BEA TUXEDO

Reference Manual

Section 3CBL – COBOL Functions

BEA TUXEDO 6.5 for WLE 5.1
Document Edition 6.5
May 2000

BEA TUXEDO Reference Manual

Document Edition	Date	Software Version
6.5	May 2000	BEA TUXEDO 6.5 for WLE 5.1

Contents

About This Document

The Tuxedo 6.5 Reference Manual for BEA WebLogic Enterprise™ 5.1 includes the following components:

- "Section 1 — Commands" provides information about shell-level commands included with Tuxedo® and WebLogic Enterprise software.
- "Section 3C — C Functions" describes C language functions that comprise the .Application-Transaction Monitor Interface (ATMI). ATMI provides routines to open and close resources, manage transactions, manage typed buffers, and invoke request/response and conversational service calls.
- "Section 3CBL — COBOL Functions" describes the COBOL bindings for the ATMI interface.
- "Section 3 FML — FML Commands" describes C language functions for defining and manipulating Field Manipulation Language (FML) storage structures.
- "Secrtion 5 — File Formats and Data Descriptions" describes various files and tables. This includes the configuration files, `UBBCONFIG` and `TUXCONFIG`, and the Tuxedo Management Information Base (TMIB) classes that provide an interface for managing WLE or Tuxedo systems.

What You Need to Know

This document is intended for administrators who configure operational parameters that support mission-critical BEA WebLogic Enterprise and BEA Tuxedo systems.

e-docs Web Site

The BEA WebLogic Enterprise product documentation is available on the BEA Systems, Inc. corporate Web site. From the BEA Home page, click the Product Documentation button 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 WebLogic Enterprise 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 WebLogic Enterprise documentation Home page, click the PDF Files button, and select the document you want to print.

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

Related Information

For more information about CORBA, Java 2 Enterprise Edition (J2EE), BEA Tuxedo, distributed object computing, transaction processing, C++ programming, and Java programming, see the WLE *Bibliography* in the WebLogic Enterprise online documentation.

Contact Us!

Your feedback on the BEA WebLogic Enterprise 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 Systems, Inc. professionals who create and update the WebLogic Enterprise documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Enterprise 5.1 release.

If you have any questions about this version of BEA WebLogic Enterprise, or if you have problems installing and running BEA WebLogic Enterprise, 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.

Convention	Item
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Examples*: LPT1 SIGNON OR
`{ }`	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.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

Convention	Item
|	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

`...`	Indicates one of the following in a command line:
	■ 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.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

. . .	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)

Introduction to the COBOL Application-Transaction Monitor Interface

Description

The application-transaction monitor interface 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.

BEA Tuxedo Request/Respons e Client/Server Model

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 system 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 system 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 either when 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 runtime 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 Client/Server Model

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 system 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 verbs that queue messages and dequeue responses are `TPENQUEUE`() and `TPDEQUEUE`(). They can be called from any type of BEA Tuxedo system application processes: client, server, or conversational.

The queued message facility is an XA-compliant resource manager. 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 verbs for defining and managing transactions: BEA Tuxedo system’s ATMI transaction demarcation verbs (which are prefaced with `TP`) and X/Open’s TX Interface (whose verbs are prefaced with `TX`). Because X/Open used ATMI’s transaction demarcation verbs as the base for the TX Interface, the syntax and semantics of the TX Interface are quite similar to ATMI. This section is an overview of ATMI’s transaction concepts. The following section introduces additional concepts of the TX Interface.

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 performed in many programs, at possibly 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 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 (that is, 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 either calling 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 verbs. An application writer may use either set of verbs when writing clients and service routines. In fact, the BEA Tuxedo system does not require all client and server programs within a single application to use one set of verbs or the other. However, the two verb 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(3) 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 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 ubbconfig(5). 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 system 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 system 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 re-issue 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.

Beginning in Release 6.4, some additional detail is provided beyond the TPESVCERR error code. If a service failed due to exceeding the timeout threshold, an event,.SysServiceTimeout, is posted and a call to tperrordetail(3) will return the TPED_SVCTIMEOUT error code.

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 at run time the set of services that it offers, 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. Refer to the C language binding `intro`(3) section for details. `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-TYPE`s.

`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(1) 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 (PICS9(9) COMP-5)`, `shorts (PICS9(4) COMP-5)`, and `characters (PICX(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. Refer to the C language binding `intro`(3c) section for details.

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 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 the program 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 only be associated with a single application and cannot act as clients.

In addition to allowing multiple application contexts per process, the BEA Tuxedo system also provides facilities to allow multiple application threads per process. However, since threading support is not provided in the COBOL language, multi-threading features are only available in the C language interface.

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

Per-Process Context Modes

Function	**States**		
	Uninitialized **S_0**	**Initialized Single-context Mode** **S_1**	**Initialized Multi-context Mode** **S_2**
`TPINITIALIZE` without `TP-MULTI-CONTEXTS`	S_1	S_1	S_2 (*error*)
`TPINITIALIZE` with `TP-MULTI-CONTEXTS`	S_2	S_1 (*error*)	S_2
implicit `TPINITIALIZE`	S_1	S_1	S_2 (*error*)
`TPTERM` - not last association			S_2
`TPTERM` - last association		S_0	S_0
`TPTERM` - 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 by the information passed in the *`TPINFDEF-REC`* data structure at `TPINITIALIZE()` time and in part by the system based on the processor at 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 `TPCALL()` and `TPFORWAR()` 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 routine `TPNOTIFY()` 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      PICS9(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 TPEMAXVAL      VALUE 24.
05 TPEVENT       PICS9(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     PICS9(9) COMP-5.
88 TPED-NOEVENT    VALUE 0.
88 TPEV-SVCTIMEOUT   VALUE 1.
88 TPEV-TERM      VALUE 2.
05 APPL-RETURN-CODE  PICS9(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 amount received.

```
*
* TPTYPE.cbl
*
05 REC-TYPE   PICX(8).
88 X-OCTET  VALUE "X_OCTET".
88 X-COMMON VALUE "X_COMMON".
05 SUB-TYPE   PICX(16).
05 LEN        PICS9(9) COMP-5.
88 NO-LENGTH VALUE 0.
05 TPTYPE-STATUSPICS9(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 TPSENDONLY        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 USRNAME       PICX(30).
05 CLTNAME       PICX(30).
05 PASSWD        PICX(30).
05 GRPNAME       PICX(30).
05 NOTIFICATION-FLAG  PICS9(9) COMP-5.
88 TPU-SIG       VALUE 1.
88 TPU-DIP       VALUE 2.
88 TPU-IGN       VALUE 3.
05 ACCESS-FLAG     PICS9(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       PICS9(9) COMP-5.
```

The `TPCONTEXTDEF` data structure is used by `TPGETCTXT`(3cbl) and `TPSETCTXT`(3cbl) 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 enqueuing the message.

```
*
* TPQUEDEF.cbl
*
05 TPBLOCK-FLAG     PICS9(9) COMP-5.
88 TPNOBLOCK      VALUE 0.
88 TPBLOCK       VALUE 1.
05 TPTRAN-FLAG     PICS9(9) COMP-5.
88 TPNOTRAN       VALUE 0.
88 TPTRAN        VALUE 1.
```

```
05 TPTIME-FLAG     PICS9(9) COMP-5.
88 TPNOTIME       VALUE 0.
88 TPTIME        VALUE 1.
05 TPSIGRSTRT-FLAG   PICS9(9) COMP-5.
88 TPNOSIGRSTRT     VALUE 0.
88 TPSIGRSTRT      VALUE 1.
05 TPNOCHANGE-FLAG   PICS9(9) COMP-5.
88 TPNOCHANGE      VALUE 0.
88 TPCHANGE       VALUE 1.
05 TPQUE-ORDER-FLAG   PICS9(9) COMP-5.
88 TPQDEFAULT      VALUE 0.
88 TPQTOP        VALUE 1.
88 TPQBEFOREMSGID    VALUE 2.
05 TPQUE-TIME-FLAG   PICS9(9) COMP-5.
88 TPQNOTIME      VALUE 0.
88 TPQTIME-ABS     VALUE 1.
88 TPQTIME-REL     VALUE 2.
05 TPQUE-PRIORITY-FLAG PICS9(9) COMP-5.
88 TPQNOPRIORITY    VALUE 0.
88 TPQPRIORITY     VALUE 1.
05 TPQUE-CORRID-FLAG  PICS9(9) COMP-5.
88 TPQNOCORRID     VALUE 0.
88 TPQCORRID      VALUE 1.
05 TPQUE-REPLYQ-FLAG  PICS9(9) COMP-5.
88 TPQNOREPLYQ     VALUE 0.
88 TPQREPLYQ      VALUE 1.
05 TPQUE-FAILQ-FLAG   PICS9(9) COMP-5.
88 TPQNOFAILUREQ    VALUE 0.
88 TPQFAILUREQ     VALUE 1.
05 TPQUE-MSGID-FLAG   PICS9(9) COMP-5.
88 TPQNOMSGID      VALUE 0.
88 TPQMSGID       VALUE 1.
05 TPQUE-GETBY-FLAG   PICS9(9) COMP-5.
88 TPQGETNEXT      VALUE 0.
88 TPQGETBYMSGID    VALUE 1.
88 TPQGETBYCORRID    VALUE 2.
05 TPQUE-WAIT-FLAG   PICS9(9) COMP-5.
88 TPQNOWAIT      VALUE 0.
88 TPQWAIT       VALUE 1.
05 DIAGNOSTIC      PICS9(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 QMENOTA       VALUE -8.
88 QMEPROTO       VALUE -9.
```

```
88 QMEBADQUEUE     VALUE -10.
88 QMENOMSG       VALUE -11.
88 QMEINUSE       VALUE -12.
88 QMENOSPACE      VALUE -13.
05 DEQ-TIME       PICS9(9) COMP-5.
05 PRIORITY       PICS9(9) COMP-5.
05 MSGID        PICX(32).
05 CORRID        PICX(32).
05 QNAME        PICX(15).
05 QSPACE-NAME     PICX(15).
05 REPLYQUEUE      PICX(15).
05 FAILUREQUEUE     PICX(15).
05 CLIENTID OCCURS4 TIMESPICS9(9) COMP-5.
05 APPL-RETURN-CODE   PICS9(9) COMP-5.
05 APPKEY        PICS9(9) COMP-5.
```

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

```
*
* TPSVCRET.cbl
*
05 TP-RETURN-VAL    PICS9(9) COMP-5.
88 TPSUCCESS      VALUE 0.
88 TPFAIL        VALUE 1.
88 TPEXIT        VALUE 2.
05 APPL-CODE      PICS9(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        PICS9(9) COMP-5 VALUE IS0.
05 TRANID       OCCURS6 TIMESPICS9(9) COMP-5.
```

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

```
*
* TPCMTDEF.cbl
*
05 CMT-FLAG          PICS9(9) COMP-5.
88 TP-CMT-LOGGED       VALUE 1.
88 TP-CMT-COMPLETE      VALUE 2.
05 PREV-CMT-FLAG       PICS9(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      PICS9(9) COMP-5.
88 TPNOAUTH      VALUE 0.
88 TPSYSAUTH      VALUE 1.
88 TPAPPAUTH      VALUE 2.
```

The *TPPRIDEF* data structure is used by `TPSPRIO()` and `TPGPRIO()` to manipulate message priorities.

```
*
* TPPRIDEF.cbl
*
05 PRIORITY       PICS9(9) COMP-5.
05 PRIO-FLAG      PICS9(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    PICS9(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      PICS9(9) COMP-5.
88 TPBLOCK         VALUE 0.
88 TPNOBLOCK        VALUE 1.
05 TPTIME-FLAG       PICS9(9) COMP-5.
88 TPTIME         VALUE 0.
88 TPNOTIME        VALUE 1.
05 TPSIGRSTRT-FLAG     PICS9(9) COMP-5.
88 TPNOSIGRSTRT      VALUE 0.
88 TPSIGRSTRT       VALUE 1.
05 LMID          PICX(30).
05 USERNAME        PICX(30).
05 CLTNAME         PICX(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.
*
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 TPEVNOTRAN      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).
```

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      PICS9(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   PICS9(9) COMP-5.
*  A value of -1 in FORMAT-ID means that the XID is null
10 GTRID-LENGTH  PICS9(9) COMP-5.
10 BRANCH-LENGTH PICS9(9) COMP-5.
10 XID-DATA        PICX(128).
05 TRANSACTION-MODE    PICS9(9) COMP-5.
*  Transaction mode settings
88 TX-NOT-IN-TRAN     VALUE 0.
88 TX-IN-TRAN       VALUE 1.
05 COMMIT-RETURN     PICS9(9) COMP-5.
*  Commit_return settings
88 TX-COMMIT-COMPLETED  VALUE 0.
88 TX-COMMIT-DECISION-LOGGED VALUE 1.
05 TRANSACTION-CONTROL  PICS9(9) COMP-5.
*  Transaction_control settings
88 TX-UNCHAINED      VALUE 0.
88 TX-CHAINED       VALUE 1.
```

```
05 TRANSACTION-TIMEOUT  PICS9(9) COMP-5.
*  Transaction_timeout value
88 NO-TIMEOUT       VALUE 0.
05 TRANSACTION-STATE   PICS9(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 request/response servers, conversational servers, and clients are allowed to call.

Note that `TPSVRINIT` and `TPSVRDONE` are not in this table since these functions are not called by applications (that is, they are application-supplied functions that are invoked by the BEA Tuxedo system).

Functions

Function	Process Type		
	Request/response Server	Conversational Server	Client
TPABORT	Y	Y	Y
TPACALL	Y	Y	Y
TPADVERTISE	Y	Y	N
TPBEGIN	Y	Y	Y
TPBROADCAST	Y	Y	Y
TPCALL	Y	Y	Y
TPCANCEL	Y	Y	Y
TPCHKAUTH	Y	Y	Y

Functions

Function	Process Type		
	Request/response Server	Conversational Server	Client
TPCHKUNSOL	N	N	Y
TPCLOSE	Y	Y	Y
TPCOMMIT	Y	Y	Y
TPCONNECT	Y	Y	Y
TPDEQUE	Y	Y	Y
TPDISCON	Y	Y	Y
TPENQUEUE	Y	Y	Y
TPFORWAR	Y	N	N
TPGETCTXT	Y	Y	Y
TPGETLEV	Y	Y	Y
TPGETRPLY	Y	Y	Y
TPGPRIO	Y	Y	Y
TPINITIALIZE	N	N	Y
TPNOTIFY	Y	Y	Y
TPOPEN	Y	Y	Y
TPPOST	Y	Y	Y
TPRECV	Y	Y	Y
TPRESUME	Y	Y	Y
TPRETURN	Y	Y	N
TPSCMT	Y	Y	Y
TPSEND	Y	Y	Y

Functions

Function	Process Type		
	Request/response Server	Conversational Server	Client
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 can not 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, in single-context mode a client may implicitly join an application by issuing one of many ATMI function calls (for example, TPACALL or TPCALL). A client must use TPINITIALIZE in any of the following circumstances:

- Application authentication is required (see TPINITIALIZE and the description of the SECURITY keyword in ubbconfig(5))
- The client wants to access an XA-compliant resource manager directly (see TPINITIALIZE(3cbl)
- 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.

Initialization States

Function	**States**	
	Uninitialized $\mathbf{I_0}$	**Initialized** $\mathbf{I_1}$
`TPCHKAUTH`	I_0	I_1
`TPGETCTXT`	I_0	I_1
`TPINITIALIZE`	I_1	I_1
`TPSETCTXT`	I_1	I_1
`TPSETUNSOL`	I_0	I_1
`TPTERM`	I_0	I_0
`tptypes`	I_0	I_1
All others		I_1

Note: *All others* refers to the remaining ATMI calls.

The remaining state tables assume a precondition of state I (regardless of whether a process arrived in this state via `tpinit` or the BEA Tuxedo *`main()`*).

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.

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
`TPSVCSTART` with `TPTRAN`		R_1
All others	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`).

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	

Transaction State of Application Association

Function	**State**		
	Not in transaction T_0	**Initiator** T_1	**Participant** T_2
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
TPCLOSE	R_0		
TPTERM	I_0	T_0	
all others	T_0	T_1	T_2

The following state table indicates the state of a single request handle returned by TPACALL(3cbl).

Asynchronous Request Descriptor States

Function	**States**	
	No Descriptor $\mathbf{A_0}$	**Valid Descriptor** $\mathbf{A_1}$
TPACALL	A_1	
TPGETRPLY		A_0
TPCANCEL		A_0 *
TPABORT	A_0	A0†

Asynchronous Request Descriptor States

Function	States	
	No Descriptor A_0	**Valid Descriptor A_1**
`TPCOMMIT`	A_0	A_0†
`TPSUSPEND`	A_0	A‡
`TPRETURN`	A_0	A_0
`TPFORWAR`	A_0	A_0
`TPTERM`	I_0	I_0
All others	A_0	A_1

Note: * This state change occurs only if the descriptor is not associated with the caller’s transaction.

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

‡ 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
```

Connection Request Handle States

Function/Event	States				
	C_0	C_1	C_2	C_3	C_4
TPCONNECT with TPSENDONLY	C_1 *				
TPCONNECT with TPRECVONLY	C_2 *				
TPSVCSTART with flag TPSENDONLY	C_3 †				
TPSVCSTART with flag TPRECVONLY	C_4 †				
TPRECV/no event			C_2		C_4
TPRECV/TPEV_SENDONLY			C_1		C_3
TPRECV/TPEV_DISCONIMM			C_0		C_0
TPRECV/TPEV_SVCERR			C_0		
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_0 ‡		

Connection Request Handle States

Function/Event	**States**				
	$\mathbf{C_0}$	$\mathbf{C_1}$	$\mathbf{C_2}$	$\mathbf{C_3}$	$\mathbf{C_4}$
`TPSUSPEND (originator only)`	C_0	C_0††	C_0††		
`TPABORT (originator only)`	C_0	C_0 ‡	C_0 ‡		
`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 others	C_0	C_1	C_2	C_3	C_4

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

† If the `TPTRAN` flag is set, the connection is in transaction mode.

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

†† 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 verbs 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 primitives 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 verbs.

The states are defined below:

S_0

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

S_1

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

S_2
: A process has opened its RM but is not in a transaction. Its `transaction_control` characteristic is `TX-CHAINED`.

S_3
: A process has opened its RM and is in a transaction. Its `transaction_control` characteristic is `TX-UNCHAINED`.

S_4
: A process has opened its RM and is in a transaction. Its `transaction_control` characteristic is `TX-CHAINED`.

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
`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
`TXSETRRANCTL` `control = TX-UNCHAINED`		S_1	S_1	S_3	S_3
`TXSETTIMEOUT`		S_1	S_2	S_3	S_4

1. TX_SET1 denotes any of 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).
2. TX_SET2 denotes any of 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).
3. If TX_FAIL is returned on any call, the application process is in an undefined state with respect to the above table.
4. 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 will be rolled back whether the application program calls tx_commit or tx_rollback.

See Also buffer(3c), TPADVERTISE(3cbl), TPBEGIN(3cbl), TPCALL(3cbl), TPCONNECT(3cbl), TPGETCTXT(3cbl), INITIALIZE(3cbl), TPOPEN(3cbl), TPSETCTXT(3cbl), TPSVCSTART(3cbl), tuxtypes(5), typesw(5)

FINIT(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 was to reinitialize a buffer to have no fields is: `Finit(frfr`, (`FLDLEN`)`Fsizeof(fbfr`));

See Also

`Fintro`()

FVFTOS(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 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 `Fintro`(), `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`(3), `Fojoin`(3), `Fjoin`(3), and `Fconcat`(3). 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 `Fintro`(), `viewfile`(5)

TPABORT(3CBL)

Name `TPABORT`—abort current BEA Tuxedo system 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 `TPFAIL`.

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.

Currently, `TPABORT`'s 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 system 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(), TPCOMMIT(), TPGETLEV()

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-TYPE`s recognized by `SERVICE-NAME`. Note that for each request sent while in transaction mode, a corresponding reply must ultimately be received.

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 re-issued. 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`]
: Can not 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 sub-types 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`]
: A timeout occurred. If the caller is in transaction mode, then a transaction timeout occurred and the transaction is marked abort-only; otherwise, a blocking timeout occurred and both `TPBLOCK` and `TPTIME` were specified. If a transaction timeout occurred, then any attempts to send new requests or receive outstanding replies will fail with [`TPETIME`] until the transaction has been aborted.

[`TPEBLOCK`]
: A blocking condition exists and `TPNOBLOCK` was specified.

[TPGOTSIG]
A signal was received and TPNOSIGRSTRT was specified.

[TPEPROTO]
TPACALL was called in an improper context.

[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(), TPCANCEL(), TPGETRPLY(), TPGPRIO(), TPSPRIO()

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 shutdown.

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 system 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 in an improper context.

[`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`()

TPBEGIN(3CBL)

Name TPBEGIN—routine to begin a BEA Tuxedo system 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 time-out 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 system 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 system transaction.

See Also `TPABORT`(), `TPCOMMIT`(), `TPGETLEV`(), `TPSCMT`()

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`, `USRNAME` 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.

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 and both `TPBLOCK` and `TPTIME` were 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 in an improper context.

[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.)

Note that signaling of clients is always done by the system so that the behavior of notification is consistent regardless of where the originating notification call is made. Because of this, only clients running as the application administrator can use signal-based notification. The id for the application administrator is identified as part of 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() for more information on notification method selection.

See Also TPINITIALIZE(), TPNOTIFY(), TPTERM(), 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 sub-type 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.

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 sub-type 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 re-issued. 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 then 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]

Can not 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 sub-types that SERVICE-NAME accepts.

[TPEOTYPE]

Either the type and sub-type 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 sub-type 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]

A timeout occurred. If the caller is in transaction mode, then a transaction timeout occurred and the transaction is marked abort-only; otherwise, a blocking timeout occurred and both TPBLOCK and TPTIME were specified. In either case, neither *ODATA-REC* nor *OTPTYPE-REC* are changed. If a transaction timeout 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.

[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 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).

[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 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 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 in an improper context.

[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(), TPFORWARD(), TPGPRIO(), TPRETURN(), TPSPRIO(), tperrordetail(3), tpstrerrordetail(3)

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 in an improper context.

[`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`()

TPCHKAUTH(3CBL)

Name TPCHKAUTH—check if authentication required to join a BEA Tuxedo system 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. However, signal-based notification is not supported on MS-DOS. If it is selected at TPCHKAUTH() time, then a USERLOG() message is generated and the method is automatically set to dip-in.

See Also TPINITIALIZE()

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 system 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 (e.g., 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) Note that signaling of clients is always done by the system so that the behavior of notification is consistent regardless of where the originating notification call is made. Because of this, only clients running as the application administrator can use signal-based notification. The id for the application administrator is identified as part of 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`() for more information on notification method selection.

See Also `TPBROADCAST`(), `TPINITIALIZE`(), `TPNOTIFY`(), `TPSETUNSOL`()

TPCLOSE(3CBL)

Name `TPCLOSE`—close the BEA Tuxedo system 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`()

TPCOMMIT(3CBL)

Name TPCOMMIT—commit current BEA Tuxedo system 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 timed out and the status of the transaction is unknown (that is, it can have been either committed or aborted). Note that if the transaction timed out and its status is known to be aborted, then [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 can 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 system 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 system transaction.

See Also `TPABORT`(), `TPBEGIN`(), `TPCONNECT`(), `TPGETLEV`(), `TPRETURN`(), `TPSCMT`()

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 sub-types 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`.

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, then the interrupted call is re-issued. Either TPNOSIGRSTRT or TPSIGRSTRT must be set.

TPNOSIGRSTRT
: When TPNOSIGRSTRT is specified and a signal is received, 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 sub-types 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]
: A timeout occurred. If the caller is in transaction mode, then a transaction timeout occurred and the transaction is marked abort-only; otherwise, a blocking timeout occurred and both TPBLOCK and TPTIME were specified. If a transaction timeout occurred, then any attempts to send or receive messages on any connections or to start a new connection will fail with [TPETIME] until the transaction has been aborted.

[TPEBLOCK]
: A blocking condition exists and TPNOBLOCK was specified.

[TPGOTSIG]
: A signal was received and TPSIGRSTRT was not specified.

[TPEPROTO]
: TPCONNECT was called in an improper context.

[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(), TPRECV(), TPSEND()

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` dequeues 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 default 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`. *`TPQUEDEF-REC`* settings can also be used to indicate that the application wants to wait for a message, in the case where a message is not currently available. See the section below describing this record.

`DATA-REC` specifies where a dequeued message is 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 sub-type 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 deleted 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 deletion of the message from the queue is also rolled back). This can be exploited to "peek" at a message on the queue, rolling back the transaction to leave the message on the queue (note that this cannot be done if `TPNOTRAN` is set as described below). 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. The message is dequeued in a separate transaction. If a communication error or a timeout occurs (either transaction or blocking timeout), the application will not know whether or not the message was successfully dequeued and the message may be lost.

Following is a list of valid settings in *`TPQUEDEF-REC`*.

`TPNOTRAN`
: If the caller is in transaction mode and this setting is used, then the message is not dequeued within the same transaction as the caller. 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, then 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.

`TPNOCHANGE`
: When this setting is used, the type of *`DATA-REC`* is not allowed to change. That is, the type and sub-type 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.

`TPNOBLOCK`
: The message is not dequeued if a blocking condition exists (for example, the internal buffers into which the message is transferred are full). This blocking condition does not include blocking on the queue itself if the `TPQWAIT` option is specified. 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 re-issued. 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.

If `TPDEQUEUE` returns successfully, the application can retrieve additional information about the message using *`TPQUEDEF-REC`*. 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 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. This is 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).
```

Following is a list of valid settings in *TPQUEDEF-REC* controlling input information for TPDEQUEUE.

TPQGETNEXT
: If set, it requests that the next message on the queue be dequeued, using the default queue order. Either TPQGETNEXT, TPQGETBYMSGID or TPQGETBYCORRID must be set.

TPQGETBYMSGID
: If set, it requests that the message identified by MSGID be dequeued. The message identifier would be one that was returned by a prior call to TPENQUEUE(). Note that the message identifier is not valid if the message has moved from one queue to another; in this case, use the correlation identifier. Either TPQGETNEXT, TPQGETBYMSGID or TPQGETBYCORRID must be set.

TPQGETBYCORRID
: If set, it requests that the message identified by CORRID be dequeued. The correlation identifier would be one that the application specified when enqueuing the message with TPENQUEUE. Either TPQGETNEXT, TPQGETBYMSGID or TPQGETBYCORRID must be set.

TPQWAIT
: This setting indicates that an error should not be returned if the queue is empty. Instead, the process should block until a message is available. Set TPQNOWAIT to not wait until a message is available. TPQWAIT cannot be set if either TPQGETBYMSGID or TPQGETBYCORRID is set.

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 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.
```

Following is a list of valid settings controlling output information from `TPDEQUEUE`. If the setting is true when `TPDEQUEUE` is called, the associated element in the record is populated if available and the setting remains true. If the value is not available, the setting will not be true after `TPDEQUEUE` completes.

`TPQPRIORITY`
: If set and the value is available, the priority at which the message was queued 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.

`TPQMSGID`
: If set and the call to `TPDEQUEUE` was successful, the message identifier will be stored in `MSGID`. If `TPQNOMSGID` is set, the message identifier is not available.

`TPQCORRID`
: If set and the call to `TPDEQUEUE` was successful and the message was queued with a correlation identifier, the value will be stored in `CORRID`. Any reply to a queue must have this correlation identifier. If `TPQNOCORRID` is set, the correlation identifier is not available.

`TPQREPLYQ`
: If set and the message is associated with a reply queue, the value will be 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 set and the message is associated with a failure queue, the value will be 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.

If the call to `TPDEQUEUE` 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.

Additionally on output, APPKEY is set to 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 (the associated TMQUEUE(1) server is not available).

[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 sub-type of the dequeued message. Neither *DATA-REC* nor *TPTYPE-REC* are changed. If the call was made on behalf of the caller's current transaction, then the transaction is marked abort-only and the message will remain 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]
: A timeout occurred. If the caller is in transaction mode, then a transaction timeout occurred and the transaction is marked abort-only; otherwise, a blocking timeout occurred and both TPBLOCK and TPTIME were specified. In either case, neither *DATA-REC* nor *TPTYPE-REC* are changed. If a transaction timeout occurred, then any attempts to call TPDEQUEUE or TPENQUEUE will fail with TPETIME until the transaction has been aborted.

[TPEBLOCK]
: A blocking condition exists and TPBLOCK was set.

[TPGOTSIG]
: A signal was received and TPNOSIGRSTRT was set.

[TPEPROTO]
: `TPDEQUEUE` was called in an improper context. 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 made with `TPNOTRAN` set and an error occurred trying to start a transaction in which to dequeue the message.

[QMEBADMSGID]
: An invalid message identifier was specified for dequeuing.

[QMEINUSE]
: When dequeuing a message by correlation or message identifier, the specified message is in-use by another transaction. Otherwise, all messages currently on the queue are in-use by other transactions.

[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 or deleted 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.

See Also `TPENQUEUE()`

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]
A timeout occurred. The communications handle is no longer valid.

[TPEPROTO]
: TPDISCON was called in an improper context.

[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(), TPCOMMIT(), TPCONNECT(), TPRECV(), TPRETURN(), TPSEND()

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 system server, the `QNAME` matches the name of a service provided by a 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 expected a reply, then the reply to the forwarded service request is stored on the originator's (stable) queue. 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 system processes (clients and/or servers). In this case, the queue name does not match a service name but some agreed upon title 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-TYPE`s and `SUB-TYPE`s 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 deleted 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. In this case, the queued message is stored on the queue in a separate transaction. Once `TPENQUEUE` returns successfully, the submitted message is guaranteed to be available. If a communication error or a timeout occurs (either transaction or blocking timeout), 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.

Following is a list of valid settings in *`TPQUEDEF-REC`*.

`TPNOTRAN`

> If the caller is in transaction mode and this setting is used, then the message is not enqueued within the same transaction as the caller. A caller in transaction mode that sets this to true is still subject to the transaction timeout (and no other). If message enqueuing 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 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 (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 re-issued. 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.

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; 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 enqueuing 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 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.

`TPQTIME-ABS`
: If set, the message is made available after the time specified by `DEQ-TIME`. `DEQ-TIME` is an absolute time value as generated by `time`() or `mktime`() (the number of seconds since 00:00:00 UTC, January 1, 1970). Set `TPQNOTIME` if neither an absolute or relative time value is set. `TPQTIME-ABS, TPQTIME-REL,` or `TPQNOTIME` must be set.

`TPQTIME-REL`
: If set, the message is made available after a time relative to the completion of the queuing transaction. `DEQ-TIME` specifies the number of seconds to delay after the transaction completes before the submitted message should be available. Set `TPQNOTIME` if neither an absolute or relative time value is set. `TPQTIME-ABS, TPQTIME-REL,` or `TPQNOTIME` must be set.

`TPQPRIORITY`
: If 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). Set `TPQNOPRIORITY` if a priority value is not available.

TPQCORRID

> If 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 such that an application can correlate a reply with a particular request. Set TPQNOCORRID if a correlation identifier is not available.

TPQREPLYQ

> If set, a reply queue named in REPLYQUEUE is associated with the queued message. Any reply to the message will be 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 set, a failure queue named in FAILUREQUEUE is associated with the queued message. If a failure occurs when the enqueued message is subsequently dequeued, a failure message will go to the named queue within the same queue space as the original request message. Set TPQNOFAILUREQ if a failure queue name is not available.

Additionally, APPL-RETURN-CODE can be set with a user-return code. This value will be 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.
```

Following is a list of valid settings controlling output information from TPENQUEUE. If the setting is true when TPENQUEUE is called, the associated element in the record is populated if available and the setting remains true. If the value is not available, the setting will not be true after TPENQUEUE completes.

TPQMSGID

> If set and the call to TPENQUEUE was successful, the message identifier will be stored in MSGID. TPQNOMSGID is set if a message identifier is not available.

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 (the associated TMQUEUE(5) server is not available).

[TPETIME]
: A timeout occurred. If the caller is in transaction mode, then a transaction timeout occurred and the transaction is marked abort-only; otherwise, a blocking timeout occurred and both TPBLOCK and TPTIME were specified. If a transaction timeout occurred, then any attempts to call TPDEQUEUE or TPENQUEUE will fail with TPETIME until the transaction has been aborted.

[TPEBLOCK]
: A blocking condition exists and TPBLOCK was set.

[TPGOTSIG]
: A signal was received and TPNOSIGRSTRT was set.

[TPEPROTO]
: TPENQUEUE was called in an improper context. 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 enqueuing 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 made with the `TPNOTRAN` setting and an error occurred trying to start a transaction in which to enqueue the message.

[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 or deleted queue name was specified.

[QMENOSPACE]
: There is no space on the queue for the message.

See Also

`TMQFORWARD`(5), `TMQUEUE`(5), `TPDEQUEUE`(), `TPSPRIO`()

TPFORWAR(3CBL)

Name `TPFORWAR`—forward a BEA Tuxedo system 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 system 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 system 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 system 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`(3) 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 system 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 (i.e., TPSIGRSTRT is implied). Currently, settings in *TPSVCDEF-REC* are reserved for future use and any specified are ignored. TPACALL()).

Return Values A service routine does not return any value to its caller, the BEA Tuxedo system 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 while in the service routine or while forwarding the request, the requester waiting for a reply with either TPCALL(), or TPGETRPLY() will get a TPETIME error return. Also, 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(), and 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 (e.g., in clients, or in `TPSVRINIT`() or `TPSVRDONE`()), then these routines simply return having no effect.

See Also `TPCONNECT`(), `TPRETURN`(), `tperrordetail`(3), `tpstrerrordetail`(3)

TPGETCTXT(3cbl)

Name TPGETCTXT - retrieves an identifier for a thread's context

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 thread

1. Calls TPINITIALIZE
2. Calls TPGETCTXT
3. Passes the value of CONTEXT in TPCONTEXTDEF-REC to another thread in the same process so the other thread can call TPSETCTXT

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 thread's context identifier in CONTEXT in TPCONTEXTDEF-REC.

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

[TPEINVAL]
: Invalid arguments have been given.

[TPEPROTO]
: TPGETCTXT has been called in a thread that has no current context.

[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 INTRO(3cbl), TPSETCTXT(3cbl)

TPGETLEV(3CBL)

Name

TPGETLEV—check if a BEA Tuxedo system 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 in an improper context.

[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 system 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 system transaction.

See Also

TPABORT(), TPBEGIN(), TPCOMMIT(), TPSCMT()

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 sub-type, 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.

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 sub-type 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 re-issued. 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 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, 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 sub-type 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 sub-type 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]
: A timeout occurred. If the caller is in transaction mode, then a transaction timeout occurred and the transaction is marked abort-only; otherwise, a blocking timeout occurred and both TPBLOCK and TPTIME were specified. In either case, neither *DATA-REC* nor *TPTYPE-REC* are changed. If TPGETHANDLE was set, COMM-HANDLE remains valid unless the caller is in transaction mode. If a transaction timeout occurred, then any attempts to send new requests or receive outstanding replies will fail with [TPETIME] until the transaction has been aborted.

[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 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).

[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 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. COMM-HANDLE remains valid.

[TPGOTSIG]
A signal was received and TPSIGRSTRT was not specified.

[TPEPROTO]
TPGETRPLY was called in an improper context.

[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(), TPCANCEL(), TPRETURN(), tperrordetail(3), tpstrerrordetail(3)

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 sub-type, 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 in an improper context.

[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`()

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 in an improper context.

[`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(), TPCALL(), TPDEQUEUE(), TPENQUEUE(), TPSPRIO()

TPINITIALIZE(3CBL)

Name `TPINITIALIZE`—joins a BEA Tuxedo system 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 system application. Before a client can use any of the BEA Tuxedo communication or transaction routines, it must first join a BEA Tuxedo system application. Because calling `TPINITIALIZE` is optional, a 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 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.

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.

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.

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 USRNAME 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 USRNAME, CLTNAME, and GRPNAME; default settings; and no application data.

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 calls 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.

Return Values Upon successful completion, TPINITIALIZE sets TP-STATUS to [TPOK].

Errors Under the following conditions, TPINITIALIZE fails and 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 in an improper context. For example, the `TP-MULTI-CONTEXTS` setting was specified in single-context mode, or 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.

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"
```

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 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.

`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`
: When connecting to BEA Tuxedo, require at least this minimum level of encryption. “0” means no encryption, while “40” and “128” specify the encryption key length (in bits). If this minimum level of encryption cannot be met, link establishment will fail. The default value is “0”.

`TMMAXENCRYPTBITS`
When connecting to BEA Tuxedo, negotiate encryption up to this level. “0” means no encryption, while “40” 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. Note that signaling of clients is always done by the system so that the behavior of notification is consistent regardless of where the originating notification call is made. Because of this, only clients running as the application administrator can use signal-based notification. 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 may fail, returning `TPGOTSIG` due to receipt of an unsolicited message if `TPSIGRSTRT` is not specified.

See Also `TPGETCTXT`(3cbl), `TPSETCTXT`(3cbl), `TPTERM`(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.

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 and both `TPBLOCK` and `TPTIME` were specified, or `TPACK` and `TPTIME` were set and no acknowledgment was received. and `TPTIME` was specified.

[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 BEA Tuxedo which does not support the acknowledgment protocol.

See Also TPBROADCAST(3), TPCHKUNSOL(3), TPINITIALIZE(3), TPSETUNSOL(3), TPTERM(3)

TPOPEN(3CBL)

Name TPOPEN—open the BEA Tuxedo system 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 system 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()

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 `TPPOST` 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 system event broker 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 event broker. 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 event broker 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 event broker but not as part of the posting process' transaction.

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 event broker 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 event broker 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 [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 re-issued. 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 event broker 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 event broker 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 system event broker.

[TPETRAN]
: The caller is in transaction mode, TPTRAN was set, and TPPOST contacted an event broker that does not support transaction propagation (that is, TMUSREVT(5) is not running in a BEA Tuxedo system group that supports transactions).

[TPETIME]
: A timeout occurred. If the caller is in transaction mode, then a transaction timeout occurred and the transaction is to be aborted; otherwise, a blocking timeout occurred and both TPBLOCK and TPTIME were specified. If a transaction timeout occurred, any attempts to do new work will fail with [TPETIME] until the transaction has been aborted.

[TPESVCFAIL]
: The event broker 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 event broker 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 in an improper context.

[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(), TPUNSUBSCRIBE(), EVENTS(5), TMUSREVT(5), TMSYSEVT(5)

TPRECV(3CBL)

Name
`TPRECV`—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 "TPRECV" USING TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

Description
`TPRECV` 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 sub-type, 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.

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 sub-type of the message received must match `REC-TYPE` and `SUB-TYPE`, respectively. Either `TPNOCHANGE` or `TPCHANGE` must be set.

`TPCHANGE`
: The type and/or sub-type 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 re-issued. 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 can not 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 [TPEEVENT] 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 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, 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 sub-type 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 sub-type 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]
: A timeout occurred. If the caller is in transaction mode, then a transaction timeout occurred and the transaction is marked abort-only; otherwise, a blocking timeout occurred and neither TPNOBLOCK nor TPNOTIME were specified. In either case, *DATA-REC* was not changed. If a transaction timeout occurred, then any attempts to send or receive messages on any connections or to start a new connection will fail with TPETIME until the transaction has been aborted.

[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(), TPDISCON(), TPSEND()

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(), TPCOMMIT(), TPOPEN(), TPSUSPEND().

TPRETURN(3CBL)

Name `TPRETURN`—return from a BEA Tuxedo system 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 system 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 system 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 (i.e., 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 where 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 system 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 (i.e., 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 TPRETURN's 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 (i.e., 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 (i.e., 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 sub-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. 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 where the application return code and the data portion are not transmitted:

\

if the connection has been terminated when the call is made (i.e., 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; however, if the originator receives the TPEV_SVCFAIL event, the return code is available in the originator's APPL-RETURN-CODE in *TPSTATUS-REC*.

Return Values Since 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 (i.e., 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 (i.e., 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().

tperrdetail(3) and tpstrerrordetail(3) can be used to get additional information about an error produced by the last BEA Tuxedo system routine called in the current thread. If an error occurred, tperrdetail returns a numeric value that can be used as an argument to tpstrerrordetail to retrieve the text of the error detail.

See Also TPCALL(), TPCONNECT(), TPFORWAR(), tperrordetail(3), tpstrerrordetail(3)

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 system 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 system application, the initial setting for this characteristic comes from a configuration file. (See the CMTRET variable in the RESOURCES section of ubbconfig(5))

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 in an improper context.

[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 system 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 system transaction.

See Also TPABORT(), TPBEGIN(), TPCOMMIT(), TPGETLEV()

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).

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 can not issue any more `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 can not issue more any `TPRECV`() calls). Either `TPRECVONLY` or `TPSENDONLY` must be set.

`TPSENDONLY`
: This setting signifies that the caller wants to remain in control of the connection. Either `TPRECVONLY` or `TPSENDONLY` 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 re-issued. 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 [TPEEVENT]. When TP-STATUS is set to [TPEEVENT] 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 exits):

[TPEINVAL]
: Invalid arguments were given.

[TPEBADDESC]
: COMM-HANDLE contains an invalid communications handle.

[TPETIME]
: A timeout occurred. If the caller is in transaction mode, then a transaction timeout occurred and the transaction is marked abort-only; otherwise, a blocking timeout occurred and neither TPNOBLOCK nor TPNOTIME were specified. In either case, neither *DATA-REC* nor *TPTYPE-REC* are changed. If a transaction timeout occurred, then any attempts to send or receive messages on any connections or to start a new connection will fail with [TPETIME] until the transaction has been aborted.

[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(), TPDISCON(), TPRECV()

TPSETCTXT(3cbl)

Name `TPSETCTXT` - sets a thread's context

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. (Multi-threaded 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 `TPNULLCONTEXT`, then the program is disassociated from any BEA Tuxedo context.

Return Values Upon successful completion, `TPSETCTXT` sets `TP-STATUS` to [`TPOK`].

Errors `TPSETCTXT` fails and sets `TPSTATUS` to one of the following values under the conditions described in the following list.

[`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 `INTRO`(3cbl), `TPSETCTXT`(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 system libraries. Before the first call to TPSETUNSOL, any unsolicited messages received by the BEA Tuxedo system 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 routine _tm_dispatch1 through _tm_dispatch8 must conform to the parameter definition described in tpsetunsol(3c). Routines TMDISPATCH9 through TMDISPATCH16 must use TPGETUNSOL() to receive the data.

Processing within the C language application unsolicited message handling routine is restricted to the following BEA Tuxedo system calls: tpalloc(3c), tpfree(3c), tpgetctxt(3c), tpgetlev(3c), tprealloc(3c), and tptypes(3c).

Processing within the COBOL language application unsolicited message handling routine is restricted to the following BEA Tuxedo call: TPGETLEV(3cbl) and TPGETCTXT(3cbl).

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 (e.g., 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), `TPINIT`(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 take the system default of 50. `TPSPRIO` has no effect on messages sent via `TPCONNECT`() or `TPSEND`().

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 in an improper context.

[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`(), `TPCALL`(), `TPDEQUEUE`(), `TPENQUEUE`(), `TPGPRIO`()

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 System Event Broker, TMUSREVT(5), 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 enqueuing 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 recomp(3). 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" (for example, "account_error" and "ERROR_STATE" events would both qualify).

EVENT-FILTER in *TPEVTDEF-REC* is a string containing a boolean filter rule associated with EVENT-EXPR that must be evaluated successfully before the event broker posts the event. Upon receiving an event to be posted, the event broker applies the filter rule, if one exists, to the posted event's data. If the data passes the filter rule, the event broker invokes the notification method associated with EVENT-EXPR; 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's boolean expression complier (see Fboolco(3) and Fvboolco(3), respectively) and evaluated against the posted record (see Fboolev(3) and Fvboolev(3), respectively). For STRING records, the filter rule is a regular expression of the form specified in recomp(3). All other record types require customized filter evaluators (see buffer(3) 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 system client process and TPEVNOTIFY in *TPEVTDEF-REC* is set, then the event broker 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 event broker 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 system 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 event broker'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 event broker 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 system service routine named by NAME-1 in *TPEVTDEF-REC*. That is, when an event name is posted that evaluates successfully against EVENT-EXPR, the event broker 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 system service name and it may or may not be active at the time the subscription is made. Service routines invoked by the event broker 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 event broker calls the subscribed service routine such that it will be part of the poster's transaction. Both the event broker, TMUSREVT(5), and the subscribed service routine must belong to server groups that support transactions (see ubbconfig(5) for details). If TPEVNOTRAN is set, then the event broker 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 event broker 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 event broker 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 system 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 event broker's enqueuing 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 event broker enqueues the posted event and its data such that it will be part of the poster's transaction. The event broker, TMUSREVT(5), must belong to a server group that supports transactions (see ubbconfig(5) for details). If TPEVNOTRAN is set, then the event broker enqueues the posted event and its data such that it will not be part of the poster's transaction.

By default, the BEA Tuxedo System Event Broker deletes subscriptions when the resource to which it is posting is not available (for example, the event broker 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 event broker 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 event broker 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 event broker'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 event broker'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 event broker.

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 event broker.

If TPEVQUEUE is set, then event broker 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 TPQNOCOORID, 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 event broker. Further, if TPQCOORID 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 event broker.

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 re-issued. 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 event broker'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 System Event Broker.

[TPELIMIT]
: The subscription failed because the event broker's maximum number of subscriptions has been reached.

[TPEMATCH]
: The subscription failed because it matched one already listed with the event broker.

[TPETIME]
: A timeout occurred. If the caller is in transaction mode, then a transaction timeout occurred and the transaction is to be aborted; otherwise, a blocking timeout occurred and both TPBLOCK and TPTIME were specified. If a transaction timeout occurred, any attempts to do new work will fail with [TPETIME] until the transaction has been aborted.

[TPEBLOCK]
: A blocking condition exists and TPNOBLOCK was specified.

[TPGOTSIG]
: A signal was received and TPNOSIGRSTRT was specified.

[TPEPROTO]
: TPSUBSCRIBE was called in an improper context.

[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(3), EVENTS(5), EVENT_MIB(5), Fboolco(3), Fboolev(3), Fvboolco(3), Fvboolev(3), recomp(3), TMSYSEVT(5), TMUSREVT(5), TPENQUEUE(), TPPOST(), TPSETUNSOL(), TPUNSUBSCRIBE(), 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(), TPBEGIN(), TPRECV(), TPRESUME().

TPSVCSTART(3CBL)

Name TPSVCSTART—start a BEA Tuxedo system 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 system 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 (i.e., 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 sub-type, 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 in 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.

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 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, `TPSVCSTART` fails and sets `TP-STATUS` to:

`[TPEINVAL]`

Invalid arguments were given.

`[TPEPROTO]`

`TPSVCSTART` was called in an improper context.

`[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

`TPSVRINIT`(), `TPSVRDONE`(), `buildserver`(1), `TPBEGIN`() `TPCONNECT`(), `TPCALL`(), `TPINIT`(), `TPOPEN`()

TPSVRDONE(3CBL)

Name `TPSVRDONE`—BEA Tuxedo system server termination routine

Synopsis

```
01 TPSTATUS-REC.
 COPY TPSTATUS.
PROCEDURE DIVISION.
* User code
EXIT PROGRAM.
```

Description The BEA Tuxedo system 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 system 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 rollback 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`(), `TPSVRINIT`()

TPSVRINIT(3CBL)

Name

TPSVRINIT—BEA Tuxedo system 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 system 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 system communication may be performed and transactions may be defined in this routine. However, if TPSVRINIT returns with open connections, asynchronous replies pending or while still in transaction mode, the BEA Tuxedo system will close the connections, ignore replies pending, abort the transaction, and the server will exit gracefully.

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 successfully started.

Application-specific options can be passed into a server and processed in TPSVRINIT (see servopts(5)). The options are passed through *ARGC* and *ARGV*. *ARGC* contain the number of arguments that have been passed and *ARGV* contains the arguments (in character format) separated by a single SPACE character. getopt(3C) 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 (and not take any service requests) by returning any value except [TPOK] in TP-STATUS.

Return Values

If either TPRETURN() or TPFORWAR() are used outside of a service routine (e.g., in clients, or in TPSVRINIT or TPSVRDONE()), then these routines return having no effect.

Usage

If either TPRETURN() or TPFORWAR() are called in TPSVRINIT, these routines simply return having no effect.

See Also

TPOPEN(), TPSVRDONE()

TPTERM(3CBL)

Name

`TPTERM`—leave a BEA Tuxedo system application

Synopsis

```
01 TPSTATUS-REC.
 COPY TPSTATUS.
CALL "TPTERM" USING TPSTATUS-REC.
```

Description

`TPTERM` removes a client from a BEA Tuxedo system application. If the client is in transaction mode, then the transaction is rolled back. When `TPTERM` returns successfully, the caller can no longer communicate with any other thread of control nor can it participate in any transactions. Any outstanding conversations are immediately disconnected.

If `TPTERM` is called more than once (that is, after the caller has already left the application), no action is taken and success is returned.

Return Values

Upon successful completion, `TPTERM` sets `TP-STATUS` to [`TPOK`].

Errors

Under the following conditions, `TPTERM` fails and sets `TP-STATUS` to:

[`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`()

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 shutdown.

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 `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`()

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 System Event Broker'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).

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 re-issued. 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 event broker'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 event broker 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 System Event Broker.

[TPETIME]
: A timeout occurred. If the caller is in transaction mode, then a transaction timeout occurred and the transaction is to be aborted; otherwise, a blocking timeout occurred and both TPBLOCK and TPTIME were specified. If a transaction timeout occurred, any attempts to do new work will fail with [TPETIME] until the transaction has been aborted.

[TPEBLOCK]
: A blocking condition exists and TPNOBLOCK was specified.

[TPGOTSIG]
: A signal was received and `TPNOSIGRSTRT` was specified.

[TPEPROTO]
: `TPUNSUBSCRIBE` was called in an improper context.

[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 `EVENTS`(5), `EVENT_MIB`(5), `TMSYSEVT`(5), `TMUSREVT`(5), `TPPOST`(), `TPSUBSCRIBE`()

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`(), `TXOPEN`(), `TXROLLBACK`(), `TXSETTIMEOUT`()

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`()

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(), TXSETCOMMITRET(), TXSETTRANCTL(), TXSETTIMEOUT()

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`(), `TXSETCOMMITRET`(), `TXSETTRANCTL`(), `TXSETTIMEOUT`()

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 re-open 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`().

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(), TXSETTRANCTL(), TXSETTIMEOUT()

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`.

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 `TXCOMMIT`(3), `TXOPEN`(3), `TXINFORM`(3), `TXGBEGIN`(3), `TXROLLBACK`(3)

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`.

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(), TXCOMMIT(), TXOPEN(), TXROLLBACK(), TXINFORM()

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 in an improper context. For example, the caller has not yet 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`(), `TXCOMMIT`(), `TXOPEN`(), `TXROLLBACK`(), `TXINFORM`()

USERLOG(3CBL)

Name USERLOG—write a message to the BEA Tuxedo system 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 system 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 userlog, 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.