(32) .  
05 CORRID     PIC X(32) .
```

The following is a list of valid settings in *TPQUEUEDEF-REC* controlling input information for *TPDEQUEUE()*.

TPQGETNEXT

Setting this value requests that the next message on the queue be dequeued, using the default queue order. One of the following must be set: *TPQGETNEXT*, *TPQGETBYMSGID*, or *TPQGETBYCORRID*.

TPQGETBYMSGID

Setting this value requests that the message identified by *MSGID* be dequeued. The message identifier may be acquired by a prior call to *TPENQUEUE()*. Note that the message identifier changes if the message has moved from one queue to another. Note also that the entire 32 bytes of the message identifier value are significant, so the value identified by *MSGID* must be completely initialized (for example, padded with spaces).

One of the following must be set: *TPQGETNEXT*, *TPQGETBYMSGID*, or *TPQGETBYCORRID*.

TPQGETBYCORRID

Setting this value requests that the message identified by *CORRID* be dequeued. The correlation identifier is specified by the application when enqueueing the message with *TPENQUEUE()*. Note that the entire 32 bytes of the correlation identifier value are significant, so the value identified by *CORRID* must be completely initialized (for example, padded with spaces).

One of the following must be set: *TPQGETNEXT*, *TPQGETBYMSGID*, or *TPQGETBYCORRID*.

TPQWAIT

Setting this value indicates that an error should not be returned if the queue is empty. Instead, the process should wait until a message is available. Set *TPQNOWAIT* to not wait until a message is available. If *TPQWAIT* is set in conjunction with *TPQGETBYMSGID* or *TPQGETBYCORRID*, it indicates that an error should not be returned if no message with the specified message identifier or correlation identifier is present in the queue. Instead, the process should wait until a message meeting the criteria is available. The process is still subject to the caller's transaction timeout, or, when not in transaction mode, the process is still subject to the timeout specified on the *TMQUEUE* process by the *-t* option.

If a message matching the desired criteria is not immediately available and the configured action resources are exhausted, *TPDEQUEUE* fails, *TP-STATUS* is set to *TPEDIAGNOSTIC*, and *DIAGNOSTIC* is set to *QMESYSTEM*.

Note that each *TPDEQUEUE()* request specifying the *TPQWAIT* control parameter requires that a queue manager (*TMQUEUE*) action object be available if a message satisfying the condition is not immediately available. If one is not available, the *TPDEQUEUE()* request fails. The number of available queue manager actions are specified when a queue space is

created or modified. When a waiting dequeue request completes, the associated action object associated is made available for another request.

TPQPEEK

If TPQPEEK is set, the specified message is read but not removed from the queue. The TPNOTRAN flag must be set. It is not possible to read messages enqueued or dequeued within a transaction before the transaction completes.

When a thread is non-destructively dequeuing a message using TPQPEEK, the message may not be seen by other non-blocking dequeuers for the brief time the system is processing the non-destructive dequeue request. This includes dequeuers using specific selection criteria (such as message identifier and correlation identifier) that are looking for the message currently being non-destructively dequeued.

On output from TPDEQUEUE (), the following elements may be set in *TPQUEDEF-REC*:

05 PRIORITY	PIC S9(9) COMP-5.
05 MSGID	PIC X(32).
05 CORRID	PIC X(32).
05 TPQUEQOS-DELIVERY-FLAG	PIC S9(9) COMP-5.
05 TPQUEQOS-REPLY-FLAG	PIC S9(9) COMP-5.
05 REPLYQUEUE	PIC X(15).
05 FAILUREQUEUE	PIC X(15).
05 DIAGNOSTIC	PIC S9(9) COMP-5.
05 CLIENTID OCCURS 4 TIMES	PIC S9(9) COMP-5
05 APPL-RETURN-CODE	PIC S9(9) COMP-5.
05 APPKEY	PIC S9(9) COMP-5.

The following is a list of valid settings in *TPQUEDEF-REC* controlling output information from TPDEQUEUE (). For any of these settings, if the setting is true when TPDEQUEUE () is called, the associated element in the record is populated with the value provided when the message was queued, and the setting remains true. If the value is not available (that is, no value was provided when the message was queued) or the setting is not true when TPDEQUEUE () is called, TPDEQUEUE () completes with the setting not true.

TPQPRIORITY

If this value is set, the call to TPDEQUEUE () is successful, and the message was queued with an explicit priority, then the priority is stored in PRIORITY. The priority is in the range 1 to 100, inclusive, and the higher the number, the higher the priority (that is, a message with a higher number is dequeued before a message with a lower number). If TPQNOPRIORITY is set, the priority is not available.

Note that if no priority was explicitly specified when the message was queued, the priority for the message is 50.

TPQMSGID

If this value is set and the call to `TPDEQUEUE()` is successful, the message identifier is stored in `MSGID`. The entire 32 bytes of the message identifier value are significant. If `TPQNOMSGID` is set, the message identifier is not available.

TPQCORRID

If this value is set, the call to `TPDEQUEUE()` is successful, and the message was queued with a correlation identifier, then the correlation identifier is stored in `CORRID`. The entire 32 bytes of the correlation identifier value are significant. Any BEA Tuxedo/Q provided reply to a message has the correlation identifier of the original message. If `TPQNOCORRID` is set, the correlation identifier is not available.

TPQDELIVERYQOS

If this value is set, the call to `TPDEQUEUE()` is successful, and the message was queued with a delivery quality of service, then the flag—`TPQQOSDELIVERYDEFAULTPERSIST`, `TPQQOSDELIVERYPERSISTENT`, or `TPQQOSDELIVERYNONPERSISTENT`—specified by `TPQEQOS-DELIVERY-FLAG` indicates the delivery quality of service. If `TPQNODELIVERYQOS` is set, the delivery quality of service is not available.

Note that if no delivery quality of service was explicitly specified when the message was queued, the default delivery policy of the target queue dictates the delivery quality of service for the message.

TPQREPLYQOS

If this value is set, the call to `TPDEQUEUE()` is successful, and the message was queued with a reply quality of service, then the flag—`TPQQOSREPLYDEFAULTPERSIST`, `TPQQOSREPLYPERSISTENT`, or `TPQQOSREPLYNONPERSISTENT`—specified by `TPQEQOS-REPLY-FLAG` indicates the reply quality of service. If `TPQNOREPLYQOS` is set, the reply quality of service is not available.

Note that if no reply quality of service was explicitly specified when the message was queued, the default delivery policy of the `REPLYQUEUE` queue dictates the delivery quality of service for any reply. The default delivery policy is determined when the reply to a message is enqueued. That is, if the default delivery policy of the reply queue is modified between the time that the original message is enqueued and the reply to the message is enqueued, the policy used is the one in effect when the reply is finally enqueued.

TPQREPLYQ

If this value is set, the call to `TPDEQUEUE()` is successful, and the message was queued with a reply queue, then the name of the reply queue is stored in `REPLYQUEUE`. Any reply

to the message should go to the named reply queue within the same queue space as the request message. If `TPQNOREPLYQ` is set, the reply queue is not available.

`TPQFAILUREQ`

If this value is set, the call to `TPDEQUEUE()` is successful, and the message was queued with a failure queue, then the name of the failure queue is stored in `FAILUREQUEUE`. Any failure message should go to the named failure queue within the same queue space as the request message. If `TPQNOFAILUREQ` is set, the failure queue is not available.

The remaining settings in `TPQUEDEF-REC` are set to the following values when `TPDEQUEUE()` is called: `TPQNOTOP`, `TPQNOBEFOREMSGID`, `TPQNOTIME_ABS`, `TPQNOTIME_REL`, `TPQNOEXPTIME_ABS`, `TPQNOEXPTIME_REL`, and `TPQNOEXPTIME_NONE`.

If the call to `TPDEQUEUE()` fails and `TP-STATUS` is set to `TPEDIAGNOSTIC`, a value indicating the reason for failure is returned in `DIAGNOSTIC`. The possible values are defined below in the `DIAGNOSTICS` section.

Additionally on output, if the call to `TPDEQUEUE()` is successful, `APPKEY` is set to the application authentication key, `CLIENTID` is set to the identifier for the client originating the request, and `APPL-RETURN-CODE` is set to the user-return code value that was set when the message was enqueued.

Return Values

Upon successful completion, `TPDEQUEUE()` sets `TP-STATUS` to `[TPOK]`.

Errors

Under the following conditions, `TPDEQUEUE()` fails and sets `TP-STATUS` to the following values (unless otherwise noted, failure does not affect the caller's transaction, if one exists):

`[TPEINVAL]`

Invalid arguments were given (for example, `QSPACE-NAME` is `SPACES` or settings in `TPQUEDEF-REC` are invalid).

`[TPENOENT]`

Cannot access the `QSPACE-NAME` because it is not available (that is, the associated [TMQUEUE \(5\)](#) server is not available), or cannot start a global transaction due to the lack of entries in the Global Transaction Table (GTT).

`[TPEOTYPE]`

Either the `REC-TYPE` and `SUB-TYPE` of the dequeued message are not known to the caller; or, `TPNOCHANGE` was set and the `REC-TYPE` and `SUB-TYPE` do not match the type and subtype of the dequeued message. Neither `DATA-REC` nor `TPTYPE-REC` are changed.

When the call is made in transaction mode and this error occurs, the transaction is marked abort-only, and the message remains on the queue.

[TPTRUNCATE]

The size of the incoming message is larger than the size specified in *LEN*. Only *LEN* amount of data was moved to *DATA-REC*, the remaining data is discarded.

[TPETIME]

This error code indicates that either a timeout has occurred or *TPDEQUEUE()* has been attempted, in spite of the fact that the current transaction is already marked rollback only.

If the caller is in transaction mode, then either the transaction is already rollback only or a transaction timeout has occurred. The transaction is marked abort-only. If the caller is not in transaction mode, a blocking timeout has occurred. (A blocking timeout can occur only if both *TPBLOCK* and *TPTIME* are specified.) In either case, no changes are made to *DATA-REC* or *TPTYPE-REC*.

If a transaction timeout has occurred, then, with one exception, any attempts to send new requests or receive outstanding replies will fail with *TPETIME* until the transaction has been aborted. The exception is a request that does not block, expects no reply, and is not sent on behalf of the caller's transaction (that is, *TPACALL()* with *TPNOTRAN*, *TPNOBLOCK*, and *TPNOREPLY* set).

When a service fails inside a transaction, the transaction is put into the *TX_ROLLBACK_ONLY* state. This state is treated, for most purposes, as though it were equivalent to a timeout. All further ATMI calls for this transaction (with the exception of those issued in the circumstances described in the previous paragraph) will fail with *TPETIME*.

[TPBLOCK]

A blocking condition exists and *TPBLOCK* was set.

[TPGOTSIG]

A signal was received and *TPNOSIGRSTRT* was set.

[TPPROTO]

TPDEQUEUE() was called improperly. There is no effect on the queue or the transaction.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file. There is no effect on the queue.

[TPEOS]

An operating system error has occurred. There is no effect on the queue.

[TPEDIAGNOSTIC]

Dequeuing a message from the specified queue failed. The reason for failure can be determined by the diagnostic value returned via *TPQUEDEF-REC*.

Diagnostics

The following diagnostic values are returned during the dequeuing of a message.

[QMEINVAL]

An invalid setting was specified.

[QMEBADRMID]

An invalid resource manager identifier was specified.

[QMENOTOPEN]

The resource manager is not currently open.

[QMETRAN]

The call was not in transaction mode or was made with *TPNOTRAN* set and an error occurred trying to start a transaction in which to dequeue the message. This diagnostic is not returned by a queue manager from BEA Tuxedo release 7.1 or later.

[QMEBADMSGID]

An invalid message identifier was specified for dequeuing.

[QMESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[QMEOS]

An operating system error has occurred.

[QMEABORTED]

The operation was aborted. When executed within a global transaction, the global transaction has been marked rollback-only. Otherwise, the queue manager aborted the operation.

[QMEPROTO]

A dequeue was done when the transaction state was not active.

[QMEBADQUEUE]

An invalid, deleted, or reserved queue name was specified.

[QMENOMSG]

No message was available for dequeuing. Note that it is possible that the message exists on the queue and another application process has read the message from the queue. In this case, the message may be put back on the queue if that other process rolls back the transaction.

[QMEINUSE]

When dequeuing a message by message identifier or correlation identifier, the specified message is in use by another transaction. Otherwise all messages currently on the queue are in use by other transactions. This diagnostic is not returned by a queue manager from BEA Tuxedo release 7.1 or later.

[QMESHARE]

When dequeuing a message from a specified queue, the specified queue is opened *exclusively* by another application. The other application is one based on a BEA product other than the BEA Tuxedo system that opened the queue for exclusive read and/or write using the Queuing Services API (QSAPI).

See Also

[qmadmin\(1\)](#), [TPENQUEUE\(3cbl\)](#), [TMQUEUE\(5\)](#)

TPDISCON(3cbl)**Name**

TPDISCON() - take down a conversational connection

Synopsis

```
01 TPSVCDEF-REC.
   COPY TPSVCDEF.

01 TPSTATUS-REC.
   COPY TPSTATUS.

CALL "TPDISCON" USING TPSVCDEF-REC TPSTATUS-REC.
```

Description

TPDISCON() immediately tears down the connection specified by COMM-HANDLE in TPSVCDEF-REC, the communications handle, and generates a TPEV-DISCONIMM event on the other end of the connection.

`TPDISCON()` can only be called by the initiator of the conversation. `TPDISCON()` can not be called within a conversational service on the communications handle with which it was invoked. Rather, a conversational service must use `TPRETURN()` to signify that it has completed its part of the conversation. Similarly, even though a program communicating with a conversational service can issue `TPDISCON()`, the preferred way is to let the service tear down the connection in `TPRETURN()`; doing so ensures correct results. If the initiator of the connection is a server, then `TPRETURN()` can also be used to cause an orderly disconnection. If the initiator of the connection is in a transaction, then `TPCOMMIT()` or `TPABORT()` can be used to cause an orderly disconnection.

`TPDISCON()` causes the connection to be torn down immediately (that is, abortive rather than orderly). Any data that has not yet reached its destination may be lost. `TPDISCON()` can be issued even when the program on the other end of the connection is participating in the caller's transaction. In this case, the transaction is aborted. Also, the caller does not need to have control of the connection when `TPDISCON()` is called.

Return Values

Upon successful completion, `TPDISCON()` sets `TP-STATUS` to `[TPOK]`.

Errors

Under the following conditions, `TPDISCON()` fails and sets `TP-STATUS` to:

[TPEBADDESC]

`COMM-HANDLE` is invalid or is the communications handle with which a conversational service was invoked.

[TPETIME]

This error code indicates that either a timeout has occurred or `TPDISCON()` has been attempted, in spite of the fact that the current transaction is already marked rollback only.

If the caller is in transaction mode, then either the transaction is already rollback only or a transaction timeout has occurred. The transaction is marked abort-only. (Note that calling `TPDISCON()` on a connection in the caller's transaction would have resulted in the transaction being marked abort-only, even if `TPDISCON()` had succeeded.)

If the caller is not in transaction mode, a blocking timeout has occurred. (A blocking timeout can occur only if both `TPBLOCK` and `TPTIME` are specified.)

If a transaction timeout has occurred, then, with one exception, any attempts to perform further conversational work, send new requests, or receive outstanding replies will fail with `TPETIME` until the transaction has been aborted. The exception is a request that does

not block, expects no reply, and is not sent on behalf of the caller's transaction (that is, `TPACALL()` with `TPNOTRAN`, `TPNOBLOCK`, and `TPNOREPLY` set).

When a service fails inside a transaction, the transaction is put into the `TX_ROLLBACK_ONLY` state. This state is treated, for most purposes, as though it were equivalent to a timeout. All further ATMI calls for this transaction (with the exception of those issued in the circumstances described in the previous paragraph) will fail with `TPETIME`.

[TPEPROTO]

`TPDISCON()` was called improperly.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file. The communications handle is no longer valid.

[TPEOS]

An operating system error has occurred. The communications handle is no longer valid.

See Also

[TPABORT\(3cbl\)](#), [TPCOMMIT\(3cbl\)](#), [TPCONNECT\(3cbl\)](#), [TPRECV\(3cbl\)](#), [TPRETURN\(3cbl\)](#), [TPSEND\(3cbl\)](#)

TPENQUEUE(3cbl)

Name

`TPENQUEUE()` - routine to enqueue a message

Synopsis

```
01 TPQUEDEF-REC.
   COPY TPQUEDEF.

01 TPTYPE-REC.
   COPY TPTYPE.

01 DATA-REC.
   COPY User data.

01 TPSTATUS-REC.
   COPY TPSTATUS.
```

CALL "TPENQUEUE" USING *TPQUEDEF-REC* *TPTYPE-REC* *DATA-REC* *TPSTATUS-REC*.

Description

`TPENQUEUE()` stores a message on the queue named by `QNAME` in the `QSPACE-NAME` queue space. A queue space is a collection of queues, one of which must be `QNAME`.

When the message is intended for a BEA Tuxedo ATMI server, the `QNAME` matches the name of a service provided by the server. The system-provided server, [TMQFORWARD \(5\)](#), provides a default mechanism for dequeuing messages from the queue and forwarding them to servers that provide a service matching the queue name. If the originator expects a reply, then the reply to the forwarded service request is stored on the originator's queue unless otherwise specified. The originator will dequeue the reply message at a subsequent time. Queues can also be used for a reliable message transfer mechanism between any pair of BEA Tuxedo ATMI processes (clients and/or servers). In this case, the queue name does not match a service name but some agreed upon name for transferring the message.

The data portion of a message is specified by `DATA-REC` and `LEN` in `TPTYPE-REC` specifies how much of `DATA-REC` to enqueue. Note that if `DATA-REC` is a record of a type that does not require a length to be specified, then `LEN` is ignored (and may be 0). If `REC-TYPE` in `TPTYPE-REC` is `SPACES`, `DATA-REC` and `LEN` are ignored and a message is enqueued with no data portion. The `REC-TYPE` and `SUB-TYPE`, both in `TPTYPE-REC`, must match one of the `REC-TYPES` and `SUB-TYPES` recognized by `QSPACE-NAME`.

The message is queued at the priority defined for `QSPACE-NAME` unless overridden by a previous call to `TPSPRIO()`.

If the caller is within a transaction and `TPTRAN` is set, the message is queued in transaction mode. This has the effect that if `TPENQUEUE()` returns successfully and the caller's transaction is committed successfully, then the message is guaranteed to be available subsequent to the transaction completing. If the caller's transaction is rolled back either explicitly or as the result of a transaction timeout or some communication error, then the message will be removed from the queue (that is, the placing of the message on the queue is also rolled back). It is not possible to enqueue then dequeue the same message within the same transaction.

The message is not queued in transaction mode if either the caller is not in transaction mode, or `TPNOTRAN` is set. Once `TPENQUEUE()` returns successfully, the submitted message is guaranteed to be in the queue. When not in transaction mode, if a communication error or a timeout occurs, the application will not know whether or not the message was successfully stored on the queue.

The order in which messages are placed on the queue is controlled by the application via *TPQUEDEF-REC* as described below; the default queue ordering is set when the queue is created.

The following is a list of valid settings in *TPQUEDEF-REC*.

TPNOTRAN

If the caller is in transaction mode and this setting is used, the message is not enqueued within the caller's transaction. A caller in transaction mode that sets this to `true` is still subject to the transaction timeout (and no other). If message enqueueing fails that was invoked with this setting, the caller's transaction is not affected. Either `TPNOTRAN` or `TPTRAN` must be set.

TPTRAN

If the caller is in transaction mode and this setting is used, the message is enqueued within the same transaction as the caller. The setting is ignored if the caller is not in transaction mode. Either `TPNOTRAN` or `TPTRAN` must be set.

TPNOBLOCK

The message is not enqueued if a blocking condition exists. If `TPNOBLOCK` is set and a blocking condition exists such as the internal buffers into which the message is transferred are full, the call fails and `TP-STATUS` is set to `TPEBLOCK`. If `TPNOBLOCK` is set and a blocking condition exists because the target queue is opened *exclusively* by another application, the call fails, `TP-STATUS` is set to `TPDIAGNOSTIC`, and the `DIAGNOSTIC` field of the *TPQUEDEF* record is set to `QMESHAPE`. In the latter case, the other application, which is based on a BEA product other than the BEA Tuxedo system, opened the queue for exclusive read and/or write using the Queuing Services API (QSAPI). Either `TPNOBLOCK` or `TPBLOCK` must be set.

TPBLOCK

When `TPBLOCK` is set and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). Either `TPNOBLOCK` or `TPBLOCK` must be set.

TPNOTIME

This setting signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur. Either `TPNOTIME` or `TPTIME` must be set.

TPTIME

This setting signifies that the caller will receive blocking timeouts if a blocking condition exists and the blocking time is reached. Either `TPNOTIME` or `TPTIME` must be set.

TPSIGRSTRT

If a signal interrupts any underlying system calls, the interrupted system call is reissued. Either `TPNOSIGRSTRT` or `TPSIGRSTRT` must be set.

TPNOSIGRSTRT

If a signal interrupts any underlying system calls, the interrupted system call is not restarted and the routine fails. Either `TPNOSIGRSTRT` or `TPSIGRSTRT` must be set.

Additional information about queuing the message can be specified via *TPQUEDEF-REC*. This information includes values to override the default queue ordering placing the message at the top of the queue or before an enqueued message; an absolute or relative time after which a queued message is made available; an absolute or relative time when a message expires and is removed from the queue; the quality of service for delivering the message; the quality of service that any replies to the message should use; a correlation identifier that aids in correlating a reply or failure message with the queued message; the name of a queue to which a reply should be enqueued; and the name of a queue to which any failure message should be enqueued.

Control Parameter

TPQUEDEF-REC is used by the application program to pass and retrieve information associated with enqueueing the message. Settings are used to indicate what elements in the record are valid.

On input to `TPENQUEUE()`, the following elements may be set in *TPQUEDEF-REC*:

05 DEQ-TIME	PIC S9(9) COMP-5.
05 PRIORITY	PIC S9(9) COMP-5.
05 MSGID	PIC X(32).
05 CORRID	PIC X(32).
05 TPQUEQOS-DELIVERY-FLAG	PIC S9(9) COMP-5.
05 TPQUEQOS-REPLY-FLAG	PIC S9(9) COMP-5.
05 EXP-TIME	PIC S9(9) COMP-5.
05 REPLYQUEUE	PIC X(15).
05 FAILUREQUEUE	PIC X(15).
05 APPL-RETURN-CODE	PIC S9(9) COMP-5.

The following values indicate what values are set in the *TPQUEDEF-REC*.

TPQTOP

Setting this value indicates that the queue ordering be overridden and the message placed at the top of the queue. This request may not be granted depending on whether or not the queue was configured to allow overriding the queue ordering. Set `TPQDEFAULT` to use default queue ordering. `TPQTOP`, `TPQBEFOREMSGID`, or `TPQDEFAULT` must be set.

TPQBEFOREMSGID

Setting this value indicates that the queue ordering be overridden and the message placed in the queue before the message identified by `MSGID`. This request may not be granted depending on whether or not the queue was configured to allow overriding the queue ordering. Set `TPQDEFAULT` to use default queue ordering. `TPQTOP`, `TPQBEFOREMSGID`, or `TPQDEFAULT` must be set.

Note that the entire 32 bytes of the message identifier value are significant, so the value identified by `MSGID` must be completely initialized (for example, padded with spaces).

TPQTIME-ABS

If this value is set, the message is made available after the time specified by `DEQ-TIME`. `DEQ-TIME` is an absolute time value as generated by `time(2)` or `mktime(3C)` (the number of seconds since 00:00:00 Universal Coordinated Time—UTC, January 1, 1970). Set `TPQNOTIME` if neither an absolute nor relative time value is set. `TPQTIME-ABS`, `TPQTIME-REL`, or `TPQNOTIME` must be set. The absolute time is determined by the clock on the machine where the queue manager process resides.

TPQTIME-REL

If this value is set, the message is made available after a time relative to the completion of the enqueueing operation. `DEQ-TIME` specifies the number of seconds to delay after the enqueueing completes before the submitted message should be available. Set `TPQNOTIME` if neither an absolute nor relative time value is set. `TPQTIME-ABS`, `TPQTIME-REL`, or `TPQNOTIME` must be set.

TPQPRIORITY

If this value is set, the priority at which the message should be enqueued is stored in `PRIORITY`. The priority must be in the range 1 to 100, inclusive. The higher the number, the higher the priority (that is, a message with a higher number is dequeued before a message with a lower number). For queues not ordered by priority, this value is informational. If `TPQNOPRIORITY` is set, the priority for the message is 50 by default.

TPQCORRID

If this value is set, the correlation identifier value specified in `CORRID` is available when a message is dequeued with `TPDEQUEUE()`. This identifier accompanies any reply or failure message that is queued so that an application can correlate a reply with a particular request. Set `TPQNOPRIORITY` if a correlation identifier is not available.

Note that the entire 32 bytes of the correlation identifier value are significant, so the value specified in `CORRID` must be completely initialized (for example, padded with spaces).

TPQREPLYQ

If this value is set, a reply queue named in `REPLYQUEUE` is associated with the queued message. Any reply to the message is queued to the named queue within the same queue space as the request message. Set `TPQNOREPLYQ` if a reply queue name is not available.

TPQFAILUREQ

If this value is set, a failure queue named in `FAILUREQUEUE` is associated with the queued message. If (1) the enqueued message is processed by `TMQFORWARD()`, (2) `TMQFORWARD` was started with the `-d` option, and (3) the service fails and returns a non-NULL reply, a failure message consisting of the reply and its associated `APPL-RETURN-CODE` in the `TPSTATUS` record is enqueued to the named queue within the same queue space as the original request message. Set `TPQNOFAILUREQ` if a failure queue name is not available.

TPQDELIVERYQOS

TPQREPLYQOS

If `TPQDELIVERYQOS` is set, the flags specified by `TPQUEQOS-DELIVERY-FLAG` control the quality of service for message delivery. One of the following mutually exclusive flags must be set: `TPQQOSDELIVERYDEFAULTPERSIST`, `TPQQOSDELIVERYPERSISTENT`, or `TPQQOSDELIVERYNONPERSISTENT`. If `TPQDELIVERYQOS` is not set, `TPQNODELIVERYQOS` must be set. When `TPQNODELIVERYQOS` is set, the default delivery policy of the target queue dictates the delivery quality of service for the message.

If `TPQREPLYQOS` is set, the flags specified by `TPQUEQOS-REPLY-FLAG` control the quality of service for reply message delivery for any reply. One of the following mutually exclusive flags must be set: `TPQQOSREPLYDEFAULTPERSIST`, `TPQQOSREPLYPERSISTENT`, or `TPQQOSREPLYNONPERSISTENT`. The `TPQREPLYQOS` flag is used when a reply is returned from messages processed by `TMQFORWARD`. Applications not using `TMQFORWARD` to invoke services may use the `TPQREPLYQOS` flag as a hint for their own reply mechanism.

If `TPQREPLYQOS` is not set, `TPQNOREPLYQOS` must be set. When `TPQNOREPLYQOS` is set, the default delivery policy of the `REPLYQUEUE` queue dictates the delivery quality of service for any reply. Note that the default delivery policy is determined when the reply to a message is enqueued. That is, if the default delivery policy of the reply queue is modified between the time that the original message is enqueued and the reply to the message is enqueued, the policy used is the one in effect when the reply is finally enqueued.

The valid `TPQUEQOS-DELIVERY-FLAG` and `TPQUEQOS-REPLY-FLAG` flags are:

`TPQQOSDELIVERYDEFAULTPERSIST`

`TPQQOSREPLYDEFAULTPERSIST`

These flags specify that the message is to be delivered using the default delivery policy specified on the target or reply queue.

TPQQOSDELIVERYPERSISTENT

TPQQOSREPLYPERSISTENT

These flags specify that the message is to be delivered in a persistent manner using the disk-based delivery method. When specified, these flags override the default delivery policy specified on the target or reply queue.

TPQQOSDELIVERYNONPERSISTENT

TPQQOSREPLYNONPERSISTENT

These flags specify that the message is to be delivered in a non-persistent manner using the memory-based delivery method; the message is queued in memory until it is dequeued. When specified, these flags override the default delivery policy specified on the target or reply queue.

If the caller is transactional, non-persistent messages are enqueued within the caller's transaction, however, non-persistent messages are lost if the system is shut down or crashes or the IPC shared memory for the queue space is removed.

TPQEXPTIME-ABS

If this value is set, the message has an absolute expiration time, which is the absolute time when the message will be removed from the queue.

The absolute expiration time is determined by the clock on the machine where the queue manager process resides.

The absolute expiration time is specified by the value stored in `EXP-TIME`. `EXP-TIME` must be set to an absolute time generated by `time(2)` or `mktime(3C)` (the number of seconds since 00:00:00 Universal Coordinated Time—UTC, January 1, 1970).

If an absolute time is specified that is earlier than the time of the enqueue operation, the operation succeeds, but the message is not counted for the purpose of calculating thresholds. If the expiration time is before the message availability time, the message is not available for dequeuing unless either the availability or expiration time is changed so that the availability time is before the expiration time. In addition, these messages are removed from the queue at expiration time even if they were never available for dequeuing. If a message expires during a transaction, the expiration does not cause the transaction to fail. Messages that expire while being enqueued or dequeued within a transaction are removed from the queue when the transaction ends. There is no acknowledgment that the message has expired.

One of the following must be set: `TPQEXPTIME-ABS`, `TPQEXPTIME-REL`, `TPQEXPTIME-NONE`, or `TPQNOEXPTIME`.

TPQEXPTIME-REL

If this value is set, the message has a relative expiration time, which is the number of seconds *after* the message arrives at the queue that the message is removed from the queue. The relative expiration time is specified by the value stored in `EXP-TIME`.

If the expiration time is before the message availability time, the message is not available for dequeuing unless either the availability or expiration time is changed so that the availability time is before the expiration time. In addition, these messages are removed from the queue at expiration time even if they were never available for dequeuing. The expiration of a message during a transaction does cause the transaction to fail. Messages that expire while being enqueued or dequeued within a transaction are removed from the queue when the transaction ends. There is no acknowledgment that the message has expired.

One of the following must be set: `TPQEXPTIME-ABS`, `TPQEXPTIME-REL`, `TPQEXPTIME-NONE`, or `TPQNOEXPTIME`.

TPQEXPTIME-NONE

Setting this value indicates that the message should not expire. This flag overrides any default expiration policy associated with the target queue. You can remove a message by dequeuing it or by deleting it via an administrative interface. One of the following must be set: `TPQEXPTIME-ABS`, `TPQEXPTIME-REL`, `TPQEXPTIME-NONE`, or `TPQNOEXPTIME`.

TPQNOEXPTIME

Setting this value specifies that the default expiration time associated with the target queue applies to the message. One of the following must be set: `TPQEXPTIME-ABS`, `TPQEXPTIME-REL`, `TPQEXPTIME-NONE`, or `TPQNOEXPTIME`.

Additionally, `APPL-RETURN-CODE` can be set with a user-return code. This value is returned to the application that dequeues the message.

On output from `TPENQUEUE()`, the following elements may be set in `TPQUEDEF-REC`:

```
05 MSGID          PIC X(32) .  
05 DIAGNOSTIC PIC S9(9) COMP-5.
```

The following is a valid setting in `TPQUEDEF-REC` controlling output information from `TPENQUEUE()`. If this setting is `true` when `TPENQUEUE()` is called, the /Q server [TMQUEUE\(5\)](#) populates the associated element in the record with a message identifier. If this setting is not `true` when `TPENQUEUE()` is called, `TMQUEUE()` does *not* populate the associated element in the record with a message identifier.

TPQMSGID

If this value is set and the call to `TPENQUEUE()` is successful, the message identifier is stored in `MSGID`. The entire 32 bytes of the message identifier value are significant, so the value stored in `MSGID` is completely initialized (for example, padded with NULL characters). The actual padding character used for initialization varies between releases of the BEA Tuxedo /Q component. If `TPQNOMSGID` is set, the message identifier is not available.

The remaining members of the control structure are not used on input to `TPENQUEUE()`.

If the call to `TPENQUEUE()` failed and `TP-STATUS` is set to `TPEDIAGNOSTIC`, a value indicating the reason for failure is returned in `DIAGNOSTIC`. The possible values are defined below in the `DIAGNOSTICS` section.

Return Values

Upon successful completion, `TPENQUEUE()` sets `TP-STATUS` to `[TPOK]`.

Errors

Under the following conditions, `TPENQUEUE()` fails and sets `TP-STATUS` to the following values (unless otherwise noted, failure does not affect the caller's transaction, if one exists).

[TPEINVAL]

Invalid arguments were given (for example, `QSPACE-NAME` is `SPACES` or settings in `TPQUEDEF-REC` are invalid).

[TPENOENT]

Cannot access the `QSPACE-NAME` because it is not available (that is, the associated [TMQUEUE \(5\)](#) server is not available), or cannot start a global transaction due to the lack of entries in the Global Transaction Table (GTT).

[TPETIME]

This error code indicates that either a timeout has occurred or `TPENQUEUE()` has been attempted, in spite of the fact that the current transaction is already marked rollback only.

If the caller is in transaction mode, then either the transaction is already rollback only or a transaction timeout has occurred. The transaction is marked abort-only. If the caller is not in transaction mode, a blocking timeout has occurred. (A blocking timeout can occur only if both `TPBLOCK` and `TPTIME` are specified.)

If a transaction timeout has occurred, then, with one exception, any attempts to send new requests or receive outstanding replies will fail with `TPETIME` until the transaction has been aborted. The exception is a request that does not block, expects no reply, and is not

sent on behalf of the caller's transaction (that is, `TPACALL()` with `TPNOTRAN`, `TPNOBLOCK`, and `TPNOREPLY` set).

When a service fails inside a transaction, the transaction is put into the `TX_ROLLBACK_ONLY` state. This state is treated, for most purposes, as though it were equivalent to a timeout. All further ATMI calls for this transaction (with the exception of those issued in the circumstances described in the previous paragraph) will fail with `TPETIME`.

[TPEBLOCK]

A blocking condition exists and `TPBLOCK` was set.

[TPGOTSIG]

A signal was received and `TPNOSIGRSTRT` was set.

[TPEPROTO]

`TPENQUEUE()` was called improperly. There is no effect on the queue or the transaction.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file. There is no effect on the queue.

[TPEOS]

An operating system error has occurred. There is no effect on the queue.

[TPEDIAGNOSTIC]

Enqueuing a message from the specified queue failed. The reason for failure can be determined by the diagnostic value returned via `TPQUEDEF-REC`.

Diagnostic Values

The following diagnostic values are returned during the enqueueing of a message.

[QMEINVAL]

An invalid setting was specified.

[QMEBADRMID]

An invalid resource manager identifier was specified.

[QMENOTOPEN]

The resource manager is not currently open.

[QMETRAN]

The call was not in transaction mode or was made with the `TPNOTRAN` setting and an error occurred trying to start a transaction in which to enqueue the message. This diagnostic is not returned by a queue manager from BEA Tuxedo release 7.1 or later.

[QMEBADMSGID]

An invalid message identifier was specified.

[QMESYSTEM]

A system error has occurred. The exact nature of the error is written to a log file.

[QMEOS]

An operating system error has occurred.

[QMEABORTED]

The operation was aborted. When executed within a global transaction, the global transaction has been marked rollback-only. Otherwise, the queue manager aborted the operation.

[QMEPROTO]

An enqueue was done when the transaction state was not active.

[QMEBADQUEUE]

An invalid, deleted, or reserved queue name was specified.

[QMENOSPACE]

Due to an insufficient resource, such as no space on the queue, the message with its required quality of service (persistent or non-persistent storage) was not enqueued. `QMENOSPACE` is returned when any of the following configured resources is exceeded: (1) the amount of disk (persistent) space allotted to the queue space, (2) the amount of memory (non-persistent) space allotted to the queue space, (3) the maximum number of simultaneously active transactions allowed for the queue space, (4) the maximum number of messages that the queue space can contain at any one time, (5) the maximum number of concurrent actions that the Queuing Services component can handle, or (6) the maximum number of authenticated users that may concurrently use the Queuing Services component.

[QMERELASE]

An attempt was made to enqueue a message to a queue manager that is from a version of the BEA Tuxedo system that does not support a newer feature.

[QMESHARE]

When enqueueing a message from a specified queue, the specified queue is opened *exclusively* by another application. The other application is one based on a BEA product

other than the BEA Tuxedo system that opened the queue for exclusive read and/or write using the Queuing Services API (QSAPI).

See Also

[qmadmin\(1\)](#), [TPDEQUEUE\(3cbl\)](#), [TPSPRIO\(3cbl\)](#), [TMQFORWARD\(5\)](#), [TMQUEUE\(5\)](#)

TPFORWAR(3cbl)

Name

TPFORWAR() - forward a BEA Tuxedo ATMI service request to another routine

Synopsis

```
01 TPSVCDEF-REC.
   COPY TPSVCDEF.

01 TPTYPE-REC.
   COPY TPTYPE.

01 DATA-REC.
   COPY User data.

01 TPSTATUS-REC.
   COPY TPSTATUS.

COPY TPFORWAR REPLACING TPSVCDEF-REC BY TPSVCDEF-REC
      TPTYPE-REC BY TPTYPE-REC
      DATA-REC BY DATA-REC
      TPSTATUS-REC BY TPSTAUS-REC
```

Description

TPFORWAR() allows a service routine to forward a client's request to another service routine for further processing. Since TPFORWAR() contains an EXIT PROGRAM statement, it should be called from within the same routine that was invoked to ensure correct return of control to the BEA Tuxedo ATMI dispatcher (that is, TPFORWAR() should not be invoked in a sub-program of the service routine since control would not return to the BEA Tuxedo ATMI dispatcher).

TPFORWAR() cannot be called from within a conversational service.

This routine forwards a request to the service named by *SERVICE-NAME* in *TPSVCDEF-REC* using data contained in *DATA-REC*. A service routine forwarding a request receives no reply. After the request is forwarded, the service routine returns to the BEA Tuxedo ATMI dispatcher and the server is free to do other work. Note that because no reply is expected from a forwarded request, the request may be forwarded without error to any service routine in the same executable as the service which forwarded the request.

If the service routine is in transaction mode, this routine puts the caller's portion of the transaction in a state where it may be completed when the originator of the transaction issues either `TPCOMMIT()` or `TPABORT()`. If a transaction was explicitly started with `TPBEGIN()` while in a service routine, the transaction must be ended with either `TPCOMMIT()` or `TPABORT()` before calling `TPFORWAR()`. Thus, all services in a "forward chain" are either all started in transaction mode or none are started in transaction mode.

The last server in a forward chain sends a reply back to the originator of the request using `TPRETURN()`. In essence, `TPFORWAR()` transfers to another server the responsibility of sending a reply back to the awaiting requester.

`TPFORWAR()` should be called after receiving all replies expected from service requests initiated by the service routine. Any outstanding replies which are not received will automatically be dropped by the BEA Tuxedo ATMI dispatcher upon receipt. In addition, the communications handle for those replies become invalid and the request is not forwarded to *SERVICE-NAME*.

DATA-REC is the record to be sent and *LEN* in *TPTYPE-REC* specifies the amount of data in *DATA-REC* that should be sent. Note that if *DATA-REC* is a record of a type that does not require a length to be specified, then *LEN* is ignored (and may be 0). If *REC-TYPE* in *TPTYPE-REC* is *SPACES*, *DATA-REC* and *LEN* are ignored and a request with zero length data is sent. If *REC-TYPE* is *STRING* and *LEN* is 0, then the request is sent with no data portion.

Since the service routine writer does not regain control after calling `TPFORWAR()`, a blocking send with signal restart is used (that is, `TPSIGRSTRT` is implied). Currently, settings in *TPSVCDEF-REC* are reserved for future use and any specified are ignored.

Return Values

A service routine does not return any value to its caller, the BEA Tuxedo ATMI dispatcher. Thus, *TP-STATUS* is not set.

Errors

If any errors occur either in the handling of the parameters passed to the routine or in its processing, a "failed" message is sent back to the original requester (unless no reply is to be sent). The existence of outstanding replies or subordinate connections, or the caller's transaction being

marked abort-only, qualify as failures which generate failed messages. Failed messages are detected by the requester with the `TPESVCERR()` error indication. When such an error occurs, the caller's data is not sent. Also, this error causes the caller's current transaction to be marked abort-only.

If a transaction timeout occurs, either during the service routine or while the request is being forwarded, the requester waiting for a reply with either `TPCALL()` or `TPGETRPLY()` will get a `TPETIME` error return. When a service fails inside a transaction, the transaction times out and is put into the `TX_ROLLBACK_ONLY` state. All further ATMI calls for that transaction will fail with `TPETIME`. The waiting requester will not receive any data. Service routines, however, are expected to terminate using either `TPRETURN()` or `TPFORWAR()`. A conversational service routine must use `TPRETURN()`; it cannot use `TPFORWAR()`.

If a service routine returns without using either `TPRETURN()` or `TPFORWAR()` or `TPFORWAR()` is called from a conversational server, the server will print a warning message in a log file and return a service error to the original requester. All open connections to subordinates will be disconnected immediately, and any outstanding asynchronous replies will be marked stale. If the server was in transaction mode at the time of failure, the transaction is marked abort-only. Note also that if either `TPRETURN()` or `TPFORWAR()` are used outside of a service routine (for example, in clients, or in `TPSVRINIT()` or `TPSVRDONE()`), then these routines simply return having no effect.

See Also

[TPCONNECT\(3cbl\)](#), [TPRETURN\(3cbl\)](#)

TPGBLKTIME(3cbl)

Name

`TPGBLKTIME()` - retrieves the blocktime value previously set by `TPSBLKTIME`

Synopsis

```
01 TPBLKDEF-REC.
    COPY TPBLKDEF.
01 TPSTATUS-REC.
    COPY TPSTATUS.

CALL "TPGBLKTIME" USING TPBLKDEF-REC TPSTATUS-REC.
```

Description

TPGBLKTIME() retrieves a previously set, per second, blocktime value and places this value in BLKTIME in TBLKDEF-REC. If TPGBLKTIME() specifies a blocktime flag value, and no such flag value has been set, the return value is 0. A blocktime flag value less than 0 produces an error.

The following is a list of valid TPBLKDEF-REC flag values:

TPBLK-NEXT

This value retrieves the per second blocktime value for the previously set TPSBLKTIME() using a TBLKNEXT value.

TPBLK-ALL

This value retrieves the per second blocktime value for the previously set TPSBLKTIME() using a TBLKALL value.

If TPGBLKTIME() does not specify a TPBLK-NEXT or TPBLK-ALL blocktime flag value, it returns the applicable blocktime value for the next blocking API set due to a previous TPSBLKTIME() call with the TPBLK-NEXT or TPBLK-ALL flag blocktime value, or a system-wide default blocktime value.

Note: When a workstation client calls TPGBLKTIME() without a TPBLK-NEXT or TPBLK-ALL blocktime flag value, the system-wide default blocktime value cannot be returned. A 0 value is returned instead.

Return Values

Upon successful completion, TPGBLKTIME() sets TP-STATUS to [TPOK] and returns the previously set blocktime, if any, in BLKTIME in TBLKDEF-REC. A BLKTIME 0 value indicates that there are no previously set input blocktime values.

Errors

Under the following conditions, TPGBLKTIME fails and sets TP-STATUS to one of the following values. The failure does not affect transaction timeout values

[TPEINVAL]

Invalid arguments were given. For example, a value other than TPBLK-NEXT or TPBLK-ALL was specified in TPBLKDEF-REC.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

See Also

[TPCALL\(3cbl\)](#), [TPCONNECT\(3cbl\)](#), [TPRECV\(3cbl\)](#), [TPSBLKTIME\(3cbl\)](#), [UBBCONFIG\(5\)](#)

TPGETCTXT(3cbl)

Name

`TPGETCTXT()` - retrieves a context identifier for the current application association

Synopsis

```
01 TPCONTEXTDEF-REC.  
   COPY TPCONTEXTDEF.  
  
01 TPSTATUS-REC.  
   COPY TPSTATUS.  
  
CALL "TPGETCTXT" USING TPCONTEXTDEF-REC TPSTATUS-REC.
```

Description

`TPGETCTXT()` retrieves an identifier that represents the current application context and places that identifier in `CONTEXT` in `TPCONTEXTDEF-REC`. Typically, a COBOL application:

1. Calls `TPINITIALIZE()` with the `TP-MULTI-CONTEXTS` flag set.
2. Calls `TPGETCTXT()` and saves the `TPCONTEXTDEF-REC`.
3. Calls `TPINITIALIZE()`, again with the `TP-MULTI-CONTEXTS` flag.
4. Calls `TPGETCTXT()` again and saves the returned context.
5. Calls `TPSETCTXT()` to switch back to the first context.

`TPGETCTXT()` may be called in single-context applications as well as in multi-context applications.

Return Values

Upon successful completion, `TPGETCTXT` sets `TP-STATUS` to `[TPOK]` and places the program's context identifier in `CONTEXT` in `TPCONTEXTDEF-REC`. `CONTEXT` is set to the current context ID, which may be represented by either:

- An actual context ID

- `TPNULLCONTEXT`, indicating that this program is not currently associated with a context

Note: `TPINVALIDCONTEXT` cannot be returned in COBOL programs because this value is possible only in multithreaded programs.

Errors

Upon failure, `TPGETCTXT` sets `TP-STATUS` to one of the following values.

`[TPEINVAL]`

Invalid arguments have been given.

`[TPESYSTEM]`

A BEA Tuxedo system error has occurred. The exact nature of the error has been written to a log file.

`[TPEOS]`

An operating system error has occurred.

See Also

[Introduction to the COBOL Application-Transaction Monitor Interface](#), `TPSETCTXT (3cbl)`

TPGETLEV(3cbl)

Name

`TPGETLEV()` - check if a BEA Tuxedo ATMI transaction is in progress

Synopsis

```
01  TPTRXLEV-REC.
```

```
   COPY  TPTRXLEV.
```

```
01  TPSTATUS-REC.
```

```
   COPY  TPSTATUS.
```

```
CALL "TPGETLEV" USING TPTRXLEV-REC TPSTATUS-REC.
```

Description

`TPGETLEV()` returns to the caller the current transaction level. Currently, the only levels defined are `TP-NOT-IN-TRAN` and `TP-IN-TRAN`.

Return Values

Upon successful completion, `TPGETLEV()` sets `TP-STATUS` to `[TPOK]` and sets values in `TPTRXLEV-REC` to either a `TP-NOT-IN-TRAN` to indicate that no transaction is in progress, or `TP-IN-TRAN` to indicate that a transaction is in progress.

Errors

Under the following conditions, `TPGETLEV()` fails and sets `TP-STATUS` to:

[TPEPROTO]

`TPGETLEV()` was called improperly.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

Notices

When using `TPBEGIN()`, `TPCOMMIT()`, and `TPABORT()` to delineate a BEA Tuxedo ATMI transaction, it is important to remember that only the work done by a resource manager that meets the XA interface (and is linked to the caller appropriately) has transactional properties. All other operations performed in a transaction are not affected by either `TPCOMMIT()` or `TPABORT()`. See [buildserver\(1\)](#) for details on linking resource managers that meet the XA interface into a server such that operations performed by that resource manager are part of a BEA Tuxedo ATMI transaction.

See Also

[TPABORT\(3cbl\)](#), [TPBEGIN\(3cbl\)](#), [TPCOMMIT\(3cbl\)](#), [TPSCMT\(3cbl\)](#)

TPGETRPLY(3cbl)

Name

`TPGETRPLY()` - get reply from asynchronous message

Synopsis

```
01  TPSVCDEF-REC.  
    COPY  TPSVCDEF.
```



```
01 TPTYPE-REC.
```

```
    COPY TPTYPE.
```

```
01 DATA-REC.
```

```
    COPY User data.
```

```
01 TPSTATUS-REC.
```

```
    COPY TPSTATUS.
```

```
CALL "TPGETRPLY" USING TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

Description

TPGETRPLY () returns a reply from a previously sent request. TPGETRPLY () either returns a reply for a particular request, or it returns any reply that is available. Both options are described below.

DATA-REC specifies where the reply is to be read into and, on input, LEN in TPTYPE-REC indicates the maximum number of bytes that should be moved into DATA-REC. Also, REC-TYPE in TPTYPE-REC must be specified. Upon successful return from TPGETRPLY (), LEN contains the actual number of bytes moved into DATA-REC, REC-TYPE and SUB-TYPE, both in TPTYPE-REC, contain the data's type and subtype, respectively. If the reply is larger than DATA-REC, then DATA-REC will contain only as many bytes as will fit in the record. The remainder of the reply is discarded and TPGETRPLY () sets TPTRUNCATE ().

If LEN is 0 upon successful return, then the reply has no data portion and DATA-REC was not modified. It is an error for LEN to be 0 on input.

The following is a list of valid settings in TPSVCDEF-REC.

TPGETANY

This setting signifies that TPGETRPLY () should ignore the communications handle indicated by COMM-HANDLE in TPSVCDEF-REC, return any reply available and set COMM-HANDLE to the communications handle for the reply returned. If no replies exist, TPGETRPLY () can wait for one to arrive. Either TPGETANY or TPGETHANDLE must be set.

TPGETHANDLE

This setting signifies that TPGETRPLY () should use the communications handle identified by COMM-HANDLE and return a reply available for that COMM-HANDLE. If no replies exist, TPGETRPLY () can wait for one to arrive. Either TPGETANY or TPGETHANDLE must be set.

TPNOCHANGE

When this value is set, the type of *DATA-REC* is not allowed to change. That is, the type and subtype of the reply record must match *REC-TYPE* and *SUB-TYPE*, respectively. Either *TPNOCHANGE* or *TPCHANGE* must be set.

TPCHANGE

The type and/or subtype of the reply record differs from *REC-TYPE* and *SUB-TYPE*, respectively, so long as the receiver recognizes the incoming record type. Either *TPNOCHANGE* or *TPCHANGE* must be set.

TPNOBLOCK

TPGETRPLY() does not wait for the reply to arrive. If the reply is available, then *TPGETRPLY()* gets the reply and returns. Either *TPNOBLOCK* or *TPBLOCK* must be set.

TPBLOCK

When *TPBLOCK* is specified and no data is available, the caller blocks until the reply arrives or a timeout occurs (either transaction or blocking timeout). Either *TPNOBLOCK* or *TPBLOCK* must be set.

TPNOTIME

This setting signifies that the caller is willing to block indefinitely for its reply and wants to be immune to blocking timeouts. Transaction timeouts may still occur. Either *TPNOTIME* or *TPTIME* must be set.

TPTIME

This setting signifies that the caller will receive blocking timeouts if a blocking condition exists and the blocking time is reached. Either *TPNOTIME* or *TPTIME* must be set.

TPSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is reissued. Either *TPNOSIGRSTRT* or *TPSIGRSTRT* must be set.

TPNOSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is not restarted and the call fails. Either *TPNOSIGRSTRT* or *TPSIGRSTRT* must be set.

Except as noted below, *COMM-HANDLE* is no longer valid after its reply is received.

Return Values

Upon successful completion, *TPGETRPLY()* sets *TP-STATUS* to *[TPOK]*. When *TP-STATUS* is set to *TPOK()* or *TPESVCFAIL()*, *APPL-RETURN-CODE* in *TPSTATUS-REC* contains an application-defined value that was sent as part of *TPRETURN()*. If the size of the incoming

message was larger than the size specified in `LEN` on input, `TPTRUNCATE()` is set and only `LEN` amount of data was moved to `DATA-REC`, the remaining data is discarded.

Errors

Under the following conditions, `TPGETRPLY()` fails and sets `TP-STATUS` as indicated below. Note that if `TPGETHANDLE` is set, then `COMM-HANDLE` is invalidated unless otherwise stated. If `TPGETANY` is set, then `COMM-HANDLE` identifies the communications handle for the reply on which the failure occurred; if an error occurred before a reply could be retrieved, then `COMM-HANDLE` is 0. Also, the failure does not affect the caller's transaction, if one exists, unless otherwise stated.

[TPEINVAL]

Invalid arguments were given (for example, settings in `TPSVCDEF-REC` are invalid).

[TPEOTYPE]

Either the type and subtype of the reply are not known to the caller; or, `TPNOCHANGE` was set and the `REC-TYPE` and `SUB-TYPE` do not match the type and subtype of the reply sent by the service. Neither `DATA-REC` nor `TPTYPE-REC` are changed. If the reply was to be received on behalf of the caller's current transaction, then the transaction is marked abort-only since the reply is discarded.

[TPEBADDESC]

`COMM-HANDLE` contains an invalid communications handle.

[TPETIME]

This error code indicates that either a timeout has occurred or `TPGETRPLY()` has been attempted, in spite of the fact that the current transaction is already marked rollback only.

If the caller is in transaction mode, then either the transaction is already rollback only or a transaction timeout has occurred. The transaction is marked abort-only. If the caller is not in transaction mode, a blocking timeout has occurred. (A blocking timeout can occur only if both `TPBLOCK` and `TPTIME` are specified.) In either case, no changes are made to `DATA-REC` or `TPTYPE-REC`. If `TPGETHANDLE` was set, `COMM-HANDLE` remains valid unless the caller is in transaction mode.

If a transaction timeout has occurred, then, with one exception, any attempts to send new requests or receive outstanding replies will fail with `TPETIME` until the transaction has been aborted. The exception is a request that does not block, expects no reply, and is not sent on behalf of the caller's transaction (that is, `TPACALL()` with `TPNOTRAN`, `TPNOBLOCK`, and `TPNOREPLY` set).

When a service fails inside a transaction, the transaction is put into the `TX_ROLLBACK_ONLY` state. This state is treated, for most purposes, as though it were

equivalent to a timeout. All further ATMI calls for this transaction (with the exception of those issued in the circumstances described in the previous paragraph) will fail with `TPETIME`.

[TPESVCFAIL]

The service routine sending the caller's reply called `TPRETURN()` with `TPFAIL()`. This is an application-level failure. The contents of the service's reply, if one was sent, is available in `DATA-REC`. `APPL-RETURN-CODE` contains an application-defined value that was sent as part of `TPRETURN()`. If the reply was received on behalf of the caller's transaction, then the transaction is marked abort-only. Note that regardless of whether the transaction has timed out, the only valid communications before the transaction is aborted are calls to `TPACALL()` with `TPNOREPLY`, `TPNOTRAN`, and `TPNOBLOCK` set.

[TPESVCERR]

An error was encountered by a service routine during its completion in `TPRETURN()` or `TPFORWAR()` (for example, bad arguments were passed). No reply data is returned when this error occurs (that is, neither `DATA-REC` nor `TPTYPE-REC` are changed). If the reply was received on behalf of the caller's transaction, then the transaction is marked abort-only. Note that regardless of whether the transaction has timed out, the only valid communications before the transaction is aborted are calls to `TPACALL()` with `TPNOREPLY`, `TPNOTRAN`, and `TPNOBLOCK` set.

[TPEBLOCK]

A blocking condition exists and `TPNOBLOCK` was specified. `COMM-HANDLE` remains valid.

[TPGOTSIG]

A signal was received and `TPSIGRSTRT` was not specified.

[TPEPROTO]

`TPGETRPLY()` was called improperly.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

See Also

[TPACALL\(3cbl\)](#), [TPCANCEL\(3cbl\)](#), [TPRETURN\(3cbl\)](#)

TPGETUNSOL(3cbl)

Name

TPGETUNSOL () - get unsolicited message

Synopsis

```

01  TPTYPE-REC.
    COPY TPTYPE.

01  DATA-REC.
    COPY User data.

01  TPSTATUS-REC.
    COPY TPSTATUS.

CALL "TPGETUNSOL" USING TPTYPE-REC DATA-REC TPSTATUS-REC.
```

Description

TPGETUNSOL () gets unsolicited messages that were sent via TPBROADCAST () or TPNOTIFY (). This routine may only be called from an unsolicited message handler.

Upon successful return, LEN IN TPTYPE_REC contains the actual number of bytes moved into DATA-REC. REC-TYPE and SUB-TYPE, both in TPTYPE-REC, contain the data's type and subtype, respectively. If the message is larger than DATA-REC, then DATA-REC will contain only as many bytes as will fit in the record. The remainder of the message is discarded and sets TPTRUNCATE (). If LEN is 0, upon successful completion, then the message has no data portion and DATA-REC was not modified.

It is an error for LEN to be 0 on input.

Return Values

Upon successful completion, TPGETUNSOL () sets TP-STATUS to [TPOK]. If the size of the incoming message was larger then the size specified in LEN on input, TPTRUNCATE () is set and only LEN amount of data was moved to DATA-REC, the remaining data is discarded.

Errors

Under the following conditions, TPGETUNSOL () fails and sets TP-STATUS to:

[TPEINVAL]

Invalid arguments were given.

[TPEPROTO]

TPGETUNSOL () was called improperly.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

See Also

[TPSETUNSOL\(3cbl\)](#)

TPGPRIO(3cbl)

Name

TPGPRIO () - get service request priority

Synopsis

```
01  TPPRIDEF-REC.  
    COPY  TPPRIDEF.
```

```
01  TPSTATUS-REC.  
    COPY  TPSTATUS.
```

```
CALL  "TPGPRIO"  USING  TPPRIDEF-REC  TPSTATUS-REC.
```

Description

TPGPRIO () returns the priority for the last request sent or received. Priorities can range from 1 to 100, inclusive, with 100 being the highest priority. TPGPRIO () may be called after TPCALL () or TPACALL (), (also TPENQUEUE () or TPDEQUEUE (), assuming the queued management facility is installed), and the priority returned is for the request sent. Also, TPGPRIO () may be called within a service routine to find out at what priority the invoked service was sent. TPGPRIO () may be called any number of times and will return the same value until the next request is sent.

Since the conversation primitives are not associated with priorities, issuing `TPSEND()` or `TPRECV()` has no effect on the priority returned by `TPGPRIO()`. Also, there is no priority associated with a conversational service routine unless a `TPCALL()` or `TPACALL()` is done within that service.

Return Values

Upon successful completion, `TPGPRIO()` sets `TP-STATUS` to `[TPOK]` and returns a request's priority in `PRIORITY` in `TPPRIDEF-REC`.

Errors

Under the following conditions, `TPGPRIO()` fails and sets `TP-STATUS` to:

[TPENOENT]

`TPGPRIO()` was called and no requests (via `TPCALL()` or `TPACALL()`) have been sent, or it is called within a conversational service for which no requests have been sent.

[TPEPROTO]

`TPGPRIO()` was called improperly.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

See Also

[TPACALL\(3cbl\)](#), [TPCALL\(3cbl\)](#), [TPDEQUEUE\(3cbl\)](#), [TPENQUEUE\(3cbl\)](#), [TPSPRIO\(3cbl\)](#)

TPINITIALIZE(3cbl)

Name

`TPINITIALIZE()` - joins a BEA Tuxedo ATMI application

Synopsis

```
01  TPINFDEF-REC.
    COPY TPINFDEF.
```

```
01  USER-DATA-REC PIC X(any-length).
```

```

01 TPSTATUS-REC.
   COPY TPSTATUS.

CALL "TPINITIALIZE" TPINFDEF-REC USER-DATA-REC TPSTATUS-REC.

```

Description

`TPINITIALIZE()` allows a client to join a BEA Tuxedo ATMI application. Before a client can use any of the BEA Tuxedo communication or transaction routines, it must first join a BEA Tuxedo ATMI application. `TPINITIALIZE()` has two modes of operation: single-context mode and multi-context mode, which will be discussed in greater detail below. Because calling `TPINITIALIE()` is optional when in single-context mode, a single-context client may also join an application by calling many ATMI routines (for example, `TPACALL()` or `TPCALL()`) which transparently call `TPINITIALIZE()` with default values for the members of `TPINFDEF-REC`. A client may want to call `TPINITIALIZE()` directly so that it can set the parameters described below. In addition, `TPINITIALIZE()` must be used when multi-context mode is required or when application authentication is required (see the description of the `SECURITY` keyword in [UBBCONFIG\(5\)](#)). After `TPINITIALIZE()` successfully returns, the client can initiate service requests and define transactions.

In single-context mode, if `TPINITIALIZE()` is called more than once (that is, after the client has already joined the application), no action is taken and success is returned.

Description of the TPINFDEF-REC Record

The `TPINFDEF-REC` record includes the following members.

```

05 USRNAME          PIC X(30) .
05 CLTNAME          PIC X(30) .
05 PASSWD           PIC X(30) .
05 GRPNAME          PIC X(30) .
05 NOTIFICATION-FLAG PIC S9(9) COMP-5.
   88 TPU-SIG        VALUE 1.
   88 TPU-DIP        VALUE 2.
   88 TPU-IGN        VALUE 3.
05 ACCESS-FLAG      PIC S9(9) COMP-5.
   88 TPSA-FASTPATH  VALUE 1.
   88 TPSA-PROTECTED VALUE 2.
05 CONTEXTS-FLAG    PIC S9(9) COMP-5.
   88 TP-SINGLE-CONTEXT VALUE 0.

```



```

88 TP-MULTI-CONTEXTS    VALUE 1.
05 DATALEN              PIC S9(9) COMP-5.

```

USRNAME is a name representing the caller. CLTNAME is a client name whose semantics are application defined. The value `sysclient` is reserved by the system for the CLTNAME field. The USRNAME and CLTNAME fields are associated with the client at `TPINITIALIZE()` time and are used for both broadcast notification and administrative statistics retrieval. PASSWD is an application password in unencrypted format that is used for validation against the application password. The PASSWD is significant up to 30 characters. GRPNAME is used to associate the client with a resource manager group name. If GRPNAME is SPACES, then the client is not associated with a resource manager and is in the default client group.

Single-context Mode Versus Multi-context Mode

`TPINITIALIZE()` has two modes of operation: single-context mode and multi-context mode. In single-context mode, a process may join at most one application at any one time. Single-context mode is specified by calling `TPINITIALIZE()` with the `TP-SINGLE-CONTEXT` setting of `CONTEXTS-FLAG` or by calling another function that invokes `TPINITIALIZE()` implicitly.

In single-context mode, if `TPINITIALIZE()` is called more than once (that is, after the client has already joined the application), no action is taken and success is returned.

Multi-context mode is entered by calling `TPINITIALIZE()` with the `TP-MULTI-CONTEXTS` setting of `CONTEXTS-FLAG`. In multi-context mode, each call to `TPINITIALIZE()` results in the creation of a separate application association.

An application association is a context that associates a process and a BEA Tuxedo application. A client may have associations with multiple BEA Tuxedo applications, and may also have multiple associations with the same application. All of a client's associations must be made to applications running the same release of the BEA Tuxedo system, and either all associations must be native clients or all associations must be Workstation clients.

For native clients, the value of the `TUXCONFIG` environment variable is used to identify the application to which the new association will be made. For Workstation clients, the value of the `WSNADDR` or `WSENVFILE` environment variable is used to identify the application to which the new association will be made. The context for the current COBOL process is set to the new association.

In multi-context mode the application can get a handle for the current context, by calling `TPGETCTXT()`, and pass that handle as a parameter to `TPSETCTXT()`, thus setting the context in which a particular COBOL process will operate.

Mixing single-context mode and multi-context mode is not allowed. Once an application has chosen one of these modes, calling `TPINITIALIZE()` in the other mode is not allowed unless `TPTERM()` is first called for all application associations.

TPINFDEF-REC Record Descriptions

The settings of *TPINFDEF-REC* are used to indicate both the client specific notification mechanism and the mode of system access. These settings may override the application default; however, in the event that they cannot, `TPINITIALIZE()` will print a warning in a log file, ignore the setting and return the application default setting in *TPINFDEF-REC* upon return from `TPINITIALIZE()`. For client notification, the possible settings are as follows:

TPU-SIG

Select unsolicited notification by signals. This setting is not allowed in conjunction with the `TP-MULTI-CONTEXTS` setting of *CONTEXTS-FLAG*.

TPU-DIP

Select unsolicited notification by dip-in.

TPU-IGN

Ignore unsolicited notification.

Only one of the above can be used at a time. If the client does not select a notification method, then the application default method will be set upon return from `TPINITIALIZE()`.

For setting the mode of system access, the possible settings are as follows:

TPSA-FASTPATH

Set system access to fastpath.

TPSA-PROTECTED

Set system access to protected.

Only one of the above can be used at a time. If the client does not select a notification method or a system access mode, then the application default method(s) will be set upon return from `TPINITIALIZE()`. See [UBBCONFIG\(5\)](#) for details on both client notification methods and system access modes.

DATALEN is the length of the application specific data that will be sent to the service. A *SPACES* value for *USERNAME* and *CLTNAME* is allowed for applications not making use of the application authentication feature of the BEA Tuxedo system. Currently, *GRPNAME* must be *SPACES*. Clients using this option will get defined in the BEA Tuxedo system with the following: default values for *USERNAME*, *CLTNAME*, and *GRPNAME*; default settings; and no application data.

Return Values

Upon successful completion, `TPINITIALIZE()` sets `TP-STATUS` to `[TPOK]`. Upon failure, `TPINITIALIZE()` leaves the calling process in its original context, returns `-1`, and sets `TP-STATUS` to indicate the error condition.

Errors

Upon failure, `TPINITIALIZE()` sets `TP-STATUS` to:

`[TPEINVAL]`

Invalid arguments were specified.

`[TPENOENT]`

The client cannot join the application because of space limitations.

`[TPEPERM]`

The client cannot join the application because it does not have permission to do so or because it has not supplied the correct application password. Permission may be denied based on an invalid application password, failure to pass application specific authentication or use of restricted names.

`[TPEPROTO]`

`TPINITIALIZE()` was called improperly. For example: (a) the caller is a server; (b) the `TP-MULTI-CONTEXTS` setting was specified in single-context mode; or (c) the `TP-MULTI-CONTEXTS` setting was not specified in multi-context mode.

`[TPESYSTEM]`

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

`[TPEOS]`

An operating system error has occurred.

Portability

The interfaces described in `TPINITIALIZE()` are supported on UNIX system and MS-DOS operating systems. However, signal-based notification is not supported on MS-DOS. If it is selected at `TPINITIALIZE()` time, then a `USERLOG()` message is generated and the method is automatically set to dip-in.

Environment Variables

TUXCONFIG

Is used within `TPINITIALIZE()` when invoked by a non-workstation native client. It indicates the application to which the client should connect. Note that this environment variable is referenced only when `TPINITIALIZE()` is called. Subsequent calls make use of the application context.

WSENVFILE

Is used within `TPINITIALIZE()` when invoked by a Workstation client. It indicates a file containing environment variable settings that should be set in the caller's environment. See [compilation\(5\)](#) for more details on environment variable settings necessary for Workstation clients. Note that this file is processed only when `TPINITIALIZE()` is called and not before.

WSNADDR

Is used within `TPINITIALIZE()` when invoked by a Workstation client. It indicates the network address(es) of the workstation listener that is to be contacted for access to the application. This variable is required for Workstation clients and is ignored for native clients.

TCP/IP addresses may be specified in the following forms:

```
//host.name:port_number  
//#. #. #. #:port_number
```

In the first format, the domain finds an address for *hostname* using the local name resolution facilities (usually DNS). *hostname* must be the local machine, and the local name resolution facilities must unambiguously resolve *hostname* to the address of the local machine.

In the second example, the "*#. #. #. #*" is in dotted-decimal format. In dotted-decimal format, each *#* should be a number from 0 to 255. This dotted-decimal number represents the IP address of the local machine.

In both of the above formats, *port_number* is the TCP port number at which the domain process will listen for incoming requests. *port_number* can either be a number between 0 and 65535 or a name. If *port_number* is a name, then it must be found in the network services database on your local machine.

The address can also be specified in hexadecimal format when preceded by the characters "0x". Each character after the initial "0x" is a number between 0 and 9 or a letter between A and F (case insensitive). The hexadecimal format is useful for arbitrary binary network addresses such as IPX/SPX or TCP/IP.

The address can also be specified as an arbitrary string. The value should be the same as that specified for the `NLSADDR` parameter in the `NETWORKS` section of the configuration file.

More than one address can be specified if desired by specifying a comma-separated list of pathnames for `WSNADDR`. Addresses are tried in order until a connection is established. Any member of an address list can be specified as a parenthesized grouping of pipe-separated network addresses. For example:

```
WSNADDR="(//m1.acme.com:3050|//m2.acme.com:3050),//m3.acme.com:3050"
```

For users running under Windows, the address string looks like the following:

```
set WSNADDR=(//m1.acme.com:3050^|//m2.acme.com:3050),//m3.acme.com:3050
```

Because the pipe symbol (`|`) is considered a special character in Windows, it must be preceded by a caret (`^`)—an escape character in the Windows environment—when it is specified on the command line. However, if `WSNADDR` is defined in an `envfile`, the BEA Tuxedo system gets the values defined by `WSNADDR` through the `tuxgetenv(3c)` function. In this context, the pipe symbol (`|`) is not considered a special character, so you do not need to escape it with a caret (`^`).

The BEA Tuxedo system randomly selects one of the parenthesized addresses. This strategy distributes the load randomly across a set of listener processes. Addresses are tried in order until a connection is established. Use the value specified in the application configuration file for the workstation listener to be called. If the value begins with the characters `"0x"`, it is interpreted as a string of hex-digits, otherwise it is interpreted as ASCII characters.

WSFADDR

Used within `TPINITIALIZE()` when invoked by a Workstation client. It specifies the network address used by the Workstation client when connecting to the workstation listener or workstation handler. This variable, along with the `WSFRANGE` variable, determines the range of TCP/IP ports to which a Workstation client will attempt to bind before making an outbound connection. This address must be a TCP/IP address. The port portion of the TCP/IP address represents the base address from which a range of TCP/IP ports can be bound by the Workstation client. The `WSFRANGE` variable specifies the size of the range. For example, if this address is `//mymachine.bea.com:30000` and `WSFRANGE` is 200, then all native processes attempting to make outbound connections from this *LMID* will bind a port on `mymachine.bea.com` between 30000 and 30200. If not set, this variable defaults to the empty string, which implies the operating system chooses a local port randomly.

WSFRANGE

Used within `TPINITIALIZE()` when invoked by a Workstation client. It specifies the range of TCP/IP ports to which a Workstation client process will attempt to bind before making an outbound connection. The `WSFADDR` parameter specifies the base address of the range. For example, if the `WSFADDR` parameter is set to `//mymachine.bea.com:30000` and `WSFRANGE` is set to 200, then all native processes attempting to make outbound connections from this *LMID* will bind a port on `mymachine.bea.com` between 30000 and 30200. The valid range is 1-65535. The default is 1.

WSDEVICE

Is used within `TPINITIALIZE()` when invoked by a Workstation client. It indicates the device name to be used to access the network. This variable is used by Workstation clients and ignored for native clients. Note that certain supported transport level network interfaces do not require a device name; for example, sockets and NetBIOS. Workstation clients supported by such interfaces need not specify `WSDEVICE`.

WSTYPE

Is used within `TPINITIALIZE()` when invoked by a Workstation client to negotiate encode/decode responsibilities with the native site. This variable is optional for Workstation clients and ignored for native clients.

WSRPLYMAX

Is used by `TPINITIALIZE()` to set the maximum amount of core memory that should be used for buffering application replies before they are dumped to file. The default value for this parameter varies with each instantiation. The instantiation specific programmer's guide should be consulted for further information.

TMMINENCRYPTBITS

Is used to establish the minimum level of encryption required to connect to the BEA Tuxedo system. "0" means no encryption, while "56" and "128" specify the encryption key length (in bits). If this minimum level of encryption cannot be met, link establishment will fail. The default is "0"

TMAXENCRYPTBITS

Is used to negotiate the level of encryption up to this level when connecting to the BEA Tuxedo system. "0" means no encryption, while "56" and "128" specify the encryption length (in bits). The default value is "128."

Warning

Signal-based notification is not allowed in multi-context mode. In addition, clients that select signal-based notification may not be able to receive signals from the system due to signal restrictions. When clients cannot receive signals, the system generates a log message that it is

switching notification for the selected client to dip-in and the client is notified then and thereafter via dip-in notification. See the description of the `NOTIFY` parameter in the `RESOURCES` section of [UBBCONFIG\(5\)](#) for a detailed discussion of notification methods.

Because signaling of clients is always done by the system, the behavior of notification is always consistent, regardless of where the originating notification call is made. Therefore to use signal-based notification:

- A native client must be running as an application administrator
- A Workstation client is not required to be running as the application administrator

The ID for the application administrator is identified in the configuration file for the application.

If signal-based notification is selected for a client, then certain ATMI calls may fail, returning `TPGOTSIG` due to receipt of an unsolicited message if `TPSIGRSTRT` is not specified.

See Also

[TPGETCTXT\(3cbl\)](#), [TPSETCTXT\(3cbl\)](#), [TPTERM\(3cbl\)](#)

TPKEYCLOSE(3cbl)

Name

`TPKEYCLOSE()` - close a previously opened key handle

Synopsis

```
01  TPKEYDEF-REC.
    COPY TPKEYDEF.

01  TPSTATUS-REC.
    COPY TPSTATUS.

CALL "TPKEYCLOSE" USING TPKEYDEF-REC TPSTATUS-REC.
```

Description

`TPKEYCLOSE()` releases a previously opened key handle and all resources associated with it. Any sensitive information, such as the principal's private key, is erased from memory.

The calling process must supply `KEY-HANDLE` in `TPKEYDEF-REC`. `KEY-HANDLE` is a key identifier returned by a previous call to `TPKEYOPEN()`.

Return Values

Upon successful completion, `TPKEYCLOSE()` sets *TP-STATUS* in *TPSTATUS-REC* to [TPOK].

Errors

Upon failure, `TPKEYCLOSE()` sets *TP-STATUS* in *TPSTATUS-REC* to one of the following values.

[TPEINVAL]

Invalid arguments were given. For example, *KEY-HANDLE* in *TPKEYDEF-REC* is not set correctly.

[TPESYSTEM]

An error occurred. Consult the system error log file for details.

See Also

[TPKEYGETINFO\(3cbl\)](#), [TPKEYOPEN\(3cbl\)](#), [TPKEYSETINFO\(3cbl\)](#)

TPKEYGETINFO(3cbl)

Name

`TPKEYGETINFO()` - get information associated with a key handle

Synopsis

```
01 TPKEYDEF-REC.  
   COPY TPKEYDEF.
```

```
01 ATTVALUE-REC.  
   COPY user data
```

```
01 TPSTATUS-REC.  
   COPY TPSTATUS.
```

```
CALL "TPKEYGETINFO" USING TPKEYDEF-REC ATTVALUE-REC TPSTATUS-REC.
```

Description

`TPKEYGETINFO()` reports information about a key handle. A key handle represents a specific principal's key and the information associated with it.

The calling process must supply *KEY-HANDLE* in *TPKEYDEF-REC*, which is a key identifier returned by a previous call to `TPKEYOPEN()`.

The attribute for which information is desired is identified by *ATTRIBUTE-NAME* in *TPKEYDEF-REC*. The attribute name may be padded with *SPACES* or *LOW-VALUES*. Some attributes are specific to a cryptographic service provider, but the following core set of attributes should be supported by all providers.

Attribute	Value
PRINCIPAL	The name identifying the principal associated with the key (key handle), represented as a NULL-terminated character string.
PKENCRYPT_ALG	An ASN.1 Distinguished Encoding Rules (DER) <i>object identifier</i> of the public key algorithm used by the key for public key encryption. The object identifier for RSA is identified in the following table.
PKENCRYPT_BITS	The key length of the public key algorithm (RSA modulus size). The value must be within the range of 512 to 2048 bits, inclusive.
SIGNATURE_ALG	An ASN.1 DER <i>object identifier</i> of the digital signature algorithm used by the key for digital signature. The object identifiers for RSA and DSA are identified in the following table.
SIGNATURE_BITS	The key length of the digital signature algorithm (RSA modulus size). The value must be within the range of 512 to 2048 bits, inclusive.
ENCRYPT_ALG	An ASN.1 DER <i>object identifier</i> of the symmetric key algorithm used by the key for bulk data encryption. The object identifiers for DES, 3DES, and RC2 are identified in the following table.
ENCRYPT_BITS	The key length of the symmetric key algorithm. The value must be within the range of 40 to 128 bits, inclusive. When an algorithm with a fixed key length is set in <i>ENCRYPT_ALG</i> , the <i>ENCRYPT_BITS</i> value is automatically set to the fixed key length. For example, if <i>ENCRYPT_ALG</i> is set to DES, the <i>ENCRYPT_BITS</i> value is automatically set to 56.

Attribute	Value
DIGEST_ALG	An ASN.1 DER <i>object identifier</i> of the message digest algorithm used by the key for digital signature. The object identifiers for MD5 and SHA-1 are identified in the following table.
PROVIDER	The name of the cryptographic service provider.
VERSION	The version number of the cryptographic service provider's software.

The ASN.1 DER algorithm object identifiers supported by the default public key implementation are given in the following table.

ASN.1 DER Algorithm Object Identifier	Algorithm
{ 0x06, 0x08, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x02, 0x05 }	MD5
{ 0x06, 0x05, 0x2b, 0x0e, 0x03, 0x02, 0x1a }	SHA1
{ 0x06, 0x09, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x01, 0x01 }	RSA
{ 0x06, 0x05, 0x2b, 0x0e, 0x03, 0x02, 0x0c }	DSA
{ 0x06, 0x05, 0x2b, 0x0e, 0x03, 0x02, 0x07 }	DES
{ 0x06, 0x08, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x03, 0x07 }	3DES
{ 0x06, 0x08, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x03, 0x02 }	RC2

The information associated with the specified attribute will be stored in user-defined `ATTVALUE-REC`, padded at the end with `SPACES`. The maximum amount of data that can be stored at this location is specified by the caller in `ATTRIBUTE-VALUE-LEN` in `TPKEYDEF-REC`.

After `TPKEYGETINFO()` completes, `ATTRIBUTE-VALUE-LEN` is set to the size of the data actually returned (not including padding values). If the number of bytes that need to be returned exceeds `ATTRIBUTE-VALUE-LEN`, `TPKEYGETINFO()` fails (with the `TPELIMIT` error code) and sets `ATTRIBUTE-VALUE-LEN` to the required amount of space.

Return Values

Upon successful completion, `TPKEYGETINFO()` sets *TP-STATUS* in *TPSTATUS-REC* to `[TPOK]`.

Errors

Upon failure, `TPKEYGETINFO()` sets *TP-STATUS* in *TPSTATUS-REC* to one of the following values:

`[TPEINVAL]`

Invalid arguments were given. For example, *KEY-HANDLE* is not a valid key.

`[TPESYSTEM]`

An error occurred. Consult the system error log file for details.

`[TPELIMIT]`

Insufficient space was provided to hold the requested attribute value.

`[TPENOENT]`

The requested attribute is not associated with this key.

See Also

[TPKEYCLOSE\(3cbl\)](#), [TPKEYOPEN\(3cbl\)](#), [TPKEYSETINFO\(3cbl\)](#)

TPKEYOPEN(3cbl)

Name

`TPKEYOPEN()` - open a key handle for digital signature generation, message encryption, or message decryption

Synopsis

```
01 TPKEYDEF-REC.
   COPY TPKEYDEF.
```

```
01 TPSTATUS-REC.
   COPY TPSTATUS.
```

```
CALL "TPKEYOPEN" USING TPKEYDEF-REC TPSTATUS-REC.
```

Description

`TPKEYOPEN()` makes a key handle available to the calling process. A key handle represents a specific principal's key and the information associated with it.

A key may be used for one or more of the following purposes:

- Automatically generating a digital signature, which protects a message's content and proves that a specific principal originated the message. (A principal may be a person or a process.) This type of key is a private key and is available only to the key's owner.

Calling `TPKEYOPEN()` with the principal's name and the `TPKEY-SIGNATURE` and `TPKEY-AUTOSIGN` settings returns a handle to the principal's public key and enables signature generation in `AUTOSIGN` mode. The public key software generates and attaches the digital signature to the message just before the message is sent.

- Verifying a digital signature, which proves that a message's content remains unaltered and that a specific principal originated the message.

Signature verification does not require a call to `TPKEYOPEN()`; the verifying process uses the public key specified in the digital certificate accompanying the digitally signed message to verify the signature.

- Automatically encrypting a message destined for a specific principal. This type of key is available to any process with access to the principal's public key and digital certificate.

Calling `TPKEYOPEN()` with the principal's name and the `TPKEY-ENCRYPT` and `TPKEY-AUTOENCRYPT` settings returns a handle to the principal's public key (via the principal's digital certificate) and enables encryption in `AUTOENCRYPT` mode. The public key software encrypts the message and attaches an encryption envelope to the message just before the message is sent; the encryption envelope enables the receiving process to decrypt the message.

- Decrypting a message intended for a specific principal. This type of key is a private key and is available only to the key's owner.

Calling `TPKEYOPEN()` with the principal's name and the `TPKEY-DECRYPT` setting returns a handle to the principal's private key and digital certificate.

The key handle returned by `TPKEYOPEN()` is stored in `KEY-HANDLE` in `TPKEYDEF-REC`.

The calling process must supply `PRINCIPAL-NAME` in `TPKEYDEF-REC`, which specifies the key owner's identity. This name may be padded at the end with `SPACES` or `LOW-VALUES`. If `PRINCIPAL-NAME` is all `SPACES` or `LOW-VALUES`, a default identity is assumed. The default identity may be based on the current login session, the current operating system account, or another attribute such as a local hardware device.

The calling process may have to supply *LOCATION* in *TPKEYDEF-REC*, which specifies the location of a key owner's identity. If the underlying provider does not require a location field, this field may be populated with *SPACES* or *LOW-VALUES*.

To authenticate the identity of *PRINCIPAL-NAME*, proof material such as a password or pass phrase may be required. If required, the proof material should be stored in *IDENTITY-PROOF* in *TPKEYDEF-REC*. Otherwise, this field may be populated with *SPACES* or *LOW-VALUES*.

The length of the proof material (in bytes) is specified by *PROOF-LEN* in *TPKEYDEF-REC*. If *PROOF-LEN* is 0, *IDENTITY-PROOF* is assumed to be a character string padded at the end with *SPACES* or *LOW-VALUES*, in which case trailing *SPACES* or *LOW-VALUES* are not considered part of the proof material.

There may be a choice of cryptographic service providers, based on the local machine's configuration and operating environment. If you need to choose one, set *CRYPTO-PROVIDER* in *TPKEYDEF-REC* to the name of the required provider. Otherwise, set this field to *SPACES* or *LOW-VALUES*, and a default provider will be assumed.

The type of key access required for a key's mode of operation is determined by specifying one or more of the following settings in *TPKEYDEF-REC*.

TPKEY-SIGNATURE:

This private key is available to generate digital signatures.

TPKEY-AUTOSIGN:

Whenever this process transmits a message, the public key software uses the signer's private key to generate a digital signature and then attaches the digital signature to the message.

TPKEY-ENCRYPT:

This public key is available to identify the recipient of an encrypted message.

TPKEY-AUTOENCRYPT:

Whenever this process transmits a message, the public key software encrypts the message, uses the recipient's public key to generate an encryption envelope, and then attaches the encryption envelope to the message.

TPKEY-DECRYPT:

This private key is available for decryption.

Various combinations of these settings are allowed. If a key is used only for encryption (*TPKEY-ENCRYPT* and *TPKEY-AUTOENCRYPT*), *IDENTITY-PROOF* is not required.

Return Values

Upon successful completion, `TPKEYOPEN()` sets *TP-STATUS* in *TPSTATUS-REC* to `[TPOK]`. In addition, *KEY-HANDLE* in *TPKEYDEF-REC* is set to a value that represents this key, for use by other functions such as `TPKEYGETINFO()`.

Errors

Upon failure, `TPKEYOPEN()` sets *TP-STATUS* in *TPSTATUS-REC* to one of the following values:

[TPEINVAL]

Invalid arguments were given. For example, the settings (flag) values are not set correctly.

[TPEPERM]

Permission failure. The cryptographic service provider was not able to access a private key for this principal, given the proof information and current environment.

[TPESYSTEM]

An error occurred. Consult the system error log file for details.

See Also

[TPKEYCLOSE\(3cbl\)](#), [TPKEYGETINFO\(3cbl\)](#), [TPKEYSETINFO\(3cbl\)](#)

TPKEYSETINFO(3cbl)

Name

`TPKEYSETINFO()` - set optional parameters associated with a key handle

Synopsis

```
01 TPKEYDEF-REC.  
   COPY TPKEYDEF.
```

```
01 ATTVALUE-REC.  
   COPY user data
```

```
01 TPSTATUS-REC.  
   COPY TPSTATUS.
```

```
CALL "TPKEYSETINFO" USING TPKEYDEF-REC ATTVALUE-REC TPSTATUS-REC.
```

Description

`TPKEYSETINFO()` sets an optional attribute parameter for a key handle. A key handle represents a specific principal's key and the information associated with it.

The key for which information is to be modified is identified by *KEY-HANDLE* in *TPKEYDEF-REC*. *KEY-HANDLE* is a key identifier returned by a previous call to `TPKEYOPEN()`.

The attribute for which information is to be modified is identified by *ATTRIBUTE-NAME* in *TPKEYDEF-REC*. The attribute name may be padded with *SPACES* or *LOW-VALUES*. Some attributes may be specific to a certain cryptographic service provider, but the core set of attributes presented on the [TPKEYGETINFO\(3cbl\)](#) reference page should be supported by all providers.

The information in user-defined *ATTVALUE-REC* is to be associated with *ATTRIBUTE-NAME*. Upon successful completion of `TPKEYSETINFO()`, the information in *ATTVALUE-REC* is stored or processed in a manner defined by the cryptographic service provider. If the data content of *ATTVALUE-REC* is self-describing, *ATTRIBUTE-VALUE-LEN* in *TPKEYDEF-REC* is ignored (and may be 0). Otherwise, *ATTRIBUTE-VALUE-LEN* must contain the length of data in *ATTVALUE-REC*.

Return Values

Upon successful completion, `TPKEYSETINFO()` sets *TP-STATUS* in *TPSTATUS-REC* to `[TPOK]`.

Errors

Upon failure, `TPKEYSETINFO()` sets *TP-STATUS* in *TPSTATUS-REC* to one of the following values:

`[TPEINVAL]`

Invalid arguments were given. For example, *KEY-HANDLE* is not set correctly.

`[TPESYSTEM]`

An error occurred. Consult the system error log file for more details.

`[TPELIMIT]`

The attribute value provided is too large.

`[TPENOENT]`

The requested attribute is not recognized by the key's cryptographic service provider.

See Also

[TPKEYCLOSE\(3cbl\)](#), [TPKEYGETINFO\(3cbl\)](#), [TPKEYOPEN\(3cbl\)](#)

TPNOTIFY(3cbl)

Name

TPNOTIFY () - send notification by client identifier

Synopsis

```
01 TPSVCDEF-REC.  
   COPY TPSVCDEF.
```

```
01 TPTYPE-REC.  
   COPY TPTYPE.
```

```
01 DATA-REC.  
   COPY User data.
```

```
01 TPSTATUS-REC.  
   COPY TPSTATUS.
```

```
CALL "TPNOTIFY" USING TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

Description

TPNOTIFY () allows a server to send an unsolicited message to an individual client.

CLIENTID in *TPSVCDEF-REC* contains a client identifier saved from the *TPSVCDEF-REC* of a previous or current service invocation.

DATA-REC is the record to be sent and LEN in *TPTYPE-REC* specifies how much of *DATA-REC* should be sent. If *DATA-REC* is a record of type that does not require a length to be specified, then LEN is ignored (and may be 0). If REC-TYPE in *TPTYPE-REC* is SPACES, *DATA-REC* and LEN are ignored and a request is sent with no data portion.

Upon successful return from TPNOTIFY () , the message has been delivered to the system for forwarding to the identified client. If TPACK () was set, then a successful return means the message has been received by the client. Furthermore, if the client has registered an unsolicited message handler, the handler will have been called.

The following is a list of valid settings in *TPSVCDEF-REC*.

TPNOBLOCK

The request is not sent if a blocking condition exists (for example, the internal buffers into which the message is transferred are full). Either `TPNOBLOCK` or `TPBLOCK` must be set.

TPBLOCK

If a blocking condition exists in sending the notification, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). Either `TPNOBLOCK` or `TPBLOCK` must be set.

TPNOTIME

This setting signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur. Either `TPNOTIME` or `TPTIME` must be set.

TPTIME

This setting signifies that the caller will receive blocking timeouts if a blocking condition exists and the blocking time is reached. Either `TPNOTIME` or `TPTIME` must be set.

TPSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is reissued. Either `TPNOSIGRSTRT` or `TPSIGRSTRT` must be set.

TPNOSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is not restarted and the call fails. Either `TPNOSIGRSTRT` or `TPSIGRSTRT` must be set.

TPACK

This setting signifies that the caller will block waiting for an acknowledgment from the client. Either `TPNOACK()` or `TPACK()` must be set.

TPNOACK

This setting signifies that the caller will not block waiting for an acknowledgment from the client. Either `TPNOACK()` or `TPACK()` must be set.

Return Values

Upon successful completion, `TPNOTIFY()` sets `TP-STATUS` to `[TPOK]`.

Errors

Under the following conditions, `TPNOTIFY()` fails and sets `TP-STATUS` to:

`[TPEINVAL]`

Invalid arguments were given.

[TPENOENT]

The target client does not exist and `TPACK()` was set.

[TPETIME]

A blocking timeout occurred. A blocking timeout can occur under either of the following circumstances: (a) `TPBLOCK` and `TPTIME` are specified, or (b) `TPACK` and `TPTIME` are specified (in which case no acknowledgment is received).

[TPEBLOCK]

A blocking condition was found on sending the notification and `TPNOBLOCK` was specified.

[TPGOTSIG]

A signal was received and `TPSIGRSTRT` was not specified.

[TPEPROTO]

`TPNOTIFY()` was called in an improper context (for example, within a client).

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

[TPERELEASE]

When `TPACK()` is specified and the target is a client from a prior release of the BEA Tuxedo system that does not support the acknowledgment protocol.

See Also

[TPBROADCAST\(3cbl\)](#), [TPCHKUNSOL\(3cbl\)](#), [TPINITIALIZE\(3cbl\)](#), [TPSETUNSOL\(3cbl\)](#),
[TPTERM\(3cbl\)](#)

TPOPEN(3cbl)

Name

`TPOPEN()` - open the BEA Tuxedo ATMI resource manager

Synopsis

```
01  TPSTATUS-REC.  
    COPY TPSTATUS.
```

```
CALL "TPOPEN" USING TPSTATUS-REC.
```

Description

TPOPEN() opens the resource manager to which the caller is linked. At most one resource manager can be linked to the caller. This routine is used in place of resource manager-specific open() calls and allows a service routine to be free of calls that may hinder portability. Since resource managers differ in their initialization semantics, the specific information needed to open a particular resource manager is placed in a configuration file.

If a resource manager is already open (that is, TPOPEN() is called more than once), no action is taken and success is returned.

Return Values

Upon successful completion, TPOPEN() sets TP-STATUS to [TPOK]. More information concerning the reason a resource manager failed to open can be gotten by interrogating the resource manager in its own specific manner. Note that any calls to determine the exact nature of a resource manager's error hinder portability.

Errors

Under the following conditions, TPOPEN() fails and sets TP-STATUS to:

[TPERMERR]

A resource manager failed to open correctly. More information concerning the reason a resource manager failed to open can be obtained by interrogating a resource manager in its own specific manner. Note that any calls to determine the exact nature of the error hinder portability.

[TPEPROTO]

TPOPEN() was called in an improper context (for example, by a client that has not joined a BEA Tuxedo ATMI server group).

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

See Also

[TPCLOSE\(3cbl\)](#)

TPPOST(3cbl)

Name

TPPOST() - post an event

Synopsis

```
01 TPEVTDEF-REC.  
   COPY TPEVTDEF.
```

```
01 TPTYPE-REC.  
   COPY TPTYPE.
```

```
01 DATA-REC.  
   COPY User data.
```

```
01 TPSTATUS-REC.  
   COPY TPSTATUS.
```

```
CALL "TPPOST" USING TPEVTDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

Description

The caller uses TPPPOST() to post an event and any accompanying data. The event is named by *EVENT-NAME* in *TPEVTDEF-REC* and *DATA-REC* contains the data to be posted. The posted event and its data are dispatched by the BEA Tuxedo EventBroker to all subscribers whose subscriptions successfully evaluate against *EVENT-NAME* and whose optional filter rules successfully evaluate against *DATA-REC*.

EVENT-NAME must be 31 characters or less, but cannot be SPACES. *EVENT-NAME*'s first character cannot be a dot (".") as this character is reserved as the starting character for all events defined by the BEA Tuxedo system itself.

DATA-REC is the typed record to be posted and *LEN* in *TPTYPE-REC* specifies the amount of data in *DATA-REC* that should be posted with the event. Note that if *DATA-REC* is a record of a type that does not require a length to be specified, then *LEN* is ignored (and may be 0). If *DATA-REC* is a record of a type that does require a length to be specified, then *LEN* must not be 0 (if it is 0, no data will be posted). If *REC-TYPE* in *TPTYPE-REC* is SPACES, *DATA-REC* and *LEN* are ignored and the event is posted with no data.

When `TPPOST()` is used within a transaction, the transaction boundary can be extended to include those servers and/or stable-storage message queues notified by the EventBroker. When a transactional posting is made, some of the recipients of the event posting are notified on behalf of the poster's transaction (for example, servers and queues), while some are not (for example, clients).

If the poster is within a transaction and `TPTRAN` is set, the posted event goes to the EventBroker in transaction mode such that it dispatches the event as part of the poster's transaction. The broker dispatches transactional event notifications only to those service routine and stable-storage queue subscriptions that had `TPEVTRAN` set in `TPEVTDEF-REC` when the subscription was made. Client notifications, and those service routine and stable-storage queue subscriptions that had `TPEVNOTRAN` set in `TPEVTDEF-REC` when the subscription was made, are also dispatched by the EventBroker but not as part of the posting process' transaction.

The following is a list of valid settings in `TPEVTDEF-REC`:

`TPNOTRAN`

If the caller is in transaction mode and this setting is used, then the event posting is not made on behalf of the caller's transaction. A caller in transaction mode that uses this setting is still subject to the transaction timeout (and no other). If the event posting fails, the caller's transaction is not affected. Either `TPNOTRAN` or `TPTRAN` must be set.

`TPTRAN`

If the caller is in transaction mode and this setting is used, then the event posting is made on behalf of the caller's transaction. This setting is ignored if the caller is not in transaction mode. Either `TPNOTRAN` or `TPTRAN` must be set.

`TPNOREPLY`

Informs `TPPOST()` not to wait for the EventBroker to process all subscriptions for `EVENT-NAME` before returning. When `TPNOREPLY` is set, `EVENT-COUNT` in `TPEVTDEF-REC` is set to zero regardless of whether `TPPOST()` returns successfully or not. When the caller is in transaction mode, this setting cannot be used when `TPTRAN` is also set. Either `TPNOREPLY` or `TPREPLY` must be set.

`TPREPLY`

Informs `TPPOST()` to wait for all subscriptions to be processed before returning. When `TPREPLY` is set, the routine returns [`TPOK`] on success and sets `EVENT-COUNT` in `TPEVTDEF-REC` to the number of event notifications dispatched by the EventBroker on behalf of `EVENT-NAME`. When the caller is in transaction mode, this setting must be used when `TPTRAN` is also set. Either `TPNOREPLY` or `TPREPLY` must be set.

TPNOBLOCK

The event is not posted if a blocking condition exists. If such a condition occurs, the call fails and sets `TP-STATUS` to `[TPBLOCK]`. Either `TPNOBLOCK` or `TPBLOCK` must be set.

TPBLOCK

When `TPBLOCK` is specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). Either `TPNOBLOCK` or `TPBLOCK` must be set.

TPNOTIME

This setting signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur. Either `TPNOTIME` or `TPTIME` must be set.

TPTIME

This setting signifies that the caller will receive blocking timeouts if a blocking condition exists and the blocking time is reached. Either `TPNOTIME` or `TPTIME` must be set.

TPSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is reissued. Either `TPNOSIGRSTRT` or `TPSIGRSTRT` must be set.

TPNOSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is not restarted, the call fails and sets `TP-STATUS` to `[TPGOTSIG]`. Either `TPNOSIGRSTRT` or `TPSIGRSTRT` must be set.

Return Values

Upon successful completion, `TPPOST()` sets `TP-STATUS` to `[TPOK]`. In addition, `EVENT-COUNT` contains the number of event notifications dispatched by the EventBroker on behalf of `EVENT-NAME` (that is, postings for those subscriptions whose event expression evaluated successfully against `EVENT-NAME` and whose filter rule evaluated successfully against `DATA-REC`). Upon return where `TP-STATUS` is set to `[TPESVCFAIL]`, `EVENT-COUNT` contains the number of non-transactional event notifications dispatched by the EventBroker on behalf of `EVENT-NAME`.

Errors

Under the following conditions, `TPPOST()` fails and sets `TP-STATUS` to one of the following values. (Unless otherwise noted, failure does not affect the caller's transaction, if one exists.)

[TPEINVAL]

Invalid arguments were given (for example, `EVENT-NAME` is `SPACES`).

[TPENOENT]

Cannot access the BEA Tuxedo User EventBroker.

[TPETRAN]

The caller is in transaction mode, TPTRAN was set, and TPPOST() contacted an EventBroker that does not support transaction propagation (that is, TMUSREVT(5) is not running in a BEA Tuxedo ATMI group that supports transactions).

[TPETIME]

This error code indicates that either a timeout has occurred or TPPOST() has been attempted, in spite of the fact that the current transaction is already marked rollback only.

If the caller is in transaction mode, then either the transaction is already rollback only or a transaction timeout has occurred. The transaction is marked abort-only. If the caller is not in transaction mode, a blocking timeout has occurred. (A blocking timeout can occur only if both TPBLOCK and TPTIME are specified.)

If a transaction timeout has occurred, then, with one exception, any attempts to send new requests or receive outstanding replies will fail with TPETIME until the transaction has been aborted. The exception is a request that does not block, expects no reply, and is not sent on behalf of the caller's transaction (that is, TPACALL() with TPNOTRAN, TPNOBLOCK, and TPNOREPLY set).

When TPPOST() fails inside a transaction, the transaction is put into the TX_ROLLBACK_ONLY state. This state is treated, for most purposes, as though it were equivalent to a timeout. All further ATMI calls for this transaction (with the exception of those issued in the circumstances described in the previous paragraph) will fail with TPETIME.

[TPESVCFAIL]

The EventBroker encountered an error posting a transactional event to either a service routine or to a stable storage queue on behalf of the caller's transaction. The caller's current transaction is marked abort-only. When this error is returned, EVENT-COUNT contains the number of non-transactional event notifications dispatched by the EventBroker on behalf of EVENT-NAME; transactional postings are not counted since their effects will be aborted upon completion of the transaction. Note that so long as the transaction has not timed out, further communication may be performed before aborting the transaction and that any work performed on behalf of the caller's transaction will be aborted upon transaction completion (that is, for subsequent communication to have any lasting effect, it should be done with TPNOTRAN set).

[TPEBLOCK]

A blocking condition exists and TPNOBLOCK was specified.

[TPGOTSIG]

A signal was received and `TPNOSIGRSTRT` was specified.

[TPEPROTO]

`TPPOST()` was called improperly.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

See Also

[TPSUBSCRIBE\(3cbl\)](#), [TPUNSUBSCRIBE\(3cbl\)](#), [EVENTS\(5\)](#), [TMSYSEVT\(5\)](#), [TMUSREVT\(5\)](#)

TPRCV(3cbl)

Name

`TPRCV()` - receive a message in a conversational connection

Synopsis

01 *TPSVCDEF-REC*.

COPY *TPSVCDEF*.

01 *TPTYPE-REC*.

COPY *TPTYPE*.

01 *DATA-REC*.

COPY User data.

01 *TPSTATUS-REC*.

COPY *TPSTATUS*.

CALL "TPRCV" USING *TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC*.

Description

`TPRCV()` is used to receive data sent across an open connection from another program. `COMM-HANDLE`, specifies on which open connection to receive data. `COMM-HANDLE` is a

communications handle returned from either `TPCONNECT()` or `TPSVCSTART()`. `DATA-REC` specifies where the message is read into, and, on input, `LEN` indicates the maximum number of bytes that should be moved into `DATA-REC`.

Upon successful and for several event types, `LEN` contains the actual number of bytes moved into `DATA-REC`. `REC-TYPE` and `SUB-TYPE` contain the data's type and subtype, respectively. If the message is larger than `DATA-REC`, then `DATA-REC` will contain only as many bytes as will fit in the record. The remainder of the reply is discarded and `TPRECV()` sets `TPTRUNCATE`.

If `LEN` is 0 upon successful return, then the reply has no data portion and `DATA-REC` was not modified. It is an error for `LEN` to be 0 on input.

`TPRECV()` can be issued only by the program that does not have control of the connection.

The following is a list of valid settings in `TPSVCDEF-REC`.

`TPNOCHANGE`

When this setting is used, the type of `DATA-REC` is not allowed to change. That is, the type and subtype of the message received must match `REC-TYPE` and `SUB-TYPE`, respectively. Either `TPNOCHANGE` or `TPCHANGE` must be set.

`TPCHANGE`

The type and/or subtype of the message received is allowed to differ from those specified in `REC-TYPE` and `SUB-TYPE`, respectively, so long as the receiver recognizes the incoming record type. Either `TPNOCHANGE` or `TPCHANGE` must be set.

`TPNOBLOCK`

`TPRECV()` does wait for data to arrive. If data is already available to receive, then `TPRECV()` gets the data and returns. Either `TPNOBLOCK` or `TPBLOCK` must be set.

`TPBLOCK`

When `TPBLOCK` is specified and no data is available to receive, the caller blocks until data arrives. Either `TPNOBLOCK` or `TPBLOCK` must be set.

`TPNOTIME`

This setting signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts will still affect the program. Either `TPNOTIME` or `TPTIME` must be set.

`TPTIME`

This setting signifies that the caller will receive blocking timeouts if a blocking condition exists and the blocking time is reached. Either `TPNOTIME` or `TPTIME` must be set.

TPSIGRSTRT

If a signal interrupts the underlying receive system call, then the call is reissued. Either `TPNOSIGRSTRT` or `TPSIGRSTRT` must be set.

TPNOSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is not restarted and the call fails. Either `TPNOSIGRSTRT` or `TPSIGRSTRT` must be set.

If an event exists for the communications handle, `COMM-HANDLE`, then `TPRECV()` will return setting `TP-STATUS` to `TPEEVENT()`. The event type is returned in `TPEVENT()`. Data can be received along with the `TPEV-SVCSUCC`, `TPEV-SVCFAIL`, and `TPEV-SENDONLY` events. Valid events for `TPRECV()` are as follows.

TPEV-DISCONIMM

Received by the subordinate of a conversation, this event indicates that the originator of the conversation has issued an immediate disconnect on the connection via `TPDISCON()`, or an error occurred when the originator issued `TPRETURN()` or `TPCOMMIT()` with the connection still open. This event is also returned to the originator or subordinate when a connection is broken due to a communications error (for example, a server, machine, or network failure). Because this is an immediate disconnection notification (that is, abortive rather than orderly), data in transit may be lost. If the two programs were participating in the same transaction, then the transaction is marked abort-only. `COMM-HANDLE` is no longer valid.

TPEV-SENDONLY

The program on the other end of the connection has relinquished control of the connection. The recipient of this event is allowed to send data but cannot receive any data until it relinquishes control.

TPEV-SVCERR

Received by the originator of a conversation, this event indicates that the subordinate of the conversation has issued `TPRETURN()`. `TPRETURN()` encountered an errors that precluded the service from returning successfully. For example, bad arguments may have been passed to `TPRETURN()` or `TPRETURN()` may have been called while the service had open connections to other subordinates. Due to the nature of this event, any application-defined data or return code are not available. The connection has been torn down and `COMM-HANDLE` is no longer valid. If this event occurred as part of the recipient's transaction, then the transaction is marked as abort-only.

TPEV-SVCFAIL

Received by the originator of a conversation, this event indicates that the subordinate service on the other end of the conversation has finished unsuccessfully as defined by the application (that is, it called `TPRETURN()` with `TPFAIL()` or `TPEXIT()`). If the

subordinate service was in control of this connection when `TPRETURN()` was called, then it can pass an application-defined return value and a record back to the originator of the connection. As part of ending the service routine, the server has torn down the connection. Thus, `COMM-HANDLE` is no longer valid. If this event occurred as part of the recipient's transaction, then the transaction is marked abort-only.

`TPEV-SVCSUCC`

Received by the originator of a conversation, this event indicates that the subordinate service on the other end of the conversation has finished successfully as defined by the application (that is, it called `TPRETURN()` with `TPSUCCESS()`). As part of ending the service routine, the server has torn down the connection. Thus, `COMM-HANDLE` is no longer valid. If the recipient is in transaction mode, then it can either commit (if it is also the initiator) or abort the transaction causing the work done by the server (if also in transaction mode) to either commit or abort.

Return Values

Upon successful completion, `TPRECV()` sets `TP-STATUS` to `[TPOK]`. When `TP-STATUS` is set to `[TPEEVEN]` and `TPEVENT()` is either `TPEV-SVCSUCC` or `TPEV-SVCFAIL`, `APPL-RETURN-CODE` contains an application-defined value that was sent as part of `TPRETURN()`. If the size of the incoming message was larger than the size specified in `LEN` on input, `TPTRUNCATE()` is set and only `LEN` amount of data was moved to `DATA-REC`, the remaining data is discarded.

Errors

Under the following conditions, `TPRECV()` fails and sets `TP-STATUS` to (unless otherwise noted, failure does not affect the caller's transaction, if one exists):

`[TPEINVAL]`

Invalid arguments were given (for example, settings in `TPSVCDEF-REC` are invalid).

`[TPEOTYPE]`

Either the type of subtype of the incoming message are not known to the caller, or `TPNOCHANGE` was set and `REC-TYPE` and `SUB-TYPE` do not match the type and subtype of the incoming message. If the conversation is part of the caller's transaction, then the transaction is marked abort-only since the incoming message is discarded.

`[TPEBADDESC]`

`COMM-HANDLE` contains an invalid communications handle.

`[TPETIME]`

This error code indicates that either a timeout has occurred or `TPRECV()` has been attempted, in spite of the fact that the current transaction is already marked rollback only.

If the caller is in transaction mode, then either the transaction is already rollback only or a transaction timeout has occurred. The transaction is marked abort-only. If the caller is not in transaction mode, a blocking timeout has occurred. (A blocking timeout can occur only if both `TPBLOCK` and `TPTIME` are specified.) In either case, no changes are made to *DATA-REC*.

If a transaction timeout has occurred, then, with one exception, any attempts to send new requests or receive outstanding replies will fail with `TPETIME` until the transaction has been aborted. The exception is a request that does not block, expects no reply, and is not sent on behalf of the caller's transaction (that is, `TPACALL()` with `TPNOTRAN`, `TPNOBLOCK`, and `TPNOREPLY` set).

When an ATMI call fails inside a transaction, the transaction is put into the `TX_ROLLBACK_ONLY` state. This state is treated, for most purposes, as though it were equivalent to a timeout. All further ATMI calls for this transaction (with the exception of those issued in the circumstances described in the previous paragraph) will fail with `TPETIME`.

[TPEEVENT]

An event occurred and its type is available in `TPEVENT()`.

[TPEBLOCK]

A blocking condition exists and `TPNOBLOCK` was specified.

[TPGOTSIG]

A signal was received and `TPSIGRSTRT` was not specified.

[TPEPROTO]

`TPRECV()` was called in an improper context (for example, the connection was established such that the calling program can only send data).

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

Usage

A server can pass an application-defined return value and typed record when calling `TPRETURN()`. The return value is available in `APPL-RETURN-CODE` and the record is available in *DATA-REC*.

See Also

`TPCONNECT(3cbl)`, `TPDISCON(3cbl)`, `TPSEND(3cbl)`

TPRESUME(3cbl)

Name

`TPRESUME()` - resume a global transaction

Synopsis

```
01  TPTRXDEF-REC.
    COPY TPTRXDEF.

01  TPSTATUS-REC.
    COPY TPSTATUS.

CALL "TPRESUME" USING TPTRXDEF-REC TPSTATUS-REC.
```

Description

`TPRESUME()` is used to resume work on behalf of a previously suspended transaction. Once the caller resumes work on a transaction, it must either suspend it with `TPSUSPEND()`, or complete it with one of `TPCOMMIT()` or `TPABORT()` at a later time.

The caller must ensure that its linked resource managers have been opened (via `TPOPEN()`) before it can resume work on any transaction.

`TPRESUME()` places the caller in transaction mode on behalf of the global transaction identifier contained in `TRANID()`.

Return Value

Upon successful completion, `TPRESUME()` sets `[TPOK]`.

Errors

Under the following conditions, `TPRESUME()` fails and sets `TP-STATUS` to:

`[TPEINVAL]`

Either `TRANID()` contains a non-existent transaction identifier (including previously completed or timed-out transactions), or it contains a transaction identifier that the caller is not allowed to resume. The caller's state with respect to the transaction is not changed.

[TPEMATCH]

TRANID () contains a transaction identifier that another program has already resumed. The caller's state with respect to the transaction is not changed.

[TPETRAN]

The BEA Tuxedo system is unable to resume the global transaction because the caller is currently participating in work outside any global transaction with one or more resource managers. All such work must be completed before a global transaction can be resumed. The caller's state with respect to the local transaction is unchanged.

[TPEPROTO]

TPRESUME () was called in an improper context (for example, the caller is already in transaction mode). The caller's state with respect to transaction mode is unchanged.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

Notes

XA-compliant resource managers must be successfully opened to be included in the global transaction. (See TPOPEN () for details.)

A program resuming a suspended transaction must reside on the same logical machine (LMID) as the program that suspended the transaction. For a Workstation client, the workstation handler (WSH) to which it is connected must reside on the same logical machine as the handler for the Workstation client that suspended the transaction.

See Also

[TPABORT \(3cb1\)](#), [TPCOMMIT \(3cb1\)](#), [TPOPEN \(3cb1\)](#), [TPSUSPEND \(3cb1\)](#)

TPRETURN(3cb1)

Name

TPRETURN () - returns from a BEA Tuxedo ATMI service routine

Synopsis

```

01 TPSVCRET-REC.
   COPY TPSVCRET.

01 TPTYPE-REC.
   COPY TPTYPE.

01 DATA-REC.
   COPY User data.

01 TPSTATUS-REC.
   COPY TPSTATUS.

COPY TPRETURN REPLACING TPSVCRET-REC BY TPSVCRET-REC

TPTYPE-REC BY TPTYPE-REC
DATA-REC BY DATA-REC T
PSTATUS-REC BY TPSTATUS-REC.

```

Description

`TPRETURN()` indicates that a service routine has completed. Since `TPRETURN()` contains an `EXIT PROGRAM` statement, it should be called from within the same routine that was invoked to ensure correct return of control to the BEA Tuxedo ATMI dispatcher (that is, `TPRETURN()` should not be invoked in a sub-program of the service routine since control would not return to the BEA Tuxedo ATMI dispatcher).

`TPRETURN()` is used to send a service's reply message. If the service receiving the reply is waiting in either `TPCALL()`, `TPGETRPLY()`, or `TPRECV()`, then after a successful call to `TPRETURN()`, the reply is available in the receiver's record.

For conversational services, `TPRETURN()` also tears down the connection. That is the service routine cannot call `TPDISCON()` directly. To ensure correct results, the program that connected to the conversation service should not call `TPDISCON()`; rather, it should wait for notification that the conversational service has completed (that is, it should wait for one of the events, like `TPEV-SVCSUCC` or `TPEV-SVCFAIL`, sent by `TPRETURN()`).

If a service routine was in transaction mode, `TPRETURN()` places the service's portion of the transaction in a state from which it may be either committed or aborted when the transaction is completed. A service may be invoked multiple times as part of the same transaction so it is not

necessarily fully committed nor aborted until either `TPCOMMIT()` or `TPABORT()` is called by the originator of the transaction.

`TPRETURN()` should be called after receiving all replies expected from request/response service requests initiated by the service routine. Otherwise, depending on the nature of the service, either a `[TPESVCERR]` status or a `TPEV-SVCERR` event will be returned to the program that initiated communications with the service routine. Any outstanding replies which are not received will automatically be dropped by the BEA Tuxedo ATMI dispatcher upon receipt. In addition, the communications handle for those replies become invalid.

`TPRETURN()` should also be called after closing all connections initiated by the service. Otherwise, depending on the nature of the service, either a `[TPESVCERR]` status or a `TPEV-SVCERR` event will be returned to the program that initiated communications with the service routine. Also, an immediate disconnect event (that is, `TPEV-DISCONIMM`) is sent over all open connections to subordinates.

Concerning control of a connection, if the service routine does not have control over the connection with which it was invoked when it issued `TPRETURN()`, then two outcomes are possible. First, if the service routine calls `TPRETURN()` with `TP-RETURN-VAL` in `TPSVCRET-REC` set to `TPFAIL()` and `REC-TYPE` in `TPTYPE-REC` set to `SPACES` (that is, no data is sent), then a `TPEV-SVCFAIL` event is sent to the originator of this conversation. Second, if any other invocation of `TPRETURN()` is used, a `TPEV-SVCERR` event is sent to the originator.

Since a conversational service has only one open connection which it did not initiate, the server knows over which communications handle the data (and any event) should be sent. For this reason, a communication handle is not passed to `TPRETURN()`.

The following is a description of the `TPRETURN()` arguments. `TP-RETURN-VAL` can be set to one of the following.

`TPSUCCESS`

The service has terminated successfully. If data is present, then it will be sent (barring any failures processing the return). If the caller is in transaction mode, then `TPRETURN()` places the caller's portion of the transaction in a state such that it can be committed when the transaction ultimately commits. Note that a call to `TPRETURN()` does not necessarily finalize an entire transaction. Also, even though the caller indicates success, if there are any outstanding replies or open connections, if any work done within the service caused its transaction to be marked abort-only, then a failed message is sent (that is, the recipient of the reply receives a `TPESVCERR()` indication or a `TPEV-SVCERR` event). Note that if a transaction becomes abort-only while in the service routine for any reason, then `TP-RETURN-VAL` should be set to `TPFAIL()`. If `TPSUCCESS()` is specified for a conversational service, a `TPEV-SVCSUCC` event is generated.

TPFAIL

The service has terminated unsuccessfully from an application standpoint. An error will be reported to the program receiving the reply. That is, the call to get the reply will fail and the recipient receives a [TPSVCERR] indication or a TPEV-SVCERR event. If the caller is in transaction mode, then TPRETURN() marks the transaction as abort-only (note that the transaction may already be marked abort-only). Barring any failures in processing the return, the caller's data is sent, if present. One reason for not sending the caller's data is when a transaction timeout has occurred. In this case, the program waiting for the reply will receive an error of [TPETIME].

TPEXIT

This value is the same as TPFail(), with respect to completing the service, but the server will exit after the transaction is marked as abort-only and the reply is sent back to the requester. If the server is restartable, then the server will automatically be restarted.

If TP-RETURN-VAL is not set to one of these three values, then it defaults to TPFail().

An application-defined return code, APPL-CODE in TPSVCRET-REC, may be sent to the program receiving the service reply. This code is sent regardless of the setting of TP-RETURN-VAL as long as a reply can be successfully sent (that is, as long as the receiving call returns success or [TPESVCFAIL], or receives one of the events TPEV-SVCSUCC or TPEV-SVCFAIL). The value of APPL-CODE is available in the receiver in the variable, APPL-RETURN-CODE in TPSTATUS-REC.

DATA-REC is a record to be sent and LEN specifies the amount of DATA-REC that should be sent. Note that if DATA-REC is a record of type and subtype that does not require a length to be specified, then LEN is ignored (and may be 0). If REC-TYPE is SPACES, DATA-REC and LEN are ignored. In this case, if a reply is expected by the program that invokes the service, then a reply is sent with no data portion. If no reply is expected, then TPRETURN() ignores any data passed to it and returns sending no reply. If REC-TYPE is STRING and LEN is 0, then the request is sent with no data portion.

If the service is conversational, there are several cases in which the application return code and the data portion are not transmitted:

- If the connection has been terminated when the call is made (that is, the caller has received TPEV-DISCONIMM on the connection), then this call simply ends the service routine and rolls back the current transaction, if one exists. In this case, the caller's data record cannot be transmitted.
- If the caller does not have control of the connection, either TPEV-SVCERR or TPEV-SVCFAIL is sent to the originator of the connection as described above. Regardless of which event the originator receives, no data record is transmitted. If the originator

receives the `TPEV_SVCFAIL` event, however, the return code is available in the originator's `APPL-RETURN-CODE` in `TPSTATUS-REC`.

Return Values

Because `TPRETURN()` contains an `EXIT PROGRAM` statement, no value is returned to the caller, nor does control return to the service routine. If a service routine returns without using `TPRETURN()` (that is, it uses an `EXIT PROGRAM` statement directly or just simply “falls out of the service routine”), the server will return a service error to the service requester. In addition, all open connections to subordinates will be disconnected immediately, and any outstanding asynchronous replies will be dropped. If the server was in transaction mode at the time of failure, the transaction is marked abort-only. Note also that if `TPRETURN()` is used outside of a service routine (that is, by routines that are not services), then it returns having no effect.

Errors

Errors encountered either in handling arguments or in processing cause `TP-STATUS` to be set to `[TPESVCERR]` for a program receiving the service's outcome via either `TPCALL()` or `TPGETRPLY()`, and cause the event, `TPEV-SVCERR`, to be sent over the conversation to a program using `TPSEND()` or `TPRECV()`.

See Also

[TPCALL\(3cbl\)](#), [TPCONNECT\(3cbl\)](#), [TPFORWAR\(3cbl\)](#)

TPSBLKTIME(3cbl)

Name

`TPSBLKTIME()` - routine for setting blocktime in seconds for the next service call or for all service calls

Synopsis

```
01 TPBLKDEF-REC.
    COPY TPBLKDEF.
01 TPSTATUS-REC.
    COPY TPSTATUS.

CALL "TPSBLKTIME" USING TPBLKDEF-REC TPSTATUS-REC.
```

Description

`TPSBLKTIME()` is used to set the blocktime value, in seconds, of a *potential blocking* API. A *potential blocking* API is defined as any system API that can use the flag `TBNOBLOCK` as a value. It does not have any effect on transaction timeout values.

`BLKTIME` in `TPBLKDEF-REC` sets blocking time in seconds. The blocktime range is 0 to 32767. A 0 blocktime value indicates that any previously set blocking time flag value is cancelled, and the blocking time set with a different blocktime flag value prevails. If `TPSBLKTIME()` is not called, the `BLOCKTIME` value in the `*SERVICES` section or the default `*RESOURCES` section of the `UBBCONFIG` file is used.

Note: Blocking timeouts set with `TPSBLKTIME()` take precedence over the `BLOCKTIME` parameter set in the `SERVICES` and `RESOURCES` section of the `UBBCONFIG` file. The precedence for blocktime checking is as follows:

`TPSBLKTIME(TPBLK-NEXT)`, `TPSBLKTIME(TPBLK-ALL)`, `*SERVICES`, `*RESOURCES`

Exactly one of the following values must be in `TPBLKDEF-REC`:

`TPBLK-NEXT`

Sets the blocktime value, in seconds, for the *next* potential blocking API.

Any API that is called containing the `TPNOBLOCK` flag is not effected by `TPSBLKTIME(TPBLK-NEXT)` and continues to be non-blocking.

A `TPBLK-NEXT` blocktime value overrides a `TPBLK-ALL` blocktime value for those API calls that immediately follow it.

`TPSBLKTIME(TPBLK-NEXT)` operates on a *per-thread* basis. Therefore, it is not necessary for applications to use any mutex around the `TPSBLKTIME(TPBLK-NEXT)` call and the subsequent API call which it affects.

`TPBLK-ALL`

This flag sets the blocktime value, in seconds, for the *all* subsequent potential blocking APIs until the next `TPSBLKTIME()` is called within that context. Any API that is called containing the `TPNOBLOCK` flag is not effected by `TPSBLKTIME(TPBLK-ALL)` and continues to be non-blocking.

`TPSBLKTIME(TPBLK-ALL)` operates on a *per-context* basis. Therefore, it is necessary to call `TPSBLKTIME(TPBLK-ALL)` in only one thread of context that is used in multiple threads.

`TPSBLKTIME(TPBLK-ALL)` will not affect any context that follows after [TPTERM\(3cbl\)](#) is called.

Notes: In order to perform blocking time values that are not affected by thread timing dependencies, it is best that `TPSBLKTIME (TPBLK-ALL)` is called in a multi-threaded context immediately after `TPINITIALIZE(3cbl)` using the `TP-MULT-ICONTEXTS` flag and before the return value of `TPGETCTXT(3cbl)` is made available to other threads.

When `TPSBLKTIME (TPBLK-ALL)` is called in a service on a multi-threaded server, it will affect the *currently* executed thread only. To set the blocktime for all services, it is best that you use `TPSBLKTIME (TPBLK-ALL)` with `TPSVRINIT(3cbl)`.

Return Values

Upon successful completion, `TPSBLKTIME()` sets `TP-STATUS` to `[TPOK]`.

Errors

Under the following conditions, `TPSBLKTIME` fails and sets `TP-STATUS` to one of the following values. The failure does not affect transaction timeout values.

[TPEINVAL]

Invalid arguments were given. For example, a value other than `TPBLK-NEXT` or `TPBLK-ALL` was specified in `TPBLKDEF-REC`.

[TPERELEASE]

`TPSBLKTIME()` was called in a client attached to a workstation handler running an earlier Tuxedo release.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

See Also

`TPCALL(3cbl)`, `TPCONNECT(3cbl)`, `TPRECV(3cbl)`, `TPGBLKTIME(3cbl)`, `UBBCONFIG(5)`

TPSCMT(3cbl)

Name

`TPSCMT()` - set when `TPCOMMIT` should return

Synopsis

```
01  TPCMTDEF-REC.
    COPY  TPCMTDEF.
```

```

01 TPSTATUS-REC.
   COPY TPSTATUS.

CALL "TPSCMT" USING TPCMTDEF-REC TPSTATUS-REC.

```

Description

TPSCMT () sets the TP-COMMIT-CONTROL characteristic to the value specified in TPCMTDEF-REC. The TP-COMMIT-CONTROL characteristic affects the way TPCOMMIT () behaves with respect to returning control to its caller. A program can call TPSCMT () regardless of whether it is in transaction mode or not. Note that if the caller is participating in a transaction that another program must commit, then its call to TPSCMT () does not affect that transaction. Rather, it affects subsequent transactions that the caller will commit.

In most cases, a transaction is committed only when a BEA Tuxedo ATMI program calls TPCOMMIT (). There is one exception: when a service is dispatched in transaction mode because the AUTOTRAN variable in the SERVICES section of the UBBCONFIG file is enabled, then the transaction completes upon calling TPRETURN (). If TPFORWAR () is called, then the transaction will be completed by the server ultimately calling TPRETURN (). Thus, the setting of the TP-COMMIT-CONTROL characteristic in the service that calls TPRETURN () determines when TPCOMMIT () returns control within a server. If TPCOMMIT () returns a heuristic error code, the server will write a message to a log file.

When a client joins a BEA Tuxedo ATMI application, the initial setting for this characteristic comes from a configuration file. (See the CMTRET variable in the RESOURCES section of [UBBCONFIG \(5\)](#).)

The following are the valid settings for TPCMTDEF-REC.

TP-CMT-LOGGED

This setting indicates that TPCOMMIT () should return after the commit decision has been logged by the first phase of the two-phase commit protocol but before the second phase has completed. This setting allows for faster response to the caller of TPCOMMIT () although there is a risk that a transaction participant might decide to heuristically complete (that is, aborted) its work due to timing delays waiting for the second phase to complete. If this occurs, there is no way to indicate this situation to the caller since TPCOMMIT () has already returned (although BEA Tuxedo writes a message to a log file when a resource manager takes a heuristic decision). Under normal conditions, participants that promise to commit during the first phase will do so during the second phase. Typically, problems caused by network or site failures are the sources for heuristic decisions being made during the second phase.

TP-CMT-COMPLETE

This setting indicates that `TPCOMMIT()` should return after the two-phase commit protocol has finished completely. This setting allows for `TPCOMMIT()` to return an indication that a heuristic decision occurred during the second phase of commit.

Return Values

Upon successful completion, `TPSCMT()` sets `TP-STATUS` to `[TPOK]` and returns the previous value of the `TP-COMMIT-CONTROL` characteristic.

Errors

Under the following conditions, `TPSCMT()` fails and sets `TP-STATUS` to:

`[TPEINVAL]`

TPCMTDEF-REC is not set to `TP-CMT-LOGGED` or `TP-CMT-COMPLETE`.

`[TPEPROTO]`

`TPSCMT()` was called improperly.

`[TPESYSTEM]`

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

`[TPEOS]`

An operating system error has occurred.

Notices

When using `TPBEGIN()`, `TPCOMMIT()`, and `TPABORT()` to delineate a BEA Tuxedo ATMI transaction, it is important to remember that only the work done by a resource manager that meets the XA interface (and is linked to the caller appropriately) has transactional properties. All other operations performed in a transaction are not affected by either `TPCOMMIT()` or `TPABORT()`. See [buildserver\(1\)](#) for details on linking resource managers that meet the XA interface into a server such that operations performed by that resource manager are part of a BEA Tuxedo ATMI transaction.

See Also

[TPABORT\(3cbl\)](#), [TPBEGIN\(3cbl\)](#), [TPCOMMIT\(3cbl\)](#), [TPGETLEV\(3cbl\)](#)

TPSEND(3cbl)

Name

TPSEND() - routine to send a message in a conversational connection

Synopsis

```

01 TPSVCDEF-REC.
   COPY TPSVCDEF.

01 TPTYPE-REC.
   COPY TPTYPE.

01 DATA-REC.
   COPY User data.

01 TPSTATUS-REC.
   COPY TPSTATUS.

CALL "TPSEND" USING TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

Description

TPSEND() is used to send data across an open connection to another program. The caller must have control of the connection. COMM-HANDLE specifies the open connection to send data over. COMM-HANDLE is a communications handle returned from either TPCONNECT() or TPSVCSTART().

DATA-REC contains the data to be sent and LEN specifies how much of the data to send. Note that if DATA-REC is a record of a type that does not require a length to be specified, then LEN is ignored (and may be 0). If REC-TYPE is SPACES, DATA-REC and LEN are ignored and a message is sent with no data (this might be done, for instance, to grant control of the connection without transmitting any data).

The following is a list of valid settings in TPSVCDEF-REC.

TPRECVONLY

This setting signifies that, after the caller's data is sent, the caller gives up control of the connection (that is, the caller cannot issue anymore TPSEND() calls). When the receiver on the other end of the connection receives the data sent by TPSEND(), it will also receive

an event (TPEV-SENDONLY) indicating that it has control of the connection (and cannot issue anymore TPRECV () calls). Either TPRECVONLY or TPEV-SENDONLY must be set.

TPSENDONLY

This setting signifies that the caller wants to remain in control of the connection. Either TPRECVONLY or TPEV-SENDONLY must be set.

TPNOBLOCK

The data and any events are not sent if a blocking condition exists (for example, the data buffers through which the message is sent are full). Either TPNOBLOCK or TPBLOCK must be set.

TPBLOCK

When TPBLOCK is specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). Either TPNOBLOCK or TPBLOCK must be set.

TPNOTIME

This setting signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts will still affect the program. Either TPNOTIME or TPTIME must be set.

TPTIME

This setting signifies that the caller will receive blocking timeouts if a blocking condition exists and the blocking time is reached. Either TPNOTIME or TPTIME must be set.

TPSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted call is reissued. Either TPNOSIGRSTRT or TPSIGRSTRT must be set.

TPNOSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is not restarted and the call fails. Either TPNOSIGRSTRT or TPSIGRSTRT must be set.

If an event exists for COMM-HANDLE, then TPSEND () will return without sending the caller's data. The event type is returned in TPEVENT () . Valid events for TPSEND () are as follows.

TPEV-DISCONIMM

Received by the subordinate of a conversation, this event indicates that the originator of the conversation has issued an immediate disconnect on the connection via TPDISCON () , or the originator of the connection issued TPRETURN () with open subordinate connections. This event is also returned to the originator or subordinate when a connection is broken due to a communications error (for example, a server, machine, or network failure).

TPEV-SVCFAIL

Received by the originator of a conversation, this event indicates that the subordinate of the conversation has issued `TPRETURN()` without having control of the conversation. In addition, `TPRETURN()` was issued with `TPFAIL()` set and no data record (that is, the `REC-TYPE` passed to `TPRETURN()` was set to `SPACES`).

TPEV-SVCERR

Received by the originator of a conversation, this event indicates that the subordinate of the conversation has issued `TPRETURN()` without having control of the conversation. In addition, `TPRETURN()` was issued in a manner different from that described for `TPEV-SVCFAIL` below.

Because each of these events indicates an immediate disconnection notification (that is, abortive rather than orderly), data in transit may be lost. The communications handle used for the connection is no longer valid. If the two programs were participating in the same transaction, then the transaction has been marked abort-only.

Return Values

Upon successful completion, `TPSEND()` sets `TP-STATUS` to `[TPOK]`. If an event exists and no errors were encountered, `TPSEND()` sets `TP-STATUS` to `[TPEEVEVENT]`. When `TP-STATUS` is set to `[TPEEVEVENT]` and `TP-EVENT` is either `TPEV-SVCSUCC` or `TPEV-SVCFAIL`, `APPL-RETURN-CODE` contains an application-defined value that was sent as part of `TPRETURN()`.

Errors

Under the following conditions, `TPSEND()` fails and sets `TP-STATUS` to (unless otherwise noted, failure does not affect caller's transaction, if one exists):

[TPEINVAL]

Invalid arguments were given.

[TPEBADDESC]

`COMM-HANDLE` contains an invalid communications handle.

[TPETIME]

This error code indicates that either a timeout has occurred or `TPSEND()` has been attempted, in spite of the fact that the current transaction is already marked rollback only.

If the caller is in transaction mode, then either the transaction is already rollback only or a transaction timeout has occurred. The transaction is marked abort-only. If the caller is not in transaction mode, a blocking timeout has occurred. (A blocking timeout can occur only if both `TPBLOCK` and `TPTIME` are specified.)

If a transaction timeout has occurred, then, with one exception, any attempts to send new requests or receive outstanding replies will fail with `TPETIME` until the transaction has been aborted. The exception is a request that does not block, expects no reply, and is not sent on behalf of the caller's transaction (that is, `TPACALL()` with `TPNOTRAN`, `TPNOBLOCK`, and `TPNOREPLY` set).

When a transactional ATMI call fails inside a transaction, the transaction is put into the `TX_ROLLBACK_ONLY` state. This state is treated, for most purposes, as though it were equivalent to a timeout. All further ATMI calls for this transaction (with the exception of those issued in the circumstances described in the previous paragraph) will fail with `TPETIME`.

[TPEEVENT]

An event occurred and its type is available in `TPEVENT()`. *DATA-REC* is not sent when this error occurs.

[TPEBLOCK]

A blocking condition exists and `TPNOBLOCK` was specified.

[TPGOTSIG]

A signal was received and `TPSIGRSTRT` was not specified.

[TPEPROTO]

`TPSEND()` was called in an improper context (for example, the connection was established such that the calling program can only receive data).

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

See Also

[TPCONNECT\(3cbl\)](#), [TPDISCON\(3cbl\)](#), [TPRECV\(3cbl\)](#)

TPSETCTXT(3cbl)

Name

`TPSETCTXT()` - sets a context identifier for the current application association

Synopsis

```

01  TPCONTEXTDEF-REC.
    COPY TPCONTEXTDEF.

01  TPSTATUS-REC.
    COPY TPSTATUS.

CALL "TPSETCTXT" USING TPCONTEXTDEF-REC TPSTATUS-REC.

```

Description

TPSETCTXT () defines the context in which the current program operates. (Multithreaded COBOL applications are not currently supported.) Subsequent BEA Tuxedo calls reference the application indicated by *CONTEXT* in TPCONTEXTDEF-REC. The value of *CONTEXT* in TPCONTEXTDEF-REC should have been provided by a previous call to TPGETCTXT () . If the value of *CONTEXT* is TPNULLEXCONTEXT, then the program is disassociated from any BEA Tuxedo context. TPINVALIDCONTEXT is not a valid input value for *CONTEXT* in TPCONTEXTDEF-REC.

Return Values

Upon successful completion, TPSETCTXT () sets TP-STATUS to [TPOK].

Upon failure, TPSETCTXT () leaves the calling process in its original context and sets TP-STATUS to indicate the error condition.

Errors

Upon failure, TPSETCTXT () sets TP-STATUS to one of the following values:

[TPEINVAL]

Invalid arguments have been given.

[TPENOENT]

The value of *CONTEXT* in TPCONTEXTDEF-REC is not a valid context.

[TPEPROTO]

TPSETCTXT () has been called in an improper context. For example, it has been called in a process that has not called TPINITIALIZE () or that has called TPINITIALIZE () without specifying the TP-MULTI-CONTEXTS setting.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error has been written to a log file.

[TPEOS]

An operating system error has occurred.

See Also

[Introduction to the COBOL Application-Transaction Monitor Interface](#), [TPGETCTXT\(3cbl\)](#)

TPSETUNSOL(3cbl)

Name

TPSETUNSOL() - sets method for handling unsolicited messages

Synopsis

```
01 CURR-ROUTINE  PIC S9(9) COMP-5.

01 PREV-ROUTINE  PIC S9(9) COMP-5.

01 TPSTATUS-REC.
   COPY TPSTATUS.

CALL "TPSETUNSOL" USING CURR-ROUTINE PREV-ROUTINE TPSTATUS-REC.
```

Description

TPSETUNSOL() allows a client to identify the routine that should be invoked when an unsolicited message is received by the BEA Tuxedo ATMI libraries. Before the first call to TPSETUNSOL(), any unsolicited messages received by the BEA Tuxedo ATMI libraries on behalf of the client are logged and ignored. A call to TPSETUNSOL() with a function number, *CURR-ROUTINE*, set to 0 has the same effect. The method used by the system for notification and detection is determined by the application default, which can be overridden on a per-client basis (see [TPINITIALIZE\(3cbl\)](#)).

The routine number passed, in *CURR-ROUTINE*, on the call to TPSETUNSOL() selects one of 16 predefined routines. The routine names must be `_tm_dispatch1` through `_tm_dispatch8` for C routines that provide unsolicited message handling and `TMDISPATCH9` through `TMDISPATCH16` for COBOL routines that provide the same message handling. The C functions (`_tm_dispatch1`

through `_tm_dispatch8`) must conform to the parameter definition described in `tpsetunsol(3c)`. The COBOL routines (`TMDISPATCH9` through `TMDISPATCH16`) must use `TPGETUNSOL()` to receive the data.

Processing within the unsolicited message handling routine in a C application is restricted to the following BEA Tuxedo functions: `tpalloc()`, `tpfree()`, `tpgetctxt()`, `tpgetlev()`, `tprealloc()`, and `tpypes()`.

Processing within the unsolicited message handling routine in a COBOL application is restricted to the following BEA Tuxedo functions: `TPGETLEV()` and `TPGETCTXT()`.

Return Values

Upon successful completion, `TPSETUNSOL()` sets `TP-STATUS` to `[TPOK]` and returns the previous setting for the unsolicited message handling routine (0 in `PREV-ROUTINE` is a successful return indicating that no message handling routine had been set previously).

Errors

Under the following conditions, `TPSETUNSOL()` fails and sets `TP-STATUS` to:

[TPEINVAL]

Invalid arguments were given (for example, `CURR-ROUTINE` is not a valid routine value).

[TPEPROTO]

`TPSETUNSOL()` was called in an improper context (for example, from within a server).

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

Portability

The interfaces described in `TPNOTIFY()` are supported on native site UNIX-based processors. In addition, the routines `TPBROADCAST()` and `TPCHKUNSOL()` as well as the routine `TPSETUNSOL()` are supported on UNIX and MS-DOS workstation processors.

`TPSETUNSOL()` is not supported on Windows, OS/2, and RS6000 due to the way that Dynamic Link Libraries and Shared Libraries work in these environments; `TPEPROTO()` will be returned if called on these platforms. Use the C-language interface `tpsetunsol()` to set up a handler function in these environments.

See Also

[TPGETCTXT\(3cbl\)](#), [TPGETUNSOL\(3cbl\)](#), [TPINITIALIZE\(3cbl\)](#), [TPTERM\(3cbl\)](#)

TPSPRIO(3cbl)

Name

`TPSPRIO()` - set service request priority

Synopsis

```
01 TPPRIDEF-REC.  
   COPY TPPRIDEF.
```

```
01 TPSTATUS-REC.  
   COPY TPSTATUS.
```

```
CALL "TPSPRIO" USING TPPRIDEF-REC TPSTATUS-REC.
```

Description

`TPSPRIO()` sets the priority for the next request sent or forwarded. The priority set affects only the next request sent. (Priority can also be set for messages enqueued or dequeued by `TPENQUEUE()` or `TPDEQUEUE()` if the queued management facility is installed.) By default, the setting of `PRIORITY` in *TPPRIDEF-REC* increments or decrements a service's default priority up to a maximum of 100 or down to a minimum of 1 depending on its sign, where 100 is the highest priority. The default priority for a request is determined by the service to which the request is being sent. This default may be specified administratively (see [UBBCONFIG\(5\)](#)), or accept the system default of 50. `TPSPRIO()` has no effect on messages sent via `TPCONNECT()` or `TPSEND()`.

The following is a list of valid settings in *TPPRIDEF-REC*.

TPABSOLUTE

The priority of the next request should be sent out at the absolute value of `PRIORITY`. The absolute value of `PRIORITY` must be within the range 1 and 100, inclusive, with 100 being the highest priority. Any value outside of this range causes a default value to be used.

TPRELATIVE

The priority of the next request should be sent out at the relative value of `PRIORITY`.

Return Values

Upon successful completion, `TPSPRIO()` sets `TP-STATUS` to `[TPOK]`.

Errors

Under the following conditions, `TPSPRIO()` fails and sets `TP-STATUS` to:

[TPEINVAL]

TPPRIDEF-REC settings are invalid.

[TPEPROTO]

`TPSPRIO()` was called improperly.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

See Also

[TPACALL\(3cbl\)](#), [TPCALL\(3cbl\)](#), [TPDEQUEUE\(3cbl\)](#), [TPENQUEUE\(3cbl\)](#), [TPGPRIOR\(3cbl\)](#)

TPSUBSCRIBE(3cbl)

Name

`TPSUBSCRIBE()` - subscribe to an event

Synopsis

```
01 TPEVTDEF-REC.
   COPY TPEVTDEF.
```

```
01 TPQUEDEF-REC.
   COPY TPQUEDEF.
```

```
01 TPSTATUS-REC.
   COPY TPSTATUS.
```

```
CALL "TPSUBSCRIBE" USING TPEVTDEF-REC TPQUEDEF-REC TPSTATUS-REC.
```

Description

The caller uses `TPSUBSCRIBE()` to subscribe to an event or set of events named by `EVENT-EXPR` in *TPEVTDEF-REC*. Subscriptions are maintained by the BEA Tuxedo EventBroker, `TMUSREVT()`,

and are used to notify subscribers when events are posted via `TPPOST()`. Each subscription specifies a notification method which can take one of three forms: client notification, service calls, or message enqueueing to stable-storage queues. Notification methods are determined by the subscriber's process type and the setting of the `TPEV-METHOD-FLAG` in `TPEVTDEF-REC`.

The event or set of events being subscribed to is named by the regular expression, `EVENT-EXPR` in `TPEVTDEF-REC`, and cannot be `SPACES`. Regular expressions are of the form specified in `tpsubscribe(3c)`. For example, if `EVENT-EXPR` is `"\e\e.*"`, the caller is subscribing to all system-generated events; if `EVENT-EXPR` is `"\e\e.SysServer.*"`, the caller is subscribing to all system-generated events related to servers. If `EVENT-EXPR` is `"[A-Z].*"`, the caller is subscribing to all user events starting with A-Z; if `EVENT-EXPR` is `".*(ERR|err).*"`, the caller is subscribing to all user events containing either the substring "ERR" or the substring "err" in the event name (for example, "account_error" and "ERROR_STATE" events would both qualify).

`EVENT-FILTER` in `TPEVTDEF-REC` is a string containing a Boolean filter rule that must be evaluated successfully before the EventBroker posts the event. Upon receiving an event to be posted, the EventBroker applies the filter rule, if one exists, to the posted event's data. If the data passes the filter rule, the EventBroker invokes the notification method; otherwise, the broker does not invoke the associated notification method. The caller can subscribe to the same event multiple times with different filter rules.

Filter rules are specific to the typed records to which they are applied. For FML and view records, the filter rule is a string that can be passed to each Boolean expression compiler (see [Fboolco](#), [Fboolco32](#), [Fvboolco](#), [Fvboolco32\(3fml\)](#)) and evaluated against the posted record (see [Fboolev](#), [Fboolev32](#), [Fvboolev](#), [Fvboolev32\(3fml\)](#)). For `STRING` records, the filter rule is a regular expression of the form specified in `tpsubscribe(3c)`. All other record types require customized filter evaluators (see [buffer\(3c\)](#) and [typesw\(5\)](#) for details on adding customized filter evaluators). If no filter rule is associated with `EVENT-EXPR`, then `EVENT-FILTER` must be `SPACES`.

If the subscriber is a BEA Tuxedo ATMI client process and `TPEVNOTIFY` in `TPEVTDEF-REC` is set, then the EventBroker sends an unsolicited message to the subscriber when the event to which it subscribed is posted. That is, when an event name is posted that evaluates successfully against `EVENT-EXPR`, the EventBroker tests the posted data against the filter rule associated with `EVENT-EXPR`. If the data passes the filter rule or if there is no filter rule for the event, then the subscriber receives an unsolicited notification along with any data posted with the event. In order to receive unsolicited notifications, the client must register (via `TPSETUNSOL()`) an unsolicited message handling routine. If a BEA Tuxedo ATMI server process calls `TPSUBSCRIBE()` with `TPEVNOTIFY` set, then `TPSUBSCRIBE()` fails and sets `TP-STATUS` in `TPSTATUS-REC` to `[TPEPROTO]`.

Clients receiving event notification via unsolicited messages should remove their subscriptions from the EventBroker's list of active subscriptions before exiting (see `TPUNSUBSCRIBE()` for details). Using `TPUNSUBSCRIBE()`'s wildcard handle, -1, clients can conveniently remove all of their "non-persistent" subscriptions which include those associated with the unsolicited notification method (see the description of `TPEVPERSIST` below for subscriptions and their associated notification methods that persist after a process exits). If a client exits without removing its non-persistent subscriptions, then the EventBroker will remove them when it detects that the client is no longer accessible.

When `TPEVNOTIFY` is set, `TPEVNOTRAN` and `TPEVNOPERSIST` must also be set; otherwise `TPSUBSCRIBE()` fails and sets `TP-STATUS` to `[TPEINVAL]`. That is, an event subscription for a client having the unsolicited notification method cannot be transactional nor can it be persistent.

If the subscriber (regardless of process type) sets `TPEVSERVICE()` in `TPEVTDEF-REC`, then event notifications are sent to the BEA Tuxedo ATMI service routine named by `NAME-1` in `TPEVTDEF-REC`. That is, when an event name is posted that evaluates successfully against `EVENT-EXPR`, the EventBroker tests the posted data against the filter rule associated with `EVENT-EXPR`. If the data passes the filter rule or if there is no filter rule for the event, then a service request is sent to `NAME-1` along with any data posted with the event. The service name in `NAME-1` can be any valid BEA Tuxedo ATMI service name and it may or may not be active at the time the subscription is made. Service routines invoked by the EventBroker should return with no reply data. That is, they should call `TPRETURN()` with `REC-TYPE` in `TPTYPE-REC` set to `SPACES`. Any data passed to `TPRETURN()` will be dropped.

If `TPEVTRAN` in `TPEVTDEF-REC` is also set, then if the process calling `TPPOST()` is in transaction mode, the EventBroker calls the subscribed service routine such that it will be part of the poster's transaction. Both the EventBroker, `TMUSREVT()`, and the subscribed service routine must belong to server groups that support transactions (see [UBBCONFIG\(5\)](#) for details). If `TPEVNOTRAN` is set, then the EventBroker calls the subscribed service routine such that it will not be part of the poster's transaction.

If the subscriber (regardless of process type) sets `TPEVQUEUE()` in `TPEVTDEF-REC`, then event notifications are enqueued to the queue space named by `NAME-1` in `TPEVTDEF-REC` and the queue named by `NAME-2` in `TPEVTDEF-REC`. That is, when an event name is posted that evaluates successfully against `EVENT-EXPR`, the EventBroker tests the posted data against the filter rule associated with `EVENT-EXPR`. If the data passes the filter rule or if there is no filter rule for the event, then the EventBroker enqueues a message to the queue space named by `NAME-1` and the queue named by `NAME-2` along with any data posted with the event. The queue space and queue name can be any valid BEA Tuxedo ATMI queue space and queue name, either of which may or may not exist at the time the subscription is made.

TPQUEDEF-REC can contain options further directing the EventBroker's enqueueing of the posted event. If the caller has no options to specify, then *TPQUEDEF-REC* should be set to `LOW-VALUE`. Otherwise, options can be set as described in the "Control Parameter" subsection of the `TPENQUEUE()` reference page (specifically, see the section describing the valid list of settings controlling input information for `TPENQUEUE()`).

If `TPEVTRAN` in *TPEVTDEF-REC* is also set, then if the process calling `TPPOST()` is in transaction mode, the EventBroker enqueues the posted event and its data such that it will be part of the poster's transaction. The EventBroker, `TMUSREVT()`, must belong to a server group that supports transactions (see [UBBCONFIG\(5\)](#) for details). If `TPEVNOTRAN` is set, then the EventBroker enqueues the posted event and its data such that it will not be part of the poster's transaction.

By default, the BEA Tuxedo EventBroker deletes subscriptions when the resource to which it is posting is not available (for example, the EventBroker cannot access a service routine and/or a queue space/queue name associated with an event subscription). Setting `TPEVPERSIST` in *TPEVTDEF-REC* indicates that the subscriber wants this subscription to persist across such errors (usually because the resource will become available again in the future). Persistent subscriptions are allowed only for `TPEVSERVICE()` and `TPEVQUEUE()` notification methods. `TPEVPERSIST` cannot be used when `TPEVNOTIFY` is set; otherwise, the function fails and sets `TP-STATUS` to `[TPEINVAL]`. When `TPEVNOPERSIST` is used, the EventBroker will remove this subscription if it encounters an error accessing either the client, the service name, or queue space/queue name designated in this subscription.

If `TPEVPERSIST` is used with `TPEVTRAN` and the resource is not available at the time of event notification, then the EventBroker will return to the poster such that its transaction must be aborted. That is, even though the subscription remains intact, the resource's unavailability will cause the poster's transaction to fail.

If the EventBroker's list of active subscriptions already contains a subscription that matches the one being requested by `TPSUBSCRIBE()`, then the function fails setting `TP-STATUS` to `[TPEMATCH]`. For a subscription to match an existing one, both `EVENT-EXPR` and `EVENT-FILTER` must match those of a subscription already in the EventBroker's active list of subscriptions. In addition, depending on the notification method, other criteria are used to determine matches.

If `TPEVNOTIFY` is set, then the caller's system-defined client identifier (known as a `CLIENTID`) is also used to detect matches. That is, `TPSUBSCRIBE()` fails if `EVENT-EXPR`, `EVENT-FILTER`, and the caller's `CLIENTID` match those of a subscription already known to the EventBroker.

If `TPEVSERVICE()` is set, then `TPSUBSCRIBE()` fails if `EVENT-EXPR`, `EVENT-FILTER`, and the service name set in `NAME-1` match those of a subscription already known to the EventBroker.

If `TPEVQUEUE()` is set, then EventBroker uses the queue space, queue name, and correlation identifier, in addition to `EVENT-EXPR` and `EVENT-FILTER`, when determining matches. The correlation identifier can be used to differentiate among several subscriptions for the same event expression and filter rule, destined for the same queue. Thus, if the caller has set both `TPEVQUEUE()` and `TPQNOCOORDID()`, then `TPSUBSCRIBE()` fails if `EVENT-EXPR`, `EVENT-FILTER`, the queue space name set in `NAME-1`, and the queue name set in `NAME-2` match those of a subscription (which also does not have a correlation identifier specified) already known to the EventBroker. Further, if `TPQCOORDID()` is set, then `TPSUBSCRIBE()` fails if `EVENT-EXPR`, `EVENT-FILTER`, `NAME-1`, `NAME-2`, and `CORRID` in `TPQUEDEF-REC` match those of a subscription (which has the same correlation identifier specified) already known to the EventBroker.

The following is a list of settings in `TPEVTDEF-REC`.

TPNOBLOCK

The subscription is not made if a blocking condition exists. If such a condition occurs, the call fails and sets `TP-STATUS` to `[TPEBLOCK]`. Either `TPNOBLOCK` or `TPBLOCK` must be set.

TPBLOCK

When `TPBLOCK` is specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). Either `TPNOBLOCK` or `TPBLOCK` must be set.

TPNOTIME

This setting signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur. Either `TPNOTIME` or `TPTIME` must be set.

TPTIME

This setting signifies that the caller will receive blocking timeouts if a blocking condition exists and the blocking time is reached. Either `TPNOTIME` or `TPTIME` must be set.

TPSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is reissued. Either `TPNOSIGRSTRT` or `TPSIGRSTRT` must be set.

TPNOSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is not restarted, the call fails and sets `TP-STATUS` to `[TPGOTSIG]`. Either `TPNOSIGRSTRT` or `TPSIGRSTRT` must be set.

Return Values

Upon successful completion, `TPSUBSCRIBE()` sets `TP-STATUS` to `[TPOK]`. In addition, `TPSUBSCRIBE()` sets `SUBSCRIPTION-HANDLE` in `TPEVTDEF-REC` to the handle for this subscription. `SUBSCRIPTION-HANDLE` can be used when calling `TPUNSUBSCRIBE()` to remove this subscription from the EventBroker's list of active subscriptions. Either the subscriber or any other process is allowed to use the returned handle to delete this subscription.

Errors

Under the following conditions, `TPSUBSCRIBE()` fails and sets `TP-STATUS` to one of the following values. (Unless otherwise noted, failure does not affect the caller's transaction, if one exists.)

`[TPEINVAL]`

Invalid arguments were given (for example, `EVENT-EXPR` is `SPACES`).

`[TPENOENT]`

Cannot access the BEA Tuxedo EventBroker.

`[TPELIMIT]`

The subscription failed because the EventBroker's maximum number of subscriptions has been reached.

`[TPEMATCH]`

The subscription failed because it matched one already listed with the EventBroker.

`[TPEPERM]`

The client is not attached as `tpsadm` and the subscription action is either a service call or the enqueueing of a message.

`[TPETIME]`

This error code indicates that either a timeout has occurred or `TPSUBSCRIBE()` has been attempted, in spite of the fact that the current transaction is already marked rollback only.

If the caller is in transaction mode, then either the transaction is already rollback only or a transaction timeout has occurred. The transaction is marked abort-only. If the caller is not in transaction mode, a blocking timeout has occurred. (A blocking timeout can occur only if both `TPBLOCK` and `TPTIME` are specified.)

If a transaction timeout has occurred, then, with one exception, any attempts to send new requests or receive outstanding replies will fail with `TPETIME` until the transaction has been aborted. The exception is a request that does not block, expects no reply, and is not

sent on behalf of the caller's transaction (that is, `TPACALL()` with `TPNOTRAN`, `TPNOBLOCK`, and `TPNOREPLY` set).

When a transactional ATMI call fails inside a transaction, the transaction is put into the `TX_ROLLBACK_ONLY` state. This state is treated, for most purposes, as though it were equivalent to a timeout. All further ATMI calls for this transaction (with the exception of those issued in the circumstances described in the previous paragraph) will fail with `TPETIME`.

[TPEBLOCK]

A blocking condition exists and `TPNOBLOCK` was specified.

[TPGOTSIG]

A signal was received and `TPNOSIGRSTRT` was specified.

[TPEPROTO]

`TPSUBSCRIBE()` was called improperly.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

See Also

`buffer(3c)`, `tpsubscribe(3c)`, `TPENQUEUE(3cbl)`, `TPPOST(3cbl)`, `TPSETUNSOL(3cbl)`, `TPUNSUBSCRIBE(3cbl)`, `Fboolco`, `Fboolco32`, `Fvboolco`, `Fvboolco32(3fml)`, `Fboolev`, `Fboolev32`, `Fvboolev`, `Fvboolev32(3fml)`, `EVENTS(5)`, `EVENT_MIB(5)`, `TMSYSEVT(5)`, `TMUSREVT(5)`, `tuxtypes(5)`, `typesw(5)`, `UBBCONFIG(5)`

TPSUSPEND(3cbl)

Name

`TPSUSPEND()` - suspend a global transaction

Synopsis

```
01  TPTRXDEF-REC.
    COPY TPTRXDEF.
```

```
01  TPSTATUS-REC.
```

```
COPY TPSTATUS.
```

```
CALL "TPSUSPEND" USING TPTRXDEF-REC TPSTATUS-REC.
```

Description

`TPSUSPEND()` is used to suspend the transaction active in the caller's program. A transaction begun with `TPBEGIN()` may be suspended with `TPSUSPEND()`. Either the suspending program or another program may use `TPRESUME()` to resume work on a suspended transaction. When `TPSUSPEND()` returns, the caller is no longer in transaction mode. However, while a transaction is suspended, all resources associated with that transaction (such as database locks) remain active. Like an active transaction, a suspended transaction is susceptible to the transaction timeout value that was assigned when the transaction first began.

For the transaction to be resumed in another process, the caller of `TPSUSPEND()` must have been the initiator of the transaction by explicitly calling `TPBEGIN()`. `TPSUSPEND()` may also be called by a process other than the originator of the transaction (for example, a server that receives a request in transaction mode). In the latter case, only the caller of `TPSUSPEND()` may call `TPRESUME()` to resume that transaction. This case is allowed so that a process can temporarily suspend a transaction to begin and do some work in another transaction before completing the original transaction (for example, to run a transaction to log a failure before rolling back the original transaction).

`TPSUSPEND()` populates `TRANID` with the transaction identifier being suspended.

To ensure success, the caller must have completed all outstanding transactional communication with servers before issuing `TPSUSPEND()`. That is, the caller must have received all replies for requests sent with `TPACALL()` that were associated with the caller's transaction. Also, the caller must have closed all connections with conversational services associated with the caller's transaction (that is, `TPRECV()` must have returned the `TPEV-SVCSUCC` event). If either rule is not followed, then `TPSUSPEND()` fails, the caller's current transaction is not suspended and all transactional communication handles remain valid. Communication handles not associated with the caller's transaction remain valid regardless of the outcome of `TPSUSPEND()`.

Return Value

Upon successful completion, `TPSUSPEND()` sets `[TPOK]`.

Errors

Under the following conditions, `TPSUSPEND()` fails and sets `TP-STATUS` to:

[TPEABORT]

The caller's active transaction has been aborted. All communication handles associated with the transaction are no longer valid.

[TPEPROTO]

TPSUSPEND() was called in an improper context (for example, the caller is not in transaction mode). The caller's state with respect to transaction mode is unchanged.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

See Also

[TPACALL\(3cbl\)](#), [TPBEGIN\(3cbl\)](#), [TPRECV\(3cbl\)](#), [TPRESUME\(3cbl\)](#)

TPSVCSTART(3cbl)**Name**

TPSVCSTART() - start a BEA Tuxedo ATMI service

Synopsis

```
01 TPSVCDEF-REC.
   COPY TPSVCDEF.
```

```
01 TPTYPE-REC.
   COPY TPTYPE.
```

```
01 DATA-REC.
   COPY User data.
```

```
01 TPSTATUS-REC.
   COPY TPSTATUS.
```

```
CALL "TPSVCSTART" USING TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

Description

`TPSVCSTART()` is the first BEA Tuxedo ATMI routine to be called when writing a service routines. In fact, it is an error to issue any other call within a service routine before calling `TPSVCSTART()`. `TPVCSTART()` is used to retrieve the service's parameters and data. This routine is used for services that receive requests via `TPCALL()` or `TPACALL()` routines as well as by services that communicate via `TPCONNECT()`, `TPSEND()`, and `TPRECV()` routines.

Service routines processing requests made via either `TPCALL()`, `TPACALL()`, or `TPFORWAR()` receive at most one incoming message (upon successfully returning from `TPSVCSTART()`) and send at most one reply (upon exiting the service routine with `TPRETURN()`).

Conversational services, on the other hand, are invoked by connection requests with at most one incoming message along with a means of referring to the open connection. Upon successfully returning from `TPSVCSTART()`, either the connecting program or the conversational service may send and receive data as defined by the application. The connection is half-duplex in nature meaning that one side controls the conversation (that is, it sends data) until it explicitly gives up control to the other side of the connection.

Concerning transactions, service routines can participate in at most one transaction if invoked in transaction mode. As far as the service routine writer is concerned, the transaction ends upon returning from the service routine. If the service routine is not invoked in transaction mode, then the service routine may originate as many transactions as it wants using `TPBEGIN()`, `TPCOMMIT()`, and `TPABORT()`. Note that `TPRETURN()` is not used to complete a transaction. Thus, it is an error to call `TPRETURN()` with an outstanding transaction that originated within the service routine.

DATA-REC specifies where the service's data is read into, and, on input, *LEN* in *TPTYPE-REC* indicates the maximum number of bytes that should be moved into *DATA-REC*. Upon successful return from `TPSVCSTART`, *LEN* contains the actual number of bytes moved into *DATA-REC*. *REC-TYPE* and *SUB-TYPE*, both in *TPTYPE-REC*, contain the data's type and subtype, respectively. If the message is larger than *DATA-REC*, then *DATA-REC* will contain only as many bytes as will fit in the record. The remainder of the message is discarded and `TPSVCSTART()` sets `TPTRUNCATE()`.

If *LEN* is 0 upon successful return, then the service has no incoming data and *DATA-REC* was not modified. It is an error for *LEN* to be 0 on input.

Upon successful return, *SERVICE-NAME* in *TPSVCDEF-REC* is populated with the service name that the requesting program used to invoke the service.

The following are the possible settings in *TPSVCDEF-REC* upon return of `TPSVCSTART()`.

TPREQRSP

The service was invoked with either `TPCALL()` or `TPACALL()`. This setting is mutually exclusive with `TPCONV`.

TPCONV

The service was invoked with `TPCONNECT()`. The communications handle for the conversation is available in `COMM-HANDLE` in *TPSVCDEF-REC*. This setting is mutually exclusive with `TPREQRSP`.

TPNOTRAN

The service routine is not in transaction mode. This setting is mutually exclusive with `TPTRAN`.

TPTRAN

The service routine is in transaction mode. This setting is mutually exclusive with `TPNOTRAN`.

TPNOREPLY

The program invoking the service routine is not expecting a reply. This setting is meaningful only when `TPREQRSP` is set. This setting is mutually exclusive with `TPREPLY`.

TPREPLY

The program invoking the service routine is expecting a reply. This setting is meaningful only when `TPREQRSP` is set. This setting is mutually exclusive with `TPNOREPLY`.

TPSENDONLY

The service is invoked such that it can send data across the connection and the program on the other end of the connection can only receive data. This setting is meaningful only when `TPCONV` is set. This setting is mutually exclusive with `TPRECVONLY`.

TPRECVONLY

The service is invoked such that it can only receive data from the connection and the program on the other end of the connection can send data. This setting is meaningful only when `TPCONV` is set. This setting is mutually exclusive with `TPSENDONLY`.

`APPKEY` in *TPSVCDEF-REC* is set to the application key assigned to the requesting client program by the application-defined authentication service. This key value is passed along with any and all service requests made while within this invocation of the service routine. `APPKEY` will have a value of -1 for originating clients that do not pass through the application authentication service. This includes clients of an earlier release level interoperating with a security application.

Return Values

Upon successful completion, `TPSVCSTART()` sets `TP-STATUS` to `[TPOK]`. If the size of the incoming message was larger than the size specified in `LEN` on input, `TPTRUNCATE()` is set and only `LEN` amount of data was moved to *DATA-REC*, the remaining data is discarded.

Errors

Under the following conditions, `TPSVCSTART()` fails and sets `TP-STATUS` to:

`[TPEINVAL]`

Invalid arguments were given.

`[TPEPROTO]`

`TPSVCSTART()` was called improperly.

`[TPESYSTEM]`

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

`[TPEOS]`

An operating system error has occurred.

See Also

`buildserver(1)`, `TPBEGIN(3cbl)`, `TPCALL(3cbl)`, `TPCONNECT(3cbl)`,
`TPINITIALIZE(3cbl)`, `TPOPEN(3cbl)`, `TPSVRDONE(3cbl)`, `TPSVRINIT(3cbl)`

TPSVRDONE(3cbl)

Name

`TPSVRDONE()` - BEA Tuxedo ATMI server termination routine

Synopsis

```
01  TPSTATUS-REC.  
    COPY TPSTATUS.  
PROCEDURE DIVISION.  
    * User code  
EXIT PROGRAM.
```

Description

The BEA Tuxedo ATMI server abstraction calls `TPSVRDONE()` after it has finished processing service requests but before it exits. When this routine is invoked, the server is still part of the

system but its own services have been unadvertised. Thus, BEA Tuxedo ATMI communication can be performed and transactions can be defined in this routine. However, if `TPSVRDONE()` returns with open connections, asynchronous replies pending or while still in transaction mode, the BEA Tuxedo system will close its connections, ignore any pending replies and roll back the transaction before the server exits.

If an application does not provide this routine in a server, then the default version provided by the BEA Tuxedo system is called instead. The default `TPSVRDONE()` calls `TPCLOSE()` and `USERLOG()` to announce that the server is about to exit.

Usage

If either `TPRETURN()` or `TPFORWAR()` are called in `TPSVRDONE()`, then these routines simply return having no effect.

See Also

[TPCLOSE\(3cbl\)](#), [TPSVRINIT\(3cbl\)](#)

TPSVRINIT(3cbl)

Name

`TPSVRINIT()` - BEA Tuxedo ATMI server initialization routine

Synopsis

```
LINKAGE SECTION.

01 CMD-LINE.
   05 ARGC PIC 9(4) COMP-5.
   05 ARGV.
   10 ARGS PIC X OCCURS 0 TO 9999 DEPENDING ON ARGC.
01 TPSTATUS-REC.
   COPY TPSTATUS.
PROCEDURE DIVISION USING CMD-LINE TPSTATUS-REC.
* User code
EXIT PROGRAM
```

Description

The BEA Tuxedo ATMI server abstraction calls `TPSVRINIT()` during its initialization. This routine is called after the program has become a server but before it handles any service requests;

thus, BEA Tuxedo ATMI communication may be performed and transactions may be defined in this routine. However, if `TPSVRINIT()` returns with either open connections or asynchronous replies pending, or while still in transaction mode, the BEA Tuxedo system closes the connections, ignores any pending replies, and aborts the transaction before the server exits.

If an application does not provide this routine in a server, then the default version provided by the BEA Tuxedo system is called, instead. The default `TPSVRINIT()` calls `TPOPEN()` and `USERLOG()` to announce that the server has started successfully.

Application-specific options can be passed into a server and processed in `TPSVRINIT()`. (For details, see [servopts\(5\)](#)). The options are passed through `ARGC` and `ARGV`. `ARGC` contains the number of arguments that have been passed; `ARGV`, the content of those arguments, specified in character format with single spaces separating arguments. `getopt()` is used in a BEA Tuxedo system.

If successful, `TPSVRINIT()` returns `[TPOK]` in `TP-STATUS` and the service can start accepting requests. If an error occurs in `TPSVRINIT`, the application can cause the server to exit gracefully (without taking any service requests) by returning any value other than `[TPOK]` in `TP-STATUS`.

When `TPSVRINIT()` returns any value other than `[TPOK]`, the system does not restart the server. Instead, the administrator must run `tmboot` to restart the server.

Return Values

When `TPRETURN()` or `TPFORWAR()` is used outside a service routine (for example, in a client, `TPSVRINIT()`, or `TPSVRDONE()`), then the routine returns with no effect.

Usage

When called in `TPSVRINIT()`, the `TPRETURN()` and `TPFORWAR()` routines simply return with no effect.

See Also

[TPOPEN\(3cbl\)](#), [TPSVRDONE\(3cbl\)](#)

TPTERM(3cbl)

Name

`TPTERM()` - leaves an application

Synopsis

```
01  TPSTATUS-REC.
    COPY TPSTATUS.
    CALL "TPTERM" USING TPSTATUS-REC.
```

Description

TPTERM() removes a client from a BEA Tuxedo ATMI application. If the client is in transaction mode, then the transaction is rolled back. When TPTERM() returns successfully, the caller can no longer perform BEA Tuxedo client operations. Any outstanding conversations are immediately disconnected.

If TPTERM() is called more than once (that is, if it is called after the caller has already left the application), no action is taken and success is returned.

Multi-contexting Issues

After invoking TPTERM(), a program is placed in the TPNULLEXEC context. Most ATMI functions invoked by a program in the TPNULLEXEC context perform an implicit TPINITIALIZE(). Whether or not the call to TPINITIALIZE() succeeds depends on the usual determining factors, unrelated to context-specific issues.

Return Values

Upon successful completion, TPTERM() sets TP-STATUS to [TPOK]. Upon success in a multi-contexted application, the application's current context is changed to TPNULLEXEC. It is the user's responsibility to use TPSETCTXT() to change the context subsequently, as desired.

Upon failure, TPTERM() returns -1 and sets TP-STATUS to indicate the error condition.

Errors

Upon failure, TPTERM() sets TP-STATUS to one of the following values:

[TPEPROTO]

TPTERM() was called in an improper context (for example, the caller is a server).

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

See Also

[TPINITIALIZE\(3cbl\)](#)

TPUNADVERTISE(3cbl)

Name

TPUNADVERTISE() - routine for unadvertising service names

Synopsis

```
01 SVC-NAME PIC X(15) .
01 TPSTATUS-REC.
   COPY TPSTATUS.
CALL "TPUNADVERTISE" USING SVC-NAME TPSTATUS-REC.
```

Description

TPUNADVERTISE() allows a server to unadvertise a service that it offers. By default, a server's services are advertised when it is booted and they are unadvertised when it is shut down.

All servers belonging to a multiple server, single queue (MSSQ) set must offer the same set of services. These routines enforce this rule by affecting the advertisements of all servers sharing an MSSQ set.

TPUNADVERTISE() removes *SVC-NAME* as an advertised service for the server (or the set of servers sharing the caller's MSSQ set). *SVC-NAME* cannot be SPACES. Also, *SVC-NAME* should be 15 characters or less. (See the SERVICES section of [UBBCONFIG\(5\)](#)). Longer names will be accepted and truncated to 15 characters. Care should be taken such that truncated names do not match other service names.

Return Values

Upon successful completion, TPUNADVERTISE() sets TP-STATUS to [TPOK].

Errors

Under the following conditions, TPUNADVERTISE() fails and sets TP-STATUS to:

[TPEINVAL]

Invalid arguments were given (for example *SVC-NAME* is SPACES).

[TPENOENT]

SVC-NAME is not currently advertised by the server.

[TPEPROTO]

TPUNADVERTISE () was called in an improper context (for example, by a client).

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

See Also

[TPADVERTISE\(3cbl\)](#)

TPUNSUBSCRIBE(3cbl)**Name**

TPUNSUBSCRIBE () - unsubscribe to an event

Synopsis

```
01 TPEVTDEF-REC.
   COPY TPEVTDEF.
01 TPSTATUS-REC.
   COPY TPSTATUS.
CALL "TPUNSUBSCRIBE" USING TPEVTDEF-REC TPSTATUS-REC.
```

Description

The caller uses TPUNSUBSCRIBE () to remove an event subscription or a set of event subscriptions from the BEA Tuxedo EventBroker's list of active subscriptions. SUBSCRIPTION-HANDLE in TPEVTDEF-REC is an event subscription handle returned by TPSubscribe (). Setting SUBSCRIPTION-HANDLE to the wildcard value, -1, directs TPUNSUBSCRIBE () to unsubscribe to all non-persistent subscriptions previously made by the calling process. Non-persistent subscriptions are those made with TPEVNOPERSIST set when TPSubscribe () was called. Persistent subscriptions can be deleted only by using the handle returned by TPSubscribe ().

Note that the -1 handle removes only those subscriptions made by the calling process and not any made by previous instantiations of the caller (for example, a server that dies and restarts cannot use the wildcard to unsubscribe to any subscriptions made by the original server).

The following is a list of valid settings in TPEVTDEF-REC.

TPNOBLOCK

The subscription is not removed if a blocking condition exists. If such a condition occurs, the call fails and sets `TP-STATUS` to `[TPEBLOCK]`. Either `TPNOBLOCK` or `TPBLOCK` must be set.

TPBLOCK

When `TPBLOCK` is specified and a blocking condition exists, the caller blocks until the condition subsides or a timeout occurs (either transaction or blocking timeout). Either `TPNOBLOCK` or `TPBLOCK` must be set.

TPNOTIME

This setting signifies that the caller is willing to block indefinitely and wants to be immune to blocking timeouts. Transaction timeouts may still occur. Either `TPNOTIME` or `TPTIME` must be set.

TPTIME

This setting signifies that the caller will receive blocking timeouts if a blocking condition exists and the blocking time is reached. Either `TPNOTIME` or `TPTIME` must be set.

TPSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is reissued. Either `TPNOSIGRSTRT` or `TPSIGRSTRT` must be set.

TPNOSIGRSTRT

If a signal interrupts any underlying system calls, then the interrupted system call is not restarted, the call fails and sets `TP-STATUS` to `[TPGOTSIG]`. Either `TPNOSIGRSTRT` or `TPSIGRSTRT` must be set.

Return Values

Upon successful completion, `TPUNSUBSCRIBE()` sets `TP-STATUS` to `[TPOK]`. In addition, `TPUNSUBSCRIBE()` sets `EVENT-COUNT` in `TPEVTDEF-REC` to the number of subscriptions deleted (zero or greater) from the EventBroker's list of active subscriptions. `EVENT-COUNT` may contain a number greater than 1 only when the wildcard handle, -1, is used. Also, `EVENT-COUNT` may contain a number greater than 0 even when `TPUNSUBSCRIBE()` completes unsuccessfully (that is, when the wildcard handle is used, the EventBroker may have successfully removed some subscriptions before it encountered an error deleting others).

Errors

Under the following conditions, `TPUNSUBSCRIBE()` fails and sets `TP-STATUS` to one of the following values. (Unless otherwise noted, failure does not affect the caller's transaction, if one exists.)

[TPEINVAL]

Invalid arguments were given (for example, `SUBSCRIPTION-HANDLE` is an invalid subscription handle).

[TPENOENT]

Cannot access the BEA Tuxedo EventBroker.

[TPETIME]

This error code indicates that either a timeout has occurred or `TPUNSUBSCRIBE()` has been attempted, in spite of the fact that the current transaction is already marked rollback only.

If the caller is in transaction mode, then either the transaction is already rollback only or a transaction timeout has occurred. The transaction is marked abort-only. If the caller is not in transaction mode, a blocking timeout has occurred. (A blocking timeout can occur only if both `TPBLOCK` and `TPTIME` are specified.)

If a transaction timeout has occurred, then, with one exception, any attempts to send new requests or receive outstanding replies will fail with `TPETIME` until the transaction has been aborted. The exception is a request that does not block, expects no reply, and is not sent on behalf of the caller's transaction (that is, `TPACALL()` with `TPNOTRAN`, `TPNOBLOCK`, and `TPNOREPLY` set).

When a transactional ATMI call fails inside a transaction, the transaction is put into the `TX_ROLLBACK_ONLY` state. This state is treated, for most purposes, as though it were equivalent to a timeout. All further ATMI calls for this transaction (with the exception of those issued in the circumstances described in the previous paragraph) will fail with `TPETIME`.

[TPEBLOCK]

A blocking condition exists and `TPNOBLOCK` was specified.

[TPGOTSIG]

A signal was received and `TPNOSIGRSTRT` was specified.

[TPEPROTO]

`TPUNSUBSCRIBE()` was called improperly.

[TPESYSTEM]

A BEA Tuxedo system error has occurred. The exact nature of the error is written to a log file.

[TPEOS]

An operating system error has occurred.

See Also

[TPPOST\(3cbl\)](#), [TPSUBSCRIBE\(3cbl\)](#), [EVENTS\(5\)](#), [EVENT_MIB\(5\)](#), [TMSYSEVT\(5\)](#),
[TMUSREVT\(5\)](#)

TXBEGIN(3cbl)

Name

`TXBEGIN()` - begin a global transaction

Synopsis

```
01 TX-RETURN-STATUS.  
   COPY TXSTATUS.  
CALL "TXBEGIN" USING TX-RETURN-STATUS.
```

Description

`TXBEGIN()` is used to place the calling thread of control in transaction mode. The calling thread must first ensure that its linked resource managers have been opened (via `TXOPEN()`) before it can start transactions. `TXBEGIN` fails (with a `TX-STATUS` value of `[TX-PROTOCOL-ERROR]`) if the caller is already in transaction mode or `TXOPEN()` has not been called.

Once in transaction mode, the calling thread must call `TXCOMMIT()` or `TXROLLBACK()` to complete its current transaction. There are certain cases related to transaction chaining where `TXBEGIN()` does not need to be called explicitly to start a transaction. See `TXCOMMIT()` and `TXROLLBACK()` for details. *TX-RETURN-STATUS* is the record used to return a value.

Optional Set-up

`TXSETTIMEOUT()`

Return Value

Upon successful completion, `TXBEGIN()` returns `TX-OK`, a non-negative return value.

Errors

Under the following conditions, `TXBEGIN()` fails and returns one of these negative values:

`[TX-OUTSIDE]`

The transaction manager is unable to start a global transaction because the calling thread of control is currently participating in work outside any global transaction with one or

more resource managers. All such work must be completed before a global transaction can be started. The caller's state with respect to the local transaction is unchanged.

[TX-PROTOCOL-ERROR]

The function was called in an improper context (for example, the caller is already in transaction mode). The caller's state with respect to transaction mode is unchanged.

[TX-ERROR]

Either the transaction manager or one or more of the resource managers encountered a transient error trying to start a new transaction. When this error is returned, the caller is not in transaction mode. The exact nature of the error is written to a log file.

[TX-FAIL]

Either the transaction manager or one or more of the resource managers encountered a fatal error. The nature of the error is such that the transaction manager and/or one or more of the resource managers can no longer perform work on behalf of the application. When this error is returned, the caller is not in transaction mode. The exact nature of the error is written to a log file.

See Also

[TXCOMMIT\(3cbl\)](#), [TXOPEN\(3cbl\)](#), [TXROLLBACK\(3cbl\)](#), [TXSETTIMEOUT\(3cbl\)](#)

Warnings

XA-compliant resource managers must be successfully opened to be included in the global transaction. (See [TXOPEN](#) for details.)

TXCLOSE(3cbl)

Name

`TXCLOSE()` - close a set of resource managers

Synopsis

```
DATA DIVISION.
  * Include TX definitions.
01 TX-RETURN-STATUS.
  COPY TXSTATUS.
PROCEDURE DIVISION.
CALL "TXCLOSE" USING TX-RETURN-STATUS.
```

Description

`TXCLOSE()` closes a set of resource managers in a portable manner. It invokes a transaction manager to read resource manager-specific information in a transaction manager-specific manner and pass this information to the resource managers linked to the caller.

`TXCLOSE()` closes all resource managers to which the caller is linked. This function is used in place of resource-manager-specific “close” calls and allows an application program to be free of calls which may hinder portability. Since resource managers differ in their termination semantics, the specific information needed to “close” a particular resource manager must be published by each resource manager.

`TXCLOSE()` should be called when an application thread of control no longer wishes to participate in global transactions. `TXCLOSE()` fails (returning `[TX-PROTOCOL-ERROR]`) if the caller is in transaction mode. That is, no resource managers are closed even though some may not be participating in the current transaction.

When `TXCLOSE()` returns success (`TX-OK`), all resource managers linked to the calling thread are closed.

TX-RETURN-STATUS is the record used to return a value.

Return Value

Upon successful completion, `TXCLOSE()` returns `TX-OK`, a non-negative value.

Errors

Under the following conditions, `TXCLOSE()` fails and returns one of these negative values:

`[TX-PROTOCOL-ERROR]`

The function was called in an improper context (for example, the caller is in transaction mode). No resource managers are closed.

`[TX-ERROR]`

Either the transaction manager or one or more of the resource managers encountered a transient error. The exact nature of the error is written to a log file. All resource managers that could be closed are closed.

`[TX-FAIL]`

Either the transaction manager or one or more of the resource managers encountered a fatal error. The nature of the error is such that the transaction manager and/or one or more of the resource managers can no longer perform work on behalf of the application. The exact nature of the error is written to a log file.

See Also

[TXOPEN\(3cbl\)](#)

TXCOMMIT(3cbl)

Name

TXCOMMIT() - commit a transaction

Synopsis

```
DATA DIVISION.
* Include TX definitions.
01 TX-RETURN-STATUS.
  COPY TXSTATUS.
PROCEDURE DIVISION.
CALL "TXCOMMIT" USING TX-RETURN-STATUS.
```

Description

TXCOMMIT() is used to commit the work of the transaction active in the caller's thread of control.

If the *transaction_control* characteristic (see TXSETTRANCTL()) is TX-UNCHAINED, then when TXCOMMIT() returns, the caller is no longer in transaction mode. However, if the *transaction_control* characteristic is TX-CHAINED, then when TXCOMMIT() returns, the caller remains in transaction mode on behalf of a new transaction (see the RETURN VALUE and ERRORS sections below).

TX-RETURN-STATUS is the record used to return a value.

Optional Set-up

- TXSETCOMMITRET()
- TXSETTRANCTL()
- TXSETTIMEOUT()

Return Value

Upon successful completion, TXCOMMIT() returns TX-OK, a non-negative return value.

Errors

Under the following conditions, TXCOMMIT() fails and returns one of these negative values:

[TX-NO-BEGIN]

The current transaction committed successfully; however, a new transaction could not be started and the caller is no longer in transaction mode. This return value may occur only when the *transaction_control* characteristic is TX-CHAINED.

[TX-ROLLBACK]

The current transaction could not commit and has been rolled back. In addition, if the *transaction_control* characteristic is TX-CHAINED, a new transaction is started.

[TX-ROLLBACK-NO-BEGIN]

The transaction could not commit and has been rolled back. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the *transaction_control* characteristic is TX-CHAINED.

[TX-MIXED]

The work done on behalf of the transaction was partially committed and partially rolled back. In addition, if the *transaction_control* characteristic is TX-CHAINED, a new transaction is started.

[TX-MIXED-NO-BEGIN]

The work done on behalf of the transaction was partially committed and partially rolled back. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the *transaction_control* characteristic is TX-CHAINED.

[TX-HAZARD]

Due to a failure, some of the work done on behalf of the transaction may have been committed and some of it may have been rolled back. In addition, if the *transaction_control* characteristic is TX-CHAINED, a new transaction is started.

[TX-HAZARD-NO-BEGIN]

Due to a failure, some of the work done on behalf of the transaction may have been committed and some of it may have been rolled back. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the *transaction_control* characteristic is TX-CHAINED.

[TX-PROTOCOL-ERROR]

The function was called in an improper context (for example, the caller is not in transaction mode). The caller's state with respect to transaction mode is not changed.

[TX-FAIL]

Either the transaction manager or one or more of the resource managers encountered a fatal error. The nature of the error is such that the transaction manager and/or one or more of the resource managers can no longer perform work on behalf of the application. The

exact nature of the error is written to a log file. The caller's state with respect to the transaction is unknown.

See Also

[TXBEGIN\(3cbl\)](#), [TXSETCOMMITRET\(3cbl\)](#), [TXSETTIMEOUT\(3cbl\)](#), [TXSETTRANCTL\(3cbl\)](#)

TXINFORM(3cbl)

Name

`TXINFORM()` - return global transaction information

Synopsis

```
DATA DIVISION.
    * Include TX definitions.
01 TX-RETURN-STATUS.
    COPY TXSTATUS.
01 TX-INFO-AREA.
    COPY TXINFDEF.
PROCEDURE DIVISION.
    CALL "TXINFORM" USING TX-INFO-AREA, TX-RETURN-STATUS.
```

Description

`TXINFORM()` returns global transaction information in *TX-INFO-AREA*. In addition, this function returns a value indicating whether the caller is currently in transaction mode or not.

`TXINFORM()` populates the *TX-INFO-AREA* record with global transaction information. The contents of the *TX-INFO-AREA* record are described under `INTRO()`.

If `TXINFORM` is called in transaction mode, then *TX-IN-TRAN* is set, *XID-REC* will be populated with a current transaction branch identifier and *TRANSACTION-STATE* will contain the state of the current transaction. If the caller is not in transaction mode, *TX-NOT-IN-TRAN* is set and *XID-REC* will be populated with the NULL *XID* (see `TXINTRO` for details). In addition, regardless of whether the caller is in transaction mode, *COMMIT-RETURN*, *TRANSACTION-CONTROL*, and *TRANSACTION-TIMEOUT* contain the current settings of the *commit_return* and *transaction_control* characteristics, and the transaction timeout value in seconds.

The transaction timeout value returned reflects the setting that will be used when the next transaction is started. Thus, it may not reflect the timeout value for the caller's current global

transaction since calls made to `TXSETTIMEOUT()` after the current transaction was begun may have changed its value.

TX-RETURN-STATUS is the record used to return a value.

Return Value

Upon successful completion, `TXINFORM()` returns `TX-OK`, a non-negative return value.

Errors

Under the following conditions, `TXINFORM()` fails and returns one of these negative values:

[TX-PROTOCOL-ERROR]

The function was called in an improper context (for example, the caller has not yet called `TXOPEN()`).

[TX-FAIL]

The transaction manager encountered a fatal error. The nature of the error is such that the transaction manager can no longer perform work on behalf of the application. The exact nature of the error is written to a log file.

See Also

[TXOPEN\(3cbl\)](#), [TXSETCOMMITRET\(3cbl\)](#), [TXSETTIMEOUT\(3cbl\)](#), [TXSETTRANCTL\(3cbl\)](#)

Warnings

Within the same global transaction, subsequent calls to `TXINFORM` are guaranteed to provide an `XID` with the same *gtrid* component, but not necessarily the same *bqual* component.

TXOPEN(3cbl)

Name

`TXOPEN()` - open a set of resource managers

Synopsis

```
DATA DIVISION.  
    * Include TX definitions.  
01 TX-RETURN-STATUS.  
    COPY TXSTATUS.  
PROCEDURE DIVISION.  
    CALL "TXOPEN" USING TX-RETURN-STATUS.
```


Description

`TXOPEN()` opens a set of resource managers in a portable manner. It invokes a transaction manager to read resource manager-specific information in a transaction manager-specific manner and pass this information to the resource managers linked to the caller.

`TXOPEN()` attempts to open all resource managers that have been linked with the application. This function is used in place of resource manager-specific “open” calls and allows an application program to be free of calls which may hinder portability. Since resource managers differ in their initialization semantics, the specific information needed to “open” a particular resource manager must be published by each resource manager.

If `TXOPEN()` returns `TX-ERROR`, then no resource managers are open. If `TXOPEN()` returns `TX-OK`, some or all of the resource managers have been opened. Resource managers that are not open will return resource manager-specific errors when accessed by the application. `TXOPEN()` must successfully return before a thread of control participates in global transactions.

Once `TXOPEN()` returns success, subsequent calls to `TXOPEN` (before an intervening call to `TXCLOSE()`) are allowed. However, such subsequent calls will return success, and the TM will not attempt to reopen any RMs.

`TX-RETURN-STATUS` is the record used to return a value.

Return Value

Upon successful completion, `TXOPEN()` returns `TX-OK`, a non-negative return value.

Errors

Under the following conditions, `TXOPEN()` fails and returns one of these negative values.

[TX-ERROR]

Either the transaction manager or one or more of the resource managers encountered a transient error. No resource managers are open. The exact nature of the error is written to a log file.

[TX-FAIL]

Either the transaction manager or one or more of the resource managers encountered a fatal error. The nature of the error is such that the transaction manager and/or one or more of the resource managers can no longer perform work on behalf of the application. The exact nature of the error is written to a log file.

See Also

[TXCLOSE\(3cbl\)](#)

TXROLLBACK(3cbl)

Name

TXROLLBACK() - roll back a transaction

Synopsis

```
DATA DIVISION.  
  * Include TX definitions.  
01 TX-RETURN-STATUS.  
  COPY TXSTATUS.  
PROCEDURE DIVISION.  
CALL "TXROLLBACK" USING TX-RETURN-STATUS.
```

Description

TXROLLBACK() is used to roll back the work of the transaction active in the caller's thread of control.

If the *transaction_control* characteristic (see TXSETTRANCTL()) is TX-UNCHAINED, then when TXROLLBACK() returns, the caller is no longer in transaction mode. However, if the *transaction_control* characteristic is TX-CHAINED, then when TXROLLBACK() returns, the caller remains in transaction mode on behalf of a new transaction (see the RETURN VALUE and ERRORS sections below).

TX-RETURN-STATUS is the record used to return a value.

Optional Set-up

- TXSETTRANCTL()
- TXSETTIMEOUT()

Return Value

Upon successful completion, TXROLLBACK() returns TX-OK, a non-negative return value.

Errors

Under the following conditions, TXROLLBACK() fails and returns one of these negative values:

[TX-NO-BEGIN]

The current transaction rolled back; however, a new transaction could not be started and the caller is no longer in transaction mode. This return value may occur only when the *transaction_control* characteristic is TX-CHAINED.

[TX-MIXED]

The work done on behalf of the transaction was partially committed and partially rolled back. In addition, if the *transaction_control* characteristic is TX-CHAINED, a new transaction is started.

[TX-MIXED-NO-BEGIN]

The work done on behalf of the transaction was partially committed and partially rolled back. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the *transaction_control* characteristic is TX-CHAINED.

[TX-HAZARD]

Due to a failure, some of the work done on behalf of the transaction may have been committed and some of it may have been rolled back. In addition, if the *transaction_control* characteristic is TX-CHAINED, a new transaction is started.

[TX-HAZARD-NO-BEGIN]

Due to a failure, some of the work done on behalf of the transaction may have been committed and some of it may have been rolled back. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the *transaction_control* characteristic is TX-CHAINED.

[TX-COMMITTED]

The work done on behalf of the transaction was heuristically committed. In addition, if the *transaction_control* characteristic is TX-CHAINED, a new transaction is started.

[TX-COMMITTED-NO-BEGIN]

The work done on behalf of the transaction was heuristically committed. In addition, a new transaction could not be started and the caller is no longer in transaction mode. This return value can occur only when the *transaction_control* characteristic is TX-CHAINED.

[TX-PROTOCOL-ERROR]

The function was called in an improper context (for example, the caller is not in transaction mode).

[TX-FAIL]

Either the transaction manager or one or more of the resource managers encountered a fatal error. The nature of the error is such that the transaction manager and/or one or more of the resource managers can no longer perform work on behalf of the application. The exact nature of the error is written to a log file. The caller's state with respect to the transaction is unknown.

See Also

`TXBEGIN(3cbl)`, `TXSETTIMEOUT(3cbl)`, `TXSETTRANCTL(3cbl)`

TXSETCOMMITRET(3cbl)

Name

`TXSETCOMMITRET()` - set `commit_return` characteristic

Synopsis

```
DATA DIVISION.  
    * Include TX definitions.  
01 TX-RETURN-STATUS.  
    COPY TXSTATUS.  
*  
01 TX-INFO-AREA.  
    COPY TXINFDEF.  
PROCEDURE DIVISION.  
CALL "TXSETCOMMITRET" USING TX-INFO-AREA TX-RETURN-STATUS.
```

Description

`TXSETCOMMITRET()` sets the *commit_return* characteristic to the value specified in *COMMIT-RETURN*. This characteristic affects the way `TXCOMMIT()` behaves with respect to returning control to its caller. `TXSETCOMMITRET()` may be called regardless of whether its caller is in transaction mode. This setting remains in effect until changed by a subsequent call to `TXSETCOMMITRET()`.

The initial setting for this characteristic is `TX-COMMIT-COMPLETED`.

The following are the valid settings for *COMMIT-RETURN*.

`TX-COMMIT-DECISION-LOGGED`

This flag indicates that `TXCOMMIT()` should return after the commit decision has been logged by the first phase of the two-phase commit protocol but before the second phase has completed. This setting allows for faster response to the caller of `TXCOMMIT()`. However, there is a risk that a transaction will have a heuristic outcome, in which case the caller will not find out about this situation via return codes from `TXCOMMIT()`. Under normal conditions, participants that promise to commit during the first phase will do so during the second phase. In certain unusual circumstances however (for example,

long-lasting network or node failures) phase 2 completion may not be possible and heuristic results may occur.

TX-COMMIT-COMPLETED

This flag indicates that `TXCOMMIT()` should return after the two-phase commit protocol has finished completely. This setting allows the caller of `TXCOMMIT()` to see return codes that indicate that a transaction had or may have had heuristic results.

TX-RETURN-STATUS is the record used to return a value.

Return Value

Upon successful completion, `TXSETCOMMITRET()` returns `TX-OK`, a non-negative return value.

Errors

Under the following conditions, `TXSETCOMMITRET()` does not change the setting of the *commit_return* characteristic and returns one of these negative values:

[TX-EINVAL]

COMMIT-RETURN is not one of `TX-COMMIT-DECISION-LOGGED` or `TX-COMMIT-COMPLETED`.

[TX-PROTOCOL-ERROR]

The function was called in an improper context (for example, the caller has not yet called `TXOPEN()`).

[TX-FAIL]

The transaction manager encountered a fatal error. The nature of the error is such that the transaction manager can no longer perform work on behalf of the application. The exact nature of the error is written to a log file.

See Also

`TXBEGIN(3cbl)`, `TXCOMMIT(3cbl)`, `TXINFORM(3cbl)`, `TXOPEN(3cbl)`, `TXROLLBACK(3cbl)`

TXSETTRANCTL(3cbl)

Name

`TXSETTRANCTL()` - set *transaction_control* characteristic

Synopsis

```
DATA DIVISION.  
  * Include TX definitions.  
01 TX-RETURN-STATUS.  
  COPY TXSTATUS.  
01 TX-INFO-AREA.  
  COPY TXINFDEF.  
PROCEDURE DIVISION.  
CALL "TXSETTRANCTL" USING TX-INFO-AREA TX-RETURN-STATUS.
```

Description

TXSETTRANCTL() sets the *transaction_control* characteristic to the value specified in *TRANSACTION-CONTROL*. This characteristic determines whether TXCOMMIT() and TXROLLBACK() start a new transaction before returning to their caller. TXSETTRANCTL() may be called regardless of whether the application program is in transaction mode. This setting remains in effect until changed by a subsequent call to TXSETTRANCTL().

The initial setting for this characteristic is TX-UNCHAINED.

The following are the valid settings for *TRANSACTION-CONTROL*.

TX-UNCHAINED

This flag indicates that TXCOMMIT() and TXROLLBACK() should not start a new transaction before returning to their caller. The caller must issue TXBEGIN() to start a new transaction.

TX-CHAINED

This flag indicates that TXCOMMIT() and TXROLLBACK() should start a new transaction before returning to their caller.

TX-RETURN-STATUS is the record used to return a value.

Return Value

Upon successful completion, TXSETTRANCTL() returns TX-OK, a non-negative return value.

Errors

Under the following conditions, TXSETTRANCTL() does not change the setting of the *transaction_control* characteristic and returns one of these negative values:

[TX-EINVAL]

TRANSACTION-CONTROL is not one of TX-UNCHAINED or TX-CHAINED.

[TX-PROTOCOL-ERROR]

The function was called in an improper context (for example, the caller has not yet called `TXOPEN()`).

[TX-FAIL]

The transaction manager encountered a fatal error. The nature of the error is such that the transaction manager can no longer perform work on behalf of the application. The exact nature of the error is written to a log file.

See Also

`TXBEGIN(3cbl)`, `TXCOMMIT(3cbl)`, `TXOPEN(3cbl)`, `TXROLLBACK(3cbl)`, `TXINFORM(3cbl)`

TXSETTIMEOUT(3cbl)**Name**

`TXSETTIMEOUT()` - set *transaction_timeout* characteristic

Synopsis

```
DATA DIVISION.
  * Include TX definitions.
01 TX-RETURN-STATUS.
  COPY TXSTATUS.
  *
01 TX-INFO-AREA.
  COPY TXINFDEF.
PROCEDURE DIVISION.
CALL "TXSETTIMEOUT" USING TX-INFO-AREA TX-RETURN-STATUS.
```

Description

`TXSETTIMEOUT()` sets the *transaction_timeout* characteristic to the value specified in *TRANSACTION-TIMEOUT*. This value specifies the time period in which the transaction must complete before becoming susceptible to transaction timeout; that is, the interval between the AP calling `TXBEGIN()` and `TXCOMMIT()` or `TXROLLBACK()`. `TXSETTIMEOUT()` may be called regardless of whether its caller is in transaction mode or not. If `TXSETTIMEOUT()` is called in transaction mode, the new timeout value does not take effect until the next transaction.

The initial *transaction_timeout* value is 0 (no timeout).

TRANSACTION-TIMEOUT specifies the number of seconds allowed before the transaction becomes susceptible to transaction timeout. It may be set to any value up to the maximum value for an S9(9) COMP-5 as defined by the system. A *TRANSACTION-TIMEOUT* value of zero disables the timeout feature.

TX-RETURN-STATUS is the record used to return a value.

Return Value

Upon successful completion, `TXSETTIMEOUT()` returns `TX-OK`, a non-negative return value.

Errors

Under the following conditions, `TXSETTIMEOUT()` does not change the setting of the *transaction_timeout* characteristic and returns one of these negative values:

[TX-EINVAL]

The timeout value specified is invalid.

[TX-PROTOCOL-ERROR]

The function was called improperly. For example, it was called before the caller called `TXOPEN()`.

[TX-FAIL]

The transaction manager encountered an error. The nature of the error is such that the transaction manager can no longer perform work on behalf of the application. The exact nature of the error is written to a log file.

See Also

[TXBEGIN\(3cbl\)](#), [TXCOMMIT\(3cbl\)](#), [TXINFORM\(3cbl\)](#), [TXOPEN\(3cbl\)](#), [TXROLLBACK\(3cbl\)](#)

USERLOG(3cbl)

Name

`USERLOG()` - write a message to the BEA Tuxedo ATMI central event log

Synopsis

```
01 LOG-REC.  
   COPY User data.  
01 LOGREC-LEN PIC S9(9) COMP-5.  
01 TPSTATUS-REC.
```



```
COPY TPSTATUS.
CALL "USERLOG" USING LOG-REC LOGREC-LEN TPSTATUS-REC.
```

Description

USERLOG () places LOG-REC into a fixed output file—the BEA Tuxedo ATMI central event log.

The central event log is an ordinary UNIX file whose pathname is composed as follows:

- If the shell variable ULOGPFX is set, its value is used as the prefix for the filename. If ULOGPFX is not set, ULOG is used. The prefix is determined the first time USERLOG () is called.
- Each time USERLOG () is called the date is determined, and the month, day, and year are concatenated to the prefix as `mmddyy` to set the name for the file.
- The first time a process writes to the user log, it first writes an additional message indicating the associated BEA Tuxedo version.

The message is then appended to the file. With this scheme, processes that call USERLOG () on successive days will write into different files.

- Messages are appended to the log file with a tag made up of the time (`hhmmss`), system name, process name, and process-id of the calling process. The tag is terminated with a colon (:).
- BEA Tuxedo system-generated error messages in the log file are prefixed by a unique identification string of the form:

```
catalog>:number>:
```

- This string gives the name of the internationalized catalog containing the message string, plus the message number. By convention, BEA Tuxedo system-generated error messages are used only once, so the string uniquely identifies a location in the source code.
- If the last character of the *format* specification is not a newline character, USERLOG () appends one.
- If the first character of the shell variable ULOGDEBUG is 1 or y, the message sent to USERLOG () is also written to the standard error of the calling process.
- USERLOG () is used by the BEA Tuxedo system to record a variety of events.
- The USERLOG mechanism is entirely independent of any database transaction logging mechanism.

Portability

The USERLOG interface is supported on UNIX and MS-DOS operating systems. The system name produced as part of the log message is not available on MS-DOS systems; therefore, the value PC is used as the system name for MS-DOS systems.

Examples

If the variable ULOGPFX is set to /application/logs/log and if the first call to USERLOG() occurred on 9/7/90, the log file created is named /application/logs/log.090790. If the call:

```
01 LOG-REC PIC X(15) VALUE "UNKNOWN USER".  
01 LOGREC-LEN PIC S9(9) VALUES IS 13.  
CALL "USERLOG" USING LOG-REC LOGREC-LEN TPSTATUS-REC.
```

is made at 4:22:14pm on the UNIX named logsys by the program whose process ID is 23431, the following line appears in the log file:

```
162214.logsys!security.23431: UNKNOWN USER
```

If the message is sent to the central event log while the process is in transaction mode, the user log entry has additional components in the tag. These components consist of the literal gtrid followed by three PIC S9(9) COMP-5 hexadecimal values. The values uniquely identify the global transaction and make up what is referred to as the global transaction identifier. This identifier is used mainly for administrative purposes, but it does make an appearance in the tag that prefixes the messages in the central event log. If the foregoing message is written to the central event log in transaction mode, the resulting log entry will look like this:

```
162214.logsys!security.23431: gtrid x2 x24e1b803 x239: UNKNOWN USER
```

If the shell variable ULOGDEBUG has a value of y, the log message is also written to the standard error of the program named security.

Errors

USERLOG() hangs if the message sent to it is larger than BUFSIZ as defined in stdio.h

Diagnostics

USERLOG() returns values include the inability to open, or write to the current log file. Inability to write to the standard error, when ULOGDEBUG is set, is not considered an error.

Notices

It is recommended that applications' use of `USERLOG` messages be limited to messages that can be used to help debug application errors; flooding the log with incidental information can make it hard to spot actual errors.

