BEA TUXEDO

Reference Manual
Section 5 - File Formats and Data Descriptions

BEA TUXEDO 6.5 for BEA WLE Version 4.2
Document Edition 6.5
July 1999

Copyright

Restricted Rights Legend

Trademarks or Service Marks

BEA TUXEDO Reference Manual

Document Edition	Date	Software Version
6.5	July 1999	BEA TUXEDO 6.5 for BEA WLE Version 4.2

Contents

Section 5 - File Formats and Data Descriptions

Section 5 - File Formats and Data Descriptions

intro(5)

Name `intro`-introduction to tables and files

Description This section describes the format of miscellaneous tables and files.

The page named `compilation`(5) summarizes information about header files, libraries and environment variables needed when compiling application source code.

The section includes descriptions of BEA TUXEDO system-supplied servers. Applications wishing to use the BEA TUXEDO system-supplied servers should specify them in the configuration file for the application.

The `servopts` page describes options that can be specified in the configuration file as the `CLOPT` parameter of application servers.

The BEA TUXEDO Management Information Base is documented in the `MIB`(5) reference page and in the following component MIB pages:

`ACL_MIB(5) APPQ_MIB(5) EVENT_MIB(5) TM_MIB(5) WS_MIB(5)`

ACL_MIB(5)

Name ACL_MIB-BEA TUXEDO ACL Management Information Base

Synopsis

```
#include <fml32.h>
#include <tpadm.h>
```

Description The BEA TUXEDO MIB defines the set of classes through which Access Control Lists (ACLs) may be managed. A BEA TUXEDO configuration with SECURITY set to USER_AUTH, ACL, or MANDATORY_ACL must be created before accessing or updating these classes. ACL_MIB(5) should be used in combination with the generic MIB reference page MIB(5) to format administrative requests and interpret administrative replies. Requests formatted as described in MIB(5) using classes and attributes described in this reference page may be used to request an administrative service using any one of a number of existing ATMI interfaces in an active application. ACL_MIB(5) consists of the following classes:

ACL_MIB Classes

Class Name	Attribute
T_ACLGROUP	ACL group
T_ACLPERM	ACL permissions
T_ACLPRINCIPAL	ACL principal (users or domains)

Each class description section has four subsections:

Overview
: High level description of the attributes associated with the class.

Attribute Table
: A table that lists the name, type, permissions, values and default for each attribute in the class. The format of the attribute table is described below.

Attribute Semantics
: Tells how each attribute should be interpreted.

Limitations
: Limitations in the access to and interpretation of this class.

Attribute Table Format

As described above, each class that is a part of this MIB is defined below in four parts. One of these parts is the attribute table. The attribute table is a reference guide to the attributes within a class and how they may used by administrators, operators and general users to interface with an application. There are five components to each attribute description in the attribute tables: name, type, permissions, values and default. Each of these components is discussed in MIB(5).

TA_FLAGS Values

MIB(5) defines the generic TA_FLAGS attribute which is a long containing both generic and component MIB specific flag values. At this time, there are no ACL_MIB(5) specific flag values defined.

FML32 Field Tables

The field tables for the attributes described in this reference page are found in the file udataobj/tpadm relative to the root directory of the BEA TUXEDO system software installed on the system. The directory ${TUXDIR}/udataobj should be included by the application in the colon separated list specified by the FLDTBLDIR environment variable and the field table name tpadm should be included in the comma separated list specified by the FIELDTBLS environment variable.

Limitations

Access to the header files and field tables for this MIB is being provided only on BEA TUXEDO 6.0 sites and later, both native and Workstation.

T_ACLGROUP CLASS DEFINITION

Overview

The T_ACLGROUP class represents groups of BEA TUXEDO application users and domains.

Attribute Table

ACL_MIB(5): T_ACLGROUP Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_GROUPNAME(r)(*)	string	rU-------	*string[1...30]*	N/A
TA_GROUPID(k)	long	rw-------	0 = *num* 16,384	lowest id
TA_STATE	string	rw-------	GET:"{INA}" SET:"{NEW|INV}"	N/A N/A

(k) - GET key field
(r) - Required field for object creation (SET TA_STATE NEW)
(*) - GET/SET key, one or more required for SET operations

Attribute Semantics

TA_GROUPNAME: *string[1...30]*

Logical name of the group. A group name is a string of printable characters and cannot contain a pound sign, comma, colon, or newline.

TA_GROUPID: 0 = *num* 16,384

Group identifier associated with this user. A value of 0 indicates the default group "other". If not specified at creation time, it defaults to the next available (unique) identifier greater than 0.

TA_STATE:

GET: {VALid}

A GET operation will retrieve configuration information for the selected T_ACLGROUP object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

VALid	T_ACLGROUP object is defined and inactive. Note that this is the only valid state for this class. ACL groups are never *active*.

SET: {NEW|INValid}

A SET operation will update configuration information for the selected T_ACLGROUP object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_ACLGROUP object for application. State change allowed only when in the INValid state. Successful return leaves the object in the VALid state.
unset	Modify an existing T_ACLGROUP object. This combination is not allowed in the INValid state. Successful return leaves the object state unchanged.
INValid	Delete T_ACLGROUP object for application. State change allowed only when in the VALid state. Successful return leaves the object in the INValid state.

Limitations

A user can be associated with exactly one ACL group. For someone to take on more than one role or be associated with more than one group, multiple user entries must be defined.

T_ACLPERM CLASS DEFINITION

Overview

The T_ACLPERM class indicates what groups are allowed to access BEA TUXEDO System entities. These entities are named via a string. The names currently represent service names, event names, and application queue names.

Attribute Table

ACL_MIB(5): T_ACLPERM Class Definition: Attribute Table

Attribute	Type	Permissions	Values	Default
TA_ACLNAME(r)(*)	string	rw-------	*string[1...30]*	N/A
TA_ACLTYPE(r)(*)	string	rw-------	ENQ|DEQ|SERVICE|	N/A
			POSTEVENT"	
TA_ACLGROUPIDS	string	rw-------	*string*	N/A
TA_STATE	string	rw-------	GET:"{INA}" SET:"{NEW|INV}"	N/A N/A

(r) - Required field for object creation (SET TA_STATE NEW)

(*) - GET/SET key, one or more required for SET operations

Attribute Semantics

TA_ACLNAME: *string*

The name of the entity for which permissions are being granted. The name can represent a service name, an event name, and/or a queue name. An ACL name is a string of printable characters and cannot contain a colon, pound sign, or newline.

TA_ACLTYPE: ENQ | DEQ | SERVICE | POSTEVENT

The type of the entity for which permissions are being granted.

TA_ACLGROUPIDS: *string*

A comma separated list of group identifiers (numbers) that are permitted access to the associated entity. The length of *string* is limited only by the amount of disk space on the machine.

TA_STATE:

GET: {VALid}

A GET operation will retrieve configuration information for the selected T_ACLPERM object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

VALid	T_ACLPERM object is defined and inactive. Note that this is the only valid state for this class. ACL permissions are never *active*.

SET: {NEW|INValid}

A SET operation will update configuration information for the selected T_ACLPERM object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_ACLPERM object for application. State change allowed only when in the INValid state. Successful return leaves the object in the VALid state.
unset	Modify an existing T_ACLPERM object. This combination is not allowed in the INValid state. Successful return leaves the object state unchanged.
INValid	Delete T_ACLPERM object for application. State change allowed only when in the VALid state. Successful return leaves the object in the INValid state.

Limitations Permissions are defined at the group level, not on individual user identifiers.

T_ACLPRINCIPAL CLASS DEFINITION

Overview

The T_ACLPRINCIPAL class represents users or domains that can access a BEA TUXEDO application and the group with which they are associated. To join the application as a specific user, it is necessary to present a user-specific password.

Attribute Table

ACL_MIB(5): T_ACLPRINCIPAL Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_PRINNAME(r)(*)	string	rU-------	*string[1...30]*	N/A
TA_PRINCLTNAME(k)	string	rw-------	*string[1...30]*	"*"
TA_PRINID(k)	long	rU-------	1 = *num* 131,072	lowest id
TA_PRINGRP(k)	long	rw-------	0 = *num* 16,384	0
TA_PRINPASSWD	string	rwx------	*string*	N/A
TA_STATE	string	rw-------	GET:"{INA}" SET:"{NEW|INV}"	N/A N/A

(k) - GET key field
(r) - Required field for object creation (SET TA_STATE NEW)
(*) - GET/SET key, one or more required for SET operations

Attribute Semantics

TA_PRINNAME: *string*

Logical name of the user or domain (a principal). A principal name is a string of printable characters and cannot contain a pound sign, colon, or newline.

TA_PRINCLTNAME: *string*

The client name associated with the user. It generally describes the role of the associated user, and provides a further qualifier on the user entry. If not specified at creation time, the default is the wildcard asterisk (*). A client name is a string of printable characters and cannot contain a colon, or newline.

TA_PRINID: 1 = *num* 131,072

Unique user identification number. If not specified at creation time, it defaults to the next available (unique) identifier greater than 0.

TA_PRINGRP: 0 = *num* 16,384

Group identifier associated with this user. A value of 0 indicates the default group "other". If not specified at creation time, the default 0 is assigned.

TA_PRINPASSWD: *string*

The clear-text authentication password for the associated user. Note that the system will automatically encrypt this information on behalf of the administrator.

TA_STATE :

GET: {VALid}

A GET operation will retrieve configuration information for the selected T_ACLPRINCIPAL object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

VALid	T_ACLPRINCIPAL object is defined and inactive. Note that this is the only valid state for this class. ACL principals are never *active*.

SET: {NEW|INValid}

A SET operation will update configuration information for the selected T_ACLPRINCIPAL object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_ACLPRINCIPAL object for application. State change allowed only when in the INValid state. Successful return leaves the object in the VALid state.
unset	Modify an existing T_ACLPRINCIPAL object. This combination is not allowed in the INValid state. Successful return leaves the object state unchanged.
INValid	Delete T_ACLPRINCIPAL object for application. State change allowed only when in the VALid state. Successful return leaves the object in the INValid state.

Limitations

A user or domain can be associated with exactly one ACL group. For someone to take on more than one role or be associated with more than one group, multiple principal entries must be defined.

Diagnostics

There are two general types of errors that may be returned to the user when interfacing with ACL_MIB(5). First, any of the three ATMI verbs (tpcall(3), tpgetrply(3) and tpdequeue(3)) used to retrieve responses to administrative requests may return any error defined for them. These errors should be interpreted as described on the appropriate reference pages.

If, however, the request is successfully routed to a system service capable of satisfying the request and that service determines that there is a problem handling the request, then failure may be returned in the form of an application level service failure. In these cases, tpcall(3) and tpgetrply(3) will return an error with tperrno set to TPESVCFAIL and return a reply message containing the original request along with TA_ERROR, TA_STATUS and TA_BADFLD fields further qualifying the error as described below. When a service failure occurs for a request forwarded to the system through the TMQFORWARD(5) server, the failure reply message will be enqueued to the failure queue identified on the original request (assuming the -d option was specified for TMQFORWARD).

When a service failure occurs during processing of an administrative request, the FML32 field TA_STATUS is set to a textual description of the failure, the FML32 field TA_ERROR is set to indicate the cause of the failure as indicated below. All error codes specified below are guaranteed to be negative.

The following diagnostic codes are returned in TA_ERROR to indicate successful completion of an administrative request. These codes are guaranteed to be non-negative.

[*other*]

Other return codes generic to any component MIB are specified in the MIB(5) reference page. These return codes are guaranteed to be mutually exclusive with any ACL_MIB(5) specific return codes defined here.

Interoperability

The header files and field tables defined in this reference page are available on BEA TUXEDO system release 6.0 and later. Fields defined in these headers and tables will not be changed from release to release. New fields may be added which are not defined on the older release site. Access to the /AdminAPI is available from any site with the header files and field tables necessary to build a request. The T_ACLPRINCIPAL, T_ACLGROUP, and T_ACLPERM classes are new with BEA TUXEDO system release 6.0.

Portability

The existing FML32 and ATMI functions necessary to support administrative interaction with BEA TUXEDO system MIBs, as well as the header file and field table defined in this reference page, are available on all supported native and workstation platforms.

Example

Following is a sequence of code fragments that adds a user to a group and adds permissions for that group to a service name.

Field Tables

The field table *tpadm* must be available in the environment to have access to attribute field identifiers. This can be done at the shell level as follows:

```
$ FIELDTBLS=tpadm
$ FLDTBLDIR=${TUXDIR}/udataobj
$ export FIELDTBLS FLDTBLDIR
```

Header Files

The following header files are included.

```
#include <atmi.h>
#include <fml32.h>
#include <tpadm.h>
```

Add User

The following code fragment adds a user to the default group "other."

```
/* Allocate input and output buffers */ ibuf = tpalloc("FML32",
NULL, 1000);
 obuf = tpalloc("FML32", NULL, 1000);

 /* Set MIB(5) attributes defining request type */
 Fchg32(ibuf, TA_OPERATION, 0, "SET", 0);
 Fchg32(ibuf, TA_CLASS, 0, "T_ACLPRINCIPAL", 0);

 /* Set ACL_MIB(5) attributes */
 Fchg32(ibuf, TA_PRINNAME, 0, ta_prinname, 0);
 Fchg32(ibuf, TA_PRINID, 0, (char *)ta_prinid, 0);
 Fchg32(ibuf, TA_STATE, 0, (char *)"NEW", 0);

 Fchg32(ibuf, TA_PRINPASSWD, 0, (char *)passwd, 0);


 /* Make the request */
 if (tpcall(".TMIB", (char *)ibuf, 0, (char **)obuf, olen, 0) 0) {
 fprintf(stderr, "tpcall failed: %s\en", tpstrerror(tperrno));
 if (tperrno == TPESVCFAIL) {
 Fget32(obuf, TA_ERROR, 0,(char *)ta_error, NULL);
 ta_status = Ffind32(obuf, TA_STATUS, 0, NULL);
 fprintf(stderr, "Failure: %ld, %s\en",
 ta_error, ta_status);
 }
 /* Additional error case processing */
 }
```

Files

`${TUXDIR}/include/tpadm.h`, `${TUXDIR}/udataobj/tpadm`,

See Also

`Fintro`(3), `Fadd32`(3), `Fchg32`(3), `Ffind32`(3), `tpalloc`(3), `tprealloc`(3), `tpcall`(3), `tpacall`(3), `tpgetrply`(3), `tpenqueue`(3), `tpdequeue`(3), `MIB`(5), `TM_MIB`(5), *BEA TUXEDO Administrator's Guide*, *BEA TUXEDO Programmer's Guide*

APPQ_MIB(5)

Name APPQ_MIB-BEA TUXEDO system /Q Management Information Base

Synopsis

```
#include <fml32.h>
#include <tpadm.h>
```

Description The /Q MIB defines classes through which application queues can be managed.

APPQ_MIB(5) should be used in combination with the generic MIB reference page MIB(5) to format administrative requests and interpret administrative replies. Requests formatted as described in MIB(5) using classes and attributes described on this reference page may be used to request an administrative service using any one of a number of existing ATMI interfaces in an active application. Application queues in an inactive application may also be administered using the tpadmcall(3) function interface.

APPQ_MIB(5) consists of the following classes:

APPQ_MIB Classes

Class Name	Attributes
T_APPQ	Application queues within a queue space
T_APPQMSG	Messages within an application queue
T_APPQSPACE	Application queue spaces
T_APPQTRANS	Transactions associated with application queues

Note that this MIB refers to application-defined reliable disk-based queues (that is, /Q queues), and not server queues (the T_QUEUE class of the TM_MIB(5) component).

Each class description section has four subsections:

Overview
: High level description of the attributes associated with the class.

Attribute Table
: A table that lists the name, type, permissions, values and default for each attribute in the class. The format of the attribute table is described below.

Attribute Semantics
: Tells how each attribute should be interpreted.

Limitations
: Limitations in the access to and interpretation of this class.

Attribute Table Format

Each class that is a part of this MIB is documented in four parts. One part is the attribute table. The attribute table is a reference guide to the attributes within a class and how they may used by administrators, operators, and general users to interface with an application.

There are five components to each attribute description in the attribute tables: name, type, permissions, values and default. Each of these components is discussed in MIB(5).

TA_FLAGS Values

MIB(5) defines the generic TA_FLAGS attribute which is a long containing both generic and component MIB-specific flag values. The following flag values are defined for the APPQ_MIB(5) component. These flag values should be or'd with any generic MIB flags.

QMIB_FORCECLOSE
: When setting the TA_STATE attribute of a T_APPQSPACE object to CLEaning, this flag indicates that the state change should succeed even if the state of the queue space is ACTive.

QMIB_FORCEDELETE
: When setting the TA_STATE attribute of a T_APPQSPACE object to INValid, this flag indicates that the state change should succeed even if the queue space is ACTive or if messages are present in any of its queues. Similarly, when setting the TA_STATE attribute of a T_APPQ object to INValid, this flag allows the queue to be deleted even if messages are present or processes are attached to the queue space.

QMIB_FORCEPURGE
: When setting the TA_STATE attribute of a T_APPQ object to INValid, this flag indicates that the state change should succeed even if messages are present on the queue. If, however, a message stored in the selected T_APPQ object is currently involved in a transaction, the state change will fail and an error will be written to the userlog.

FML32 Field Table

The field table for the attributes described on this reference page is found in the file udataobj/tpadm relative to the root directory of the BEA TUXEDO software installed on the system. The directory ${TUXDIR}/udataobj should be included by the application in the path list (semi-colon separated on Netware/NT and colon separated otherwise) specified by the FLDTBLDIR environment variable and the field table name tpadm should be included in the comma-separated list specified by the FIELDTBLS environment variable.

Limitations This MIB is provided only on BEA TUXEDO system 6.0 sites and later, both native and Workstation.

If a site running a BEA TUXEDO system release earlier than Release 6.0 is active in the application, then administrative access through this MIB is limited as follows.

- `SET` operations are not allowed.
- Local information access for sites earlier than Release 6.0 is not available.

T_APPQ Class Definition

Overview

The T_APPQ class represents application queues. One or more application queues may exist in a single application queue space.

Limitations

It is not possible to retrieve all instances of this class by leaving all key fields unset. Instead, sufficient key fields must be supplied to explicitly target a single application queue space. These required key fields are TA_APPQSPACENAME, TA_QMCONFIG, and TA_LMID, except when the application is unconfigured (i.e., the TUXCONFIG environment variable is not set), in which case TA_LMID must be omitted. For example, if the TA_APPQSPACENAME, TA_QMCONFIG, and TA_LMID attributes are set in a request using tpcall(3), then all T_APPQ objects within the specified queue space will be retrieved.

Attribute Table

APPQ_MIB(5): T_APPQ Class Definition Attribute Table

Attributea[a]	Type	Permissions	Values	Default
TA_APPQNAME(k)(r)(*)	string	ru-r--r--	*string*[1..15]	N/A
TA_APPQSPACENAME(k)(r)(*)	string	ru-r--r--	*string*[1..15]	N/A
TA_QMCONFIG(k)(r)(*)	string	ru-r--r--	*string*[1..78]	N/A
TA_LMID(k)(r)(*)b	string	ru-r--r--	*string*[1..30]	N/A
TA_STATE (Note 3)c	string	rw-r--r--	GET:{VAL} SET:{NEW|INV}	N/A N/A
TA_APPQORDERd	string	rw-r--r--	{PRIO|TIME|LIFO|FIFO}	FIFO
TA_CMD	string	rw-r--r--	*shell-command -string*[0..78]	""
TA_CMDHW	string	rw-r--r--	0 = *num*[Bbm%]	100%
TA_CMDLW	string	rw-r--r--	0 = *num*[Bbm%]	0%
TA_MAXRETRIES	long	rw-r--r--	0 = *num*	0
TA_OUTOFORDER	string	rw-r--r--	{NONE|TOP |MSGID}	NONE
TA_RETRYDELAY	long	rw-r--r--	0 = *num*	0
TA_CURBLOCKS	long	r--r--r--	0 = *num*	N/A
TA_CURMSG	long	r--r--r--	0 = *num*	N/A

APPQ_MIB(5): T_APPQ Class Definition Attribute Table

Attributea[a]	Type	Permissions	Values	Default
(k) - GET key field)[e] (r) - Required field for object creation (*) - Required SET key field				

[a]All attributes of class T_APPQ are local attributes.

[b]TA_LMID must be specified as a key field except when the application is unconfigured (i.e., the TUXCONFIG environment variable is not set).

[c]All operations on T_APPQ objects—both GET and SET—silently open the associated queue space (i.e., implicitly set the state of the queue space to OPEn if it is not already OPEn or ACTive). This may be a time-consuming operation if the queue space is large.

[d]TA_APPQORDER can not be modified after the application queue is created.

[e]Sufficient key fields must be supplied in a GET operation to explicitly target a single application queue space.

Attribute Semantics

TA_APPQNAME: *string*[1..15]
Name of the application queue.

TA_APPQSPACENAME: *string*[1..15]
Name of the application queue space containing the application queue.

TA_QMCONFIG: *string*[1..78]
Absolute pathname of the file or device where the application queue space is located.

TA_LMID: *string*[1..30] (no comma)
Identifier of the logical machine where the application queue space is located.

TA_STATE:

GET: {VALid}
A GET operation retrieves information about the selected application queues. The following list describes the meaning of the TA_STATE attribute returned in response to a GET request. States not listed will not be returned.

VALid	The specified queue exists. This state is INActive equivalent for purposes of permissions checking.

SET: {NEW|INValid}

A SET operation changes characteristics of the selected application queue or creates a new queue. The following list describes the meaning of the TA_STATE attribute returned by a SET request. States not listed can not be set.

NEW	Create a new queue in the specified queue space. The queue is left in state VALid following successful creation.
INValid	Delete the specified queue. The queue must be in state VALid to be deleted. If the queue space has processes attached to it (i.e., it is in the ACTive state), the queue will not be deleted unless the TA_FLAGS attribute includes the QMIB_FORCEDELETE flag. In addition, if the queue has messages in it, it will not be deleted unless QMIB_FORCEPURGE is specified. Successful return leaves the object in the INValid state.
unset	Modify an application queue. Successful return leaves the state unchanged.

TA_APPQORDER:

The order in which messages in the queue are to be processed. Legal values are PRIO or TIME, followed by a comma, optionally followed by another occurrence of PRIO or TIME, followed by one of the values LIFO or FIFO. If neither FIFO nor LIFO is specified, FIFO is assumed. If nothing is specified when a queue is created, the default is FIFO. For example, these are some legal settings:

```
PRIO
PRIO,TIME,LIFO
TIME,PRIO,FIFO
TIME,FIFO
```

TA_CMD:*shell-command-string*[0..78]
: The command to be automatically executed when the high water mark, TA_CMDHW, is reached. The command will be re-executed when the high water mark is reached again after the low water mark, TA_CMDLW, has been reached.

TA_CMDHW: 0 = *num*[Bbm%]

TA_CMDLW: 0 = *num*[Bbm%]
: The high and low water marks that control the automatic execution of the command specified in the TA_CMD attribute. Each is an integer greater than or equal to zero optionally followed by one of the following keyletters. The keyletters must be consistent for TA_CMDHW and TA_CMDLW.

  b
  : The high and low water marks pertain to the number of bytes used by messages in the queue.

  B
  : The high and low water marks pertain to the number of blocks used by messages in the queue.

  m
  : The high and low water marks pertain to the number of messages in the queue.

  %
  : The high and low water marks are expressed in terms of a percentage of queue capacity.

  For example, if TA_CMDLW is 50m and TA_CMDHW is 100m, then the command specified in TA_CMD will be executed when 100 messages are on the queue, and it will not be executed again until the queue has been drained below 50 messages and has filled again to 100 messages.

TA_CURBLOCKS: 0 = *num*
: The number of disk pages currently consumed by the queue.

TA_CURMSG: 0 = *num*
: The number of messages currently in the queue.

TA_MAXRETRIES: 0 = *num*
: The maximum number of retries for a failed queue message. When the number of retries is exhausted, the message is placed on the error queue of the associated application queue space. If there is no error queue, the message is dropped. The default is zero.

`TA_OUTOFORDER`: {`MSGID`|`TOP`|`NONE`}

The way in which out-of-order message processing is to be handled. The default is `NONE`.

`TA_RETRYDELAY`: 0 = *num*

The delay, in seconds, between retries for a failed queue message. The default is zero.

T_APPQMSG Class Definition

Overview

The T_APPQMSG class represents messages stored in application queues. A message is not created by an administrator; instead, it comes into existence as a result of a call to tpenqueue(3). A message can be destroyed either by a call to tpdequeue(3) or by an administrator. In addition, certain attributes of a message can be modified by an administrator. For example, an administrator can move a message from one queue to another queue within the same queue space or change its priority.

Limitations

It is not possible to retrieve all instances of this class by leaving all key fields unset. Instead, sufficient key fields must be supplied to explicitly target a single application queue space. These required key fields are TA_APPQSPACENAME, TA_QMCONFIG, and TA_LMID, except when the application is unconfigured (i.e., the TUXCONFIG environment variable is not set), in which case TA_LMID must be omitted. For example, if the TA_APPQSPACENAME, TA_QMCONFIG, and TA_LMID attributes are set in a request using tpcall(3), then all T_APPQMSG objects in all queues of the specified queue space will be retrieved.

Attribute Table

APPQ_MIB(5): T_APPQMSG Class Definition Attribute Table

Attribute[a]	Type	Permissions	Values	Default
TA_APPQMSGID(k)(*)	string	r--r--r--	*string*[1..32]	N/A
TA_APPQNAME(k)(*)	string	r--r--r--	*string*[1..15]	N/A
TA_APPQSPACENAME(k)(*)	string	r--r--r--	*string*[1..15]	N/A
TA_QMCONFIG(k)(*)	string	r--r--r--	*string*[1..78]	N/A
TA_LMID(k)(*)[b]	string	r--r--r--	*string*[1..30]	N/A
TA_STATE[c]	string	rw-r--r--	GET:{VAL}	N/A
			SET:{INV}	N/A
TA_NEWAPPQNAME	string	-w--w----	*string*[1..15]	N/A
TA_PRIORITY	long	rw-rw-r--	{ 1 = *num* = 100 | -1 }	N/A
TA_TIME	string	rw-rw-r--	{ *YY*[*MM*[*DD*[*hh*[*mm*[*ss*]]]]] |+*seconds* }	N/A

APPQ_MIB(5): T_APPQMSG Class Definition Attribute Table

Attribute[a]	Type	Permissions	Values	Default
TA_CORRID(k)	long	r--r--r--	*string*[0..32]	N/A
TA_LOWPRIORITY(k)	long	k--k--k--	1 = *num* = 100	1
TA_HIGHPRIORITY(k)	long	k--k--k--	1 = *num* = 100	100
TA_MSGENDTIME(k)	string	k--k--k--	{ *YY*[*MM*[*DD*[*hh*[*mm*[*ss*]]]]] |+*seconds* }	MAXLONG
TA_MSGSTARTTIME(k)	string	k--k--k--	{ *YY*[*MM*[*DD*[*hh*[*mm*[*ss*]]]]] |+*seconds* }	0
TA_CURRETRIES	long	r--r--r--	0 = *num*	N/A
TA_MSGSIZE	long	r--r--r--	0 = *num*	N/A
(k) - GET key field[d] (*) - Required SET key field				

[a] All attributes of class T_APPQMSG are local attributes.

[b] TA_LMID must be specified as a key field except when the application is unconfigured (i.e., the TUXCONFIG environment variable is not set).

[c] All operations on T_APPQMSG objects—both GET and SET—silently open the associated queue space (i.e., implicitly set the state of the queue space to OPEn if it is not already OPEn or ACTive). This may be a time-consuming operation if the queue space is large.

[d] Sufficient key fields must be supplied in a GET operation to explicitly target a single application queue space.

Attribute Semantics

TA_APPQMSGID: *string*[1..32]
: A unique identifier for the queue message, which can be used to select the message for GET or SET operations. No significance should be placed on this value beyond using it for equality comparisons.

TA_APPQNAME: *string*[1..15]
: Name of the application queue in which the message is stored.

TA_APPQSPACENAME: *string*[1..15]
: Name of the application queue space containing the message.

TA_QMCONFIG: *string*[1..78]
: Absolute pathname of the file or device where the application queue space is located.

TA_LMID: *string*[1..30] (no comma)
: Identifier of the logical machine where the application queue space is located.

TA_STATE:

GET: {VALid}
: A GET operation retrieves information about the selected messages. The following list describes the meaning of the TA_STATE attribute returned in response to a GET request. States not listed will not be returned.

VALid	The message exists. This state is INActive equivalent for purposes of permissions checking.

SET: {INValid}
: A SET operation changes characteristics of the selected message. The following list describes the meaning of the TA_STATE attribute returned by a SET request. States not listed can not be set.

INValid	The message is deleted from its queue space. The message must be in state VALid before attempting this operation. Successful return leaves the object in the INValid state.
unset	Modify a message. Successful return leaves the state unchanged.

TA_CURRETRIES: 0 = *num*
: The number of retries that have been attempted so far on this message.

TA_CORRID: *string*[0..32]
: The correlation identifier for this message provided by the application in the tpenqueue(3) request. The empty string indicates that a correlation identifier is not present.

TA_LOWPRIORITY: 1 = *num* = 100
TA_HIGHPRIORITY: 1 = *num* = 100

The lowest and highest priority within which to search for occurrences of T_APPQMSG objects. These attributes may only be used as key fields with GET operations and are valid only for PRIO-based queues.

TA_MSGSTARTTIME:
TA_MSGENDTIME:

The start and end time within which to search for occurrences of T_APPQMSG objects. The range is inclusive. See TA_TIME for the format. These attributes may only be used as key fields with GET operations and are valid only for TIME-based queues.

TA_NEWAPPQNAME: *string*[1..15]

Name of the queue into which to move the selected message. This queue must be an existing queue in the same queue space. The message must be in state VALid for this operation to succeed. This attribute is not returned by a GET operation.

TA_PRIORITY: 1 = *num* = 100

The priority of the message. This attribute is valid only for PRIO-based queues. The value -1 is returned by a GET operation if the queue is not PRIO-based.

TA_TIME:

The time when the message will be processed. This attribute is valid only for TIME-based queues. The empty string is returned by a GET operation if the queue is not TIME-based. The format is one of the following:

+*seconds*

Specifies that the message will be processed *seconds* in the future. The value zero specifies that the message should be processed immediately.

YY[*MM*[*DD*[*hh*[*mm*[*ss*]]]]]

Specifies the year, month, day, hour, minute, and second when the message should be processed. Omitted units default to their minimum possible values. For example, 9506 is equivalent to 950601000000. The years 00 through 37 are treated as 2000 through 20037, 70 through 99 are treated as 1970 through 1999, and 38 through 69 are invalid.

`TA_MSGSIZE`: 0 = *num*

The size of the message, in bytes.

T_APPQSPACE Class Definition

Overview

The T_APPQSPACE class represents application queue spaces. An application queue space is an area in a BEA TUXEDO system device; see the T_DEVICE class in TM_MIB(5) for more information about devices and their attributes. Each queue space typically contains one or more application queues, and each queue may have messages stored in it.

A queue space is uniquely identified by several attributes: its name (TA_APPQSPACENAME attribute), the device that contains it (TA_QMCONFIG attribute), and the logical machine where the device is located (TA_LMID attribute).

A queue space is typically associated with exactly one server group in a configured application. The queue space name as well as the device name are components of the TA_OPENINFO attribute of the T_GROUP object.

Limitations

It is not possible to retrieve all instances of this class by leaving all key fields unset. Instead, all three key fields must be supplied to explicitly target a single application queue space. The single exception occurs when accessing a local queue space via tpadmcall(3) in the context of an unconfigured application (i.e., the TUXCONFIG environment variable is not set). In this case the TA_LMID key field must be omitted.

The above limitation regarding accessibility of queue spaces also applies to T_APPQ, T_APPQMSG, and T_APPQTRANS objects because operations on all objects in the /Q MIB implicitly involve queue spaces.

Attribute Table

APPQ_MIB(5): T_APPQSPACE Class Definition Attribute Table

Attribute[a]	Type	Permissions	Values	Default
TA_APPQSPACENAME(k)(r)(*)	string	ru-r--r--	*string*[1..15]	N/A
TA_QMCONFIG(k)(r)(*)	string	ru-r--r--	*string*[1..78]	N/A
TA_LMID(k)(r)(*))[b]	string	ru-r--r--	*string*[1..30]	N/A
TA_STATE(k)[c]	string	rwxrwxr--	GET:{INA|INI |OPE|ACT} SET:{NEW|OPE |CLE|INV}	N/A N/A

APPQ_MIB(5): T_APPQSPACE Class Definition Attribute Table

Attribute[a]	Type	Permissions	Values	Default
TA_BLOCKING	long	rw-r--r--	0 = *num*	16
TA_ERRORQNAME	string	rw-r--r--	*string*[0..15]	""
TA_FORCEINIT	string	rw-r--r--	{Y|N}	N
TA_IPCKEY(r)	long	rw-r--r--	32769 = *num* = 262143	N/A
TA_MAXMSG(r)	long	rw-r--r--	0 = *num*	N/A
TA_MAXPAGES(r)	long	rw-r--r--	0 = *num*	N/A
TA_MAXPROC(r)	long	rw-r--r--	0 = *num*	N/A
TA_MAXQUEUES(r)[d]	long	rw-r--r--	0 = *num*	N/A
TA_MAXTRANS(r)	long	rw-r--r--	0 = *num*	N/A
TA_CUREXTENT	long	r--r--r--	0 = *num* = 100	N/A
TA_CURMSG	long	r--r--r--	{ 0 = *num* | -1 }	N/A
TA_CURPROC	long	r--r--r--	0 = *num*	N/A
TA_CURQUEUES	long	r--r--r--	{ 0 = *num* | -1 }	N/A
TA_CURTRANS	long	R--R--R--	0 = *num*	N/A
TA_HWMSG	long	R--R--R--	0 = *num*	N/A
TA_HWPROC	long	R--R--R--	0 = *num*	N/A
TA_HWQUEUES	long	R--R--R--	0 = *num*	N/A
TA_HWTRANS	long	R--R--R--	0 = *num*	N/A
TA_PERCENTINIT	long	r--r--r--	0 = *num* = 100	N/A

(k) - GET key field
(r) - Required field for object creation
(*) - Required SET key field

a. All attributes of class T_APPQSPACE are local attributes.

b. TA_LMID must be specified as a key field except when the application is unconfigured (i.e., the TUXCONFIG environment variable is not set).

c. All operations on T_APPQ, T_APPQMSG, and T_APPQTRANS objects (both GET and SET) silently open the associated queue space (i.e., implicitly set the state of the queue space to OPEn if it is not already OPEn or ACTive). This may be a time-consuming operation if the queue space is large.

d. TA_MAXQUEUES can not be modified after the queue space is created.

Attribute Semantics

TA_APPQSPACENAME: string[1..15]
Name of the application queue space.

TA_QMCONFIG: string[1..78]
Absolute pathname of the file or device where the application queue space is located.

TA_LMID: *string*[1..30] (no comma)
Identifier of the logical machine where the application queue space is located.

TA_STATE:

GET: {INActive|INItializing|OPEn|ACTive}
A GET operation retrieves information about the selected application queue space. The following list describes the meaning of the TA_STATE attribute returned in response to a GET request. States not listed will not be returned.

INActive	The queue space exists; i.e., disk space for it has been reserved in a device and the space has been initialized (if requested or if necessary).
INItializing	Disk space for the queue space is currently being initialized. This state is ACTive equivalent for purposes of permissions checking.
OPEn	Shared memory and other IPC resources for the queue space have been allocated and initialized, but no processes are currently attached to the shared memory. This state is INActive equivalent for purposes of permissions checking.
ACTive	Shared memory and other IPC resources for the queue space have been allocated and initialized, and at least one process is currently attached to the shared memory. These processes can be the queue servers (TMS_QM, TMQUEUE, and perhaps TMQFORWARD) associated with the queue space, or they can be administrative processes such as qmadmin(1), or they can be processes associated with another application.

SET: {NEW|OPEn|CLEaning|INValid}

A SET operation changes the selected application queue space or creates a new one. The following list describes the meaning of the TA_STATE attribute returned by a SET request. States not listed can not be set.

NEW	Create a new queue space. The state of the queue space becomes either INItializing or INActive following a successful SET to this state.
OPEn	Allocate and initialize shared memory and other IPC resources for the queue space. This is allowed only if the queue space is in the INActive state.
CLEaning	Remove the shared memory and other IPC resources for the queue space. This is allowed only when the queue space is in the OPEn or ACTive state. The QMIB_FORCECLOSE flag must be specified if the state is ACTive. Successful return leaves the object in the INActive state.
INValid	Delete the queue space. Unless the QMIB_FORCEDELETE flag is passed, an error is reported if the state is ACTive or if messages exist on any queues in the queue space. Successful return leaves the object in the INValid state.
unset	Modify an application queue space. Successful return leaves the state unchanged.

TA_BLOCKING: 0 = *num*

The blocking factor used for disk space management of the queue space. The default when a new queue space is created is 16.

TA_CUREXTENT: 0 = *num* = 100

The current number of extents used by the queue space. The largest number allowed is 100. Each time the TA_MAXPAGES attribute is increased, a new extent is allocated.

TA_CURMSG: 0 = *num*

The current number of messages in the queue space. This number can be determined only if the queue space is OPEn or ACTive, or if the queue space is newly created. If none of the conditions apply, the value -1 is returned.

TA_CURPROC: 0 = *num*

The current number of processes accessing the queue space.

TA_CURQUEUES: 0 = *num*

The current number of queues existing in the queue space. This number can be determined only if the queue space is OPEn or ACTive, or if the queue space is newly created. If none of these conditions apply, the value -1 is returned.

TA_CURTRANS: 0 = *num*

The current number of outstanding transactions involving the queue space.

TA_ERRORQNAME: *string*[0..15]

Name of the error queue associated with the queue space. If there is no error queue, an empty string is returned by a GET request.

TA_FORCEINIT: {Y|N}

Whether or not to initialize disk pages on new extents for the queue space. The default is not to initialize. Depending on the device type (e.g., regular file or raw slice), initialization can occur even if not requested.

TA_HWMSG: 0 = *num*

The highest number of messages in the queue space since the queue space was last opened. The number is reset to 0 when the queue space state is set to CLEaning.

TA_HWPROC: 0 = *num*

The highest number of processes simultaneously attached to the queue space since the queue space was last opened. The number is reset to 0 when the queue space state is set to CLEaning.

TA_HWQUEUES: 0 = *num*

The highest number of queues existing in the queue space since the queue space was last opened. The number is reset to 0 when the queue space state is set to CLEaning.

TA_HWTRANS: 0 = *num*

The highest number of outstanding transactions involving the queue space since the queue space was last opened. If the queue space is accessed by more than one application, this number reflects all applications \(em not just the application represented by the TUXCONFIG environment variable. The number is reset to 0 when the queue space state is set to CLEaning.

TA_IPCKEY: 32769 = *num* = 262143

The IPC key used to access queue space shared memory.

TA_MAXMSG: 0 = *num*

The maximum number of messages that the queue space can contain.

TA_MAXPAGES: 0 = *num*

The maximum number of disk pages for all queues in the queue space. Each time the TA_MAXPAGES attribute is increased, a new extent is allocated (see TA_CUREXTENT). It is not possible to decrease the number of pages by setting this attribute to a lower number; an error is reported in this case.

TA_MAXPROC: 0 = *num*

The maximum number of processes that can attach to the queue space.

TA_MAXQUEUES: 0 = *num*

The maximum number of queues that the queue space can contain.

TA_MAXTRANS: 0 = *num*

The maximum number of simultaneously active transactions allowed by the queue space.

TA_PERCENTINIT: 0 = *num* = 100

The percentage (as an integer between 0 and 100 inclusive) of disk space that has been initialized for the queue space.

T_APPQTRANS Class Definition

Overview

The T_APPQTRANS class represents run-time attributes of transactions associated with application queues.

Limitations

It is not possible to retrieve all instances of this class by leaving all key fields unset. Instead, sufficient key fields must be specified to explicitly target a single application queue space. For example, if all key fields except TA_XID are set in an request using tpcall(3), then all T_APPQTRANS objects associated with the specified queue space will be retrieved.

It is important to keep in mind that transactions represented by objects of this class are not necessarily associated with the application in which they are retrieved. Care must be taken when heuristically committing or aborting a transaction because the transaction may actually belong to or have an effect on another application. The value of the TA_XID attribute is not guaranteed to be unique across applications.

Attribute Table

APPQ_MIB(5): T_APPQTRANS Class Definition Attribute Table

Attribute[a]	Type	Permissions	Values	Default
TA_XID(k)(*)	string	R--R--R--	*string*[1..78]	N/A
TA_APPQSPACENAME(k)(*)	string	r--r--r--	*string*[1..15]	N/A
TA_QMCONFIG(k)(*)	string	r--r--r--	*string*[1..78]	N/A
TA_LMID(k)(*)	string	r--r--r--	*string*[1..30]	N/A
TA_STATE[b]	string	R-XR-XR--	GET:{ACT|ABY|ABD|COM|REA|DEC|HAB|HCO}	N/A
			SET:{HAB|HCO}	N/A

(k) - GET key field[c]
(*) - Required SET key field

a. All attributes of class T_APPQTRANS are local attributes.

b. All operations on T_APPQTRANS objects \(em both GET and SET \(em silently open the associated queue space (i.e., implicitly set the state of the queue space to OPEn if it is not already OPEn or ACTive). This may be a time-consuming operation if the queue space is large.

c. Sufficient key fields must be supplied in a GET operation to explicitly target a single application queue space.

Attribute Semantics

TA_XID: string[1..78]
Transaction identifier as returned by tx_info(3) and mapped to a string representation. The data in this field should not be interpreted directly by the user except for equality comparison.

TA_APPQSPACENAME: *string*[1..15]
Name of the application queue space associated with the transaction.

TA_QMCONFIG: *string*[1..78]
Absolute pathname of the file or device where the application queue space is located.

TA_LMID: *string*[1..30] (no comma)
Identifier of the logical machine where the application queue space is located.

TA_STATE:

GET:
{ACTive|ABortonlY|ABorteD|COMcalled|REAdy|DECided|HAbord|HCommit}
A GET operation retrieves run-time information about the selected transactions. The following list describes the meaning of the TA_STATE attribute returned in response to a GET request. States not listed will not be returned. All states are ACTive equivalent for purposes of permissions checking.

ACTive	The transaction is active.
ABortonlY	The transaction has been identified for rollback.
ABorteD	The transaction has been identified for rollback and rollback has been initiated.
COMcalled	The initiator of the transaction has called tpcommit(3) and the first phase of 2-phase commit has begun.
REAdy	All of the participating groups on the retrieval site have successfully completed the first phase of two-phase commit and are ready to be committed.

DECided	The second phase of the 2-phase commit has begun.
SUSpended	The initiator of the transaction has suspended processing on the transaction.

SET: {HABort|HCOmmit}

A SET operation updates the state of the selected transactions. The following list describes the meaning of the TA_STATE attribute returned by a SET request. States not listed can not be set.

HABort	Heuristically abort the transaction. Successful return leaves the object in the HABort state.
HCOmmit	Heuristically commit the transaction. Successful return leaves the object in the HCOmmit state.

Portability The existing FML32 and ATMI functions necessary to support administrative interaction with BEA TUXEDO system MIBs, as well as the header file and field table mentioned on this reference page, are available on all supported native and workstation platforms.

Interoperability This MIB is provided only on BEA TUXEDO sites and later, both native and Workstation.

If a site running a BEA TUXEDO release earlier than Release 6.0 is active in the application, then administrative access through this MIB is limited as follows.

- SET operations are not allowed.
- Local information access for sites earlier than Release 6.0 is not available. If the class being accessed also has global information, then the global information only is returned. Otherwise, an error is returned.

If sites of differing releases, both greater than or equal to Release 6.0, are interoperating, then information on the older site is available for access and update as defined on the MIB reference page for that release and may be a subset of the information available in the later release.

Examples Following is a set of code fragments that illustrate how to perform various operations on application queue spaces, queues, messages, and transactions.

Each fragment should be preceded by code that allocates an FML32 typed buffer, such as the following:

```
rqbuf = tpalloc("FML32", NULL, 0);
```

After the buffer is populated, each fragment should be followed by code that sends the request and receives the reply, such as the following:

```
flags = TPNOTRAN | TPNOCHANGE | TPSIGRSTRT;
rval = tpcall(".TMIB", rqbuf, 0, rpbuf, rplen, flags);
```

See MIB(5) for additional information.

Field Tables

The field table tpadm must be available in the environment to allow access to attribute field identifiers. This can be done at the shell level as follows:

```
$ FIELDTBLS=tpadm
$ FLDTBLDIR=${TUXDIR}/udataobj
$ export FIELDTBLS FLDTBLDIR
```

Header Files

The following header files are needed.

```
#include <atmi.h>
#include <fml32.h>
#include <tpadm.h>
```

Create an Application Queue Space

Creating an application queue space typically involves two operations: the first to create the BEA TUXEDO system device in which the queue space will be allocated, and the second to create the queue space itself.

```
/* Allocate the buffer; see above */

/* Build the request to create a new device on SITE1 */
Fchg32(rqbuf, TA_OPERATION, 0, "SET", 0);
Fchg32(rqbuf, TA_CLASS, 0, "T_DEVICE", 0);
Fchg32(rqbuf, TA_STATE, 0, "NEW", 0);
Fchg32(rqbuf, TA_CFGDEVICE, 0, "/dev/q/dsk001", 0);
Fchg32(rqbuf, TA_LMID, 0, "SITE1", 0);
size = 500;
Fchg32(rqbuf, TA_DEVSIZE, 0, (char *)size, 0);

/* Make the request; see above */

/* Reinitialize the same buffer for reuse */
Finit32(rqbuf, (FLDLEN) Fsizeof32(rqbuf));

/* Build the request to create the queue space */
Fchg32(rqbuf, TA_OPERATION, 0, "SET", 0);
Fchg32(rqbuf, TA_CLASS, 0, "T_APPQSPACE", 0);
```

```
Fchg32(rqbuf, TA_STATE, 0, "NEW", 0);
Fchg32(rqbuf, TA_APPQSPACENAME, 0, "QSPACE1", 0);
Fchg32(rqbuf, TA_QMCONFIG, 0, "/dev/q/dsk001", 0);
Fchg32(rqbuf, TA_LMID, 0, "SITE1", 0);
Fchg32(rqbuf, TA_ERRORQNAME, 0, "errque", 0);
ipckey = 123456;
Fchg32(rqbuf, TA_IPCKEY, 0, (char *)ipckey, 0);
maxmsg = 100;
Fchg32(rqbuf, TA_MAXMSG, 0, (char *)maxmsg, 0);
maxpages = 200;
Fchg32(rqbuf, TA_MAXPAGES, 0, (char *)maxpages, 0);
maxproc = 50;
Fchg32(rqbuf, TA_MAXPROC, 0, (char *)maxproc, 0);
maxqueues = 10;
Fchg32(rqbuf, TA_MAXQUEUES, 0, (char *)maxqueues, 0);
maxtrans = 100;
Fchg32(rqbuf, TA_MAXTRANS, 0, (char *)maxtrans, 0);

/* Make the request; see above */
```

Add a Queue to an Application Queue Space

The following code creates a new queue in the queue space created in the previous example.

```
/* Build the request */
Fchg32(rqbuf, TA_OPERATION, 0, "SET", 0);
Fchg32(rqbuf, TA_CLASS, 0, "T_APPQ", 0);
Fchg32(rqbuf, TA_STATE, 0, "NEW", 0);
Fchg32(rqbuf, TA_APPQNAME, 0, "errque", 0);
Fchg32(rqbuf, TA_APPQSPACENAME, 0, "QSPACE1", 0);
Fchg32(rqbuf, TA_QMCONFIG, 0, "/dev/q/dsk001", 0);
Fchg32(rqbuf, TA_LMID, 0, "SITE1", 0);
Fchg32(rqbuf, TA_APPQORDER, 0, "PRIO", 0);

/* Make the request; see above */
```

List Application Queue Spaces Known to the Application

To list the application queue spaces known to an application, a two-level search is used. First, the groups using the /Q transaction manager `TMS_QM` are retrieved from the application configuration, and then the queue space referenced by each group is retrieved. The following code fragment assumes that each `GROUP` entry involving a queue space has a single logical machine associated with it (i.e., server migration is not used).

Listing 0-1 List Application Queue Spaces Known to the Application

```
/* Build the request to retrieve all TMS_QM groups */
Fchg32(rqbuf, TA_OPERATION, 0, "GET", 0);
```

```
Fchg32(rqbuf, TA_CLASS, 0, "T_GROUP", 0);
Fchg32(rqbuf, TA_TMSNAME, 0, "TMS_QM", 0);
fldid1 = TA_OPENINFO;
fldid2 = TA_LMID;
Fchg32(rqbuf, TA_FILTER, 0, (char *)fldid1, 0);
Fchg32(rqbuf, TA_FILTER, 0, (char *)fldid2, 1);

/* Make the request, assuming we are joined to the application */
rval = tpcall(".TMIB", rqbuf, 0, rpbuf, rplen, flags);

/* For each TMS_QM group, build the request to retrieve its queue space */
rval = Fget32(*rpbuf, TA_OCCURS, 0, (char *)occurs, NULL);
for (i = 0; i occurs; i++) {


  /* Reinitialize the buffer and set all common attributes */
  Finit32(rqbuf, (FLDLEN) Fsizeof32(rqbuf));
  Fchg32(rqbuf, TA_OPERATION, 0, "GET", 0);
  Fchg32(rqbuf, TA_CLASS, 0, "T_APPQSPACE", 0);

  /* Get the OPENINFO to determine device and queue space name */
  /* OPENINFO has the format <resource-mgr>:<qmconfig>:<appqspacename> */
  /* or on NT/Netware    <resource-mgr>:<qmconfig>;<appqspacename> */
  rval = Fget32(rpbuf, TA_OPENINFO, i, openinfo, NULL);

  /* The device is the 2nd field in OPENINFO */
  qmconfig = strchr(openinfo, ':') + 1;
  /* The queue space name is the 3rd field in OPENINFO */

#if defined(_TMDOWN) || defined(_TM_NETWARE)
#define pathsep ";" /* separtor for PATH */
#else
#define pathsep ":" /* separtor for PATH */
#endif
  appqspacename = strchr(qmconfig, pathsep);
  appqspacename[0] = '\e0'; /* null-terminate qmconfig */
  appqspacename++; /* bump past the null */

  /* Set the APPQSPACENAME and QMCONFIG keys */
  Fchg32(rqbuf, TA_APPQSPACENAME, 0, appqspacename, 0);
  Fchg32(rqbuf, TA_QMCONFIG, 0, qmconfig, 0);

  /* Get the LMID (assume no migration for this group) */
  rval = Fget32(rpbuf, TA_LMID, i, lmid, NULL);
  Fchg32(rqbuf, TA_LMID, 0, lmid, 0);

  /* Make the request */
  rval = tpcall(".TMIB", rqbuf, 0, rpbuf2, rplen2, flags);
}
```

The above technique does not find any queue space that has been created but does not yet have a corresponding GROUP entry in the application configuration. Such queue spaces must be retrieved by knowing *a priori* the key fields (i.e., TA_APPQSPACENAME, TA_QMCONFIG, and TA_LMID) for the queue space.

List Messages in an Application Queue

The following code retrieves all messages in the queue STRING in the queue space QSPACE1 in device /dev/q/dsk001 on logical machine SITE1.

```
/* Build the request */ Fchg32(rqbuf, TA_OPERATION, 0, "GET", 0);
Fchg32(rqbuf, TA_CLASS, 0, "T_APPQMSG", 0);
Fchg32(rqbuf, TA_APPQNAME, 0, "STRING", 0);
Fchg32(rqbuf, TA_APPQSPACENAME, 0, "QSPACE1", 0);
Fchg32(rqbuf, TA_QMCONFIG, 0, "/dev/q/dsk001", 0);
Fchg32(rqbuf, TA_LMID, 0, "SITE1", 0);
/* Make the request; see above */
```

List Transactions Involving a Queue Space

The following fragment retrieves all transactions involving (any queue in) the queue space QSPACE1.

```
/* Build the request */ Fchg32(rqbuf, TA_OPERATION, 0, "GET", 0);
Fchg32(rqbuf, TA_CLASS, 0, "T_APPQTRANS", 0);
Fchg32(rqbuf, TA_APPQSPACENAME, 0, "QSPACE1", 0);
Fchg32(rqbuf, TA_QMCONFIG, 0, "/dev/q/dsk001", 0);
Fchg32(rqbuf, TA_LMID, 0, "SITE1", 0);
/* Make the request; see above */
```

Files

```
${TUXDIR}/include/tpadm.h
${TUXDIR}/udataobj/tpadm
```

See Also

Fintro(3), Fadd32(3), Fchg32(3), Ffind32(3), tpalloc(3), tprealloc(3), tpcall(3), tpacall(3), tpgetrply(3), tpadmcall(3), tpenqueue(3), tpdequeue(3), MIB(5), TM_MIB(5), *BEA TUXEDO Administrator's Guide*, *TUXEDO Programmer's Guide*

AUTHSVR (5)

Name AUTHSVR-BEA TUXEDO server providing per\-user authentication

Synopsis `AUTHSVR SRVGRP="identifier" SRVID=number other_parms CLOPT="\-A"`

Description AUTHSVR is a BEA TUXEDO provided server that offers the authentication service. This server may be used in a secure application to provide per-user authentication when clients join the application. This server accepts service requests containing TPINIT typed buffers for client processes requesting access to the application. It uses the data field of the TPINIT typed buffer as a user password and validates it against the configured password. If the request passes validation, then an application key is returned with a successful return as the ticket to be used by the client.

SECURITY USER_AUTH

If SECURITY is set to USER_AUTH, then per-user authentication is enforced. The name of the authentication service can be configured for the application. If not specified, it defaults to AUTHSVC, which is the default service advertised by AUTHSVR.

By default, the file $APPDIR/tpusr is searched for password information; /etc/passwd is used if this file does not exist (although this file cannot be used correctly on systems that have a shadow password file). The file can be overridden by specifying the file name using a "-f *filename*" option in the server command line options (for example, CLOPT="-A -- -f /usr/tuxedo/users"). Note that automatic propagation of the user file from the master machine to other machines in the configuration is done only if $APPDIR/tpusr is used.

The user file is searched for a matching user name and client name. There are four types of entries in the user file. They are listed below in order of matching precedence when validating a user against the file.

1. Exact username/exact clientname
2. Wildcard username (*)/exact clientname
3. Exact username/wildcard clientname (*)
4. Wildcard username (*)/wildcard clientname (*)

An authentication request is authenticated against only the first matching password file entry. These semantics allow for a single user to have multiple entries (usually with different client names) and the user name may be a wild-card. These semantics are allowed if the user file is maintained using tpaddusr(1), tpdelusr(1), and tpmodusr(1). Note that use of these semantics is not compatible with the semantics for ACL and MANDATORY_ACL and will make migration to these security levels difficult. To get the restricted semantics for compatibility with ACL security, use the tpusradd(1), tpusrdel(1), and tpusrmod(1) programs to maintain the user file.

The reserved client name values tpsysadm (system administrator) and tpsysop (system operator) are treated specially by AUTHSVR(5) when processing authentication requests. These values are not allowed to match wildcard client names in the user file.

The application key that is returned by the AUTHSVR is the user identifier. This application key is passed to every service in the *appkey* element of the TPSVCINFO structure.

Note that a standard AUTHSVR is shipped as part of the system in ${TUXDIR}/bin/AUTHSVR and has the semantics as described above. Sample source code is provided in ${TUXDIR}/lib/AUTHSVR.c. The AUTHSVR can be replaced by an application authentication server that validates users and user data (which may not be a password) in an application-dependent fashion (e.g., using Kerberos). If you plan to replace AUTHSVR, take special note of the warning later in this reference page. It is also up to the application to determine what value is returned from the authentication service to be used for the application key (which is passed to each service).

The application keys that correspond to tpsysadm and tpsysop are 0x80000000 and 0xC0000000, respectively.

SECURITY ACL or MANDATORY_ACL

If SECURITY is set to ACL or MANDATORY_ACL, then per-user authentication is enforced and access control lists are supported for access to services, application queues, and events. The name of the authentication service must be ..AUTHSVC which is the default service advertised by AUTHSVR for these SECURITY levels.

The user file must be $APPDIR/tpusr. It is automatically propagated from the master machine to other active machines in the configuration. One instance of the AUTHSVR must be run on the master machine. Additional copies can be run on other active machines in the configuration.

The user file is searched for a matching user name and client name. The entry must match exactly on the user name. The client name must either match exactly, or the client name value in the user file can be specified as the wildcard (*) which will match any client name. A single user can have only one entry in the user file and cannot be a wild-card. The user file can be maintained through the tpusradd(1), tpusrdel(1), and tpusrmod(1) programs, the graphical user interface, or the administrative interface.

The reserved client name values tpsysadm (system administrator) and tpsysop (system operator) are treated specially by AUTHSVR(5) when processing authentication requests. These values are not allowed to match wildcard client names in the user file.

The application key that is returned by the AUTHSVR is the user identifier in the low-order 17 bits and the group identifier in the next 14 bits (the high order bit is reserved for administrative keys). The application keys that correspond to tpsysadm and tpsysop are 0x80000000 and 0xC0000000, respectively. This application key is passed to every service in the *appkey* element of the TPSVCINFO structure.

For SECURITY ACL or MANDATORY_ACL, the standard AUTHSVR that is shipped as part of the system in ${TUXDIR}/bin/AUTHSVR must be used.

Usage

Warning: ${TUXDIR}/lib/AUTHSVR.c is not the source file used to generate ${TUXDIR}/bin/AUTHSVR (don't clobber this executable); if you provide your own AUTHSVR, it is recommended that you install it in ${APPDIR}.

Portability

AUTHSVR is supported as a BEA TUXEDO-supplied server on non-Workstation platforms.

Examples

```
# Using ACL's
*RESOURCES
AUTHSVC  "..AUTHSVC"
SECURITY  ACL

*SERVERS
AUTHSVR SRVGRP="AUTH" SRVID=100 RESTART=Y GRACE=0 MAXGEN=2
#
#
# Using USER_AUTH
*RESOURCES
AUTHSVC  "AUTHSVC"
SECURITY  USER_AUTH

*SERVERS
AUTHSVR SRVGRP="AUTH" CLOPT="-A -- -f /usr/tuxedo/users" \e
 SRVID=100 RESTART=Y GRACE=0 MAXGEN=2
```

See Also `tpaddusr`(1), `tpusradd`(1), `ubbconfig`(5), *BEA TUXEDO Administrator's Guide*, *BEA TUXEDO Programmer's Guide*

compilation(5)

Name compilation-how to compile BEA TUXEDO system application components

Description In order to compile application clients and servers, and subroutines that are link edited with the BEA TUXEDO system-supplied client, `mio`, programmers need to know:

\(bu
what header files to include, and the order in which to specify them

\(bu
what utilities are used to compile the application modules

\(bu
what environment variables to set and export

This reference page provides this information.

BEA TUXEDO System, In General Header File Sequence

UNIX header files should always be included before any BEA TUXEDO system header files. Commonly used UNIX header files are `stdio.h` and `ctype.h`.

Environment Variables In general, the following environment variables should be set and exported:

`TERM`
(indicates terminal type, if a terminal is used)

`TUXDIR`
(specifies the topmost directory where the BEA TUXEDO system software resides)

`PATH`
(should include `$TUXDIR/bin`)

`ULOGPFX`
(prefix of the filename of the central event log; default, `ULOG`)

More information about these variables can be found in the *`Transaction Monitor Programmer's Guide`* and the *`Transaction Monitor Administrator's Guide`*.

BEA TUXEDO System

Header File Sequence

Form files do not contain header information.

C programs that call `mio` functions should include the following header files, in the order listed below:

```
#include <UNIX header files> (if needed by the application)
#include "fml.h"
#include "Usysflds.h"
```

C programs that call BEA TUXEDO system functions that access the Bulletin Board should include the following header files, in the order listed below:

```
#include <UNIX header files> (if needed by the application)
#include "atmi.h"
```

Additionally, programs that call `amaskout` should include the following:

```
#include <signal.h>
#include "Usignal.h"
```

Note: If a program uses `amaskout` and other BEA TUXEDO system functions, UNIX header file(s) should be included before including BEA TUXEDO system header files.

`Usignal.h`, `atmi.h`, `fml.h`, and `Usysflds.h` may be specified in any order, as long as `fml.h` is specified before `Usysflds.h`. For example, the following are both valid:

Sample 1

```
#include <stdio.h>
#include "atmi.h"
#include "fml.h"
#include "Usysflds.h"
#include "fml.h"
#include "Usysflds.h"
```

Sample 2

```
#include <stdio.h>
#include signal.h>
#include "Usignal.h"
#include "atmi.h"
```

Applications using the functions on the `tpservice`(3), `tpalloc`(3), `tpbegin`(3), `tpcall`(3), `tpinit`(3), and `tpopen`(3) reference pages should include the `atmi.h` header file.

Compilation of UFORM Masks

UFORM masks are compiled as follows:

Create field tables and field table header files.
Set and export FIELDTBLS and FLDTBLDIR.

Execute the command:

```
mc filename.m
```

where

- *filename.m* is the name of the file containing UFORM mask definitions.
- FIELDTBLS and FLDTDIR are defined in the Environment
- VARIABLES section below.
- If vuform(1) is used to develop UFORM masks, compilation is included.

To build an mio program, do the following:

5. Create the forms using UFORM or vuform(1).
6. Write and archive lastval and firstval functions (if desired).
7. Write and archive default and validation functions.
8. Build the mio program by executing:

   buildmio [-w] [-o *outfile*] [-v *vallib*] [-d *dfltlib*] [-f *lib*] [-a *genlib*]

where

- -w indicates that a workstation mio should be built
- *outfile* is the name of the executable
- *vallib* is the archive file of validation functions
- *dfltlib* is the archive file of default functions.
- *lib* is a library or object file containing externals needed by the validation or default functions.
- *genlib* is a library or object file containing both validation and default functions.

To build a non-mio client program, execute:

```
buildclient [ -v ] [ -w ] [ -o name] [ -f firstfiles] [ -l lastfiles]
[ -r rmname]
```

where

- *-v* indicates verbose mode
- *-w* indicates workstation mode
- *name* is the executable file name
- *firstfiles* are files to be included before system libraries, and *lastfiles* are files to be included after system libraries.
- *rmname* is specified as the resource manager associated with the client.

To compile a BEA TUXEDO system server, execute:

```
buildserver [ -o pgm] [ -s services] [ -f objectfiles] [ -r rmname]
```

where

- *pgm* is the name of the executable file
- *services* is a comma separated list of services to be offered by the server.
- *objectfiles* is a list of the objectfiles to be linked.
- If more than one file is specified, file names must be separated by white space and the entire list must be enclosed in double quotes.
- *rmname* is specified as the resource manager associated with the server.

Note: See the buildserver(1) reference page for additional information and options. The environment variable CC can be used to designate the compiler to use. CFLAGS can be used to pass parameters to the compiler. For esql programs, follow the esqlc instructions, then run buildserver.

Environment Variables

The following environment variables should be set and exported before executing `mc`(1) or `vuform`(1):

`FIELDTBLS`
(comma-separated list of field table files)

`FLDTBLDIR`
(colon-separated list of directories to search for the `FIELDTBLS`)

The following environment variables can be set and exported before executing `mio`.

`MASKPATH`
(full path of the directory containing the compiled masks, default is the current directory)

`MASKIPCKEY`
(key to mask cache; must be specified if cache is used)

If a default or validation routine references FML fields, set:

`FIELDTBLS`
(comma-separated list of field table files)

`FLDTBLDIR`
(colon-separated list of directories to search for the `FIELDTBLS`)

To restrict access to forms, set:

`OKXACTS`
(comma-separated list of forms that the user can access, default is all)

`NGXACTS`
(comma-separated list of forms that the user cannot access, default is none)

Note: `OKXACTS` is checked first. If the form is listed in `OKXACTS` (or `OKXACTS` is `ALL`), then `NGXACTS` is checked. If the form is listed in `NGXACTS`, then an error is reported. More information about running `mio` can be found in the *`Transaction Monitor Programmer's Guide`*, the *`Transaction Monitor Administrator's Guide`* and the *`Data Entry System Guide`*.

When the system has been built with shared libraries, the following environment variable must be set as shown before executing a client:

```
LD_LIBRARY_PATH=$TUXDIR/lib
```

The following environment variable should be set before executing a server:

`TUXCONFIG`
(full pathname of the binary configuration file, default is the current directory)

The following environment variable should be set and exported before executing system supplied clients (`bkenq`(1), `mio`, `tmadmin`(1), `ud`(1)) in an application with security turned on:

`APP_PW`
(application password)

The following environment variables should be set and exported before executing workstation clients:

`WSENVFILE`
(file containing environment variable settings)

`WSDEVICE`
(network device to use for connection)

`WSTYPE`
(workstation machine type)

More information about options for servers can be found on the `servopts`(5) reference page.

FML Programs

Header File Sequence

C programs that call FML functions should include the following header files, in the order listed below:

```
#include <UNIX header files> (if needed by the application)
#include "fml.h"
```

Note: `mio` programs that use FML must also include `fml.h`.

Compilation of FML Programs

To compile a program that contains FML functions, execute:

```
cc pgm.c -I $TUXDIR/include -L $TUXDIR/lib -lfml -lgp -o pgm
```

where *pgm* is the name of the executable file.

If the `-L` option is not locally supported, use the following command instead:

```
cc pgm.c -I $TUXDIR/include $TUXDIR/lib/libfml.a $TUXDIR/lib/libgp.a -o pgm
```

Note: The order in which libraries are specified is significant. Use the order given above.

Compiling FML VIEWS

To use the FML view compiler, execute:

```
viewc view_file
```

where

view_file is one or more files containing source view descriptions.

Note: `viewc` invokes the C compiler. The environment variable CC can be used to designate the compiler to use. The environment variable CFLAGS can be used to pass a set of parameters to the compiler.

Environment Variables for FML

The following environment variables should be set and exported when running an application that uses FML:

`FIELDTBLS`
(comma-separated list of field table files)

`FLDTBLDIR`
(colon-separated list of directories to search for the `FIELDTBLS`)

The following environment variables should be set and exported when executing `viewc`:

`FIELDTBLS`
(comma-separated list of field table files)

`FLDTBLDIR`
(colon-separated list of directories to search for the `FIELDTBLS`)

`VIEWDIR`
(directory containing view files, default is current directory)

Portability The `buildclient`(1) alias `buildclt`(1) must be used on MS-DOS and OS/2 machines because of file naming restrictions. Differences in file naming semantics and syntax also affect the following environment variables:

- `FLDTBLDIR`
- `NLSPATH`
- `TUXDIR`
- `ULOGPFX`
- `VIEWDIR`

See Also `buildclient`(1), `buildserver`(1), `buildmio`(1), `mc`(1), `viewc`(1), `cc`(1) in a UNIX reference manual

DMADM (5)

Name DMADM-Domains administrative server

Synopsis

```
DMADM SRVGRP = "identifier"
SRVID = "number"
REPLYQ = "N"
```

Description The Domains administrative server (DMADM) is a BEA TUXEDO system-supplied server that provides run-time access to the BDMCONFIG file.

DMADM is described in the SERVERS section of the UBBCONFIG file as a server running within a group, e.g., DMADMGRP. There should be only one instance of the DMADM running in this group and it must not have a reply queue (REPLYQ must be set to "N").

The following server parameters can also be specified for the DMADM server in the SERVERS section: SEQUENCE, ENVFILE, MAXGEN, GRACE, RESTART, RQPERM and SYSTEM_ACCESS.

The BDMCONFIG environment variable should be set to the pathname of the file containing the binary version of the DMCONFIG file.

Portability DMADM is supported as a BEA TUXEDO system-supplied server on UNIX System operating systems.

Interoperability DMADM must be installed on a BEA TUXEDO system Release 5.0 or later; other machines in the same domain with a R5.0 gateway may be Release 4.1 or later.

Examples The following example illustrates the definition of the administrative server and a gateway group in the UBBCONFIG file.

```
#
*GROUPS
DMADMGRP LMID=mach1 GRPNO=1
gwgrp   LMID=mach1 GRPNO=2
#
*SERVERS
DMADM SRVGRP="DMADMGRP" SRVID=1001 REPLYQ=N RESTART=Y GRACE=0
GWADM SRVGRP="gwgrp" SRVID=1002 REPLYQ=N RESTART=Y GRACE=0
GWTDOMAIN SRVGRP="gwgrp" SRVID=1003 RQADDR="gwgrp" REPLYQ=Y RESTART=Y MIN=1 MAX=1
```

See Also dmadmin(1), tmboot(1), dmconfig(5), GWADM(5), servopts(5), ubbconfig(5), *BEA TUXEDO Domains Guide*, *BEA TUXEDO Administrator's Guide*

dmconfig(5)

Name dmconfig BEA TUXEDO system ASCII domain configuration file

Description dmconfig is the ASCII version of a BEA TUXEDO system domain configuration file; it is also referred to by its environmental variable name: DMCONFIG. The dmconfig file is parsed and loaded into a binary version by the dmloadcf(1) utility. The binary configuration file, called the BDMCONFIG file, contains information used by domain gateways to initialize the context required for communications with other domains. dmadmin(1) uses the binary file (or a copy of it) in its monitoring activity. There will be one BDMCONFIG file for each BEA TUXEDO system domain application that uses the Domains feature.

A DMCONFIG file, and its binary BDMCONFIG counterpart, are analogous to the UBBCONFIG and TUXCONFIG files of a non-Domains BEA TUXEDO system application. The DMCONFIG file extends the definition of a non-Domains BEA TUXEDO system application so that the application becomes a domain.

Definitions A BEA TUXEDO system domain *Application* is defined as the environment described in a single TUXCONFIG file. A BEA TUXEDO system application can communicate with another BEA TUXEDO system application or with another TP application via a domain gateway group. In "BEA TUXEDO system domain" terms, an application is the same as a TP Domain.

A *Gateway Group* is a collection of domain gateway processes that provide communication services with a specific type of TP Domain.

A *Domain Gateway* is a BEA TUXEDO system domain process that relays requests to another TP Domain and receives replies.

A *Local Domain* is a part of the application (set or subset of services) that is made available to other domains. A Local Domain is always represented by a Domain Gateway Group, and both terms are used as synonyms.

A *Remote Domain* is a remote application that is accessed through a Gateway Group. The remote application may be another BEA TUXEDO system domain application or an application running under another TP system.

A *Remote Service* is a service provided by a remote domain that is made available to the application through a Gateway Group.

A *Local Service* is a service of a local domain that is made available to remote domains through a Gateway Group.

Configuration File Format

The format of a domain configuration file is as follows:

The file is made up of eight possible specification sections. Allowable section names are: DM_LOCAL_DOMAINS, DM_REMOTE_DOMAINS, DM_LOCAL_SERVICES, DM_REMOTE_SERVICES, DM_ROUTING, DM_ACCESS_CONTROL and DM_*domtype*, where *domtype* is either OSITP, SNAX or TDOMAIN. (This reference page describes only how to configure a domain of type TDOMAIN. See BEA Connect documentation for information about how to configure an OSITP or an SNAX domain.) The DM_LOCAL_DOMAINS section must precede the DM_REMOTE_DOMAINS section.

Parameters are generally specified by: *KEYWORD* = *value*. This sets *KEYWORD* to *value*. Valid keywords are described below within each section. *KEYWORD*s are reserved; they can not be used as *value*s unless they are quoted.

Lines beginning with the reserved word, DEFAULT:, contain parameter specifications that apply to all lines that follow them in the section in which they appear. Default specifications can be used in all sections. They can appear more than once in the same section. The format for these lines is:

```
DEFAULT: [ KEYWORD1 = value1 [ KEYWORD2 = value2 [...]]]
```

The values set on this line remain in effect until reset by another DEFAULT: line, or until the end of the section is reached. These values can also be overridden on non-DEFAULT: lines by placing the optional parameter setting on the line. If on a non-DEFAULT: line, the parameter setting is valid for that line only; lines that follow revert to the default setting. If DEFAULT: appears on a line by itself, all previously set defaults are cleared and their values revert to the system defaults.

If a value is *numeric*, standard C notation is used to denote the base (that is, 0x prefix for base 16 (hexadecimal), 0 prefix for base 8 (octal), and no prefix for base 10 (decimal)). The range of values acceptable for a numeric parameter are given under the description of that parameter.

If a value is an *identifier*, standard C rules are used. An *identifier* must start with an alphabetic character or underscore and contain only alphanumeric characters or underscores. The maximum allowable length of an identifier is 30 (not including the terminating null). An identifier cannot be the same as any *KEYWORD*.

A value that is neither an integer number or an identifier must be enclosed in double quotes.

Input fields are separated by at least one space (or tab) character.

"#" introduces a comment. A newline ends a comment.

Blank lines and comments are ignored.

Comments can be freely attached to the end of any line.

Lines are continued by placing at least one tab after the newline. Comments can not be continued.

`VERSION`=*string_value* is a field that is not checked by the software; it is provided simply as a place where customers can enter a string that may have some documentation value to the application.

The DM_LOCAL_DOMAINS Section

This section identifies local domains and their associated gateway groups. The section must have an entry for each gateway group (Local Domain). Each entry specifies the parameters required for the domain gateway processes running in that group.

Entries have the form: *LDOM* required parameters [optional parameters] where *LDOM* is an *identifier* value used to name the local domain. *LDOM* must be unique within a particular configuration. As you will see in the description of the `DM_LOCAL_SERVICES` section, *LDOM* is the identifier that associates the local services with a particular gateway group.

The following are required parameters:

`GWGRP` = *identifier*
: specifies the name of the gateway server group (the name provided in the `TUXCONFIG` file) representing this local domain. There is a one-to-one relationship between a *DOMAINID* (see below) and the name of the gateway server group.

`TYPE` = *identifier*
: is used for grouping local domains into classes. `TYPE` can be set to one of the following values: `TDOMAIN`, `SNAX` or `OSITP`. The `TDOMAIN` value indicates that this local domain can only communicate with another BEA TUXEDO system domain. The `SNAX` value indicates that this local domain communicates with another TP domain via the `SNA` protocol. The `OSITP` value indicates that this local domain communicates with another TP Domain via the OSI-TP protocol. Domain types must be defined in the `$TUXDIR/udataobj/DMTYPE` file.

`DOMAINID` = *string*
: is used to identify the local domain. `DOMAINID` must be unique across both local and remote domains. The value of *string* can be a sequence of characters (for example, "`BA.CENTRAL01`"), or a sequence of hexadecimal digits preceded by "0x" (for example, "`0x0002FF98C0000B9D6`"). `DOMAINID` must

be 30 octets or fewer in length. If the value is a string, it must be 30 characters or fewer (counting the trailing null).

Optional parameters describe resources and limits used in the operation of domain gateways:

AUDITLOG = *string*

specifies the name of the audit log file for this local domain. The audit log feature is activated from the dmadmin(1) command and records all the operations within this local domain. If the audit log feature is active and this parameter is not specified, the file DMmmddyy.LOG (where mm=month, dd=day, and yy=year) is created in the directory specified by the $APPDIR environment variable or the APPDIR keyword of the MACHINES section of the TUXCONFIG file.

BLOCKTIME = *numeric*

specifies the maximum wait time allowed for a blocking call. The value sets a multiplier of the SCANUNIT parameters specified in the TUXCONFIG file. The value SCANUNIT * BLOCKTIME must be greater than or equal to SCANUNIT and less than 32,768 seconds. If this parameter is not specified, the default is set to the value of the BLOCKTIME parameter specified in the TUXCONFIG file. A timeout always implies a failure of the affected request. Notice that the timeout specified for transactions in the TUXCONFIG will always be used when the request is issued within a transaction.

CONNECTION_POLICY=*string*

specifies the conditions under which a local domain gateway tries to establish a connection to a remote domain. Supported values are: ON_DEMAND, ON_STARTUP, or INCOMING_ONLY.

A connection policy of ON_DEMAND means that a connection will be attempted only when requested by either a client request to a remote service or an administrative "connect" command. The default setting for CONNECTION_POLICY is ON_DEMAND. The ON_DEMAND policy provides the equivalent behavior to previous releases, in which the CONNECTION_POLICY was not explicitly available. Connection retry processing is not allowed when the connection policy is ON_DEMAND.

A connection policy of ON_STARTUP means that a domain gateway will attempt to establish a connection with its remote domains at gateway server initialization time. If the value chosen for the CONNECTION_POLICY is set to ON_STARTUP, then remote services (that is, services advertised by the local domain gateway) will be advertised only if a connection is successfully established to that remote domain. Thus, if there is no active connection to the

remote domain, then the remote services will be suspended. By default, this connection policy will retry failed connections every 60 seconds, but you can specify a different value for this interval (see MAXRETRY and RETRY_INTERVAL).

A connection policy of INCOMING_ONLY means that a domain gateway will not attempt an initial connection to remote domains upon starting and remote services will initially be suspended. The domain gateway will be available for incoming connections from remote domains, and remote services will be advertised when the local domain gateway receives an incoming connection. Connection retry processing is not allowed when the connection policy is INCOMING_ONLY. (The CONNECTION_POLICY parameter does not apply to domains of type OSI or SNA.)

DMTLOGDEV = *string*

specifies the BEA TUXEDO file system that contains the Domain transaction log (DMTLOG) for this machine. The DMTLOG is stored as a BEA TUXEDO system VTOC table on the device. If this parameter is not specified, the domain gateway group is not allowed to process requests in transaction mode. Local domains running on the same machine can share the same DMTLOGDEV filesystem, but each local domain must have its own log (a table in the DMTLOGDEV) named as specified by the DMTLOGNAME keyword (see below).

DMTLOGNAME = *identifier*

specifies the name of the domain transaction log for this domain. This name must be unique when the same DMTLOGDEV is used for several local domains. If not specified, the default is the string "DMTLOG". The name must be 30 characters or less.

DMTLOGSIZE = *numeric*

specifies the numeric size, in pages, of the Domain transaction log for this machine. It must be greater than 0 and less than the amount of available space on the BEA TUXEDO file system. If not specified, the default is 100 pages.

MAXRDOM = *numeric*

specifies the maximum number of connections allowed per gateway. It applies only to OSITP and SNA domains.

MAXRDTRAN = *numeric*

specifies the maximum number of domains that can be involved in a transaction. It must be greater than 0 and less than 32,768. If not specified, the default is 16.

MAXRETRY={*numeric*|MAXLONG}

specifies the number of times that a domain gateway will try to establish connections to remote domains. The minimum value is 0 and the maximum is MAXLONG. MAXLONG indicates that retry processing will be repeated indefinitely, or until a connection is established. For a connection policy of ON_STARTUP, the default setting for MAXRETRY is MAXLONG. Setting MAXRETRY=0 turns off the auto retry mechanism. For other connection policies, auto retries are disabled.

The MAXRETRY parameter is valid only when the connection policy is ON_STARTUP. The MAXRETRY parameter does not apply to domains of type OSI or SNA.

MAXTRAN = *numeric*

specifies the maximum number of simultaneous global transactions allowed on this local domain. It must be greater than or equal to 0 and less than or equal to the MAXGTT parameter specified in the TUXCONFIG file. If not specified, the default is the value of MAXGTT.

MTYPE = *value*

is used for grouping domains so that encoding/decoding of messages between domains can be bypassed. If MTYPE is not specified, the default is to turn encoding/decoding on. If the *value* set for the MTYPE field is the same in both the DM_LOCAL_DOMAINS and the DM_REMOTE_DOMAINS section of a domain configuration file, data encoding/decoding is bypassed. The *value* set for MTYPE can be any string value up to 15 characters in length. It is used only for comparison.

RETRY_INTERVAL=*numeric*

specifies the number of seconds between automatic attempts to establish a connection to remote domains. The minimum value is 0 and the maximum value is 2147483647. The default setting for RETRY_INTERVAL is 60. If MAXRETRY is set to 0, then setting RETRY_INTERVAL is not allowed.

The RETRY_INTERVAL parameter is valid only when the connection policy is ON_STARTUP. For other connection policies, automatic retries are disabled. The RETRY_INTERVAL parameter does not apply to domains of type OSI or SNA.

SECURITY = *value*

specifies the type of application security to be enforced. The SECURITY parameter currently has three valid values: NONE, APP_PW or DM_PW. The value NONE indicates that no security is used. This is the default. The value APP_PW indicates that the application password security is to be enforced when a connection is established from a remote domain. The application

password should be defined in the TUXCONFIG file. The value DM_PW indicates that domain password security is to be enforced when a connection is established from a remote domain. Domain passwords must be defined through the dmadmin(1) command. This option does not apply to domains of type OSITP.

The DM_REMOTE_DOMAINS Section

This section identifies the known set of remote domains and their characteristics.

Entries have the form: *RDOM* required parameters [optional MTYPE parameter] where *RDOM* is an *identifier* value used to identify each remote domain known to this configuration. *RDOM* must be unique within the configuration.

The TYPE and DOMAINID parameters are required; MTYPE is optional:

TYPE = *identifier*
: is used for grouping remote domain into classes. TYPE can be set to one of the following values: TDOMAIN, SNAX or OSITP. The TDOMAIN value indicates that this remote domain can only communicate with another BEA TUXEDO system domain. The SNAX value indicates that this domain communicates with another TP domain via SNA protocol. The OSITP value indicates that this remote domain communicates with another TP domain via the OSI-TP protocol.

DOMAINID = *string*
: is used to identify a remote domain. DOMAINID must be 30 octets or fewer in length. If the value is a string, it must be 30 characters or fewer (counting the trailing null). DOMAINID must be unique across remote domains. The value of *string* can be a sequence of characters or a sequence of hexadecimal digits preceded by "0x".

MTYPE = *value*
: is used for grouping domains so that encoding/decoding of messages between domains can be bypassed. If MTYPE is not specified, the default is to turn encoding/decoding on. If the *value* set for the MTYPE field is the same in both the DM_LOCAL_DOMAINS and the DM_REMOTE_DOMAINS section of a domain configuration file, data encoding/decoding is bypassed. The *value* set for MTYPE can be any string value up to 15 characters. It is used only for comparison.

Entries associated with a remote domain can be specified more than once. The first one specified is considered to be the primary address, which means it is the first one tried when a connection is being attempted to a remote domain. If a network connection cannot be established using the NWADDR of the primary entry , the NWADDR associated with the secondary entry is used.

The DM_TDOMAIN Section

This section defines the addressing information required by domains of type TDOMAIN. This section should have one entry per local domain if requests from remote domains to local services are accepted on that local domain (gateway group), and one entry per remote domain accessible by the defined local domains.

Entries have the form:

```
DOM required parameters [optional parameters]
```

where DOM is an identifier value used to identify either a local domain (LDOM) or a remote domain (RDOM) in the DM_LOCAL_DOMAINS section or in the DM_REMOTE_DOMAINS section. The DOM identifier must match a previously defined LDOM in the DM_LOCAL_DOMAINS section or RDOM in the DM_REMOTE_DOMAINS section.

The following parameter is required:

NWADDR = *string*

This parameter specifies the network address associate with a local domain or a remote domain. If the association is with a local domain, the NWADDR is used to accept connections from other BEA TUXEDO system domains. If the association is with a remote domain, the NWADDR is used to initiate a connection. Specifies the network address to be used by the process as its listening address. The listening address for a domain gateway is the means by which it is contacted by other gateway processes participating in the application.

If string has the form "0xhex-digits" or "\\xhex-digits",it must contain an even number of valid hex digits. These forms are translated internally into a character array containing TCP/IP addresses may also be in either of the following two forms:

```
"//host.name:port_number"
```

```
"//#.#.#.#:port_number"
```

In the first of these formats, *hostname* is resolved to a TCP/IP host address at the time the address is bound using the locally configured name resolution facilities accessed via gethostbyname(3c). The "#.#.#.#" is the dotted

decimal format where each # represents a decimal number in the range 0 to 255.

Port_number is a decimal number in the range 0 to 65535. the hexadecimal representations of the string specified. This parameter specifies the network address used by a local or a remote domain to accept connections from other BEA TUXEDO system domain domains. If string has the form "0xhex-digits", it must contain an even number of valid hexadecimal digits.

The following parameters are optional:

NWDEVICE = *string*

Specifies the device file name to be used when binding to the listening address of a local or a remote domain. The NWDEVICE parameter is not required. In prior releases, if the networking functionality is TLI-based, the device name must be an absolute pathname.

CMPLIMIT = *numeric*

This parameter specifies the compression threshold to be used when sending data to the remote domain. Application buffers larger than this size will be compressed. This attribute defaults to 2,147,483,647.

MINENCRYPTBITS={0|40|128}

When establishing a network link for this domain, require at least this minimum level of encryption. Zero 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 is zero.

MAXENCRYPTBITS={0|40|128}

When establishing a network link, negotiate encryption up to this level. Zero means no encryption, while 40 and 128 specify the encryption length (in bits). The default is 128

Entries associated with a remote domain can be specified more than once. The first one specified is considered to be the primary address, which means it is the first one tried when a connection is being attempted to a remote domain. If a network connection cannot be established using the primary entry's NWADDR, the NWADDR associated with the secondary entry is used.

If this /TDOMAIN is a local domain (that is, if *DOM* matches a previously specified *LDOM*), then NWADDRs are network addresses to be used to listen for incoming connections. A secondary entry cannot be used for local domain entries.

If this /TDOMAIN entry points to a secondary remote domain (that is, if *DOM* matches a previously specified *RDOM*), then the entry points to a gateway that is only used when a network connection cannot be established using the NWADDR of the primary entry. The secondary remote gateway must reside in a different BEA TUXEDO Domain from the primary. However, the secondary gateway must have the same DOMAINID defined in its DM_LOCAL_DOMAINS section as the primary remote gateway; this arrangement is often referred to as a *mirrored* gateway. This feature is not recommended for use with transactions or conversations. In addition, the mirrored gateway is not recommended for use when the primary gateway is available.

The DM_ACCESS_CONTROL Section

This section specifies the access control lists used by local domain. Lines in this section are of the form: *ACL_NAME* required parameters where *ACL_NAME* is an (*identifier*) name used to identify a particular access control list; it must be 15 characters or less in length.

The only required parameter is:

```
ACLIST = identifier [,identifier]
```

where an ACLIST is composed of one or more remote domain names (RDOM) separated by commas. The wildcard character (*) can be used to specify that all the remote domains defined in the DM_REMOTE_DOMAINS section can access a local domain.

The DM_LOCAL_SERVICES Section

This section provides information on the services exported by each local domain. This section is optional; if it is not specified, then all local domains defined in the DM_LOCAL_DOMAINS section accept requests to all of the services advertised by the BEA TUXEDO system domain application. If this section is specified then it should be used to restrict the set of local services that can be requested from a remote domain.

Lines within this section have the form:

```
service [optional parameters]
```

where *service* is the (*identifier*) local name of the exported service, and it must be 15 characters or fewer in length. This name corresponds to a name advertised by one or more servers running with the local BEA TUXEDO system domain application. Notice that exported services inherit the properties specified for the service in an entry in the SERVICES section of the TUXCONFIG file, or their defaults. Some of the properties that may be inherited are: LOAD, PRIO, AUTOTRAN, ROUTING, BUFTYPE, and TRANTIME.

Optional parameters are:

ACL = *identifier*
: specifies the name of the access control list (ACL) to be used by the local domain to restrict requests made to this service by remote domains. The name of the ACL is defined in the DM_ACCESS_CONTROL section. If this parameter is not specified then access control will not be performed for requests to this service.

LDOM = *identifier*
: specifies the name identifying the local domain exporting this service. If this keyword is not specified then all the local domains defined in the DM_LOCAL_DOMAINS section will accept requests to this local service.

INBUFTYPE = *type[:subtype]*
: restricts the buffer type naming space of data types accepted by this service to a single buffer type. This parameter should be defined when the service is going to be used from an OSITP type gateway that uses the UDT ASE Application Context. It does not apply to /TDOMAIN.

OUTBUFTYPE = *type[:subtype]*
: restricts the buffer type naming space of data types returned by this service to a single buffer type. This parameter should be defined when the service is going to be used from an OSITP type gateway that uses the UDT ASE Application Context. The FML buffer type cannot be used for OSITP type gateways. It does not apply to /TDOMAIN.

RNAME = *string*
: specifies the name exported to remote domains. This name will be used by the remote domains for request to this service. If this parameter is not specified, the local service name is supposed to be the name used by any remote domain.

The DM_REMOTE_SERVICES Section

This section provides information on services "imported" and available on remote domains. Lines within this DM_REMOTE_SERVICES section have the form:

service [optional parameters]

where *service* is the (*identifier*) name used by the local BEA TUXEDO system domain application for a particular remote service. Remote services are associated with a particular remote domain.

Optional parameters are:

CONV = { Y | N }
: specifies whether (Y) or not (N) the remote service is a conversational service. The default is N.

LDOM = *identifier*

specifies the name of a local domain in charge of routing requests to this remote service. The gateway group associated with the local domain advertises *service* in the BEA TUXEDO system domain Bulletin Board. If this parameter is not specified then all the local domains will be able to accept requests to this remote service. The service request will be then redirected to a remote domain of the same type (see RDOM keyword below).

INBUFTYPE = *type[:subtype]*

restricts the buffer type naming space of data types accepted by this service to a single buffer type. This parameter should be defined when the service is going to be used from an OSITP type gateway that uses the UDT ASE Application Context. The FML buffer type cannot be used for OSITP type gateways. It does not apply to /TDOMAIN.

OUTBUFTYPE = *type*[*:subtype*]

restricts the buffer type naming space of data types returned by this service to a single buffer type. This parameter should be defined when the service is going to be used from an OSITP type gateway that uses the UDT ASE Application Context. The FML buffer type cannot be used for OSITP type gateways. It does not apply to /TDOMAIN.

RDOM = *identifier1*[,*identifier2*][,*identifier3*]

specifies the name of the remote domain responsible for the execution of this service. If this parameter is not specified and a routing criteria is not specified, the local domain assumes that any remote domain of the same type accepts requests for this service.

You must specify ON_STARTUP as the value of the CONNECTION_POLICY parameter if you want to configure alternate remote domains with the *identifier2* and *identifier3* arguments. If *identifier2* is configured, it is used for failover. (When the remote domain specified by *identifier1* is unavailable, the remote domain specified by *identifier2* is used.) Similarly, if *identifier3* is configured, it is used for failover. (When the remote domain specified by *identifier1* and *identifier2* are unavailable, the remote domain specified by *identifier3* is used.)

RNAME = *string*

specifies the actual service name expected by the remote domain. If this parameter is not specified, the remote service name is the same as the name specified in *service*.

ROUTING = *identifier*

when more than one remote domain offers the same service, a local domain can perform data dependent routing if this optional parameter is specified. The *identifier* specifies the name of the routing criteria used for this data dependent routing. If not specified, data dependent routing is not done for this service. *identifier* must be 15 characters or less in length. If multiple entries exist for the same service name but with different RDOM parameters, the ROUTING parameter should be the same for all of these entries.

TRANTIME = *integer*

specifies the default timeout value in seconds for a transaction automatically started for the associated service. The value must be greater than or equal to 0 and less than 2147483648. The default is 30 seconds. A value of 0 implies the maximum timeout value for the machine.

The DM_ROUTING Section

This section provides information for data dependent routing of service requests using FML, VIEW, X_C_TYPE, and X_COMMON typed buffers. Lines within the DM_ROUTING section have the form:

CRITERION_NAME required parameters

where *CRITERION_NAME* is the (*identifier*) name of the routing entry that was specified on the services entry. *CRITERION_NAME* must be 15 characters or less in length.

Required parameters are:

FIELD = *identifier*

specifies the name of the routing field. It must be 30 characters or less. This field is assumed to be a field name that is identified in an FML field table (for FML buffers) or an FML view table (for VIEW, X_C_TYPE, or X_COMMON buffers). The FLDTBLDIR and FIELDTBLS environment variables are used to locate FML field tables, and the VIEWDIR and VIEWFILES environment variables are used to locate FML view tables. If a field in an FML32 buffer will be used for routing, it must have a field number less than or equal to 8191.

RANGES = *string*

specifies the ranges and associated remote domain names (RDOM) for the routing field. *string* must be enclosed in double quotes. The format of *string* is a comma-separated ordered list of range/RDOM pairs (see EXAMPLES below).

A range is either a single value (signed numeric value or character string in single quotes), or a range of the form ‘lower - upper” (where lower and upper

are both signed numeric values or character strings in single quotes). Note that "lower" must be less than or equal to "upper".

To embed a single quote in a character string value (as in `O'Brien`, for example), it must be preceded by two backslashes (`O\\'Brien`).

The value `MIN` can be used to indicate the minimum value for the data type of the associated `FIELD`; for strings and carrays, it is the null string; for character fields, it is 0; for numeric values, it is the minimum numeric value that can be stored in the field.

The value `MAX` can be used to indicate the maximum value for the data type of the associated `FIELD`; for strings and carrays, it is effectively an unlimited string of octal-255 characters; for a character field, it is a single octal-255 character; for numeric values, it is the maximum numeric value that can be stored in the field. Thus, ``MIN - -5" is all numbers less than or equal to -5 and ``6 - MAX" is all numbers greater than or equal to 6. The meta-character "`*`" (wild-card) in the position of a range indicates any values not covered by the other ranges previously seen in the entry; only one wild-card range is allowed per entry and it should be last (ranges following it will be ignored).

The routing field can be of any data type supported in `FML`. A numeric routing field must have numeric range values and a string routing field must have string range values. String range values for string, carray, and character field types must be placed inside a pair of single quotes and can not be preceded by a sign. Short and long integer values are a string of digits, optionally preceded by a plus or minus sign. Floating point numbers are of the form accepted by the C compiler or `atof()`: an optional sign, then a string of digits optionally containing a decimal point, then an optional e or E followed by an optional sign or space, followed by an integer.

When a field value matches a range, the associated `RDOM` value specifies the remote domain to which the request should be routed. A `RDOM` value of "*" indicates that the request can go to any remote domain known by the gateway group.

Within a range/`RDOM` pair, the range is separated from the `RDOM` by a ":".

`BUFTYPE = ~`*`type1`*`[:`*`subtype1[,subtype2 . . .`*`]][;`*`type2[:subtype3`*`[, . . . ]]] . . .~`

is a list of types and subtypes of data buffers for which this routing entry is valid. The types are restricted to be either `FML, VIEW, X_C_TYPE,` or `X_COMMON`. No subtype can be specified for type `FML` and subtypes are

required; for the other types; "*" is not allowed). Duplicate type/subtype pairs can not be specified for the same routing criteria name; more than one routing entry can have the same criteria name as long as the type/subtype pairs are unique. This parameter is required. If multiple buffer types are specified for a single routing entry, the data types of the routing field for each buffer type must be the same.

If the field value is not set (for `FML` buffers), or does not match any specific range and a wild-card range has not been specified, an error is returned to the application process that requested the execution of the remote service.

Files The `BDMCONFIG` environment variable is used to find the `BDMCONFIG` configuration file.

Example1 The following configuration file defines a 5-site domain configuration. The example shows 4 Bank Branch domains communicating with a Central Bank Branch. Three of the Bank Branches run within other BEA TUXEDO system domain domains. The fourth Branch runs under the control of another TP Domain and OSI-TP is used in the communication with that domain. The example shows the BEA TUXEDO system domain Domain configuration file from the Central Bank point of view.

```
# TUXEDO DOMAIN CONFIGURATION FILE FOR THE CENTRAL BANK
#
#
*DM_LOCAL_DOMAINS
# <local domain name> <Gateway Group name> <domain type> <domain id> <log device>
#       [<audit log>] [<blocktime>]
#       [<log name>] [<log offset>] [<log size>]
#       [<maxrdom>] [<maxrdtran>] [<maxtran>]
#       [<maxdatalen>] [<security>]
#       [<tuxconfig>] [<tuxoffset>]
#
#
DEFAULT: SECURITY = NONE

c01  GWGRP = bankg1
     TYPE = TDOMAIN
     DOMAINID = "BA.CENTRAL01"
     DMTLOGDEV = "/usr/apps/bank/DMTLOG"
     DMTLOGNAME = "DMTLG_C01"

c02  GWGRP = bankg2
     TYPE = OSITP
     DOMAINID = "BA.CENTRAL01"
     DMTLOGDEV = "/usr/apps/bank/DMTLOG"
     DMTLOGNAME = "DMTLG_C02"
```

```
    NWDEVICE = "OSITP"
    URCH = "ABCD"

#
*DM_REMOTE_DOMAINS
#remote <domain name> <domain type> <domain id>
#
b01  TYPE = TDOMAIN
    DOMAINID = "BA.BANK01"

b02  TYPE = TDOMAIN
    DOMAINID = "BA.BANK02"

b03  TYPE = TDOMAIN
    DOMAINID = "BA.BANK03"

b04  TYPE = OSITP
    DOMAINID = "BA.BANK04"
    NWDEVICE = "/dev/osi"
    URCH = "ABCD"

*DM_TDOMAIN
#
# <local or remote domain name> <network address> [<nwdevice>]
#
# Local network addresses
c01  NWADDR = "//newyork.acme.com:65432"  NWDEVICE ="/dev/tcp"
c02  NWADDR = "//192.76.7.47:65433"  NWDEVICE ="/dev/tcp"
# Remote network addresses
b01  NWADDR = "//192.11.109.5:1025" NWDEVICE = "/dev/tcp"
b02  NWADDR = "//dallas.acme.com:65432" NWDEVICE = "/dev/tcp"
b03  NWADDR = "//192.11.109.156:4244" NWDEVICE = "/dev/tcp"

*DM_OSITP
#
#<local or remote domain name> <apt> <aeq>
#   [<aet>] [<acn>] [<apid>] [<aeid>]
#   [<profile>]
#
c02  APT = "BA.CENTRAL01"
   AEQ = "TUXEDO.R.4.2.1"
   AET = "{1.3.15.0.3},{1}"
   ACN = "XATMI"
b04  APT = "BA.BANK04"
   AEQ = "TUXEDO.R.4.2.1"
   AET = "{1.3.15.0.4},{1}"
   ACN = "XATMI"

*DM_LOCAL_SERVICES
```

```
#<service_name> [<Local Domain name>] [<access control>] [<exported svcname>]
#       [<inbuftype>] [<outbuftype>]
#
open_act ACL = branch
close_act ACL = branch
credit
debit
balance
loan   LDOM = c02 ACL = loans

*DM_REMOTE_SERVICES
#<service_name> [<Remote domain name>] [<local domain name>]
#        [<remote svcname>] [<routing>] [<conv>]
#        [<trantime>] [<inbuftype>] [<outbuftype>]
#
tlr_add LDOM = c01 ROUTING = ACCOUNT
tlr_bal LDOM = c01 ROUTING = ACCOUNT
tlr_add RDOM = b04 LDOM = c02 RNAME ="TPSU002"
tlr_bal RDOM = b04 LDOM = c02 RNAME ="TPSU003"
tlr_bal RDOM = b04, b03 LDOM = c01
*DM_ROUTING
# <routing criteria> <field> <typed buffer> <ranges>
#
ACCOUNT FIELD = branchid BUFTYPE ="VIEW:account"
   RANGES ="MIN - 1000:b01, 1001-3000:b02, *:b03"

*DM_ACCESS_CONTROL
#<acl name> <Remote domain list>
#
branch ACLIST = b01, b02, b03
loans  ACLIST = b04
```

Example2 This example shows the BEA TUXEDO system domain Domain Configuration file required at one of the Bank Branches (`BANK01`).

```
#
#TUXEDO DOMAIN CONFIGURATION FILE FOR A BANK BRANCH
#
#
*DM_LOCAL_DOMAINS
#
b01  GWGRP = auth
   TYPE = TDOMAIN
   DOMAINID = "BA.BANK01"
   DMTLOGDEV = "/usr/apps/bank/DMTLOG"

*DM_REMOTE_DOMAINS
#
c01  TYPE = TDOMAIN
```

```
   DOMAINID = "BA.CENTRAL01"

*DM_TDOMAIN
#
b01  NWADDR = "//192.11.109.156:4244" NWDEVICE = "/dev/tcp"
c01  NWADDR = "//newyork.acme.com:65432"  NWDEVICE ="/dev/tcp"
*DM_LOCAL_SERVICES
#
tlr_add   ACL = central
tlr_bal   ACL = central

*DM_REMOTE_SERVICES
#

OPA001   RNAME = "open_act"
CLA001   RNAME = "close_act"
CRD001   RNAME = "credit"
DBT001   RNAME = "debit"
BAL001   RNAME = "balance"

*DM_ACCESS_CONTROL
#
central   ACLIST = c01
```

Network Addresses

Suppose the local machine on which a /T domain is being run is using TCP/IP addressing and is named *backus.company.com*, with address `155.2.193.18`. Further suppose that the port number at which the /T domain should accept requests is `2334`. Assume that port number `2334` has been added to the network services database under the name *bankapp-gwtaddr*. The address can be represented in the following ways:

//155.2.193.18:*bankapp-gwtaddr*
//155.2.193.18:2334
//*backus.company.com:bankapp-gwtaddr*
//*backus.company.com:*2334
0x0002091E9B02C112

The last of these representations is hexadecimal format. The `0002` is the first part of a TCP/IP address. The `091E` is the port number `2334` translated into a hexadecimal number. After that each element of the IP address `155.2.193.12` is translated into a hexadecimal number. Thus the `155` becomes `9B`, `2` becomes `02` and so on.

See Also

`build_dgw`(1), `dmadmin`(1), `tmboot`(1), `tmshutdown`(1), `DMADM`(5), `dmloadcf`(1), `dmunloadcf`(1), `GWADM`(5), `GWTDOMAIN`(5), *BEA TUXEDO Domains Guide, BEA TUXEDO Administrator's Guide*, *BEA TUXEDO Programmer's Guide*

EVENTS(5)

Name EVENTS-list of system-generated events

Description The System Event Monitor feature detects and reports certain pre-defined events, primarily failures, that a system operator should be aware of. Each event report is an FML32 buffer containing generic fields that describe the event plus other fields that describe the object associated with the event.

This reference page first defines the generic event reporting fields, and then lists all events detected in the current BEA TUXEDO system release.

Limitations Event reporting is currently limited to classes defined in TM_MIB(5). Event reporting uses the MIB information base. See MIB(5) and TM_MIB(5) for a definition and the availability of "local attributes" and be aware that the availability of a local attribute depends on the state of communication within the application's network.

Generic Event Reporting Fields

TA_OPERATION: *string*
: The literal string "EVT", which identifies this buffer as an event report notification.

TA_EVENT_NAME: *string*
: A string that uniquely identifies this event. All system-generated events begin with ".Sys".

TA_EVENT_SEVERITY: *string*
: The string "ERROR", "WARN", or "INFO", to indicate the severity of this event.

TA_EVENT_LMID: *string*
: A string identifying the machine where the event was detected.

TA_EVENT_TIME: *long*
: A long integer containing the event detection time, in seconds, according to the clock on the machine where detection took place.

TA_EVENT_USEC: *long*
: A long integer containing the event detection time, in microseconds, according to the clock on the machine where detection took place. While the units of this value will always be microseconds, the actual resolution depends on the underlying operating system and hardware.

TA_EVENT_DESCRIPTION: *string*
: A one-line string summarizing the event.

TA_CLASS: *string*
The class of the object associated with the event. Depending on TA_CLASS, the event notification buffer will contain additional fields specific to an object of this class.

TA_ULOGCAT: *string*
Catalog name from which the message was derived, if any.

TA_ULOGMSGNUM: *num*
Catalog message number, if the message was derived from a catalog.

Event Lists

T_ACLPERM Event List

```
.SysAclPerm
        INFO: .SysACLPerm: system ACL permission change
```

T_DOMAIN Event List

```
.SysResourceConfig
        INFO: .SysResourceConfig: system configuration change
```

T_GROUP Event List

```
.SysGroupState
        INFO: .SysGroupState: system configuration change
```

T_MACHINE Event List

```
.SysMachineBroadcast
        WARN: .SysMachineBroadcast: %TA_LMID broadcast delivery
        failure

.SysMachineConfig
        INFO: .SysMachineConfig: %TA_LMID configuration change

.SysMachineFullMaxaccessers
        WARN: .SysMachineFullMaxaccessers: %TA_LMID capacity limit

.SysMachineFullMaxconv
        WARN: .SysMachineFullMaxconv: %TA_LMID capacity limit

.SysMachineFullMaxgtt
        WARN: .SysMachineFullMaxgtt: %TA_LMID capacity limit

.SysMachineFullMaxwsclients
        WARN: .SysMachineFullMaxwsclients: %TA_LMID capacity limit
```

```
.SysMachineMsgq
        WARN: .SysMachineMsgq: %TA_LMID message queue blocking

.SysMachinePartitioned
        ERROR: .SysMachinePartitioned: %TA_LMID is partitioned

.SysMachineSlow
        WARN: .SysMachineSlow: %TA_LMID slow responding to DBBL

.SysMachineState
        INFO: .SysMachineState: %TA_LMID state change to %TA_STATE

.SysMachineUnpartitioned
        ERROR: .SysMachinePartitioned: %TA_LMID is unpartitioned
```

T_BRIDGE Event List

```
.SysNetworkConfig
        INFO: .SysNetworkConfig: %TA_LMID[0]->%TA_LMID[1]
        configuration change

.SysNetworkDropped
        ERROR: .SysNetworkDropped: %TA_LMID[0]->%TA_LMID[1]
        connection dropped

.SysNetworkFailure
        ERROR: .SysNetworkFailure: %TA_LMID[0]->%TA_LMID[1]
        connection failure

.SysNetworkFlow
        WARN: .SysNetworkFlow: %TA_LMID[0]->%TA_LMID[1] flow control

.SysNetworkState
        INFO: .SysNetworkState: %TA_LMID[0]->%TA_LMID[1] state
        change to %TA_STATE
```

T_SERVER Event List

```
.SysServerCleaning
        ERROR: .SysServerCleaning: %TA_SERVERNAME, group %TA_SRVGRP,
        id %TA_SRVID server cleaning

.SysServerConfig
        INFO: .SysServerConfig: %TA_SERVERNAME, group %TA_SRVGRP, id
        %TA_SRVID configuration change
```

```
.SysServerDied
       ERROR: .SysServerDied: %TA_SERVERNAME, group %TA_SRVGRP, id
       %TA_SRVID server died

.SysServerInit
       ERROR: .SysServerInit: %TA_SERVERNAME, group %TA_SRVGRP, id
       %TA_SRVID server initialization failure

.SysServerMaxgen
       ERROR: .SysServerMaxgen: %TA_SERVERNAME, group %TA_SRVGRP,
       id %TA_SRVID server exceeded MAXGEN restart limit

.SysServerRestarting
       ERROR: .SysServerRestarting: %TA_SERVERNAME, group
       %TA_SRVGRP, id %TA_SRVID server restarting

.SysServerState
       INFO: .SysServerState: %TA_SERVERNAME, group %TA_SRVGRP, id
       %TA_SRVID state change to %TA_STATE

.SysServerTpexit
       ERROR: .SysServerTpexit: %TA_SERVERNAME, group %TA_SRVGRP,
       id %TA_SRVID server requested TPEXIT
```

T_SERVICE Event List

```
.SysServiceTimeout
       ERROR: .SysServiceTimeout: %TA_SERVERNAME, group %TA_SRVGRP,
       id %TA_SRVID server killed due to a service time-out
```

T_CLIENT Event List

```
.SysClientConfig
       INFO: .SysClientConfig: User %TA_USRNAME on %TA_LMID
       configuration change

.SysClientDied
       WARN: .SysClientDied: User %TA_USRNAME on %TA_LMID client
       died

.SysClientSecurity
       WARN: .SysClientSecurity: User %TA_USRNAME on %TA_LMID
       authentication failure

.SysClientState
       INFO: .SysClientState: User %TA_USRNAME on %TA_LMID state
       change to %TA_STATE
```

`T_TRANSACTION` Event List

.SysTransactionHeuristicAbort
```
        ERROR: .SysTransactionHeuristicAbort: Transaction %TA_GTRID
        in group %TA_GRPNO
```

.SysTransactionHeuristicCommit
```
        ERROR: .SysTransactionHeuristicCommit: Transaction
        %TA_GTRID in group %TA_GRPNO
```

T_EVENT Event List

```
.SysEventDelivery
        ERROR: .SysEventDelivery: System Event Monitor delivery
        failure on %TA_LMID

.SysEventFailure
        ERROR: .SysEventFailure: System Event Monitor subsystem
        failure on %TA_LMID
```

Files ${TUXDIR}/udataobj/evt_mib

See Also MIB(5), TM_MIB(5)

EVENT_MIB(5)

Name

EVENT_MIB-BEA TUXEDO Event Broker Management Information Base

Synopsis

```
#include <tpadm.h>
#include <fml32.h>
#include <evt_mib.h>
```

Description

The BEA TUXEDO Event Broker MIB defines the set of classes through which the Event Broker can be managed.

EVENT_MIB(5) should be used in combination with the generic MIB reference page, MIB(5), to format administrative requests and interpret administrative replies. Requests formatted as described in MIB(5) and a component MIB reference page may be used to request an administrative service using any one of a number of existing ATMI interfaces in an active application.

EVENT_MIB consists of the following classes:

EVENT_MIB **Classes**

Class Name	Attributes
T_EVENT_CLIENT	Subscriptions that trigger unsolicited notification
T_EVENT_COMMAND	Subscriptions that trigger system commands
T_EVENT_QUEUE	Subscriptions for queue-based notification
T_EVENT_SERVICE	Subscriptions for server-based notification
T_EVENT_USERLOG	Subscriptions for writing userlog messages

Each object in these classes represents a single subscription request.

FML32 Field Tables

The field table for the attributes described in this reference page is found in the file udataobj/evt_mib (relative to the root directory of the BEA TUXEDO system software). The directory ${TUXDIR}/udataobj should be included by the application in the colon-separated list specified by the FLDTBLDIR32 environment variable and the field table name evt_mib should be included in the comma-separated list specified by the FIELDTBLS32 environment variable.

T_EVENT_CLIENT CLASS DEFINITION

Overview

The T_EVENT_CLIENT class represents a set of subscriptions registered with the Event Broker for client-based notification.

When an event is detected, it is compared to each T_EVENT_CLIENT object. If the event name matches the value in TA_EVENT_EXPR and the optional filter rule is true, then the event buffer is sent to the specified client's unsolicited message handling routine.

Attribute Table

T_EVENT_CLIENT Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_EVENT_EXPR(r)	string	R--R--R--	*string[1...255]*	N/A
TA_EVENT_FILTER(k)	string	R--R--R--	*string[1...255]*	none
TA_EVENT_FILTER_BINARY(k)	carray	R--R--R--	*carray[1...64000]*	none
TA_STATE(r)	string	R-xR-xR-x	GET:"{ACT}"	N/A
			SET:"{NEW|INV}"	N/A
TA_CLIENTID(r)	string	R--R--R--	*string[1...78]*	N/A
(k) - a key field for object retrieval (r) - the field is required when a new object is created				

Check MIB(5) for an explanation of Permissions

Attribute Semantics

TA_EVENT_EXPR: *string[1...255]*

Event pattern expression. This expression, in recomp(3) format, controls which event names match this subscription.

TA_EVENT_FILTER: *string[1...255]*

Event filter expression. This expression, if present, is evaluated with respect to the posted buffer's contents. It must evaluate to TRUE or this subscription is not matched.

TA_EVENT_FILTER_BINARY: *carray[1...64000]*

Event filter expression, in binary (carray) format. Same as TA_EVENT_FILTER, but may contain arbitrary binary data. Only one of TA_EVENT_FILTER or TA_EVENT_FILTER_BINARY may be specified.

TA_STATE:

GET: ACTive
: A GET operation will retrieve configuration information for the matching T_EVENT_CLIENT object(s).

SET: {NEW|INValid}
: A SET operation will update configuration information for the T_EVENT_CLIENT object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_EVENT_CLIENT object. Successful return leaves the object in the ACTive state.
INValid	Delete T_EVENT_CLIENT object. Successful return leaves the object in the INValid state.

TA_CLIENTID: *string[1...78]*
: Send an unsolicited notification message to this client when a matching event is detected.

T_EVENT_COMMAND CLASS DEFINITION

Overview

The T_EVENT_COMMAND class represents a set of subscriptions registered with the Event Broker that trigger execution of system commands. When an event is detected, it is compared to each T_EVENT_COMMAND object. If the event name matches the value in TA_EVENT_EXPR and the optional filter rule is true, then the event buffer is formatted and passed to the system's command interpreter.

Attribute Table

T_EVENT_COMMAND Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_EVENT_EXPR(r)	string	R--------	*string[1...255]*	N/A
TA_EVENT_FILTER(k)	string	R--------	*string[1...255]*	none
TA_EVENT_FILTER_BINARY(k)	carray	R--------	*carray[1...64000]*	none
TA_STATE(r)	string	R-x------	GET:"{ACT}"	N/A
			SET:"{NEW|INV}"	N/A
TA_COMMAND(r)	string	R--------	*string[1...255]*	N/A
(k) - a key field for object retrieval (r) - the field is required when a new object is created				

Check MIB(5) for an explanation of Permissions

Attribute Semantics

TA_EVENT_EXPR: *string[1...255]*

Event pattern expression. This expression, in recomp(3) format, controls which event names match this subscription.

TA_EVENT_FILTER: *string[1...255]*

Event filter expression. This expression, if present, is evaluated with respect to the posted buffer's contents. It must evaluate to TRUE or this subscription is not matched.

TA_EVENT_FILTER_BINARY: *carray[1...64000]*

Event filter expression, in binary (carray) format. Same as TA_EVENT_FILTER, but may contain arbitrary binary data. Only one of TA_EVENT_FILTER or TA_EVENT_FILTER_BINARY may be specified.

TA_STATE:

GET: ACTive
: A GET operation will retrieve configuration information for the matching T_EVENT_COMMAND object(s).

SET: {NEW|INValid}
: A SET operation will update configuration information for the T_EVENT_COMMAND object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_EVENT_COMMAND object. Successful return leaves the object in the ACTive state.
INValid	Delete T_EVENT_COMMAND object. Successful return leaves the object in the INValid state.

TA_COMMAND: *string[1...255]*
: Execute this system command when an event matching this object is detected. For UNIX System platforms, the command is executed in the background using system(3).

T_EVENT_QUEUE CLASS DEFINITION

Overview

The T_EVENT_QUEUE class represents a set of subscriptions registered with the Event Broker for queue-based notification. When an event is detected, it is compared to each T_EVENT_QUEUE object. If the event name matches the value in TA_EVENT_EXPR and the optional filter rule is true, then the event buffer is stored in the specified reliable queue.

Attribute Table

T_EVENT_QUEUE Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_EVENT_EXPR(r)	string	R-x------	*string[1...255]*	N/A
TA_EVENT_FILTER(k)	string	R-x------	*string[1...255]*	none
TA_EVENT_FILTER_BINARY(k)	carray	R-x------	*carray[1...64000]*	none
TA_STATE(r)	string	R-x------	GET:"{ACT}"	N/A
			SET:"{NEW|INV}"	N/A
TA_QSPACE(r)	string	R-x------	*string[1...15]*	N/A
TA_QNAME(r)	string	R-x------	*string[1...15]*	N/A
TA_QCTL_QTOP	short	R-x------	*short*	0
TA_QCTL_BEFOREMSGID	short	R-x------	*short*	0
TA_QCTL_QTIME_ABS	short	R-x------	*short*	0
TA_QCTL_QTIME_REL	short	R-x------	*short*	0
TA_QCTL_DEQ_TIME	short	R-x------	*short*	0
TA_QCTL_PRIORITY	short	R-x------	*short*	0
TA_QCTL_MSGID	string	R-x------	*string[1...31]*	none
TA_QCTL_CORRID(k)	string	R-x------	*string[1...31]*	none
TA_QCTL_REPLYQUEUE	string	R-x------	*string[1...15]*	none
TA_QCTL_FAILUREQUEUE	string	R-x------	*string[1...15]*	none
TA_EVENT_PERSIST	short	R-x------	*short*	0
TA_EVENT_TRAN	short	R-x------	*short*	0

(k) - a key field for object retrieval
(r) - the field is required when a new object is created

Check MIB(5) for an explanation of Permissions

Attribute Semantics

TA_EVENT_EXPR: *string[1...255]*

Event pattern expression. This expression, in recomp(3) format, controls which event names match this subscription.

TA_EVENT_FILTER: *string[1...255]*

Event filter expression. This expression, if present, is evaluated with respect to the posted buffer's contents. It must evaluate to TRUE or this subscription is not matched.

TA_EVENT_FILTER_BINARY: *carray[1...64000]*

Event filter expression, in binary (carray) format. Same as TA_EVENT_FILTER, but may contain arbitrary binary data. Only one of TA_EVENT_FILTER or TA_EVENT_FILTER_BINARY may be specified.

TA_STATE:

GET: ACTive

A GET operation will retrieve configuration information for the matching T_EVENT_QUEUE object(s).

SET: {NEW|INValid}

A SET operation will update configuration information for the T_EVENT_QUEUE object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_EVENT_QUEUE object. Successful return leaves the object in the ACTive state.
INValid	Delete T_EVENT_QUEUE object. Successful return leaves the object in the INValid state.

TA_QSPACE: *string[1...15]*

Enqueue a notification message to a reliable queue in this queue space when a matching event is detected.

TA_QNAME: *string[1...15]*

Enqueue a notification message to this reliable queue when a matching event is detected.

TA_QCTL_QTOP: *short*
: This value, if present, is passed in to tpenqueue(3)'s TPQCTL control structure to request notification via the /Q subsystem with the message to be placed at the top of the queue.

TA_QCTL_BEFOREMSGID: *short*
: This value, if present, is passed in to tpenqueue(3)'s TPQCTL control structure to request notification via the /Q subsystem with the message to be placed on the queue ahead of the specified message.

TA_QCTL_QTIME_ABS: *short*
: This value, if present, is passed in to tpenqueue(3)'s TPQCTL control structure to request notification via the /Q subsystem with the message to be processed at the specified time.

TA_QCTL_QTIME_REL: *short*
: This value, if present, is passed in to tpenqueue(3)'s TPQCTL control structure to request notification via the /Q subsystem with the message to be processed relative to the dequeue time.

TA_QCTL_DEQ_TIME: *short*
: This value, if present, is passed in to tpenqueue(3)'s TPQCTL control structure.

TA_QCTL_PRIORITY: *short*
: This value, if present, is passed in to tpenqueue(3)'s TPQCTL control structure.

TA_QCTL_MSGID: *string[1...31]*
: This value, if present, is passed in to tpenqueue(3)'s TPQCTL structure.

TA_QCTL_CORRID: *string[1...31]*
: This value, if present, is passed in to tpenqueue(3)'s TPQCTL control structure.

TA_QCTL_REPLYQUEUE: *string[1...15]*
: This value, if present, is passed in to tpenqueue(3)'s TPQCTL control structure.

TA_QCTL_FAILUREQUEUE: *string[1...15]*
: This value, if present, is passed in to tpenqueue(3)'s TPQCTL control structure.

TA_EVENT_PERSIST: *short*

If non-zero, do not cancel this subscription if the designated queue is no longer available.

TA_EVENT_TRAN: *short*

If non-zero and the client's tppost(3) call is transactional, include the tpenqueue(3) call in the client's transaction.

T_EVENT_SERVICE CLASS DEFINITION

Overview

The T_EVENT_SERVICE class represents a set of subscriptions registered with the Event Broker for service-based notification. When an event is detected, it is compared to each T_EVENT_SERVICE object. If the event name matches the value in TA_EVENT_EXPR and the optional filter rule is true, then the event buffer is sent to the specified BEA TUXEDO service routine.

Attribute Table

T_EVENT_SERVICE Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_EVENT_EXPR(r)	string	R--R--R--	*string[1...255]*	N/A
TA_EVENT_FILTER(k)	string	R--R--R--	*string[1...255]*	none
TA_EVENT_FILTER_BINARY(k)	carray	R--R--R--	*carray[1...64000]*	none
TA_STATE(r)	string	R-xR-xR-x	GET:"{ACT}"	N/A
			SET:"{NEW|INV}"	N/A
TA_SERVICENAME(r)	string	R--R--R--	*string[1...15]*	N/A
TA_EVENT_PERSIST	short	R-xR-xR-x	*short*	0
TA_EVENT_TRAN	short	R-xR-xR-x	*short*	0
(k) - a key field for object retrieval (r) - the field is required when a new object is created				

Check MIB(5) for an explanation of Permissions

Attribute Semantics

TA_EVENT_EXPR: *string[1...255]*

Event pattern expression. This expression, in recomp(3) format, controls which event names match this subscription.

TA_EVENT_FILTER: *string[1...255]*

Event filter expression. This expression, if present, is evaluated with respect to the posted buffer's contents. It must evaluate to TRUE or this subscription is not matched.

TA_EVENT_FILTER_BINARY: *carray[1...64000]*

Event filter expression, in binary (carray) format. Same as TA_EVENT_FILTER, but may contain arbitrary binary data. Only one of TA_EVENT_FILTER or TA_EVENT_FILTER_BINARY may be specified.

TA_STATE:

GET: ACTive
: A GET operation will retrieve configuration information for the matching T_EVENT_SERVICE object(s).

SET: {NEW|INValid}
: A SET operation will update configuration information for the T_EVENT_SERVICE object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_EVENT_SERVICE object. Successful return leaves the object in the ACTive state.
INValid	Delete T_EVENT_SERVICE object. Successful return leaves the object in the INValid state.

TA_SERVICENAME: *string[1...15]*
: Call this BEA TUXEDO service when a matching event is detected.

TA_EVENT_PERSIST: *short*
: If non-zero, do not cancel this subscription if the TA_SERVICENAME service is no longer available.

TA_EVENT_TRAN: *short*
: If non-zero and the client's tppost(3) call is transactional, include the TA_SERVICENAME service call in the client's transaction.

T_EVENT_USERLOG CLASS DEFINITION

Overview

The T_EVENT_USERLOG class represents a set of subscriptions registered with the Event Broker for writing system userlog(3) messages. When an event is detected, it is compared to each T_EVENT_USERLOG object. If the event name matches the value in TA_EVENT_EXPR and the optional filter rule is true, then the event buffer is formatted and passed to BEA TUXEDO's userlog(3) function.

Attribute Table

T_EVENT_USERLOG Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_EVENT_EXPR(r)	string	R--R-----	*string[1...255]*	N/A
TA_EVENT_FILTER(k)	string	R--R-----	*string[1...255]*	none
TA_EVENT_FILTER_BINARY(k)	carray	R--R-----	*carray[1...64000]*	none
TA_STATE(r)	string	R-xR-x---	GET:"{ACT}"	N/A
			SET:"{NEW|INV}"	N/A
TA_USERLOG(r)	string	R--R-----	*string[1...255]*	N/A
(k) - a key field for object retrieval (r) - the field is required when a new object is created				

Check MIB(5) for an explanation of Permissions

Attribute Semantics

TA_EVENT_EXPR: *string[1...255]*

Event pattern expression. This expression, in recomp(3) format, controls which event names match this subscription.

TA_EVENT_FILTER: *string[1...255]*

Event filter expression. This expression, if present, is evaluated with respect to the posted buffer's contents. It must evaluate to TRUE or this subscription is not matched.

TA_EVENT_FILTER_BINARY: *carray[1...64000]*

Event filter expression, in binary (carray) format. Same as TA_EVENT_FILTER, but may contain arbitrary binary data. Only one of TA_EVENT_FILTER or TA_EVENT_FILTER_BINARY may be specified.

TA_STATE:

GET: ACTive
: A GET operation will retrieve configuration information for the matching T_EVENT_USERLOG object(s).

SET: {NEW|INValid}
: A SET operation will update configuration information for the T_EVENT_USERLOG object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_EVENT_USERLOG object. Successful return leaves the object in the ACTive state.
INValid	Delete T_EVENT_USERLOG object. Successful return leaves the object in the INValid state.

TA_USERLOG: *string[1...255]*
: Write a userlog(3) message when a matching event is detected.

Files ${TUXDIR}/udataobj/evt_mib ${TUXDIR}/include/evt_mib.h

See Also EVENTS(5), TM_MIB(5)

Ferror (5)

Name `Ferror, Ferror32`-BEA TUXEDO system FML error codes

Synopsis
```
#include “fml.h”
#include “fml32.h”
```

Description The numerical value represented by the symbolic name of an error condition is assigned to `Ferror` for errors that occur when executing many FML library routines.

The name `Ferror` expands to a modifiable *`lvalue`* that has type `int`, the value of which is set to a positive error number by several FML library routines. `Ferror` need not be the identifier of an object; it might expand to a modifiable *`lvalue`* resulting from a function call. It is unspecified whether `Ferror` is a macro or an identifier declared with external linkage. If a `tperrno` macro definition is suppressed to access an actual object, or if a program defines an identifier with the name `Ferror`, the behavior is undefined.

The reference pages for FML routines list possible error conditions for each routine and the meaning of the error in that context. The order in which possible errors are listed is not significant and does not imply precedence. The value of `Ferror` should be checked only after an error has been indicated; that is, when the return value of the component indicates an error and the component definition specifies that `tperrno` be set. An application that checks the value of `Ferror` must include the `fml.h` header file.

`Ferror32` provides a similar capability for users of fml32 routines. An application that checks the value of `Ferror32` must include the `fml32.h` header file.

```
The following list shows error codes that may be returned by FML
and FML32 routines.
#define FMINVAL 0   /* bottom of error message codes */
#define FALIGNERR 1  /* fielded buffer not aligned */
#define FNOTFLD 2   /* buffer not fielded */
#define FNOSPACE 3   /* no space in fielded buffer */
#define FNOTPRES 4   /* field not present */
#define FBADFLD 5   /* unknown field number or type */
#define FTYPERR 6   /* illegal field type */
#define FEUNIX 7    /* unix system call error */
#define FBADNAME 8   /* unknown field name */
#define FMALLOC 9   /* malloc failed */
#define FSYNTAX 10   /* bad syntax in boolean expression */
#define FFTOPEN 11   /* cannot find or open field table */
#define FFTSYNTAX 12  /* syntax error in field table */
#define FEINVAL 13   /* invalid argument to function */
#define FBADTBL 14  /* destructive concurrent access to field table
*/
```

```
#define FBADVIEW 15  /* cannot find or get view */
#define FVFSYNTAX 16  /* bad viewfile */
#define FVFOPEN 17   /* cannot find or open viewfile */
#define FBADACM 18   /* ACM contains negative value */
#define FNOCNAME 19   /* cname not found */
```

Usage

Some routines do not have an error return value. Because no routine sets `Ferror` to zero, an application can set `Ferror` to zero, call a routine and then check `Ferror` again to see if an error has occurred.

In DOS and OS/2 environments, this variable is known as `FMLerror`.

See Also

See the `ERRORS` section of the individual FML library routines for a more detailed description of the meaning of the error codes returned by each routine.

`Fintro`(3fml), `intro`(3c), `tpstrerror`(3c), `tperrordetail`(3c), `tpstrerrordetail`(3c), `F_error`(3fml)

field_tables(5)

Name `field_tables`-FML mapping files for field names

Description The Field Manipulation Language functions implement and manage fielded buffers. Each field in a fielded buffer is tagged with an identifying integer. Fields that can variable in length (for example, a string) have an additional length modifier. The buffer then consists of a series of numeric-identifier/data pairs and numeric-identifier/length/data triples.

The numeric-identifier of a field is called its "field identifier" (fldid), and is typedef'd by `FLDID`. A field is named by relating an alphanumeric string (the name) to a `FLDID` in a field table.

The original FML interface supports 16-bit field identifiers, field lengths, and buffer sizes. A newer 32-bit interface, FML32, supports larger identifiers, field lengths, and buffer sizes. All types, function names, etc. are suffixed with "32" (for example, the field identifier type definition is `FLDID32`).

Field Identifiers FML functions allow field values to be typed. Currently supported types include char, string, short, long, float, double, and character array. Constants for field types are defined in `fml.h` (fml32.h for FML32). So that fielded buffers can be truly self-describing, the type of a field is carried along with the field by encoding the field type in the FLDID. Thus, a FLDID is composed of two elements: a field type, and a field number. Field numbers must be above 100; the numbers 1-100 are reserved for system use.

Field Mapping For efficiency, it is desirable that the field name to field identifier mapping be available at compile time. For utility, it is also desirable that these mappings be available at run time. To accommodate both these goals, FML represents field tables in text files, and provides commands to generate corresponding C header files. Thus, compile time mapping is done by the C preprocessor, `cpp`, by the usual `#define` macro. Run-time mapping is done by the function `Fldid(\|)` (Fldid32(\|) for FML32), which maps its argument, a field name, to a field identifier by consulting the source field table files.

Field Table Files Files containing field tables have the following format:

- blank lines and lines beginning with # are ignored.
- lines beginning with $ are ignored by the mapping functions but are passed through (without the $) to header files generated by mkfldhdr(1) (the command name is `mkfldhdr32` for FML32). For example, this would allow the application to pass C comments, `what` strings, etc. to the generated header file.

- lines beginning with the string `*base` contain a base for offsetting subsequent field numbers. This optional feature provides an easy way to group and renumber sets of related fields.
- lines that don't begin with either * nor # should have the form:

  ```
  name    rel-numb   type
  ```

  where:

  - name is the identifier for the field. It should not exceed `cpp` restrictions.
  - `rel-numb` is the relative numeric value of the field. It is added to the current base to obtain the field number of the field.
  - `type` is the type of the field, and is specified as one of: char, string, short, long, float, double, carray.

Entries are white-space separated (any combination of tabs and spaces).

Conversion of Field Tables to Header Files

The command `mkfldhdr` (or `mkfldhdr32`) converts a field table, as described above, into a file suitable for processing by the C compiler. Each line of the generated header file is of the form:

```
#define  name  fldid
```

where `name` is the name of the field, and `fldid` is its field identifier. The field identifier includes the field type and field number, as previously discussed. The field number is an absolute number, that is, base + rel-number. The resulting file is suitable for inclusion in a C program.

Environment Variables

Functions such as `Fldid`(), which access field tables, and commands such as `mkfldhdr`(1) and `vuform`(1), which use them, both need the shell variables `FLDTBLDIR` and `FIELDTBLS` (`FLDTBLDIR32` and `FIELDTBLS32` for FML32) to specify the source directories and files, respectively, from which the in-memory version of field tables should be created. `FIELDTBLS` specifies a comma-separated list of field table file names. If `FIELDTBLS` has no value, `fld.tbl` is used as the name of the field table file. The `FLDTBLDIR` environment variable is a colon-separated list of \%directories in which to look for each field table whose name is not an absolute path name. (The search for field tables is very similar to the search for executable commands using the `PATH` variable) If `FLDTBLDIR` is not defined, it is taken to be the current directory. Thus, if `FIELDTBLS` and `FLDTBLDIR` are not set, the default is to take `fld.tbl` from the current directory.

The use of multiple field tables is a convenient way to separate groups of fields, such as groups of fields that exist in a database from those which are used only by the application. However, in general field names should be unique across all field tables, since such tables are capable of being converted to C header files (by the `mkfldhdr` command), and identical field names would produce a compiler name conflict warning. In addition, the function `Fldid`, which maps a name to a `FLDID`, does so by searching the multiple tables, and stops upon finding the first successful match.

Example The following is a sample field table in which the base shifts from 500 to 700:

```
# employee ID fields are based at 500
*base 500

#name  rel-numb  type  comment
#----  --------  ----  -------
EMPNAM  1     string emp's name
EMPID   2     long  emp's id
EMPJOB  3     char  job type: D,M,F or T
SRVCDAY  4     carray service date

# address fields are based at 700

*base 700

EMPADDR  1     string street address
EMPCITY  2     string city
EMPSTATE 3     string state
EMPZIP  4     long  zip code
```

The associated header file would be

```
#define EMPADDR  ((FLDID)41661) /* number: 701 type: string */
#define EMPCITY  ((FLDID)41662) /* number: 702 type: string */
#define EMPID   ((FLDID)8694)  /* number: 502 type: long */
#define EMPJOB  ((FLDID)16887) /* number: 503 type: char */
#define EMPNAM  ((FLDID)41461) /* number: 501 type: string */
#define EMPSTATE ((FLDID)41663) /* number: 703 type: string */
#define EMPZIP  ((FLDID)8896)  /* number: 704 type: long */
#define SRVCDAY  ((FLDID)49656) /* number: 504 type: carray */
```

See Also `mkfldhdr`(1), *TUXEDO FML Guide*

GWADM(5)

Name GWADM-Domains gateway administrative server

Synopsis

```
GWADM SRVGRP = "identifier" SRVID = "number" REPLYQ = "N"
 CLOPT = "-A -- [-a { on | off } ] [-t { on | off } ]"
```

Description The gateway administrative server (GWADM) is a BEA TUXEDO system-supplied server that provides administrative functions for a Domains gateway group.

GWADM should be defined in the SERVERS section of the UBBCONFIG file as a server running within a particular gateway group, that is, SRVGRP must be set to the corresponding GRPNAME tag specified in the GROUPS section. The SVRID parameter is also required and its value must consider the maximum number of gateways allowed within the gateway group.

There should be only one instance of a GWADM per Domains gateway group, and it should *NOT* be part of the MSSQ defined for the gateways associated with the group. Also, GWADM should have the REPLYQ attribute set to N.

The CLOPT option is a string of command line options that is passed to the GWADM when it is booted. This string has the following format:

```
CLOPT="-A -- gateway group run-time parameters>"
```

The following run-time parameters are recognized for a gateway group:

-a { *on* | *off* }
: This option turns off or on the audit log feature for this local domain. The default is off. The dmadmin program can be used to change this setting while the gateway group is running (see dmadmin(1)).

-t { *on* | *off* }
: This option turns off or on the statistics gathering feature for the local domain. The default is off. The dmadmin program can be used to change this setting while the gateway group is running (see dmadmin(1)).

The GWADM server must be booted before the corresponding gateways.

Portability GWADM is supported a BEA TUXEDO system-supplied server on UNIX System operating systems.

Interoperability GWADM must be installed on a BEA TUXEDO system Release 4.2.1 or later; other machines in the same domain with an R4.2.2 gateway can be Release 4.1 or later.

Examples

The following example illustrates the definition of the administrative server in the UBBCONFIG file.

```
#
*GROUPS
DMADMGRP GRPNO=1
gwgrp  GRPNO=2
#
*SERVERS
DMADM SRVGRP="DMADMGRP" SRVID=1001 REPLYQ=N RESTART=Y GRACE=0
GWADM SRVGRP="gwgrp" SRVID=1002 REPLYQ=N RESTART=Y GRACE=0
  CLOPT="-A -- -a on -t on"
GWTDOMAIN SRVGRP="gwgrp" SRVID=1003 RQADDR="gwgrp" REPLYQ=Y
RESTART=Y MIN=1 MAX=1
```

See Also

dmadmin(1), tmboot(1), dmconfig(5), DMADM(5), servopts(5), ubbconfig(5), *BEA TUXEDO Domains Guide*, *BEA TUXEDO Administrator's Guide*

GWTDOMAIN(5)

Name BEA TUXEDO system domain gateway process

Synopsis *GWTDOMAIN* SRVGRP = "*identifier*" SRVID = "*number*" RQADDR = "*queue_name*" REPLYQ = N RESTART = Y [MAXGEN = *value*] [GRACE = *value*]

Description GWTDOMAIN is the domain gateway process that provides interdomain communication. GWTDOMAIN processes comunicate with other GWTDOMAIN processes in remote domains.

Domain gateways are described in the SERVERS section of the UBBCONFIG file and the BDMCONFIG file. Domain gateways must be always associated with a particular group, that is, SRVGRP must be set to the corresponding GRPNAME tag specified in the GROUPS section. The SVRID parameter is also required and its value must consider the maximum number of gateways allowed within the domain group. The RESTART parameter should be set to Y. The REPLYQ parameter should be set to N.

The GWTDOMAIN process must be in the same group as the GWADM(5) process, with the GWADM listed first. Multiple GWTDOMAIN processes can be configured for a domain; each must be configured in a different BEA TUXEDO group.

Examples The following example shows the definition of a Domains gateway group in the UBBCONFIG file.

```
*GROUPS
DMADMGRP LMID=mach1 GRPNO=1
gwgrp  LMID=mach1 GRPNO=2
*SERVERS
DMADM SRVGRP="DMADMGRP" SRVID=1001 REPLYQ=N RESTART=Y MAXGEN=5
GRACE=3600
GWADM SRVGRP="gwgrp" SRVID=1002 REPLYQ=N RESTART=Y MAXGEN=5
GRACE=3600
GWTDOMAIN SRVGRP="gwgrp" SRVID=1003 RQADDR="gwgrp" REPLYQ=N
RESTART=Y MAXGEN=5 GRACE=3600
```

See also the EXAMPLES section of ubbconfig(5) and dmconfig(5).

See Also DMADM(5), dmconfig(5), GWADM(5), servopts(5), tmadmin(1), tmboot(1), ubbconfig(5), *BEA TUXEDO Domains Guide*, *Administering the BEA TUXEDO System*

GWTUX2TE (5)

Name

GWTUX2TE, GWTE2TUX - BEA TUXEDO / BEA TOP END gateway servers

Synopsis

```
GWTUX2TE SRVGRP = "identifier" SRVID = "number"
       CLOPT = "-- -f service_definition_file
       [-c TOPEND_remote_configuration_file]
       [-R sec] [-w wait_time] [[-u username] [-p password_file]]"

GWTE2TUX SRVGRP = "identifier" SRVID = "number"
       CLOPT = "-- -f service_definition_file
       [-c TOPEND_remote_configuration_file]
       [-R sec] [[-u username] [-g groupname]]"
```

Description

GWTUX2TE and GWTE2TUX are gateway servers. GWTUX2TE provides connectivity between BEA TUXEDO clients and BEA TOP END servers. GWTE2TUX provides connectivity between BEA TOP END clients and BEA TUXEDO servers. One or both of these gateway servers may be configured for a domain.

GWTUX2TE and GWTE2TUX are defined in the SERVERS section of the UBBCONFIG file as servers running within a particular server group. Therefore, SRVGRP must be set to the value of the corresponding GRPNAME parameter (as specified in the GROUPS section). The SVRID parameter is also required. GWTUX2TE and GWTE2TUX allow for MIN and MAX values of gateway instances to be specified. Although the gateway servers are synchronous, you may use multiple instances to provide better throughput.

CLOPT is an "umbrella parameter" that passes a set of command-line options to the gateway servers when the servers are booted. To specify options with CLOPT, use the following format.

```
CLOPT="-- gateway_group_runtime_parameters"
```

The following CLOPT options are recognized.

`-f service_definition_file`
: This file lists the services and functions to be advertised by the gateway server. (The file format is described in "Configuration" later in this reference page.) If -f is not specified or if the specified file has an invalid syntax, the gateway server logs an error and exits.

`-c TOP_END_remote_configuration_file`
: This file defines the connectivity between the gateway servers and the BEA TOP END system. If this option is not specified, $APPDIR/TOPENDRC.cfg is used, by default, as the configuration file. If there is no configuration file or if the file specified has an invalid syntax, the gateway server logs an error and exits.

`-u` *`username`* `-p` *`password_file`*

If security is enabled on the BEA TOP END system, then the `-u` and `-p` options should be specified for the `GWTUX2TE` gateway.

After the `-p` option, specify the file that contains the password associated with the user specified by `–u`. The password file must be in ASCII format; the password must be provided on a single line. To ensure security, the file should be read and write protected; only the BEA TUXEDO administrator should be granted access.

See "Security" for additional information.

`-R` *`Retry_interval`*

If the gateway server is unable to establish a connection with the BEA TOP END system, or if an existing connection is broken, the server will, by default, retry to establish a connection every 60 seconds. This time interval (in seconds) may be set to a different value using `–R`. Setting `–R` to 0 turns off retry. If you do so and specify `RESTART=Y` then, when a connection cannot be established or is broken, the gateway server exits and restarts.

If a connection to the BEA TOP END system is not available, the services on that system are not offered by the gateway server.

`-w` *`wait_time`*

When the `GWTUX2TE` gateway server sends a request to the BEA TOP END system, it waits for a response, by default, for 30 seconds. The `–w` parameter allows the waiting time to be specified. Specifying a waiting time of 0 indicates infinite waiting time.

The `GWTE2TUX` gateway server does not provide a configurable waiting time, so normal timeout parameters may be configured in `tuxconfig`.

`-u` *`username`* `–g` *`groupname`*

If access control lists are being used for BEA TUXEDO services, then both the `-u` and `-g` options should be specified for a `GWTE2TUX` gateway server. By default, the gateway uses guest privileges.

See "Security" for additional information.

Programming Paradigms

The `GWTUX2TE` and `GWTE2TUX` gateway servers support request/response messages only. The following BEA TUXEDO client API calls for sending and receiving are allowed:

- `tpcall`

- tpacall (with or without TPNOREPLY flag)
- tpgetrply
- tpforward

BEA TOP END servers cannot set the APPL_CONTEXT flag. If this flag is set, the gateway server dissolves the BEA TOP END dialog and returns an error (TPESVCFAIL) to the BEA TUXEDO client.

The following BEA TOP END client API calls are allowed:

- tp_client_send
- tp_client_receive

Buffer Types

The GWTUX2TE and GWTE2TUX gateway servers support BEA TUXEDO CARRAY (X_OCTET) buffers only. Attempts to send other types of buffers from a BEA TUXEDO application generate an error, which is logged by the gateway server.

Configuration

The GWTUX2TE and GWTE2TUX gateway servers use the BEA TOP END remote client and remote server services. GWTUX2TE assumes the role of a BEA TOP END client and makes use of the remote client services. GWTE2TUX assumes the role of a BEA TOP END server and makes use of the remote server services. Therefore, you must provide a BEA TOP END remote client/server configuration file on any BEA TUXEDO node running these gateway processes.

BEA TOP END Remote Client/Server Configuration File

The BEA TOP END remote client/server configuration file is described in the *BEA TOP END Remote Client Services Guide*; this section provides a brief description of the file.

Entries in this configuration file are formatted as follows.

```
[top end configuration file]
[component type] remote server
[system] sysname
[primary node] machine_name        portnum
```

The component type entry should be set to remote server. The system entry should match the name of the BEA TOP END system. The primary node entry should be set to the machine name and port number of the BEA TOP END Network Agent (NA).

A secondary node may also be specified. This node can be used when a connection to the primary node cannot be established. If multiple secondary nodes are specified, the BEA TOP END system uses a "round robin" technique to load balance the connections. This feature enables multiple instances of a gateway server to connect to different nodes on the BEA TOP END system, as shown here.

```
[secondary node] machine    28001
[secondary node] machine2   28001
```

The optional `target` parameter is also supported by the `GWTUX2TE` and `GWTE2TUX` gateway servers.

The following parameters are not supported by the `GWTUX2TE` and `GWTE2TUX` gateway servers; do not include them in the configuration file.

- `shutdown`
- `codeset`
- `maxconctx`

Each gateway process may connect to only one BEA TOP END system, as specified by `[system]` in the `TOPENDRC.cfg` file. A second gateway process may be configured to connect to a different BEA TOP END system. Use the `CLOPT -c` parameter to point to a second configuration file.

Service Definition File

The service definition file has the following syntax.

```
*TE_LOCAL_SERVICES # For TUXEDO services accessible by TOP END clients
Servicename PRODUCT=product_name FUNCTION=function_name
    QUALIFIER=function_qualifier

*TE_REMOTE_SERVICES # For TOP END services accessible by TUXEDO clients
Servicename PRODUCT=product_name FUNCTION=function_name
    QUALIFIER=function_qualifier TARGET=target_name
```

Servicename indicates the BEA TUXEDO service to be imported (`TE_REMOTE_SERVICE`) or exported (`TE_LOCAL_SERVICE`).

While the `PRODUCT` parameter must be specified, the `FUNCTION`, `QUALIFIER`, and `TARGET` parameters are optional. In addition, the `TARGET` parameter is valid for `TE_REMOTE_SERVICES` only.

You can define any service definition file parameter as a default by using the following syntax.

```
DEFAULT: PRODUCT=product_name
```

All services in the *TE_LOCAL_SERVICES section must have the same PRODUCT name.

If the FUNCTION parameter is not specified, the function name is assumed to be the service name. If the QUALIFIER and TARGET parameters are not specified for a service entry, no function qualifier or target name is used for that service.

Refer to *Administering the BEA TUXEDO System* for information on valid values for BEA TUXEDO service names. Refer to the *BEA TOP END Administrator's Guide* for information on valid values for the PRODUCT, FUNCTION, QUALIFIER, and TARGET parameters.

Limitations

The gateways do not support the following:

- Transactions
- Conversations
- Events
- Unsolicited notifications
- Queues (/Q, RTQ)
- Encryption
- Compression
- Message size above 30K
- Migration
- Formats
- MCC and LMA

Security

The following table lists the appropriate security settings for various configurations.

Gateway Server Security

For this Server	If. . .	Then. . .
GWTUX2TE	The BEA TOP END system is configured with authentication.	♦ Set the user name using the -u option. ♦ Set the password using the -p option. ♦ Protect this file using operating system protection.
GWTE2TUX	The BEA TUXEDO system is configured with SECURITY=APP_PW, USER_AUTH	No action is required.
GWTE2TUX	The BEA TUXEDO system is configured with SECURITY=ACL, MANDATORY ACL.	Set the user name using the -u option, and the group name, using the –g option.

The username and groupname or username and password specified with CLOPT must also be entered into the corresponding BEA TUXEDO or BEA TOP END security database. For the BEA TUXEDO security database, the user name is typically created using tpusradd. The group name is typically created using tpgrpadd.

Portability

The GWTUX2TE and GWTE2TUX gateway servers are supported on Windows NT, Sun Solaris, HP-UX, IBM AIX, and NCR MP-RAS.

Interoperability

The GWTUX2TE and GWTE2TUX gateway servers must run on BEA TUXEDO 6.5 or later. These gateway servers inter-operate with BEA TOP END 2.05 or later.

Examples

The following example shows how gateway servers are defined in the BEA TUXEDO UBBCONFIG file and in the BEA TOP END service definition file.

In this example, a BEA TUXEDO client issues tpcall to the RSERVICE service. The request is forwarded (via the GWTUX2TE gateway) to a BEA TOP END system (pluto) and invokes a BEA TOP END service (RPRODUCT:RFUNC).

Similarly, a BEA TOP END client issues `tp_client_send`, specifying `LPRODUCT` as the `PRODUCT` and `LFUNC` as the `FUNCTION`. The request is forwarded (via the `GWTE2TUX` gateway) to the BEA TUXEDO system and invokes a BEA TUXEDO service (`LSERVICE`).

Listing 0-2 BEA TUXEDO UBBCONFIG File

```
##############
#UBBCONFIG
*GROUPS
TOPENDGRP  GRPNO=1

#
*SERVERS
GWTE2TUX SRVGRP="TOPENDGRP" SRVID=1001 RESTART=Y MAXGEN=3 GRACE=10
       CLOPT="-- -f servicedefs -R 30"
GWTUX2TE SRVGRP="TOPENDGRP" SRVID=1002 RESTART=Y MAXGEN=3 GRACE=10
       MIN=5 MAX=5
       CLOPT="-- -f servicedefs"
```

Listing 0-3 BEA TOP END Service Definition File

```
############
#service definition file
*TE_LOCAL_SERVICES
DEFAULT: PRODUCT=LPRODUCT
LSERVICE FUNCTION=LFUNC

*TE_REMOTE_SERVICES
RSERVICE PRODUCT=RPRODUCT FUNCTION=RFUNC
```

Listing 0-4 BEA TOP END Remote Configuration File

```
# TOP END remote configuration file
[top end configuration file]
[component type] remote server
[system] pluto
[primary node] topendmach         28001
```

Note: Remember that the value of *port* in the primary node entry (which is 28001 in Listing 0-4) must match the port number of the BEA TOP END Network Agent.

Software Requirements

The following software components are required:

- BEA TUXEDO 6.5
- BEA TOP END 2.05

Failures

A BEA TUXEDO client receives a TPESVCFAIL under any of the following conditions:

- A BEA TOP END service is unreachable.
- A TOP END service returns an error.
- The network link to the BEA TOP END system is unavailable.
- A buffer type other than CARRAY or X_OCTET is sent by the BEA TUXEDO client.

A BEA TOP END client receives an error of TP_RESET, with the TP_EXT_SERVER_APPL extended status, under any of the following conditions:

- A BEA TUXEDO service is unreachable (for example, because it is suspended).
- A BEA TUXEDO service times out.
- A BEA TUXEDO service returns with TPFAIL or TPEXIT.

Note that if a gateway offers a service that is not available on the corresponding system, the client receives an error (TPESVCFAIL), as indicated above, that is different from the error returned after a local service invocation. In the latter case, the client receives TPENOENT for the BEA TUXEDO system or TP_SERVICE for the BEA TOP END system.

See Also

tmboot(1), servopts(5), ubbconfig(5)
Administering the BEA TUXEDO System
BEA TOP END Remote Client/Server Services Guide

langinfo(5)

Name `langinfo`-language information constants

Synopsis `#include <langinfo.h>`

Description This header file contains the constants used to identify items of `langinfo` data. The mode of *items* is given in `nl_types`(5).

`DAY_1`
Locale's equivalent of “sunday”

`DAY_2`
Locale's equivalent of ``monday''

`DAY_3`
Locale's equivalent of ``tuesday''

`DAY_4`
Locale's equivalent of ``wednesday''

`DAY_5`
Locale's equivalent of ``thursday''

`DAY_6`
Locale's equivalent of ``friday''

`DAY_7`
Locale's equivalent of ``saturday''

`ABDAY_1`
Locale's equivalent of ``sun''

`ABDAY_2`
Locale's equivalent of ``mon''

`ABDAY_3`
Locale's equivalent of ``tue''

`ABDAY_4`
Locale's equivalent of ``wed''

`ABDAY_5`
Locale's equivalent of ``thur''

`ABDAY_6`
: Locale's equivalent of ''fri"

`ABDAY_7`
: Locale's equivalent of ''sat"

`MON_1`
: Locale's equivalent of ''january"

`MON_2`
: Locale's equivalent of ''february"

`MON_3`
: Locale's equivalent of ''march"

`MON_4`
: Locale's equivalent of ''april"

`MON_5`
: Locale's equivalent of ''may"

`MON_6`
: Locale's equivalent of ''june"

`MON_7`
: Locale's equivalent of ''july"

`MON_8`
: Locale's equivalent of ''august"

`MON_9`
: Locale's equivalent of ''september"

`MON_10`
: Locale's equivalent of ''october"

`MON_11`
: Locale's equivalent of ''november"

`MON_12`
: Locale's equivalent of ''december"

`ABMON_1`
: Locale's equivalent of ''jan"

`ABMON_2`
: Locale's equivalent of ''feb"

`ABMON_3`
: Locale's equivalent of ''mar"

`ABMON_4`
: Locale's equivalent of ''apr"

`ABMON_5`
: Locale's equivalent of ''may"

`ABMON_6`
: Locale's equivalent of ''jun"

`ABMON_7`
: Locale's equivalent of ''jul"

`ABMON_8`
: Locale's equivalent of ''aug"

`ABMON_9`
: Locale's equivalent of ''sep"

`ABMON_10`
: Locale's equivalent of ''oct"

`ABMON_11`
: Locale's equivalent of ''nov"

`ABMON_12`
: Locale's equivalent of ''dec"

`RADIXCHAR`
: Locale's equivalent of ''."

`THOUSEP`
: Locale's equivalent of '',"

`YESSTR`
: Locale's equivalent of ''yes"

`NOSTR`
: Locale's equivalent of ''no"

`CRNCYSTR`
: Locale's currency symbol

`D_T_FMT`
: Locale's default format for date and time

`D_FMT`
: Locale's default format for the date

`T_FMT`
: Locale's default format for the time

`AM_STR`
: Locale's equivalent of ''AM"

`PM_STR`
: Locale's equivalent of ''PM"

This information is retrieved by `nl_langinfo`(3).

The items are retrieved from a special message catalog named `LANGINFO`, which should be generated for each locale supported and installed in the appropriate directory. (See `mklanginfo`(1))

See Also `mklanginfo`(1), `nl_langinfo`(3), `strftime`(3) `nl_types`(5).

MIB(5)

Name MIB-BEA BEA TUXEDO system Management Information Base

```
#include <fml32.h>
#include <fml1632.h> /* Optional */
#include <tpadm.h>
#include cmib.h>  /* Component MIB Header */
```

Description A BEA TUXEDO system application consists of distinct components (for example, BEA TUXEDO, Workstation), each administered using a Management Information Base (MIB) defined specifically for that component. These component MIBs are defined in individual reference pages each addressing the MIB for a particular part of the system. For example, the reference page TM_MIB(5) defines the MIB used to administer the fundamental aspects of a BEA TUXEDO application.

However, component MIBs do not provide sufficient definition of the interfaces involved to provide the necessary access. This reference page, MIB(5), describes the generic interfaces through which an administrator, operator or user interacts with any of the defined component MIBs. The generic interface to each BEA TUXEDO system MIB consists of two main parts.

The first part of the generic interface is a description of how existing BEA TUXEDO system interfaces are used to provide access to administrative services responsible for supporting the component MIBs. FML32, a BEA TUXEDO system buffer type, is used as the vehicle for passing input to and receiving output from component MIBs. ATMI request/response verbs are used as the interface to component MIBs, which are implemented as system supplied services. Details on interaction between an administrative user and component MIBs using FML32 buffers ATMI verbs are provided in the "FML32" and "ATMI" sections later in this reference page.

The second part of the generic interface is the definition of additional input and output FML32 fields that are used in interactions with all component MIBs. The additional FML32 fields extend the power of requests (for example, by allowing operation codes to be specified) and add generic response attributes (for example, error codes and explanatory text). Details on additional FML32 fields are provided in the "INPUT" and "OUTPUT" sections found later in this reference page.

The "USAGE" section gives examples of the use of existing ATMI verbs and the additional FML32 fields as they might be used for administrative interaction with component MIBs.

In addition to defining how users interface with component MIBs to administer an application, this reference page establishes the format used in the component MIB reference pages to define classes (see "`CLASS DESCRIPTIONS`").

Two generic classes are defined in this reference page: `T_CLASS` and `T_CLASSATT`. These two classes are used to identify administrative classes and to tune class/attribute permissions.

Finally, the "`DIAGNOSTICS`" section lists error codes that may be returned by component MIB system services.

Authentication

Users are authenticated as they attempt to join the application (see tpinit(3)). At `tpinit` time, administrators and operators can ask to join the application with a client name of either `tpsysadm` or `tpsysop`. These two `cltname` values are reserved and can only be associated with administrators and operators of the application.

The administrator who initially configures an application determines the level of security to be included by choosing a particular security type. Available security types are:

- no security
- application password authentication
- application password plus an application specific authentication service

The choice of security type determines the flexibility and security in allowing administrator and operator access to the component MIBs via the AdminAPI.

The most secure and flexible security type is an application password plus an application-specific authentication server (see `AUTHSVR`(5)). This method allows the administrator to permit access to any user or to only specified users provided they supply the appropriate password to the authentication server.

In the absence of an application specific authentication server, a client must satisfy the authentication requirements of the application (either none or application password), specify one of the special client names in the `cltname` field of the `TPINIT` structure and be running as the BEA TUXEDO administrator for the local UNIX System to qualify for special administrator or operator permissions. In any case, a successfully joined client is assigned a key by the system; the key is delivered with all requests it makes. Clients properly authenticated as either `tpsysadm` or `tpsysop` are assigned an authentication key that lets the system know they have special privileges.

Administrative authentication, as specified, is applicable only to clients that join the system prior to accessing the API. Servers making use of the API are treated the same as the client on whose behalf they are processing. Service requests made from within tpsvrinit(3) or tpsvrdone(3) are treated as coming from the administrator.

FML32 Application administration using BEA TUXEDO system defined component MIBs is supported exclusively through the FML32 buffer type. Application programs accessing MIB information must be written to allocate, manipulate and update FML32 typed buffers. There are two main approaches to using FML32 as detailed in Fintro(3) and summarized here.

The most direct way to interface to FML32 is to include the <fml32.h> header file instead of the standard <fml.h> header file and then to use the FML32 version of each relevant FML interface specified in the *TUXEDO Reference Manual*. For example, one would use Fchg32(3) instead of using Fchg(3).

Another method for interfacing with FML32 is to include both the <fml32.h> header file and the <fml1632.h> header file. These two header files work together to allow the user to program to the base FML interfaces (for example, Fchg(3)) and yet actually invoke the FML32 version of each interface.

ATMI Application programs access and update component MIB specific attribute information by allocating FML32 typed buffers, populating them with request data, sending the requests for servicing, receiving the replies to the service requests and extracting information regarding the results from the reply. The population and extraction of information to and from the FML32 typed buffers involves the FML32 interfaces as described above. Buffer allocation, sending requests and receiving replies is done using the general purpose ATMI routines listed below within the guidelines and restrictions listed. MIB requests for all components should be sent to the core BEA TUXEDO component MIB service, ".TMIB". This service not only acts as an agent for servicing TM_MIB(5) requests, it also directs requests targeted for other component MIBs so that the user need not be concerned with matching service names to MIBs and classes.

tpalloc
: Allocate FML32 typed buffers to be used in sending requests and/or receiving replies to/from BEA TUXEDO system MIB services. The FML32 buffer type has no subtypes and a minimum default size of 1024 bytes.

tprealloc
: Reallocate FML32 typed buffers.

`tpcall`

Call BEA TUXEDO system MIB service, "`.TMIB`", with a populated `FML32` typed buffer as input and with an allocated `FML32` typed buffer in which to store the output returned from the service. The buffer length for the input buffer may be specified as 0 since `FML32` is a self-describing buffer type. The `TPNOTRAN` flag should be used if the call is being made within a transaction; otherwise, there are no specific requirements or restrictions on the use of the flags defined for this verb.

`tpacall`

Asynchronously call BEA TUXEDO system MIB service, "`.TMIB`", with a populated `FML32` typed buffer as input. The buffer length for the input buffer may be specified as 0 since `FML32` is a self-describing buffer type. The `TPNOTRAN` flag should be used if the call is being made within a transaction; otherwise, there are no specific requirements or restrictions on the use of the flags defined for this verb.

`tpgetrply`

Get reply for a previously generated asynchronous call to the BEA TUXEDO system MIB service, "`.TMIB`". The reply is received into a previously allocated `FML32` typed buffer. There are no specific requirements or restrictions on the use of the flags defined for this verb.

`tpenqueue`

Enqueue a request to the BEA TUXEDO system MIB service, "`.TMIB`", for later processing. The buffer length for the input buffer may be specified as 0 since `FML32` is a self-describing buffer type. There are no specific requirements or restrictions on the use of the flags defined for this verb; however, the `TMQFORWARD`(5) server configured by the application to handle forwarding of these requests should be started with the `-n` (`tpcall`() with `TPNOTRAN` flag set) and `-d` (delete) options.

`tpdequeue`

Dequeue the reply for a previously enqueued request to the BEA TUXEDO system MIB service, "`.TMIB`". The reply is received into a previously allocated `FML32` typed buffer. There are no specific requirements or restrictions on the use of the flags defined for this verb.

Input

There are certain FML32 fields used to characterize and control administrative requests to any BEA TUXEDO system MIB. These fields are defined in this reference page as well as in the header file <tpadm.h>. The corresponding field table file can be found in ${TUXDIR}/udataobj/tpadm. These fields are added to an FML32 request buffer in addition to any component MIB specific fields necessary before making the administrative service request. The fields are described below and followed by a table summarizing the operations for which each field is required, optional or unused.

TA_OPERATION
: String valued field identifying the operation to be performed. Valid operations are GET, GETNEXT and SET.

TA_CLASS
: String valued field identifying the class being accessed. Class names are defined within component MIB specific reference pages.

TA_CURSOR
: String valued FML32 field returned by the system on a previous GET or GETNEXT operation. The value returned must be transferred by the application to the subsequent request buffer so that the system can determine current retrieval position.

TA_OCCURS
: Long valued FML32 field identifying how many objects are to be retrieved on a GET or GETNEXT operation. If this field is not specified, then all matching objects are returned, space permitting.

TA_FLAGS
: Long valued FML32 field identifying generic and component MIB specific flag values. Component MIB specific values that may be set in this attribute are defined within each component MIB reference page. Generic flag values and uses are listed below.

  MIB_LOCAL
  : This flag is used to modify retrievals from certain classes defined in this MIB. For a number of classes in this MIB, there exists both global information (available at any site in an active application) and local information (available on the particular site where the object is active). Requests to retrieve information from these classes will by default retrieve only the global information and not the local for efficiency. If the application user is willing to wait for local information to be collected, possibly from multiple sites, then this flag should be set on the retrieval request. Classes with local

information have local attributes listed last in the attribute table with a subheading indicating that they are local attributes. Classes which have only local information will automatically default to retrieving local information even if this flag value is not set.

`MIB_PREIMAGE`
: indicates that a pre-image check must be passed before a `SET` operation will be performed. A pre-image check insures that occurrence 0 of any `MIB` specific class attributes match the existing object. If so, then the object is updated using occurrence 1 of any `MIB` specific class attributes. Attributes occurring less than two times are not considered for pre-image checking. Multiply occurring fields are checked if their associated count attribute is specified twice.

`MIB_SELF`
: This flag is used as a shorthand to indicate that identification attributes for the client or server originating the request should be added to the request buffer prior to processing. For clients, `TA_CLIENTID` is added and for servers, `TA_GRPNO` and `TA_SRVID` are added.

`TA_FILTER`
: Long valued `FML32` field that may be specified with up to 32 occurrences to indicate the specific class attributes that should be returned. An occurrence with the value 0 may be specified to end the list but is not required. A list with an initial attribute value of 0 will return no class specific attributes but will return a count of class objects matched.

`TA_MIBTIMEOUT`
: Long valued `FML32` field identifying the time, in seconds, that should be allowed within the component MIB service to satisfy the request. A value less than or equal to 0 indicates that the component MIB service should not undertake any blocking operation. If unspecified, this value defaults to 20.

`TA_CURSORHOLD`
: Long valued `FML32` field identifying the time, in seconds, that a system snapshot generated from an initial `GET` operation should be held after the current `GET` or `GETNEXT` operation is satisfied before disposing of it. A value less than or equal to 0 indicates that the snapshot should be disposed of after satisfying the current request. If unspecified, this value defaults to 120.

In the following table, R indicates a required `INPUT` attribute, O an optional `INPUT` attribute, and — an unused `INPUT` attribute.

Input Table

Attribute	Type	GET	GETNEXT	SET
TA_OPERATION	string	R	R	R
TA_CLASS	string	R	—	R
TA_CURSOR	string	—	R	—
TA_OCCURS	long	O	O	—
TA_FLAGS	long	O	O	O
TA_FILTER	long	O	—	—
TA_MIBTIMEOUT	long	O	O	O
TA_CURSORHOLD	long	O	O	—

Output

Output from successful administrative requests consists of one or more MIB specific objects and one occurrence of the generic output fields. In general, multiple MIB specific objects are reflected in the output buffer by multiple occurrences of each class attribute returned. Occurrence 0 of each attribute relates to the first object, occurrence 1 to the second object, and so on. Exceptions to this guideline are noted in the component MIB reference pages. Intermediate occurrences without values for certain attributes may have FML32 defined NULL field values inserted as place holders. A successful SET operation returns a single object reflecting the object after the operation was performed. A successful GET or GETNEXT operation may return 0 or more occurrences depending on how many occurrences were requested (see TA_OCCURS below), how many occurrences were matched by the specified key fields and space limitations within the MIB specific system service.

It is important to note that not all attributes defined for any class may necessarily be returned for any request depending on object state, interoperating release environments and/or input request filters. Administrative programmers should avoid implicit dependencies on the presence of certain attributes in output buffers and should instead explicitly check for the presence of attribute values.

To repeat, the reply to a successfully processed administrative request includes certain generic fields that apply to all MIBs. The fields are defined in the header file <tpadm.h>. The corresponding field table file can be found in ${TUXDIR}/udataobj/tpadm. The generic reply fields are added to a the reply buffer and returned with the component MIB specific fields. The generic reply fields are described below.

TA_CLASS
: String valued field identifying the class represented in the reply buffer. Class names are defined within component MIB specific reference pages.

TA_OCCURS
: Long valued FML32 field identifying how many objects are in the reply buffer.

TA_MORE
: Long valued FML32 field identifying how many additional objects matching the request key fields are being held in a system snapshot for later retrieval. This field is not returned for SET operations.

TA_CURSOR
: String valued FML32 field identifying the position within a system held snapshot. This field must be added to the request buffer for a subsequent GETNEXT operation. The value of this field should not be interpreted or modified by the application user. This field is not returned for SET operations.

TA_ERROR
: Long valued FML32 field identifying a non-negative return code characterizing the successful return. Generic return codes and their meaning are defined below.

  TAOK
  : the operation was successfully performed. No updates were made to the application.

  TAUPDATED
  : an update was successfully made to the application.

  TAPARTIAL
  : a partial update was successfully made to the application.

Administrative requests that fail within MIB specific system service processing return an application service failure to the application including the original request and generic fields used to characterize the error. Application service failures are indicated by a TPESVCFAIL error return from tpcall(3) or tpgetrply(3). Application service failures returned via the TMQFORWARD(5) server will appear on the error queue specified on the original request (assuming the -d option was specified on the server command line). Generic fields used to characterize failed administrative requests are listed below.

`TA_ERROR`

Long valued `FML32` field identifying the particular error that occurred. Error codes may be generic in which case they are listed in the "`DIAGNOSTICS`" section of this reference page, or they may be specific to a component MIB, in which case they are described on the individual component MIB reference page.

`TA_STATUS`

String valued `FML32` field providing a textual description of the error.

`TA_BADFLD`

Long valued `FML32` field providing the field identifier of the offending field in cases where an error can be attributed to the value in a particular field. In cases where errors are caused by the combination of values in multiple fields, there may be multiple occurrences of this field.

Usage

Include Files

Application programs written to interface with component MIBs must include certain header files. <fml32.h> defines macros, structures and function interfaces necessary for accessing and updating FML32 typed buffers. <fml1632.h> defines a mapping from the generic FML interface macros, structures and functions to the FML32 versions and may optionally be included. <tpadm.h> defines the FML32 field names contained in this reference page. Additionally, any component MIB specific header files must be included to gain access to FML32 field definitions specific to that component MIB.

Example:

```
#include fml32.h>
 #include tpadm.h>
 #include cmib.h>  /* Component MIB Header */
```

Buffer Allocation

Interaction with a component MIB requires an FML32 typed buffer to carry the request to the service that acts on it. The ATMI verb tpalloc(3) allocates the buffer using FMLTYPE32 (defined in fml32.h>) as the value for the *type* argument. There is no subtype for FML32 buffers so the *subtype* argument of tpalloc can be NULL. The default minimum size for an FML32 buffer is 1024 bytes. Specifying 0 for the *size* argument of tpalloc results in a buffer of minimum size. If the user knows that a larger buffer is needed, it may be allocated by specifying a value larger than the system minimum for *size*.

Example:

```
rqbuf = tpalloc(FMLTYPE32, NULL, 0);
```

Building MIB Requests

Once an FML32 typed buffer is allocated, the user needs to populate it with both generic MIB field values and values specific to the component MIB being addressed. The most common interfaces used to add values to a request buffer are the FML verbs Fadd32(3) and Fchg32(3). In the event that a field cannot be added because the request buffer is full, then the buffer may need to be reallocated using the ATMI verb tprealloc(3).

Example:

```
/*
 * Does not include error processing, bigger_size provided
 * by the user, not by the system. Fchg32 used to insure that
 * field occurrence 0 is set if we are reusing a buffer.
 */
if (Fchg32(rqbuf, TA_MIBFIELD, 0, "ABC", 0) == -1) {
  if (Ferror32 == FNOSPACE) {
    rqbuf = tprealloc(rqbuf, bigger_size);
```

```
    Fchg32(rqbuf, TA_MIBFIELD, 0, "ABC", 0);
  }
}
```

Controlling MIB Requests

In addition to attributes specific to each component MIB, there are required and optional attributes defined in this reference page that control the operation requested of the component MIB.

The required generic attributes are TA_OPERATION and TA_CLASS.

TA_OPERATION specifies the operation to be performed on the MIB being accessed. Valid operations are GET, GETNEXT and SET.

TA_CLASS specifies the MIB class being accessed. Class names are defined within the component MIB reference pages. If TA_OPERATION is GETNEXT, then an additional attribute, TA_CURSOR, is required. TA_CURSOR is a field returned on a previous GET or GETNEXT operation. It is used by the system on the subsequent request to determine retrieval position.

The optional attributes TA_OCCURS, TA_FLAGS, TA_FILTER, TA_MIBTIMEOUT and TA_CURSORHOLD may be used in addition to the required attributes to further tailor the request.

TA_OCCURS
: specifies how many objects are to be retrieved on a GET or GETNEXT operation. If unspecified, all occurrences are retrieved, space permitting.

TA_FLAGS
: is used to specify flag values. Some generic flags are defined in this reference page; others are defined in each component MIB reference page.

TA_FILTER
: restricts the attribute values returned for a GET operation. If unspecified, is a long valued FML32 field used to all available class attribute values are returned.

TA_MIBTIMEOUT
: specifies the time, in seconds, that should be allowed within the component MIB service to satisfy the request. A value less than or equal to 0 indicates that the component MIB service should not undertake any blocking operation. If unspecified, this value defaults to 20.

TA_CURSORHOLD

specifies the time, in seconds, that a system snapshot generated from an initial GET operation should be held after the current GET or GETNEXT operation is satisfied before disposing of it. A value less than or equal to 0 indicates that the snapshot should be disposed of after satisfying the current request. If unspecified, this value defaults to 120.

Example:

```
/* GET 1st 5 objects */
Fchg32(rqbuf, TA_OPERATION, 0, "GET", 0);
Fchg32(rqbuf, TA_CLASS, 0, "classname", 0);
n = 5;
Fchg32(rqbuf, TA_OCCURS, 0, n, 0);
/* Make request, see Sending MIB Requests below */
/* Reply is stored in rpbuf and contains cursor */
/*
 * GETNEXT 5 objects. Transfer TA_CURSOR from rpbuf.
 * Reuse rqbuf generated above. Dispose of snapshot after
 * request, that is, set TA_CURSORHOLD to 0.
 */
Fchg32(rqbuf, TA_OPERATION, 0, "GETNEXT", 0);
Fchg32(rqbuf, TA_CURSOR, 0, Ffind32(rpbuf, TA_CURSOR, 0, NULL), 0);
n = 0;
Fchg32(rqbuf, TA_CURSORHOLD, 0, n, 0);
/* Make request, see Sending MIB Requests below */
```

Component MIB Fields

Component MIB key fields specified on a GET or GETNEXT are used to select a set of objects. Non-key fields are ignored by the component MIB.

Component MIB key fields specified on a SET operation are used to identify the particular object to be updated. Non-key fields are processed as updates to the object identified by the key fields. The user may optionally specify a pre-image which must match the current object image before an update (SET) is allowed. A user indicates that a pre-image is provided by setting the MIB_PREIMAGE bit in the TA_FLAGS attribute of the request. The key fields specifying the object to be updated are taken from the pre-image (field occurrence 0). If key fields are also specified in the post-image, then they must match exactly or the request fails. Only attributes that are part of the class and have two attribute values specified in the input buffer are considered for pre-image matching. Attributes with single values are processed as new values to be set for the indicated class object.

Example:

```
Fchg32(rqbuf, TA_OPERATION, 0, "GET", 0);
Fchg32(rqbuf, TA_CLASS, 0, "classname", 0);
Fchg32(rqbuf, TA_MIBKEY, 0, "keyvalue", 0);
n = 1;
Fchg32(rqbuf, TA_OCCURS, 0, n, 0); /* GET 1st matching occurrence */
/* Make request, see Sending MIB Requests below, reply in rpbuf */
/* Use rpbuf as pre-image and update TA_MIBFIELD value
 * if matching
 */
Fcpy32(newrq, rpbuf);
Fconcat32(newrq, rpbuf);   /* Add 2nd identical copy */
Fchg32(newrq, TA_OPERATION, 0, "SET", 0);
n = MIB_PREIMAGE;
Fchg32(newrq, TA_FLAGS, 0, n, 0);
Fchg32(newrq, TA_MIBFIELD, 1, "newval", 0); /* Post-image */
/* Make request, see Sending MIB Requests below */
```

Sending MIB Requests

All component MIB requests flow through the core BEA TUXEDO component MIB service, "`.TMIB`". This service not only acts as an agent for servicing `TM_MIB`(5) requests, it also directs requests targeted for other component MIBs so that the user need not be concerned with matching service names to MIBs and classes. Service requests can be generated using any of the request/response oriented service verbs in ATMI: `tpcall`(3), `tpacall`(3) and `tpenqueue`(3). The user has access to all flags and capabilities defined for these interface functions. The only constraint imposed here is that the "`.TMIB`" service must be invoked outside the scope of any transaction. This means that when using `tpcall`(3) or `tpacall`(3) to direct administrative requests within a transaction, the `TPNOTRAN` flag should be used or the user will get a failure (`TPETRAN`). When using `tpenqueue`(3) to direct requests, the `TMQFORWARD` server must be started with the `-n` option so that the forwarded service requests may be made outside of transactional boundaries.

Example:

```
/* Build request as shown above */
/* Send request and wait for reply */
flags = TPNOTRAN | TPNOCHANGE | TPSIGRSTRT;
rval = tpcall(".TMIB", rqbuf, 0, rpbuf, rplen, flags);
/* Send request and get descriptor back */
flags = TPNOTRAN | TPSIGRSTRT;
cd = tpacall(".TMIB", rqbuf, 0, flags);
/* Enqueue request, assumes qctl already setup */
flags = TPSIGRSTRT;
rval = tpenqueue("queue", ".TMIB", qctl, rqbuf, 0, flags);
```

Receiving MIB Replies

Replies from component MIBs may be received in one of three ways depending on how the original request was generated. If the original request was generated using tpcall(3), then a successful return from tpcall(3) indicates that the reply has been received. If the original request was generated using tpacall(3), then the reply may be received using `tpgetrply`(3). If the original request was generated using `tpenqueue`(3) and a reply queue was specified in the queue control structure, then the reply may be received using tpdequeue(3). All supported flags on these various calls may be used as appropriate.

Example:

```
/* Build request as shown above */
/* Send request and wait for reply */
flags = TPNOTRAN | TPNOCHANGE | TPSIGRSTRT;
rval = tpcall(".TMIB", rqbuf, 0, rpbuf, rplen, flags);
/* Receive reply using call descriptor */
flags = TPNOCHANGE | TPSIGRSTRT;
rval = tpgetrply(cd, rpbuf, rplen, flags);
/* Receive reply using TPGETANY, may need to change buffer type */
flags = TPGETANY | TPSIGRSTRT;
rval = tpgetrply(rd, rpbuf, rplen, flags);
/* Dequeue reply, assumes qctl already setup */
flags = TPNOCHANGE | TPSIGRSTRT;
rval = tpdequeue("queue", "replyq", qctl, rpbuf, rplen, flags);
```

Interpreting MIB Replies

In addition to attributes specific to a component MIB certain generic MIB fields may be returned in response to an administrative request, These additional attributes characterize the results of the original request and provide values that can be used in subsequent requests if necessary.

Successful `GET` or `GETNEXT` operations return:

- `TA_CLASS`

  Class name.

- `TA_OCCURS`

  Number of matching objects retrieved.

- `TA_MORE`

  Number of matching objects left to be retrieved.

- `TA_CURSOR`

  Cursor to be provided on subsequent retrieval.

- TA_ERROR

  Set to the non-negative return value TAOK.

- All available component MIB specific attributes

  Occurrence 0 of each attribute represents the first retrieved object, occurrence 1 the second, and so on. Exceptions to this rule are identified as appropriate in the component MIB reference pages.

Successful SET operations return:

- TA_CLASS

  Class name.

- TA_ERROR

  Set to a non-negative return value. TAOK indicates that the request was successful but no information was updated. This can happen because no changes were specified or because the changes specified match the current state of the object. TAUPDATED indicates that the request was successful and the information was updated. TAPARTIAL indicates that the request was successful but the update was only made partially within the system. This may occur because of network failures or message congestion and the system will synchronize the unupdated sites as soon as possible.

- All available component MIB specific attributes

  Since only one object may be updated at once, only one object will be returned. The returned attributes reflect the object after the update.

Failed operations of any type return:

- Fields specified on the original request

- TA_ERROR

  Set to a negative return value indicating the cause of the failure. Generic error codes are specified in the "DIAGNOSTICS" section of this reference page. Component MIB specific error codes (non-overlapping, both with each other and with the generic codes) are specified on each MIB reference page.

- TA_BADFLD

  Field identifier of the offending field.

- TA_STATUS

  Textual description of error condition.

Limitations

FML32 buffers with multiple occurrences of fields do not allow for empty fields in a sequence of occurrences. For example, if you set a value for occurrence 1 and occurrence 0 does not yet exist, FML32 automatically creates occurrence 0 with an FML32 defined `NULL` value. `FML32` defined NULL values are 0 for numeric fields, 0-length strings for string fields and the character '\0' for character fields. Because of this limitation, `GET` operations, which may at times return objects with different sets of attributes, may artificially break up the sets of objects returned to the user so as to not include `NULL FML32` fields that do not accurately reflect the state of the object.

Workstation clients on DOS, Windows and OS/2 are currently limited to 64K `FML32` buffers; therefore, the system restricts return buffers to be less than 64K per buffer.

Administrative API access is not available through the COBOL version of ATMI since COBOL has limited support for FML32 buffer type.

Requests to any component MIB cannot be part of an application transaction. Therefore, any calls to `tpcall`(3) or `tpacall`(3) directed to a component MIB and made within an active transaction should set the `TPNOTRAN` flag on the call. However, requests may be enqueued for future delivery to a component MIB using the ATMI verb `tpenqueue`(3) within a transaction. The enqueuing of the request will take place within a transaction while the processing within the component MIB will not. The use of the `TMQFORWARD`(5) server in this context requires that `TMQFORWARD` be started with the `-n` command line option so that request may be forwarded to the MIB service in non-transactional mode. Because of the non-transactional nature of component MIB services, it is also recommended that the `-d` option for `TMQFORWARD` be used so that service failures are delivered to the failure queue immediately rather than retrying the request.

Field identifiers for generic MIB fields and for component MIBs will be allocated in the range 6,000 to 8,000 inclusive. Therefore, applications which intend to mix administrative actions with user actions should make sure to allocate field identifiers appropriately.

Class Descriptions

Each class description section has four subsections:

Overview
: High level description of the attributes associated with the class.

Attribute Table
: A table that lists the name, type, permissions, values and default for each attribute in the class. The format of the attribute table is described below.

Attribute Semantics
: Tells how each attribute should be interpreted.

Limitations
: Limitations in the access to and interpretation of this class.

Attribute Table Format

As described above, each class is defined in four parts. One part is the attribute table. The attribute table is a reference guide to the attributes within a class and how they may used by administrators, operators and general users to interface with an application. There are five components to each attribute description in the attribute tables: name, type, permissions, values and default. Each of these components is discussed in detail below:

Name:
: `FML32` field identifier name used to identify this attribute value within an `FML32` buffer. Attributes may be arranged in groups of closely related attributes. No special meaning should be implied from the groupings; they are intended only to improve the usability of the table. A notation (r), (k), (x) or (*) may appear after an attribute name or value. The meaning of the notation is as follows:

  (r) - the field is required when a new object is created

  (k) - indicates a key field for object retrieval

  (x) - indicates a regular expression key field for object retrieval

  (*) - the field is a `SET` key for object modification

  `SET` operations on classes with one or more `SET` keys defined (see * above) must include values for one or more of the attribute values defined as `SET` keys. The `SET` keys specified must be sufficient to identify exactly one object within the class. `SET` keys are always key fields for object retrieval and therefore the (k) notation is implied though not specified. `SET` keys are not

however always required fields when creating NEW objects and will be marked with the (r) notation if they are required.

Type:

Data type of the attribute value. Data types are defined in C language notation, that is, long, char and string. In a program, data type can be determined by using the FML32 function Fldtype32(3) which returns the FML32 define representing the data type; that is, FLD_LONG, FLD_CHAR and FLD_STRING.

Permissions:

Access and update permissions are split into three groups of three each, in the manner of UNIX System permissions. However, in the attribute tables the three groups represent permissions for administrators, operators and others rather than for owner, group and others as is the case in UNIX. For each group there are three permissions positions that have the following meanings:

Position 1 - Retrieval permissions

r	Attribute may be retrieved.
R	Attribute may be retrieved only when the object state is ACTive or ACTive equivalent. See the description of the TA_STATE attribute value for each class to determine which states qualify as ACTive equivalent. This attribute represents transient information that is not persistent across distinct activations of the object.
k	Attribute may be specified only as a key field for retrieval or update.
K	Attribute may be specified only as a key field for retrieval or update and then only when the object state is ACTive or ACTive equivalent. See the description of the TA_STATE attribute value for each class to determine which states qualify as ACTive equivalent.

Position 2 - Inactive update permissions

w	Attribute may be updated when the object is in an INActive or INActive equivalent state. See the description of the TA_STATE attribute value for each class to determine which states qualify as INActive equivalent.
u	Attribute may be updated as described for the w permissions value. In addition, the combination of all attribute values identified with the u permissions character must be unique within the class.
U	Attribute may be updated as described for the w permissions value. In addition, the attribute value must be unique for the attribute within the class.

Position 3 - Active update permissions

x	Attribute may be updated when the object is in an ACTive or ACTive equivalent state. See the description of the TA_STATE attribute value for each class to determine which states qualify as ACTive equivalent.
X	Attribute may be updated when the object is in an ACTive or ACTive equivalent state. See the description of the TA_STATE attribute value for each class to determine which states qualify as ACTive equivalent. This attribute represents transient information and updates to this attribute value are not persistent across distinct activations of the object.
y	Attribute may be updated when the object is in an ACTive or ACTive equivalent state. However, there are limitations on when the change will affect objects of this or other classes. Consult the textual description of the attribute in the Attribute Semantics section for the class for more details. See the description of the TA_STATE attribute value for each class to determine which states qualify as ACTive equivalent.

Values

Values that may be set and/or retrieved with respect to this attribute. Certain formatting conventions are followed in listing attribute values.

`LITSTRING`	Literal string value.
num	Numeric value.
string[x..y]	String value between *x* and *y* characters in length, not including the terminating `NULL` character.
LMID	Shorthand for *string[1..30] (no commas allowed).* Represents a logical machine identifier.
{*x*|*y*|*z*}	Select one of *x, y* or *z.*
[*x*|*y*|*z*]	Select zero or one of *x, y* or *z*.
[*x*|*y*|*z*],*	Zero or more occurrences of *x, y* or *z* in a comma-separated list.
low = num	Numeric value greater than or equal to *low*.
low = num high	Numeric value greater than or equal to *low* and less than *high.*
`GET:`	State attribute values that may be returned or specified as key values on a retrieve (`GET`) operation. Values shown are always the three letter state abbreviation. The expanded state name is shown in the text describing the `TA_STATE` for the class. Input specifications may be made in either the shorthand or expanded form and are case-insensitive. Output states are always returned in expanded format with all upper case.
`SET:`	State attribute values that may be set on an update (`SET`) operation. Use of abbreviations is allowed as described above.

Default:

Default used when creating a new object, that is, state change from `INValid` to `NEW`. The value N/A is shown in this column for attributes that are required, derived or only available when the object is active.

TA_STATE Syntax

The TA_STATE attribute field is a member of each class defined. The semantics of this attribute are defined on a class by class basis. For the sake of brevity, TA_STATE values are often specified in a three character shorthand notation. When an expanded version of a TA_STATE value is shown, the three shorthand letters are capitalized and the rest of the letters (if any) are displayed in lower case. Input TA_STATE values may be in either shorthand or long notation and are case insensitive. Output TA_STATE values are always full length upper case. The following example should help clarify the use of the TA_STATE attribute.

```
Full Name  : ACTive
Shorthand  : ACT
Output Value : ACTIVE
Valid Input : ACT, act, AcTiVe, active
```

T_CLASS CLASS DEFINITION

Overview

The T_CLASS class represents attributes of administrative classes within a BEA TUXEDO system application. Its primary use is to identify class names.

Attribute Table

T_CLASS Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_CLASSNAME(k)	string	r--r--r--	*string*	N/A
TA_STATE(k)	string	r--r--r--	GET:"{VAL}" SET: N/A	GET:N/A SET: N/A
TA_GETSTATES TA_INASTATES TA_SETSTATES	string string string	r--r--r-- r--r--r-- r--r--r--	*string* *string* *string*	N/A N/A N/A
(k) - a key field for object retrieval				

Attribute Semantics

TA_CLASSNAME: *string*

Class name.

TA_STATE:

GET:

A GET operation retrieves information for the selected T_CLASS object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed are not returned.

VALid	T_CLASS object is defined. All objects of this class exist in this state. This state is INActive-equivalent for the purposes of permissions checking.

SET:

SET operations are not permitted on this class.

TA_GETSTATES: *string*

Delimited list ('|' delimiter) of the states that may be returned for an object in this class or as the result of a GET operation. States are returned in their full length uppercase format.

TA_INASTATES: *string*
: Delimited list ('|' delimiter) of the inactive equivalent states that may be returned for an object in this class or as the result of a GET operation. States are returned in their full length uppercase format.

TA_SETSTATES: *string*
: Delimited list ('|' delimiter) of the states that may be set for an object in this class as part of a SET operation. States are returned in their full length uppercase format.

Limitations None identified.

T_CLASSATT CLASS DEFINITION

Overview

The T_CLASSATT class represents characteristics of administrative attributes on a class/attribute basis.

Attribute Table

T_CLASSATT Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_CLASSNAME(r)(*) TA_ATTRIBUTE(r)(*)	string long	ru-r--r-- ru-r--r--	*string* 0 = *num*	N/A N/A
TA_STATE(k)	string	rw-r--r--	GET:"{VAL}" SET:"{NEW|INV}"	GET: N/A SET: N/A
TA_PERM(r) TA_FACTPERM TA_MAXPERM	long long long	rw-r--r-- r--r--r-- r--r--r--	0000 = *num* = 0777 0000 = *num* = 0777 0000 = *num* = 0777	N/A N/A N/A
TA_ATTFLAGS TA_DEFAULT TA_VALIDATION	long string string	r--r--r-- r--r--r-- r--r--r--	*long* *string* *string*	N/A N/A N/A
(k) - GET key field (r) - Required field for object creation (SET TA_STATE NEW) (*) - GET/SET key, one or more required for SET operations				

Attribute Semantics

TA_CLASSNAME : *string*

Class name. Only class names known to the system are accessible.

TA_ATTRIBUTE: *long*

Attribute field identifier as defined in the system provided header file, for example, tpadm.h.

TA_STATE :

GET: VALid

A GET operation will retrieve information for the selected T_CLASSATT object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

VALid	T_CLASSATT object is defined. All objects of this class exist in this state. This state is INActive equivalent for the purposes of permissions checking.

SET: {NEW|INValid}

A SET operation will update configuration information for the selected T_CLASSATT object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_CLASSATT object for application. State change allowed only when in the INValid state. Successful return leaves the object in the VALid state.
unset	Modify T_CLASSATT object. Allowed only when in the VALid state. Successful return leaves the object state unchanged.
INValid	Delete or reset T_CLASSATT object for application. State change allowed only when in the VALid state. Successful return leaves the object in either the INValid state or the VALid state. Objects of this class that are built-in, that is, explicitly known to the system, will revert to their default permissions on this state change and continue to exist in the VALid state. Objects of this class that belong to add-on components for which the class attributes are not explicitly known will be deleted on this state change and transition to the INValid state.

TA_PERM : 0000 = *num* = 0777

Access permissions for this class attribute combination. When setting permissions, the actual value set may be automatically reset if the requested setting exceeds the permissions available for the attribute. The maximum

permissions available for an attribute are the permissions documented for the administrator repeated in the operator and other permissions positions. For example, the TA_TYPE attribute of the T_MACHINE class is documented with permissions rw-r--r-- and has maximum permissions of rw-rw-rw-.

TA_FACTPERM : 0000 = *num* = 0777
: Permissions for this class attribute combination as set on delivery of the BEA TUXEDO system from the factory. These permissions will apply after a SET operation changing the TA_STATE of an object to INValid.

TA_MAXPERM : 0000 = *num* = 0777
: Maximum permissions for this class attribute combination.

TA_ATTFLAGS : *long*
: Bitwise or of none, some or all of the following flags indicating special characteristics of this attribute.

  MIBATT_KEYFIELD
  : Attribute is a key field for this class.

  MIBATT_LOCAL
  : Attribute represents local information.

  MIBATT_REGEXKEY
  : Attribute is a regular expression key field for this class.

  MIBATT_REQUIRED
  : Attribute is required when creating a NEW object in this class.

  MIBATT_SETKEY
  : Attribute is a SET key for this class.

  MIBATT_NEWONLY
  : Attribute is writable for inactive equivalent objects in this class only when creating a NEW object by changing the TA_STATE from INValid to NEW.

TA_DEFAULT : *string*
: Default for this attribute when creating a NEW object in this class. Note that for classes where NEW objects may not be created through the Admin API, this attribute will always be returned as a 0 length string. Attributes that may not be SET when creating a NEW object are also returned as 0 length strings. Attributes which have *long* values will have defaults returned as the string

representing the long value. Some attributes have special characteristics indicated by the special values indicated below that may be returned here.

`# Inherited:`*Classname*`[:`*Attribute*`]`
: Attribute default is inherited from the attribute of the same name in the indicated class. If *Attribute* is specified, then the value is inherited from the indicated attribute rather than the one of the same name.

`# Required`
: Attribute is required when creating a `NEW` object.

`# Special`
: Attribute has special rules for defining the default. The appropriate component MIB reference page should be consulted for further details.

`TA_VALIDATION :` *string*
: String representing the validation rule applied to this class/attribute combination when a new value is being `SET`. This string will take one of the following formats:

`CHOICES=`*string1*`|`*string2*`|...`
: String attribute value that must match exactly one of the choices shown.

`RANGE=`*min*`\(em`*max*
: Numeric attribute value that must be between *min* and *max*, inclusive.

`SIZE=`*min*`\(em`*max*
: String or carray attribute value that must have a length between *min* and *max* bytes long, inclusive.

`READONLY=Y`
: Readonly attribute with no validation rule for write operations.

`SPECIAL=Y`
: Special validation rule. Consult the appropriate component MIB reference page for more details.

UNKNOWN=Y
: Unknown validation rule. Commonly associated with add-on component attribute entries for which the details are not known by the core system.

Limitations

None identified.

Diagnostics

There are two general types of errors that may be returned to the user when interfacing with component MIBs. First, any of the three ATMI verbs (tpcall(3), tpgetrply(3) and tpdequeue(3)) used to retrieve responses to administrative requests may return any error defined on their respective reference pages.

Second, if the request is successfully routed to a system service capable of satisfying the request and that service determines that there is a problem handling the request, then failure may be returned in the form of an application level service failure. In these cases, tpcall(3) or tpgetrply(3) returns an error with tperrno set to TPESVCFAIL and returns a reply message containing the original request along with TA_ERROR, TA_STATUS or TA_BADFLD fields further qualifying the error as described below. When a service failure occurs for a request forwarded to the system through the TMQFORWARD(5) server, the failure reply message will be enqueued to the failure queue identified on the original request (assuming the -d option was specified for TMQFORWARD).

When a service failure occurs during processing of an administrative request, the FML32 field TA_STATUS is set to a textual description of the failure, the FML32 field TA_ERROR is set to indicate the cause of the failure as indicated below. TA_BADFLD is set as indicated in the description of the individual errors below. All error codes specified below are guaranteed to be negative.

[TAEAPP]
: The originating request required application cooperation to be successfully completed and the application did not allow the operation to be completed. For example, server shutdown requires application cooperation.

[TAECONFIG]
: The configuration file associated with the component MIB could not be accessed as needed to satisfy the requested operation.

[TAEINVAL]
: A specified field is invalid. TA_BADFLD is set to indicate the invalid field identifier.

[TAEOS]
: An operating system error occurred while attempting to satisfy the request. TA_STATUS is updated with the translation of the system error code errno.

[TAEPERM]
: An attempt was made to SET an attribute for which the user does not have write permissions or the user attempted a GET on a class for which the user does not have read permissions. TA_BADFLD is set to indicate the field identifier that failed permissions checking.

[TAEPREIMAGE]
: A SET operation failed due to a mismatch between the specified pre-image and the current object. TA_BADFLD is set to indicate the field identifier that failed the pre-image checking.

[TAEPROTO]
: The administrative request was made in an improper context. TA_STATUS is populated with additional information.

[TAEREQUIRED]
: A required field value is not present. TA_BADFLD is set to indicate the missing field identifier.

[TAESUPPORT]
: The administrative request is not supported in the current version of the system.

[TAESYSTEM]
: A BEA TUXEDO system error occurred while attempting to satisfy the request. TA_STATUS is updated with more information on the error condition.

[TAEUNIQ]
: A SET operation did not specify class keys identifying a unique object to be updated.

[*other*]
: Other error return codes specific to particular component MIBs are specified in the component MIB reference pages. These error codes are guaranteed to be mutually exclusive both amongst all component MIBs and with generic codes defined here.

The following diagnostic codes are returned in TA_ERROR to indicate successful completion of an administrative request. These codes are guaranteed to be non-negative.

[TAOK]
The operation succeeded. No updates were done to the component MIB object(s).

[TAUPDATED]
The operation succeeded. Updates were made to the component MIB object.

[TAPARTIAL]
The operation partially succeeded. Updates were made to the component MIB object.

Interoperability Access to the FML32 interfaces, and therefore to the component MIBs available for administration of a BEA TUXEDO system application, are available on BEA TUXEDO Release 4.2.2 and later. The header files and field tables defining generic MIB attributes are available on TUXEDO Release 5.0 and later. Interoperability concerns specific to a particular component MIB are discussed in the reference page for that component MIB.

Portability The existing FML32 and ATMI functions necessary to support administrative interaction with BEA TUXEDO system MIBs, as well as the header file and field table defined in this reference page, are available on all supported native and workstation platforms.

Examples See the "USAGE" section earlier for some brief example uses of existing APIs in interfacing with generic MIB processing. More detailed examples are provided with each component MIB reference page that make use of real component MIB classes and attributes.

Files
```
${TUXDIR}/include/tpadm.h,
${TUXDIR}/udataobj/tpadm
```

See Also Fintro(3), Fadd32(3), Fchg32(3), Ffind32(3) tpalloc(3), tprealloc(3), tpcall(3), tpacall(3), tpgetrply(3), tpenqueue(3), tpdequeue(3), AUTHSVR(5), TM_MIB(5), TMQFORWARD(5), *Administering the BEA TUXEDO System*, *TUXEDO Programmer's Guide*

nl_types(5)

Name `nl_types`-native language data types

Synopsis
```
#include <nl_types.h>
```

Description The `nl_types.h` header file contains the following definitions:

`nl_catd`
: used by the message catalog functions `catopen`(3), `catgets`(3) and `catclose`(3) to identify a catalogue

`nl_item`
: used by `nl_langinfo`(3) to identify items of `langinfo`(5) data. Values for objects of type `nl_item` are defined in `langinfo.h`.

`NL_SETD`
: used by `gencat`(1) when no `$set` directive is specified in a message text source file. This constant can be used in subsequent calls to `catgets`() as the value of the set identifier parameter.

`NL_MGSMAX`
: maximum number of messages per set

`NL_SETMAX`
: maximum number of sets per catalogue.

`NL_TEXTMAX`
: maximum size of a message.

`DEF_NLSPATH`
: the default search path for locating catalogues.

See Also `gencat`(1), `catgets`(3), `catopen`(3), `nl_langinfo`(3), `langinfo`(5).

servopts(5)

Name

servopts-run-time options for BEA TUXEDO system server processes

Synopsis

```
AOUT CLOPT= [-A][-s{@filename|service[,service...][:func]}]
[-e stderr_file][-p [L][low_water][,[terminate_time]]
[:[high_water][,create_time]][-h][-l locktype][-n prio]
[-o stdout_file][-r][ -- uargs]
```

Description

servopts is not a command. Rather, it is a list of run-time options recognized by servers in a BEA TUXEDO system.

The server using these options may be one of the BEA TUXEDO system-supplied servers such as FRMPRT(5), or it may be an application-supplied server built with the buildserver(1) command.

Running servers in a BEA TUXEDO system is accomplished through the tmboot(1) and tmadmin(1) commands working with servers (and other resources) specified in the application configuration file. Desired selections from the servopts list are specified with the server in the configuration file. The following options are recognized:

-A

indicates that the server should initially offer all services with which it was constructed. For BEA TUXEDO system-supplied servers, -A is the only way of specifying services.

-s { *@filename* | *service*[,*service*...][:*func*] }

specifies the names of services to be advertised when the server is booted. In the most common case, a service is performed by a function that carries the same name; that is, the x service is performed by function x. For example, the specification

```
-s x,y,z
```

will run the associated server initially offering services x, y, and z, each processed by a function of the same name. In other cases, a service (or several services) may be performed by a function of a different name. The specification

```
-s x,y,z:abc
```

runs the associated server with initial services x, y, and z, each processed by the function abc.

Spaces are not allowed between commas. Function name is preceded by a colon. Service names (and implicit function names) must be less than or equal to 15 characters in length. An explicit function name (that is, a name specified after a colon) can be up to 128 characters in length. Names longer than these limits are truncated with a warning message. When retrieved by `tmadmin`(1) or `TM_MIB`(5), only the first 15 characters of a name are displayed.

A filename can be specified with the -s option by prefacing the filename with the '@' character. Each line of this file is treated as an argument to the -s option. You may put comments in this file. All comments start with '#' or ':'. The -s option may be specified multiple times.

`-e`

specifies the name of a file to be opened as the server's standard error file. Providing this option ensures that a restarted server has the same standard error file as its predecessors. If this option is not used, a default diversion file called `stderr` is created in the directory specified by `$APPDIR`.

`-p` [`L`][*low_water*][,[*terminate_time*]][:[*high_water*][,*create_time*]]

This option can be used to support automatic spawning/decay of servers. It may be used for servers on an MSSQ with MAX greater than 1; it is not allowed (and not necessary) for conversational servers. Arguments to the option have the following meanings: L The decision to spawn more servers is based on load rather than number of servers or messages. -- the remaining arguments, *low_water*, *terminate_time*, *high_water*, and *create_time* are used to control when servers are spawned or deactivated. The algorithm is: if the load meets or exceeds *high_water* for at least *create_time* seconds, a new server is spawned. If the load drops below *low_water* for at least *terminate_time* seconds, a server is deactivated.

The L option works only in SHM mode with load balancing turned on. If SHM/LDBAL+Y is not set, then a userlog message (LIBTUX_CAT:1542) is printed and no spawning is done.

low_water defaults to an average of 1 server or message on the MSSQ or a workload of 50. high_water defaults to an average of 2 servers or messages, or a workload of 100. create_time defaults to 50: terminate_time defaults to 60.

`-h`

do not run the server immune to hangups. If not supplied, the server ignores the hangup signal.

-l *locktype*

lock the server in core. The argument for *locktype* is t, d, or p according to whether the text (TXTLOCK), data (DATLOCK), or the entire process (text and data - PROCLOCK), should be locked. See plock(2) for details. The lock fails if the server is not run as root. There is no way to unlock a server once it is locked.

-n *prio*

nice the server according to the *prio* argument. Giving the process better priority (a negative argument) requires it to be run with the uid of root. See nice(2) for details.

-o *stdout_file*

specifies the name of a file to be opened as the server's standard output file. Providing this option ensures that a restarted server has the same standard output file as its predecessors. If this option is not used, a default diversion file called stdout is created in the directory specified by $APPDIR.

-r

specifies that the server should record, on its standard error file, a log of services performed. This log may be analyzed by the txrpt(1) command. When the -r option is used, make sure that the ULOGDEBUG variable is not set to "y". The ULOGDEBUG variable prevents debugging messages from being sent to stderr. Debugging messages in the file will be misinterpreted by txrpt.

--

marks the end of system-recognized arguments and the start of arguments to be passed to a subroutine within the server. This option is needed only if the user wishes to supply application-specific arguments to the server. The system-recognized options precede the --; application arguments should follow it. Application arguments may be processed by a user-supplied version of the tpsvrinit(3c) function. getopt(3) should be used to parse them. Because all system arguments are processed prior to the call to tpsvrinit(3c), when the call is made the external integer, optind points to the start of the user flags. The same option letters (for example, -A) may be reused after the -- argument, and given any meaning appropriate to the application.

Note: At run time the BEA TUXEDO system automatically adds the following option to each command line for each server:

-c dom=*domainid*

The -c option adds a comment line, in which the specified domain ID is reported, to any command output that reports on the processes associated with the domain in question, such as the output of the ps command. This comment helps an administrator who is managing multiple domains to interpret a single output stream that refers to several domains.

Examples

See the EXAMPLES section of ubbconfig(5).

See Also

buildserver(1), tmadmin(1), tmboot(1), txrpt(1), tpsvrinit(3c), BQ(5), FRMPRT(5), ubbconfig(5), *Administering the BEA TUXEDO System*, nice(2), plock(2), getopt(3) in UNIX reference manuals

TM_MIB(5)

Name BEA TUXEDO System Management Information Base

Synopsis

```
#include <fml32.h>
#include <tpadm.h>
```

Description The BEA TUXEDO system MIB defines the set of classes through which the fundamental aspects of an application can be configured and managed. This includes management of machines, servers, networking.

TM_MIB(5) should be used in combination with the generic MIB reference page MIB(5) to format administrative requests and interpret administrative replies. Requests formatted as described in MIB(5) using classes and attributes described in this reference page may be used to request an administrative service using any one of a number of existing ATMI interfaces in an active application. Inactive applications may also be administered using the tpadmcall (3c) function interface. TM_MIB(5) consists of the following classes:

TM_MIB Classes

Class Name	controls
T_BRIDGE	Network connections
T_CLIENT	Clients
T_CONN	Conversations
T_DEVICE	Devices
T_DOMAIN	Global application attributes
T_FACTORY	
T_GROUP	Server groups
T_IFQUEUE	
T_INTERFACE	
T_MACHINE	Machine specific attributes
T_MSG	Message queues
T_NETGROUP	Network groups

TM_MIB Classes

Class Name	controls
T_NETMAP	Machines to Netgroups
T_QUEUE	Server queue
T_ROUTING	Routing criteria
T_SERVER	Servers
T_SERVICE	Services
T_SVCGRP	Service group
T_TLISTEN	/T listeners
T_TLOG	Transaction log
T_TRANSACTION	Transaction
T_ULOG	Userlog

Each class description consists of four sections:

- OVERVIEW - High level description of the attributes associated with the class.
- ATTRIBUTE TABLE - The format of the attribute table is summarized below and described in detail in MIB(5).
- ATTRIBUTE SEMANTICS - Defines the interpretation of each attribute that is part of the class.
- LIMITATIONS - Limitations in the access to and interpretation of this class.

Attribute Table Format

Each class that is a part of this MIB is defined in four parts in sections that follow. One of the four parts is the attribute table. The attribute table is a reference guide to the attributes within a class and how they may used by administrators, operators, and general users to interface with an application.

There are five columns for each attribute described in an attribute table: name, type, permissions, values, and default. Each of these components is discussed in MIB(5).

TA_FLAGS Values

MIB(5) defines the generic TA_FLAGS attribute, which is a long containing both generic and component MIB specific flag values. The following are the TM_MIB(5) specific flag values supported. These flag values should be or'd with any generic MIB flags.

TMIB_ADMONLY
: A flag used to indicate that only administrative processes should be activated when changing the state of a T_MACHINE object from INActive to ACTive.

TMIB_APPONLY
: A flag used to indicate that only application processes should be considered when activating or deactivating a T_MACHINE object. It may also be used on T_SERVER retrievals to restrict the retrieval to application servers only.

TMIB_CONFIG
: A flag used to indicate that only configured groups and servers should be considered in satisfying the request.

TMIB_NOTIFY
: A flag used when activating or deactivating T_MACHINE, T_GROUP, or T_SERVER objects to cause unsolicited notification messages to be sent to the originating client just prior to and just after the activation or deactivation of each server object selected.

FML32 Field Tables

The field table for the attributes described in this reference page is found in the file udataobj/tpadm relative to the root directory of the BEA TUXEDO system software installed on the system. The directory ${TUXDIR}/udataobj should be included by the application in the colon-separated list specified by the FLDTBLDIR environment variable, and the field table name tpadm should be included in the comma-separated list specified by the FIELDTBLS environment variable.

Limitations

Access to the header files and field tables for this MIB is being provided only on BEA TUXEDO system Release 5.0 sites and later, both native and Workstation.

Workstation access to this MIB is limited to runtime only access; the function tpadmcall (3c) is not supported on workstations.

For the purpose of preimage processing (MIB_PREIMAGE flag bit set), local attributes for classes that have global attributes are not considered. Additionally, indexed fields and the indexes that go with them are not considered, for example, T_TLOG class, TA_TLOGCOUNT, TA_TLOGINDEX, TA_GRPNO, TA_TLOGDATA attributes.

T_BRIDGE CLASS

Overview

The T_BRIDGE class represents run-time attributes pertaining to connectivity between logical machines making up an application. These attribute values represent connection status and statistics.

Attribute Table

Table 1: T_BRIDGE Class Definition Attribute Table

Attribute[1]	Type	Permissions	Values	Default
TA_LMID(*)2	string	r--r--r--	"*LMID1[,LMID2]*"	N/A
TA_NETGROUP(k)3	string	R--R--R--	"*string[1. . . 30]*"	DEFAULTNET
TA_STATE(k)	string	rwxrwxr--	GET:"{ACT|INA|SUS|PEN}" SET:"{ACT|INA|SUS|PEN}"	N/A N/A
TA_CURTIME	long	R--R--R--	0 <= *num*	N/A
TA_CONTIME	long	R-XR-XR--	0 <= *num*	N/A
TA_SUSPTIME	long	rwxrwxr--	0 <= *num*	300[4]
TA_RCVDBYT	long	R-XR-XR--	0 <= *num*	N/A
TA_SENTBYT	long	R-XR-XR--	0 <= *num*	N/A
TA_RCVDNUM	long	R-XR-XR--	0 <= *num*	N/A
TA_SENTNUM	long	R-XR-XR--	0 <= *num*	N/A
TA_FLOWCNT	long	R-XR-XR--	0 <= *num*	N/A
TA_CURENCRYPTBIT	string	R--R-----	{ 0 | 40 | 128 }	N/A

(k) - GET key field
(*) - GET/SET key, one or more required for SET operations

[1]All attributes in Class T_BRIDGE are local attributes.

[2]The TA_LMID attribute must be fully specified for SET operations, that is, *LMID1,LMID2*.

[3]SET operation may only use TA_NETGROUP DEFAULTNET in Release 6.4. GET operation may use any TA_NETGROUP defined for both LMID values.

[4]TA_SUSPTIME may be SET only if the TA_STATE is currently SUSPENDED or is being SET to SUSPENDED.

Attribute Semantics

TA_LMID: *LMID1[,LMID2]*

Source logical machine identifier (*LMID1*) and destination logical machine identifier (*LMID2*) for network connection.

TA_NETGROUP:*string[1 . . . 30]*

Logical name of the network group.When both source and destination TA_LMID identifiers are in the same TA_NETGROUP, the T_BRIDGE class will present all instances of related fields per TA_NETGROUP. TA_NETGROUP may be used as a key field on GET requests. TA_NETGROUP values other than DEFAULTNET may not be used on SET operations in this release (6.4).

TA_STATE:

GET: {ACTive|INActive|SUSpended|PENding}

A GET operation will retrieve run-time information for the selected T_BRIDGE object(s). A TA_LMID attribute value with only one logical machine identifier matches all active connections from *LMID1* to other machines in the application. In this case, each retrieved record will contain an expanded TA_LMID attribute value with the destination LMID filled in. The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

ACTive	The connection is established and active.
INActive	The connection is inactive. This state is only returned when status is requested on a particular connection, that is, both Lands specified in the TA_LMID attribute and the source logical machine is reachable.
SUSpended	An established connection was terminated due to an error condition, and reconnection has been suspended for at least the amount of time indicated in the TA_SUSPTIME attribute value. This state is ACTive equivalent for the purpose of determining permissions.
PENding	An asynchronous connection has been requested, but has not yet been completed. The final outcome of the connection request has not been determined.

SET: {ACTive|INActive|SUSpended|PENding}

A SET operation will update run-time information for the selected T_BRIDGE object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

unset	Modify an existing T_BRIDGE object. This combination is allowed only when in the ACTive or SUSpended state. Successful return leaves the object state unchanged.
ACTive	Activate the T_BRIDGE object by establishing a connection between the indicated logical machines. This operation will fail if only one logical machine is specified, if either of the two machines is not active, or if the source logical machine is not reachable. While the T_BRIDGE object is establishing the asynchronous connection, the BRIDGE will do other work. Using the state change to PENding is recommended. State change allowed in the INActive and SUSpended states. For the purpose of determining permissions for this state transition, the active object permissions are considered (that is, --x--x--x). Successful return leaves the object in the PENding state.
INActive	Deactivate the T_BRIDGE object by closing the connection between the indicated logical machines. This operation will fail if only one logical machine is specified or if the two machines are not connected. State change allowed only when in the ACTive state. Successful return leaves the object in the INActive state.

SUSpended	Suspend the T_BRIDGE object by closing the connection between the indicated logical machines and by setting the TA_SUSPTIME parameter as indicated. State change allowed only when in the ACTive state. Successful return leaves the object in the SUSpended state. Limitation: Note that since the statistics reported are from the viewpoint of the source logical machine, resetting those statistics will cause them to be out of sync with the statistics reported by the destination logical machine for the same connection.
PENding	Activate the T_BRIDGE object by establishing an asynchronous connection between the indicated logical machines. This operation will fail if only one logical machine is specified, if either of the two machines is not active, or if the source machine is not reachable. When in the PENding state, the success or failure of the connection request has not yet been determined. However, the BRIDGE may continue to process other events and data while the connection is outstanding. State change allowed in the INActive and SUSpended states. For the purpose of determining permissions for this state transition, the active object permissions are considered (that is, --x--x--x). Successful return leaves the object in the PENding state.

TA_CURTIME: 0 <= *num*

Current time, in seconds, since 00:00:00 UTC, January 1, 1970, as returned by the time(2) system call on T_BRIDGE:TA_LMID. This attribute can be used to compute elapsed time from the following attribute value.

TA_CONTIME: 0 <= *num*

Time, in seconds, since 00:00:00 UTC, January 1, 1970, as returned by the time(2) system call on T_BRIDGE:TA_LMID, when this connection was first established. Elapsed open time in seconds can be computed using TA_CURTIME - TA_CONTIME.

TA_SUSPTIME: 0 <= *num*

Time, in seconds, remaining in the suspension of this connection. After this amount of time, the connection will automatically change to a TA_STATE of INACTIVE and may be activated by normal application traffic.

TA_RCVDBYT: 0 <= *num*
: Number of bytes sent from the destination logical machine to the source logical machine.

TA_SENTBYT: 0 <= *num*
: Number of bytes sent from the source logical machine to the destination logical machine.

TA_RCVDNUM: 0 <= *num*
: Number of messages sent from the destination logical machine to the source logical machine.

TA_SENTNUM: 0 <= *num*
: Number of messages sent from the source logical machine to the destination logical machine.

TA_FLOWCNT: 0 <= *num*
: Number of times flow control has been encountered over this connection.

TA_CURENCRYPTBITS:{0|40|128}
: The current encryption level for this link. The level is negotiated between machines when the link is established.

Limitations None.

T_CLIENT CLASS

Overview

The `T_CLIENT` class represents run-time attributes of active clients within an application. These attribute values identify and track the activity of clients within a running application.

Attribute Table

T_CLIENT Class Definition Attribute Table

Attribute[1]	Type	Permissions	Values	Default
`TA_STATE( k )`	string	R-XR-XR--	GET:"`{ACT|SUS|DEA}`"	N/A
			SET:"`{ACT|SUS|DEA}`"	N/A
`TA_CLIENTID( * )`	string	R--R--R--	*string[1...78]*	N/A
`TA_CLTNAME( k )`	string	R--R--R--	*string[0...30]*	N/A
`TA_IDLETIME( k )`	long	R--R--R--	0 <= *num*	N/A
`TA_LMID( k )`	string	R--R--R--	*LMID*	N/A
`TA_PID( k )`	long	R--R--R--	1 <= *num*	N/A
`TA_SRVGRP( k )`	string	R--R--R--	*string[0...30]*	N/A
`TA_USRNAME( k )`	string	R--R--R--	*string[0...30]*	N/A
`TA_WSC( k )`	string	R--R--R--	"`{Y|N}`"	N/A
`TA_WSH( k )`	string	R--R--R--	"`{Y|N}`"	N/A
`TA_WSHCLIENTID( k )`	string	R--R--R--	*string[1...78]*	N/A
`TA_RELEASE`	long	R--R--R--	0 <= *num*	N/A
`TA_WSPROTO`	long	R--R--R--	0 <= *num*	N/A
`TA_NUMCONV`	long	R-XR-XR--	0 <= *num*	N/A
`TA_NUMDEQUEUE`	long	R-XR-XR--	0 <= *num*	N/A
`TA_NUMENQUEUE`	long	R-XR-XR--	0 <= *num*	N/A
`TA_NUMPOST`	long	R-XR-XR--	0 <= *num*	N/A
`TA_NUMREQ`	long	R-XR-XR--	0 <= *num*	N/A
`TA_NUMSUBSCRIBE`	long	R-XR-XR--	0 <= *num*	N/A
`TA_NUMTRAN`	long	R-XR-XR--	0 <= *num*	N/A
`TA_NUMTRANABT`	long	R-XR-XR--	0 <= *num*	N/A
`TA_NUMTRANCMT`	long	R-XR-XR--	0 <= *num*	N/A

T_CLIENT Class Definition Attribute Table

Attribute[1]	Type	Permissions	Values	Default
TA_CMTRET	string	R--R--R--	"{COMPLETE|LOGGED}"	N/A
TA_CURCONV	long	R--R--R--	0 <= *num*	N/A
TA_CURENCRYPTBIT	string	R--R-----	{ 0 | 40 | 128 }	N/A
TA_CURREQ	long	R--R--R--	0 <= *num*	N/A
TA_CURTIME	long	R--R--R--	1 <= *num*	N/A
TA_LASTGRP	long	R--R--R--	1 <= *num* < < 30,000	N/A
TA_NADDR	string	R--R--R--	*string[1...78]*	N/A
TA_NOTIFY	string	R--R--R--	"{DIPIN|SIGNAL|IGNORE}"	N/A
TA_NUMUNSOL	long	R--R--R--	0 <= *num*	N/A
TA_RPID	long	R--R--R--	1 <= *num*	N/A
TA_TIMELEFT	long	R--R--R--	0 <= *num*	N/A
TA_TIMESTART	long	R--R--R--	1 <= *num*	N/A
TA_TRANLEV	long	R--R--R--	0 <= *num*	N/A

(k) - GET key field
(*) - GET/SET key, one or more required for SET operations

[1]All attributes in Class T_CLIENT are local attributes.

Attribute Semantics

TA_STATE:

GET: {ACTive|SUSpended|DEAd}

A GET operation will retrieve run-time information for the selected T_CLIENT object(s). Note that client information is kept in local bulletin board tables only. Therefore, for maximum performance, inquiries on client status should be restricted using key fields as much as possible. The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

ACTive	T_CLIENT object active. This is not an indication of whether the client is idle or busy. A non 0 value retrieved for either the TA_CURCONV attribute or the TA_CURREQ attribute indicates a busy client.

SUSpended	T_CLIENT object active and suspended from making further service requests (tpcall (3c) or tpacall (3c)) and from initiating further conversations (tpconnect (3c)). See SET SUSpended below for details. This state is ACTive equivalent for the purpose of determining permissions.
DEAd	T_CLIENT object identified as active in the bulletin board but currently not running due to an abnormal death. This state will exist only until the BBL local to the client notices the death and takes action to clean up the client's bulletin board resources. This state is ACTive equivalent for the purpose of determining permissions.

SET: {ACTive|SUSpended|DEAd}

A SET operation will update run-time information for the selected T_CLIENT object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

ACTive	Activate a SUSpended T_CLIENT object. State change allowed only when in the SUSpended state. Successful return leaves the object in the ACTive state.
unset	Modify an existing T_CLIENT object. This combination is allowed only when in the ACTive or SUSpended state. Successful return leaves the object state unchanged.

SUSpended	Suspend the T_CLIENT object from making service requests (tpcall (3c) or tpacall (3c)), initiating conversations (tpconnect (3c)), beginning transactions (tpbegin (3c)), and enqueuing new requests (tpenqueue (3c)). Clients within a transaction will be permitted to make these calls until they abort or commit the current transaction, at which time they will become suspended. Invocations of these routines will result in a TPESYSTEM error return and a system log message being generated indicating the situation. State change allowed only when in the ACTive state. Successful return leaves the object in the SUSpended state.
DEAd	Abortively deactivate the T_CLIENT object. State change allowed only when in the ACTive or SUSpended state. The recommended method for deactivating clients is to first broadcast a warning message (tpbroadcast (3c)), then to suspend them (see SET SUSpended above), and finally to abortively deactivate them by setting the state to DEAd. Successful return leaves the object in the DEAd state. Limitation: Workstation handlers (T_CLIENT:TA_WSH == Y) may not be set to a state of DEAd. The system may not be able to *kill* the client due to platform or signaling restrictions. In this case, a native client will be abortively terminated at its next access to ATMI, and a workstation client's connection to a WSH will be preemptively torn down.

TA_CLIENTID: *string[1...78]*

Client identifier. The data in this field should not be interpreted directly by the end user except for equality comparison.

TA_CLTNAME: *string[0...30]*

Client name associated with client at tpinit (3c) time via the cltname element of the TPINIT structure.

TA_IDLETIME: 0 <= *num*

Approximate amount of time, in seconds, since this client last interacted with the system via an ATMI call. This value is accurate to within TA_SCANUNIT (see the T_DOMAIN class) seconds. When specified as a key field, a positive value indicates that all clients with idle times of at least the indicated value match, a negative value indicates that all clients with no more than the indicated value match, and a 0 value matches all clients.

TA_LMID: *LMID*

Logical machine where client is running (native clients) or where client is connected (workstation clients).

TA_PID: 1 <= *num*

Process identifier of client. Note that for workstation clients, this identifier indicates the workstation handler through which the workstation client is connected. A negative number may be specified on a GET operation for the purpose of retrieving client information for the calling process. If the calling process is not a client, then an error will be returned.

TA_SRVGRP: *string[0...30]*

Server group with which the client is associated. This information is set via the grpname element of the TPINIT structure at tpinit (3c) time.

TA_USRNAME: *string[0...30]*

User name associated with client at tpinit (3c) time via the usrname element of the TPINIT structure.

TA_WSC: {Y|N}

Workstation client. If this attribute is set to "Y", then the indicated client is logged in to the application from a remote workstation.

TA_WSH: {Y|N}

Workstation handler. If this attribute is set to "Y", then the indicated client is a workstation handler process.

TA_WSHCLIENTID: *string[1...78]*

Client identifier for the associated workstation handler (WSH) if this client is a workstation client (TA_WSH == Y); otherwise, this attribute will be returned as a 0-length string.

TA_RELEASE: 0 <= *num*

The BEA TUXEDO system major protocol release number for the machine where the client is running. This may be different from the TA_SWRELEASE

for the same machine. Note that for Workstation clients (TA_WSC == Y), this value may be different than the major release associated with the application administered machine through which the Workstation client accesses the application.

TA_WSPROTO: 0 <= *num*
: The BEA TUXEDO system Workstation protocol version number for a workstation client. This value is changed with each update to the Workstation protocol. A value of 0 is returned for this attribute when associated with non-Workstation clients (TA_WSC == N).

TA_NUMCONV: 0 <= *num*
: Number of conversations initiated by this client via tpconnect (3c).

TA_NUMDEQUEUE: 0 <= *num*
: Number of dequeue operations initiated by this client via tpdequeue (3c).

TA_NUMENQUEUE: 0 <= *num*
: Number of enqueue operations initiated by this client via tpenqueue (3c).

TA_NUMPOST: 0 <= *num*
: Number of postings initiated by this client via tppost (3c).

TA_NUMREQ: 0 <= *num*
: Number of requests made by this client via tpcall (3c) or tpacall (3c).

TA_NUMSUBSCRIBE: 0 <= *num*
: Number of subscriptions made by this client via tpsubscribe (3c).

TA_NUMTRAN: 0 <= *num*
: Number of transactions begun by this client.

TA_NUMTRANABT: 0 <= *num*
: Number of transactions aborted by this client.

TA_NUMTRANCMT: 0 <= *num*
: Number of transactions committed by this client.

TA_CMTRET: {COMPLETE|LOGGED}
: Setting of the TP_COMMIT_CONTROL characteristic for this client. See the description of the BEA TUXEDO system ATMI function tpscmt (3c) for details on this characteristic.

TA_CURCONV: 0 <= *num*

Number of conversations initiated by this client via tpconnect (3c) that are still active.

TA_CURENCRYPTBITS:{0|40|128}

The current encryption level for this client. The level is negotiated when the link is established.

TA_CURREQ: 0 <= *num*

Number of requests initiated by this client via tpcall (3c) or tpacall (3c) that are still active.

TA_CURTIME: 1 <= *num*

Current time, in seconds, since 00:00:00 UTC, January 1, 1970, as returned by the time(2) system call on T_CLIENT:TA_LMID. This attribute can be used to compute elapsed time from the T_CLIENT:TA_TIMESTART attribute value.

TA_LASTGRP: 1 <= *num* < 30,000

Server group number (T_GROUP:TA_GRPNO) of the last service request made or conversation initiated from this client.

TA_NADDR: *string[1...78]*

For workstation clients, this attribute indicates the network address of the client. Network addresses with unprintable characters will be converted to the "0x..." network address format as described in the T_MACHINE class for the T_NADDR attribute. If the address is a TCP/IP address, then it is returned in the dotted_decimal:port_number format:

"*//#.#.#.#:port_number*"

Each # represents a decimal number in the range 0 to 255. *Port_number* is a decimal number in the range 0 to 65535. Non-workstation clients will have a 0-length string associated with them for this attribute value. Limitation: The ability of the system to provide this information is determined by the transport provider in use. In some cases, workstation clients may not have addresses associated with them if the provider does not make this information available.

TA_NOTIFY: {DIPIN|SIGNAL|IGNORE}

Setting of the notification characteristic for this client. See the T_DOMAIN class description of this attribute for more details.

TA_NUMUNSOL: 0 <= *num*

Number of unsolicited messages queued for this client awaiting processing.

TA_RPID: 1 <= *num*

UNIX System message queue identifier for the client's reply queue. Limitation: This is a UNIX System specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_TIMELEFT: 0 <= *num*

Time left, in seconds, for this client to receive the reply for which it is currently waiting before it will timeout. This timeout may be a transactional timeout or a blocking timeout.

TA_TIMESTART: 1 <= *num*

Time, in seconds, since 00:00:00 UTC, January 1, 1970, as returned by the time(2) system call on T_CLIENT:TA_LMID, since the client joined the application.

TA_TRANLEV: 0 <= *num*

Current transaction level for this client. 0 indicates that the client is not currently involved in a transaction.

Limitations None.

T_CONN CLASS

Overview

The T_CONN class represents run-time attributes of active conversations within an application.

Attribute Table

T_CONN Class Definition Attribute Table

Attribute[1]	Type	Permissions	Values	Default
TA_LMID(k)	string	R--R--R--	*LMID*	N/A
TA_STATE(k)	string	R--R--R--	GET:"{ACT}"	N/A
			SET:N/A	N/A
TA_SERVICENAME	string	R--R--R--	*string[1...15]*	N/A
TA_CLIENTID(k)	string	R--R--R--	*string[1...78]*	N/A
TA_CONNOGRPNO	long	R--R--R--	1 <= *num* < 30,001	N/A
TA_CONNOLMID	string	R--R--R--	*LMID*	N/A
TA_CONNOPID	long	R--R--R--	1 <= *num*	N/A
TA_CONNOSNDCNT	long	R--R--R--	0 <= *num*	N/A
TA_CONNOSRVID	long	R--R--R--	1 <= *num* < 30,001	N/A
TA_CONNSGRPNO	long	R--R--R--	1 <= *num* < 30,001	N/A
TA_CONNSLMID	string	R--R--R--	*LMID*	N/A
TA_CONNSPID	long	R--R--R--	1 <= *num*	N/A
TA_CONNSSNDCNT	long	R--R--R--	0 <= *num*	N/A
TA_CONNSSRVID	long	R--R--R--	1 <= *num* < 30,001	N/A

(k) - GET key field

[1]All attributes in Class T_CONN are local attributes.

Attribute Semantics

TA_LMID: *LMID*

Retrieval machine logical machine identifier.

TA_STATE:

GET: {ACTive}

A GET operation will retrieve run-time information for the selected T_CONN object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

ACTive	The object returned reflects one or both sides of an active conversation within the application.

SET:

SET operations are not permitted on this class.

TA_SERVICENAME: *string[1...15]*

Service name of the conversational service invoked by the originator and processed by the subordinate.

TA_CLIENTID: *string[1...78]*

Client identifier. The data in this field should not be interpreted directly by the end user except for equality comparison.

TA_CONNOGRPNO: 1 <= *num* < 30,001

Server group number for the originator of the conversation. If the originator is a client, then 30,000 is returned as the value for this attribute.

TA_CONNOLMID: *LMID*

Logical machine identifier indicating where the originator is running or is accessing the application (in the case of Workstation clients).

TA_CONNOPID: 1 <= *num*

Process identifier for the originator of the conversation.

TA_CONNOSNDCNT: 0 <= *num*

Number of tpsend (3c) calls done by the originator.

TA_CONNOSRVID: 1 <= *num* < 30,001

Server identifier for the originator of the conversation.

TA_CONNSGRPNO: 1 <= *num* < 30,001

Server group number for the subordinate of the conversation.

TA_CONNSLMID: *LMID*

Logical machine identifier indicating where the subordinate is running or is accessing the application (in the case of Workstation clients).

TA_CONNSPID: 1 <= *num*

Process identifier for the subordinate in the conversation.

`TA_CONNSSNDCNT`: 0 <= *num*
: Number of `tpsend` (3c) calls done by the subordinate.

`TA_CONNSSRVID`: 1 <= *num* < 30,001
: Server identifier for the subordinate in the conversation.

limitations None.

T_DEVICE CLASS

Overview

The T_DEVICE class represents configuration and run-time attributes of raw disk slices or UNIX System files being used to store BEA TUXEDO system device lists. This class allows for the creation and deletion of device list entries within a raw disk slice or UNIX System file.

Attribute Table

T_DEVICE Class Definition Attribute Table

Attribute[1]	Type	Permissions	Values	Default
TA_LMID(*)	string	ru-r--r--	*LMID*	local_lmid
TA_CFGDEVICE(r)(*)	string	ru-r--r--	*string[2...64]*	N/A
TA_DEVICE(*)	string	ru-r--r--	*string[2...64]*	TA_CFGDEVICE
TA_DEVOFFSET(*)	long	ru-r--r--	0 <= *num*	0
TA_DEVSIZE(r)	long	rw-r--r--	0 <= *num*	1000[3]
TA_DEVINDEX(*)2	long	r--r--r--	0 <= *num*	N/A
TA_STATE(k)	string	rwxr--r--	GET:"{VAL}"	N/A
			SET:"{NEW|INV}"	N/A

(k) - GET key field
(r) - Required field for object creation (SET TA_STATE NEW)
(*) - GET/SET key, one or more required for SET operations

[1]All attributes in Class T_DEVICE are local attributes.

[2]TA_DEVINDEX is required for SET operations to identify the particular device list entry except when setting the state to NEW for the purpose of creating a new device list entry. In the latter case, TA_DEVINDEX must not be set; a value will be assigned by the system and returned after a successful creation.

[3]TA_DEVSIZE may only be SET on object creation.

Attribute Semantics

TA_LMID: *LMID*

Logical machine identifier where the device is located. Note that this attribute may be used as a key field in both unbooted and booted applications as long as they are already configured (that is, at least one T_MACHINE entry is defined). It is required as a key field on SET operations when accessing a booted application. If specified when accessing the T_DEVICE class in an unconfigured application, this attribute is ignored.

TA_CFGDEVICE: *string[2...64]*

Absolute pathname of the file or device where the BEA TUXEDO filesystem is stored or is to be stored.

TA_DEVICE: *string[2...64]*

Absolute pathname of the device list entry.

TA_DEVOFFSET: 0 <= *num*

The offset, in blocks, at which space on this TA_DEVICE begins for use within the BEA TUXEDO system VTOC specified by TA_CFGDEVICE. Limitation: This attribute must be set to 0 for the first device list entry (TA_DEVICE) on the BEA TUXEDO filesystem (TA_CFGDEVICE).

TA_DEVSIZE: 0 <= *num*

The size in pages of the disk area to be used for the device list entry. Limitation: This attribute may be set only in conjunction with a state change to NEW.

TA_DEVINDEX: 0 <= *num*

Device index for TA_DEVICE within the device list addressed by TA_CFGDEVICE. This attribute value is used for identification purposes only in getting and setting attribute values relating to particular devices within a BEA TUXEDO filesystem.

TA_STATE:

GET: {VALid}

A GET operation will retrieve run-time information for the selected T_DEVICE object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

VALid	The BEA TUXEDO filesystem indicated by TA_CFGDEVICE exists and contains a valid device list. TA_DEVICE is a valid device within that filesystem with the device index telnet lchome3.

SET: {NEW|INValid}

A SET operation will update information for the selected T_DEVICE object or add the indicated object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create or reinitialize T_DEVICE object for application. State change allowed only when in the INValid or VALid state. Successful return leaves the object in the VALid state. If this state transition is invoked in the INValid state, then the object is created; otherwise, it is reinitialized. The creation of the first TA_DEVICE device list entry on the TA_CFGDEVICE BEA TUXEDO filesystem will automatically create and initialize the necessary VTOC and UDL structures on TA_CFGDEVICE. The first device list entry created for a particular TA_CFGDEVICE must have equivalent values for the TA_DEVICE attribute.
INValid	Delete T_DEVICE object for application. State change allowed only when in the VALid state. Successful return leaves the object in the INValid state. Note that TA_DEVINDEX 0 is special and must be deleted last.

Limitations None.

T_DOMAIN CLASS

Overview

The T_DOMAIN class represents global application attributes. These attribute values serve to identify, customize, size, secure, and tune a BEA TUXEDO system application. Many of the attribute values represented here serve as application defaults for other classes represented in this MIB.

There is exactly one object of the T_DOMAIN class for each application. Because of this, there are no key fields defined for this class. A GET operation on this class will always return information representing this single object. Likewise, a SET operation will update it. GETNEXT is not permitted with this class.

Attribute Table

T_DOMAIN Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_IPCKEY(r)	long	rw-r--r--	32K+1 <= *num* < 262,144	N/A
TA_MASTER(r)	string	rwxr-xr--	"*LMID1[,LMID2]*"	N/A
TA_MODEL(r)	string	rw-r--r--	"{SHM|MP}"	N/A
TA_STATE	string	rwxr--r--	GET: "{ACT|INA}"	N/A
			SET:"{NEW|INV |ACT|INA|FIN}"	N/A
TA_DOMAINID	string	rwxr--r--	*string[0...30]*	""
TA_PREFERENCES	string	rwxr--r--	*string[0...1023]*	""
TA_UID	long	rwyr--r--	0 <= *num*	(1)
TA_GID	long	rwyr--r--	0 <= *num*	(1)
TA_PERM	long	rwyr--r--	0001 <= *num* <= 0777	0666
TA_LICEXPIRE	long	R--R--R--	*string[0 . . . 78]*	N/A
TA_LICMAXUSERS	long	R--R--R--	0 <= *num* < 32K	N/A
TA_LICSERIAL	string	R--R--R--	*string[0 . . . 78]*	N/A
TA_MIBMASK	long	rwx------	0 <= *num* <= 0777	0000
TA_MAXACCESSERS	long	rwyr--r--	1 <= *num* < 32K	50
TA_MAXCONV	long	rwyr--r--	0 <= *num* < 32K	10
TA_MAXGTT	long	rwyr--r--	0 <= *num* < 32K	100

T_DOMAIN Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_MAXBUFSTYPE	long	rw-r--r--	1 <= *num* < 32K	32
TA_MAXBUFTYPE	long	rw-r--r--	1 <= *num* < 32K	16
TA_MAXDRT	long	rw-r--r--	0 <= *num* < 32K	0
TA_MAXGROUPS	long	rw-r--r--	100 <= *num* < 32,766	100
TA_MAXNETGROUPS	long	rw-r--r--	1 <= *num* <= 8192	8
TA_MAXMACHINES	long	rw-r--r--	256 = *num* < 8K-1	256
TA_MAXQUEUES	long	rw-r--r--	1 <= *num* < 8K	50
TA_MAXRFT	long	rw-r--r--	0 <= *num* < 32,766	0
TA_MAXRTDATA	long	rw-r--r--	0 <= *num* < 32,761	0
TA_MAXSERVERS	long	rw-r--r--	1 <= *num* < 8K	50
TA_MAXSERVICES	long	rw-r--r--	1 <= *num* < 32,766	100
TA_MAXACLGROUPS	long	rw-r--r--	1 <= *num* = < 16K	16K
TA_CMTRET	string	rwyr--r--	"{COMPLETE|LOGGED}"	"COMPLETE"
TA_LDBAL	string	rwyr--r--	"{Y|N}"	"Y"
TA_NOTIFY	string	rwyr--r--	"{DIPIN|SIGNAL|IGNORE}"	"DIPIN"
TA_SYSTEM_ACCESS	string	rwyr--r--	"{FASTPATH|PROTECTED}[,NO_OVERRIDE]"	"FASTPATH"
TA_OPTIONS	string	rwyr--r--	"{LAN|MIGRATE|ACCSTATS],*}"	""
TA_USIGNAL	string	rw-r--r--	"{SIGUSR1|SIGUSR2}"	"SIGUSR2"
TA_SECURITY	string	rw-r--r--	{""|"NONE|APP_PW|USER_AUTH|ACL|MANDATORY_ACL"}	"NONE"
TA_PASSWORD	string	-wx------	*string*[0...30]	N/A
TA_AUTHSVC	string	rwxr--r--	*string*[0...15]	""
TA_SCANUNIT	long	rwxr-xr--	0 <= *num* <= 60	10^2
TA_BBLQUERY	long	rwxr-xr--	0 <= *num* 32K	300^3
TA_BLOCKTIME	long	rwxr-xr--	0 <= *num* < 32K	60^3
TA_DBBLWAIT	long	rwxr-xr--	0 <= *num* < 32K	20^3
TA_SANITYSCAN	long	rwxr-xr--	0 <= *num* < 32K	120^3

T_DOMAIN Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_CURDRT	long	R--R--R--	0 <= *num* < 32K	N/A
TA_CURGROUPS	long	R--R--R--	0 <= *num* < 32K	N/A
TA_CURMACHINES	long	R--R--R--	0 <= *num* < 32K	N/A
TA_CURQUEUES	long	R--R--R--	0 <= *num* < 32K	N/A
TA_CURRFT	long	R--R--R--	0 <= *num* < 32K	N/A
TA_CURRTDATA	long	R--R--R--	0 <= *num* < 32K	N/A
TA_CURSERVERS	long	R--R--R--	0 <= *num* < 32K	N/A
TA_CURSERVICES	long	R--R--R--	0 <= *num* < 32K	N/A
TA_CURSTYPE	long	R--R--R--	0 <= *num* < 32K	N/A
TA_CURTYPE	long	R--R--R--	0 <= *num* < 32K	N/A
TA_HWDRT	long	R--R--R--	0 <= *num* < 32K	N/A
TA_HWGROUPS	long	R--R--R--	0 <= *num* < 32K	N/A
TA_HWMACHINES	long	R--R--R--	0 <= *num* < 32K	N/A
TA_HWQUEUES	long	R--R--R--	0 <= *num* < 32K	N/A
TA_HWRFT	long	R--R--R--	0 <= *num* < 32K	N/A
TA_HWRTDATA	long	R--R--R--	0 <= *num* < 32K	N/A
TA_HWSERVERS	long	R--R--R--	0 <= *num* < 32K	N/A
TA_HWSERVICES	long	R--R--R--	0 <= *num* < 32K	N/A

(r) - Required field for object creation (SET TA_STATE NEW)

[1] UID and GID as known to the UNIX System

[2] *num* must be a multiple of 5

[3] Specify *num* so that *num* times TA_SCANUNIT is approximately "Default"

Attribute Semantics

TA_IPCKEY: 32K+1 <= *num* < 262,144

Numeric key for the well-known address in a BEA TUXEDO system bulletin board. In a single processor environment, this key "names" the bulletin board. In a multiple processor or LAN environment, this key names the message queue of the DBBL. In addition, this key is used as a basis for deriving the names of resources other than the well-known address, such as the names for bulletin boards throughout the application.

TA_MASTER: *LMID1[,LMID2]*

Master (*LMID1*) and backup (*LMID2*) logical machine identifiers. The master identifier (*LMID1*) must correspond to the local machine for INActive applications. SHM mode applications (see TA_MODEL below) may set only the master logical machine identifier. Modifications to this attribute value in an ACTive MP application (see TA_MODEL below) have the following semantics:

Assuming current active master LMID A, current backup master LMID B, and secondary LMIDs C, D, . . ., the following scenarios define the semantics of permitted changes to the TA_MASTER attribute in a running MP mode application.

```
A,B -> B,A - Master migration from A to B.

A,B -> A,C - Change backup master LMID designation to C.
```

Note that master migration may be either orderly or partitioned. Orderly migration takes place when the master machine is ACTive and reachable. Otherwise, partitioned migration takes place. All newly established or reestablished network connections will verify that the two sites connecting share a common view of where the master machine is. Otherwise, the connection will be refused and an appropriate log message generated. The master and backup machines in an ACTive application must always have a BEA TUXEDO system release number greater than or equal to all other machines active in the application. The master and backup machines must be of the same release. Modifications to the TA_MASTER attribute must preserve this relationship.

TA_MODEL: {SHM|MP}

Configuration type. SHM specifies a single machine configuration; only one T_MACHINE object may be specified. MP specifies a multi-machine or network configuration; MP must be specified if a networked application is being defined.

TA_STATE:

GET: {ACTive|INActive}

A GET operation will retrieve configuration and run-time information for the T_DOMAIN object. The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

ACTive	T_DOMAIN object defined and the master machine is active.
INActive	T_DOMAIN object defined and application is inactive.

SET: {NEW|INValid|ACTive|INActive|FINactive}

A SET operation will update configuration and run-time information for the T_DOMAIN object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_DOMAIN object for application. State change allowed only when in the INValid state. Successful return leaves the object in the INActive state. Note that this state change will also create a NEW T_MACHINE object with TA_LMID inferred from TA_MASTER, TA_PMID based on the local system name, and TA_TUXCONFIG and TA_TUXDIR determined from the environment variables TUXCONFIG and TUXDIR respectively. Other configurable attributes of the T_MACHINE class may be set at this time by including values in the T_DOMAIN NEW request. If a value for TA_APPDIR is not specified, then it will default to the current directory.
unset	Modify T_DOMAIN object. Allowed only when in the ACTive or INActive state. Successful return leaves the object state unchanged.
INValid	Delete T_DOMAIN object for application. State change allowed only when in the INActive state. Successful return leaves the object in the INValid state.

ACTive	Activate administrative processes (DBBL, BBL, etc.) on the master machine. For the purpose of determining permissions for this state transition, the active object permissions are considered (that is, --x--x--x). State change allowed only when in the INActive state. Successful return leaves the object in the INActive state.
INActive	Deactivate administrative processes (DBBL, BBL, etc.) on the master machine. State change allowed only when in the ACTive state. Successful return leaves the object in the INActive state.
FINactive	Forcibly deactivate administrative processes (DBBL, BBL, etc.) on the master machine. Attached clients will be ignored for the purpose of determining if shutdown should be allowed. State change allowed only when in the ACTive state. Successful return leaves the object in the INActive state.

TA_DOMAINID: *string[0...30]*
: Domain identification string.

TA_PREFERENCES: *string[0...1023]*
: Application defined field. This field is used by the BEA TUXEDO system /Admin GUI product to store and save GUI display preferences.

TA_UID: 0 <= *num*
: Default attribute setting for newly configured objects in the T_MACHINE class. Limitation: Changes to this attribute do not affect active or already configured T_MACHINE objects.

TA_GID: 0 <= *num*
: Default attribute setting for newly configured objects in the T_MACHINE class. Limitation: Changes to this attribute do not affect active or already configured T_MACHINE objects.

TA_PERM: 0001 <= *num* <= 0777
: Default attribute setting for newly configured objects in the T_MACHINE class. Limitation: Changes to this attribute do not affect active or already configured T_MACHINE objects.

TA_LICEXPIRE: *string[0 . . . 78]*
: Expiration date for the binary on that machine or a 0-length string if binary is not a BEA TUXEDO system master binary.

TA_LICMAXUSERS: 0 <= *num* < 32K
: Licensed maximum number of users on that machine or -1 if binary is not a BEA TUXEDO system master binary.

TA_LICSERIAL: *string [0 . . . 78]*
: Serial number of license.

TA_MIBMASK: 0 <= *num* <= 0777
: Attribute access mask. User type/access mode combinations specified by this attribute value will no longer be allowed for all class/attribute combinations defined in this reference page. For example, a setting of 0003 disallows all updates to users other than the administrator or the operator.

TA_MAXACCESSERS: 1 <= *num* < 32K
: Default attribute setting for newly configured objects in the T_MACHINE class. Limitation: Changes to this attribute do not affect active or already configured T_MACHINE objects.

TA_MAXCONV: 0 <= *num* < 32K
: Default attribute setting for newly configured objects in the T_MACHINE class. Limitation: Changes to this attribute do not affect active or already configured T_MACHINE objects.

TA_MAXGTT: 0 <= *num* < 32K
: Default attribute setting for newly configured objects in the T_MACHINE class. Limitation: Changes to this attribute do not affect active or already configured T_MACHINE objects.

TA_MAXBUFSTYPE: 1 <= *num* < 32K
: Maximum number of buffer subtypes that can be accommodated in the bulletin board buffer subtype table.

TA_MAXBUFTYPE: 1 <= *num* < 32K
: Maximum number of buffer types that can be accommodated in the bulletin board buffer type table.

TA_MAXDRT: 0 <= *num* < 32K
: Maximum number of routing table entries that can be accommodated in the bulletin board routing table. One entry per T_ROUTING class object is required. Additional entries should be allocated to allow for run-time growth.

TA_MAXGROUPS: 100 <= *num* < 32,766

Maximum number of server groups that can be accommodated in the bulletin board server group table.

Limitation: BEA TUXEDO System Release 4.2.2 and earlier sites have a fixed setting of 100 for this attribute. Interoperability with these sites requires that no more than 100 server group entries be in use at any time. Release 4.2.2 and earlier sites will not be allowed to join an application that has more than 100 defined server groups. Additionally, applications already including Release 4.2.2 or earlier sites will not be allowed to add server groups beyond 100.

TA_MAXNETGROUPS: 1 <= *num* < 8192

specifies the maximum number of configured network groups to be accommodated in the NETWORK section of the TUXCONFIG file. This value must be greater than or equal to 1 and less than 8192. If not specified, the default is 8.

TA_MAXMACHINES: 256 = *num* 8K-1

Maximum number of machines that can be accommodated in the bulletin board machine table.

Limitation: BEA TUXEDO system Release 4.2.2 has a fixed setting of 256 for this attribute. Releases prior to Release 4.2.2 have a fixed setting of 50 for this attribute. Interoperability with Release 4.2.2 and earlier sites requires that no more than the lowest fixed setting number of machine table entries be in use at any time. Release 4.2.2 sites will not be allowed to join an application that has more than 256 defined machines. Pre-Release 4.2.2 sites will not be allowed to join an application that has more than 50 defined machines. Additionally, applications already including active Release 4.2.2 or earlier sites will not be allowed to add machines beyond the lowest applicable limit.

TA_MAXQUEUES: 1 <= *num* 8K

Maximum number of queues to be accommodated in the bulletin board queue table.

Limitation: Release 4.2.2 and earlier sites may join an active application only if the setting for TA_MAXQUEUES is equal to the setting for TA_MAXSERVERS.

TA_MAXRFT: 0 <= *num* < 32K

Maximum number of routing criteria range table entries to be accommodated in the bulletin board range criteria table. One entry per individual range within a TA_RANGES specification is required plus one additional entry per T_ROUTING class object. Additional entries should be allocated to allow for run-time growth.

TA_MAXRTDATA: 0 <= *num* < 32,761

Maximum string pool space to be accommodated in the bulletin board string pool table. Strings and carrays specified within TA_RANGES values are stored in the string pool. Additional space should be allocated to allow for run-time growth.

TA_MAXSERVERS: 1 <= *num*< 8K

Maximum number of servers to be accommodated in the bulletin board server table. Allowances should be made in setting this attribute for system supplied administrative servers. Administration of each BEA TUXEDO system site adds approximately one server. Additionally, if TMSs are specified for any server groups (see T_GROUP TA_TMSNAME attribute), then they will be booted along with their server group and should be accounted for in setting TA_MAXSERVERS.

TA_MAXSERVICES: 1 <= *num* < 32766

Maximum number of services to be accommodated in the bulletin board service table. Allowances should be made in setting this attribute for system supplied servers offering services for administrative purposes. Administration of each BEA TUXEDO system site adds approximately five services. Other administrative components such as Workstation, /Q, and /DM may also add administrative services that should be accounted for.

TA_MAXACLGROUPS: 1 <= *num* < 16K

Maximum number of group identifiers that can be used for ACL permissions checking. The maximum group identifier that can be defined is TA_MAXACLGROUPS - 1.

TA_CMTRET: {COMPLETE|LOGGED}

Initial setting of the TP_COMMIT_CONTROL characteristic for all client and server processes in a BEA TUXEDO system application. LOGGED initializes the TP_COMMIT_CONTROL characteristic to TP_CMT_LOGGED; otherwise, it is initialized to TP_CMT_COMPLETE. See the description of the BEA TUXEDO system ATMI function tpscmt (3c) for details on the setting of this characteristic.

Limitation: Run-time modifications to this attribute do not affect active clients and servers.

TA_LDBAL: {Y|N}

Load balancing is/will be on ("Y") or off ("N").
Limitation: Run-time modifications to this attribute do not affect active clients and servers.

TA_NOTIFY: {DIPIN|SIGNAL|IGNORE}

Default notification detection method to be used by the system for unsolicited messages sent to client processes. This default can be overridden on a per-client basis using the appropriate tpinit (3c) flag value. Note that once unsolicited messages are detected, they are made available to the application through the application defined unsolicited message handling routine identified via the tpsetunsol (3c) function.

The value DIPIN specifies that dip-in-based notification detection should be used. This means that the system will only detect notification messages on behalf of a client process while within ATMI calls. The point of detection within any particular ATMI call is not defined by the system, and dip-in detection will not interrupt blocking system calls. DIPIN is the default notification detection method.

The value SIGNAL specifies that signal-based notification detection should be used. This means that the system sends a signal to the target client process after the notification message has been made available. The system installs a signal catching routine on behalf of clients selecting this method of notification.

The value IGNORE specifies that by default, notification messages are to be ignored by application clients. This would be appropriate in applications where only clients that request notification at tpinit (3c) time should receive unsolicited messages.

Limitations: Run-time modifications to this attribute do not affect active clients. All signaling of client processes is done by administrative system processes and not by application processes. Therefore, only clients running with the same UNIX System user identifier can be notified using the SIGNAL method.

TA_SYSTEM_ACCESS: {FASTPATH|PROTECTED}[,NO_OVERRIDE]

Default mode used by BEA TUXEDO system libraries within application processes to gain access to BEA TUXEDO system's internal tables. FASTPATH specifies that BEA TUXEDO system's internal tables are accessible by BEA TUXEDO system libraries via unprotected shared memory for fast access. PROTECTED specifies that BEA TUXEDO system's internal tables are accessible by BEA TUXEDO system libraries via protected shared memory for safety against corruption by application code. NO_OVERRIDE can be specified to indicate that the mode selected cannot be

overridden by an application process using flags available for use with `tpinit` (3c).

Limitations: Updates to this attribute value in a running application affect only newly started clients and newly configured `T_SERVER` objects.

`TA_OPTIONS`: {`LAN`|`MIGRATE`|`ACCSTATS`],*}
Comma separated list of application options in effect. Valid options are defined below:

`LAN` - Networked application.

`MIGRATE` - Allow server group migration.

`ACCSTATS` - Exact statistics (SHM mode only).

Limitation: Only the `ACCSTATS` may be set or reset in an active application.

`TA_USIGNAL`: {`SIGUSR1`|`SIGUSR2`}
Signal to be used for signal-based notification (see `TA_NOTIFY` above).

`TA_SECURITY`: {`NONE`|`APP_PW`|`USER_AUTH`|`ACL`|`MANDATORY_ACL`}
Type of application security. A 0-length string value or `NONE` for this attribute indicates that security is/will be turned off. The identifier `APP_PW` indicates that application password security is to be enforced (clients must provide the application password during initialization). Setting this attribute requires a non-0 length `TA_PASSWORD` attribute. The identifier `USER_AUTH` is similar to `APP_PW` but, in addition, indicates that per-user authentication will be done during client initialization. The identifier `ACL` is similar to `USER_AUTH` but, in addition, indicates that access control checks will be done on service names, queue names, and event names. If an associated ACL is not found for a name, it is assumed that permission is granted. The identifier `MANDATORY_ACL` is similar to `ACL` but permission is denied if an associated ACL is not found for the name.

`TA_PASSWORD`: *`string`*[`0...30`]
Clear text application password. This attribute is ignored if the `TA_SECURITY` attribute is set to nothing. The system automatically encrypts this information on behalf of the administrator.

`TA_AUTHSVC`: *string[0...15]*
Application authentication service invoked by the system for each client joining the system. This attribute is ignored if the `TA_SECURITY` attribute is set to nothing or to `APP_PW`.

TA_SCANUNIT: 0 <= *num* <= 60 (multiple of 5)

Interval of time (in seconds) between periodic scans by the system. Periodic scans are used to detect old transactions and timed-out blocking calls within service requests. The TA_BBLQUERY, TA_BLOCKTIME, TA_DBBLWAIT, and TA_SANITYSCAN attributes are multipliers of this value. Passing a value of 0 for this attribute on a SET operation will cause the attribute to be reset to its default.

TA_BBLQUERY: 0 <= *num* < 32K

Multiplier of the TA_SCANUNIT attribute indicating time between DBBL status checks on registered BBLs. The DBBL checks to ensure that all BBLs have reported in within the TA_BBLQUERY cycle. If a BBL has not been heard from, the DBBL sends a message to that BBL asking for status. If no reply is received, the BBL is partitioned. Passing a value of 0 for this attribute on a SET operation will cause the attribute to be reset to its default. This attribute value should be set to at least twice the value set for the TA_SANITYSCAN attribute value (see below).

TA_BLOCKTIME: 0 <= *num* < 32K

Multiplier of the TA_SCANUNIT attribute indicating the minimum amount of time a blocking ATMI call will block before timing out. Passing a value of 0 for this attribute on a SET operation will cause the attribute to be reset to its default.

TA_DBBLWAIT: 0 <= *num* < 32K

Multiplier of the TA_SCANUNIT attribute indicating maximum amount of time a DBBL should wait for replies from its BBLs before timing out. Passing a value of 0 for this attribute on a SET operation will cause the attribute to be reset to its default.

TA_SANITYSCAN: 0 <= *num* < 32K

Multiplier of the TA_SCANUNIT attribute indicating time between basic sanity checks of the system. Sanity checking includes client/server viability checks done by each BBL for clients/servers running on the local machine as well as BBL status check-ins (MP mode only). Passing a value of 0 for this attribute on a SET operation will cause the attribute to be reset to its default.

TA_CURDRT: 0 <= *num* < 32K

Current number of in use bulletin board routing table entries.

TA_CURGROUPS: 0 <= *num* < 32K

Current number of in use bulletin board server group table entries.

TA_CURMACHINES: 0 <= *num* < 32K
: Current number of configured machines.

TA_CURQUEUES: 0 <= *num* < 32K
: Current number of in use bulletin board queue table entries.

TA_CURRFT: 0 <= *num* < 32K
: Current number of in use bulletin board routing criteria range table entries.

TA_CURRTDATA: 0 <= *num* < 32K

: Current size of routing table string pool.

TA_CURSERVERS: 0 <= *num* < 32K
: Current number of in use bulletin board server table entries.

TA_CURSERVICES: 0 <= *num* < 32K
: Current number of in use bulletin board service table entries.

TA_CURSTYPE: 0 <= *num* < 32K
: Current number of in use bulletin board subtype table entries.

TA_CURTYPE: 0 <= *num* < 32K
: Current number of in use bulletin board type table entries.

TA_HWDRT: 0 <= *num* < 32K
: High water number of in use bulletin board routing table entries.

TA_HWGROUPS: 0 <= *num* < 32K
: High water number of in use bulletin board server group table entries.

TA_HWMACHINES: 0 <= *num* < 32K
: High water number of configured machines.

TA_HWQUEUES: 0 <= *num* < 32K
: High water number of in use bulletin board queue table entries.

TA_HWRFT: 0 <= *num* < 32K
: High water number of in use bulletin board routing criteria range table entries.

TA_HWRTDATA: 0 <= *num* < 32K
: High water size of routing table string pool.

TA_HWSERVERS: 0 <= *num* < 32K
: High water number of in use bulletin board server table entries.

TA_HWSERVICES: 0 <= *num* < 32K
: High water number of in use bulletin board service table entries.

Limitations

Many attributes of this class are tunable only when the application is inactive. Therefore, use of the ATMI interface routines to administer the application is not possible. The function `tpadmcall` (3c) is being provided as a means of configuring or reconfiguring an unbooted application. This interface may only be used for configuration (`SET` operations) in an inactive application and only on the site being configured as the master site for the application. Once an initial configuration is created and activated, administration is available through the standard ATMI interfaces as described in `MIB`(5).

T_GROUP CLASS

Overview

The T_GROUP class represents application attributes pertaining to a particular server group. These attribute values represent group identification, location, and DTP information.

Attribute Table

T_GROUP Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_SRVGRP(r)(*)	string	rU-r--r--	*string[1...30]*	N/A
TA_GRPNO(k)(r)	long	rU-r--r--	1 <= *num* < 30,000	N/A
TA_LMID(k)(r)	string	rwyr--r--	"*LMID1[,LMID2]*"	N/A
TA_STATE(k)	string	rwxr-xr--	GET:"{ACT|INA|MIG}"	N/A
			SET:"{NEW|INV|ACT|RAC|INA|MIG}"	N/A
TA_CURLMID(k)	string	R--R--R--	*LMID*	N/A
TA_ENVFILE	string	rwyr--r--	*string[0...78]*	""
TA_CLOSEINFO	string	rwyr--r--	*string[0...256]*	""
TA_OPENINFO	string	rwyr--r--	*string[0...256]*	""
TA_TMSCOUNT	long	rw-r--r--	0 or 2 = *num* < 11	3
TA_TMSNAME(k)	string	rw-r--r--	*string[0...78]*	""

(k) - GET key field
(r) - Required field for object creation (SET TA_STATE NEW)
(*) - GET/SET key, one or more required for SET operations

Attribute Semantics

TA_SRVGRP: *string[1...30]*

Logical name of the server group. The group name must be unique within all group names in the T_GROUP class and TA_LMID values in the T_MACHINE class. Server group names cannot contain an asterisk (*), comma, or colon.

TA_GRPNO: 1 <= *num* < 30,000

Group number associated with this server group.

TA_LMID: *LMID1[,LMID2]*

Primary machine logical machine identifier for this server group (*LMID1*) and optional secondary logical machine identifier (*LMID2*). The secondary LMID indicates the machine to which the server group can be migrated (if the MIGRATE option is specified in the T_DOMAIN:TA_OPTIONS attribute). A single LMID specified on a GET operation will match either the primary or secondary LMID. Note that the location of an active group is available in the

TA_CURLMID attribute. Logical machine identifiers specified with the TA_LMID attribute must be already configured. Limitation: Modifications to this attribute for an active object may only change the backup LMID designation for the group.

TA_STATE:

GET: {ACTive|INActive|MIGrating}

A GET operation will retrieve configuration and run-time information for the selected T_GROUP object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

ACTive	T_GROUP object defined and active (TMS and/or application servers). Server groups with non 0-length values for the TA_TMSNAME attribute are considered active if the TMSs associated with the group are active. Otherwise, a group is considered active if any server in the group is active.
INActive	T_GROUP object defined and inactive.
MIGrating	T_GROUP object defined and currently in a state of migration to the secondary logical machine. The secondary logical machine is the one listed in TA_LMID that does not match TA_CURLMID. This state is ACTive equivalent for the purpose of determining permissions.

SET: {NEW|INValid|ACTive|ReACtivate|INActive|MIGrating}

A SET operation will update configuration and run-time information for the selected T_GROUP object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_GROUP object for application. State change allowed only when in the INValid state. Successful return leaves the object in the INActive state.
unset	Modify an existing T_GROUP object. This combination is allowed only when in the ACTive or INActive state. Successful return leaves the object state unchanged.

INValid	Delete T_GROUP object for application. State change allowed only when in the INActive state. Successful return leaves the object in the INValid state.
ACTive	Activate the T_GROUP object. State change allowed only when in the INActive or MIGrating state. For the purpose of determining permissions for this state transition, the active object permissions are considered (that is, --x--x--x). If the group is currently in the INActive state, then TMS and application servers (subject to restriction by TA_FLAGS settings) are started on the primary logical machine if the primary logical machine is active; otherwise, the TMS and application servers are started on the secondary logical machine if it is active. If neither machine is active, then the request fails. If the group is currently in the MIGrating state, then the active secondary logical machine (identified as the alternate to TA_CURLMID in the TA_LMID list) is used to start TMS and application servers if it is active. Otherwise, the request fails. ~The TMIB_NOTIFY TA_FLAG value should be used when activating a server group if status on individual servers is required. Successful return leaves the object in the ACTive state.
ReACtivate	Identical to a transition to the ACTive state except that this state change is also allowed in the ACTive state in addition to being allowed in the INActive and MIGrating states. The TMIB_NOTIFY TA_FLAG value should be used when reactivating a server group if status on individual servers is required.
INActive	Deactivate the T_GROUP object. TMS and application servers (subject to restriction by TA_FLAGS settings) are deactivated. State change allowed only when in the ACTive or MIGrating state. Successful return leaves the object in the INActive state. The TMIB_NOTIFY TA_FLAG value should be used when deactivating a server group if status on individual servers is required.

MIGrating	Deactivate the T_GROUP object on its active primary logical machine (TA_CURLMID) and prepare the group to be migrated to the secondary logical machine. State change allowed only when in the ACTive state. Successful return leaves the object in the MIGrating state.
UnAVailable	Suspend all application services in the group. (Note: Individual services can be suspended through the T_SVCGROUP class.) A SET operation to this state is allowed only when the group is in the ACTive state. The operation leaves the group in the ACTive state, but with all its application services in a suspended state. Limitation: Operation will fail in a mixed-release application where any pre-release 6.4 machine is active.
AVaiLable	Unsuspend all application services in the group marked as suspended. A SET operation to this state value is allowed only when the group is in the ACTive state. The operation leaves the group in the ACTive state.

Limitation: Operation will fail in a mixed-release application where any pre-release 6.4 machine is active.

TA_CURLMID: *LMID*

Current logical machine on which the server group is running. This attribute will not be returned for server groups that are not active.

TA_ENVFILE: *string[0...78]*

Environment file for servers running in this group. If the value specifies an invalid file name, no values are added to the environment. the value of *string* is placed in the environment.

When booted, local servers inherit the environment of tmboot(1) and remote servers (not on the MASTER) inherit the environment of tlisten(1). TUXCONFIG, TUXDIR, and APPDIR are also put in the environment when a server is booted based on the information in the associated T_GROUP object.

PATH is set in the environment to

```
APPDIR:TUXDIR/bin:/bin:/usr/bin:<path>
```

where <*path*> is the value of the first PATH= line in the machine environment file, if one exists (subsequent PATH= lines is ignored). This PATH is used as a

search path for servers that are specified with a simple or relative pathname (that is, one that doesn't begin with slash).

LD_LIBRARY_PATH is set in the environment to

```
APPDIR:TUXDIR/lib:/lib:/usr/lib:<lib>
```

where <*lib*> is the value of the first LD_LIBRARY_PATH= line appearing in the machine environment file, if one exists (subsequent LD_LIBRARY_PATH= lines are ignored).

As part of server initialization (before tpsvrinit (3c) is called), a server reads and exports variables from both the machine and server ENVFILE files. If a variable is set in both the machine and server ENVFILE, the value in the server ENVFILE will override the value in the machine ENVFILE with the exception of PATH which is appended. A client processes only the machine ENVFILE file. When the machine and server ENVFILE files are processed, lines that are not of the form <*ident*>= is ignored, where <*ident*> contains only underscore or alphanumeric characters.

If a PATH= line is encountered, PATH is set to

```
APPDIR:TUXDIR/bin:/bin:/usr/bin:<path>
```

where <*path*> is the value of the first PATH= line appearing in the environment file (subsequent PATH= lines are ignored). If PATH appears in both the machine and server files, then <*path*> is defined as <*path1*>:<*path2*> where <*path1*> is from the machine ENVFILE and <*path2*> is from the server ENVFILE. If a LD_LIBRARY_PATH= line is encountered, LD_LIBRARY_PATH is set to

```
APPDIR:TUXDIR/lib:/lib:/usr/lib:<lib>
```

where <*lib*> is the value of the first LD_LIBRARY_PATH= line appearing in the environment file (subsequent LD_LIBRARY_PATH= lines are ignored). Attempts to reset TUXDIR, APPDIR, or TUXCONFIG are ignored and a warning is displayed if the value does not match the corresponding T_GROUP attribute value. Limitation: Modifications to this attribute for an active object DO not affect running servers or clients.

TA_CLOSEINFO: *string[0...256]*

If a non 0-length value other than "TMS" is specified for the TA_TMSNAME attribute, then this attribute value indicates the resource manager-dependent information needed when terminating access to the resource manager. Otherwise, this attribute value is ignored.

The format for this attribute value is dependent on the requirements of the vendor providing the underlying resource manager. The information required by the vendor must be prefixed with "*rm_name*:", which is the published name of the vendor's transaction (XA) interface followed immediately by a colon (:).

A 0-length string value for this attribute means that the resource manager for this group (if specified) does not require any application specific information to `close` access to the resource.

Limitation: Run-time modifications to this attribute will not affect active servers in the group.

`TA_OPENINFO`: *string[0...256]*

If a non 0-length value other than "`TMS`" is specified for the `TA_TMSNAME` attribute, then this attribute value indicates the resource manager dependent information needed when initiating access to the resource manager. Otherwise, this attribute value is ignored.

The format for this attribute value is dependent on the requirements of the vendor providing the underlying resource manager. The information required by the vendor must be prefixed with ``*rm_name*:'', which is the published name of the vendor's transaction (XA) interface followed immediately by a colon (:).

A 0-length string value for this attribute means that the resource manager for this group (if specified) does not require any application specific information to `open` access to the resource.

Limitation: Run-time modifications to this attribute will not affect active servers in the group.

`TA_TMSCOUNT`: 0 or 2 <= *num* < 11

If a non 0-length value is specified for the `TA_TMSNAME` attribute, then this attribute value indicates the number of transaction manager servers to start for the associated group. Otherwise, this attribute value is ignored.

`TA_TMSNAME`: *string[0...78]*

Transaction manager server `a.out` associated with this group. This parameter must be specified for any group entry whose servers will participate in distributed transactions (transactions across multiple resource managers and possibly machines that are started with `tpbegin`(3), and ended with `tpcommit`(3)/`tpabort`(3)).

The value "TMS" is reserved to indicate use of the null XA interface. If a non-empty value other than "TMS" is specified, then a TLOGDEVICE must be specified for the machine(s) associated with the primary and secondary logical machines for this object.

A unique server identifier is selected automatically for each TM server, and the servers will be restartable an unlimited number of times.

Limitations None.

T_MACHINE CLASS

Overview

The T_MACHINE class represents application attributes pertaining to a particular machine. These attribute values represent machine characteristics, per-machine sizing, statistics, customization options, and UNIX System filenames.

Attribute Table

T_MACHINE Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_LMID(r)(*)1	string	rU-r--r--	*string[1...30]*	N/A
TA_PMID(r)(*)1	string	rU-r--r--	*string[1...30]*	N/A
TA_TUXCONFIG(r)	string	rw-r--r--	*string[2...64]*	N/A
TA_TUXDIR(r)	string	rw-r--r--	*string[2...78]*	N/A
TA_APPDIR(r)	string	rw-r--r--	*string[2...78]*	N/A
TA_STATE(k)	string	rwyr-yr--	GET:"{ACT|INA|PAR}" SET:"{NEW|INV|ACT|RAC|INA|FIN|CLE}"	N/A N/A
TA_UID	long	rw-r--r--	0 <= *num*	(2)
TA_GID	long	rw-r--r--	0 <= *num*	(2)
TA_ENVFILE	string	rwyr--r--	*string[0...78]*	""
TA_PERM	long	rwyr--r--	0001 <= *num* <= 0777	(2)
TA_ULOGPFX	string	rwyr--r--	*string[0...78]*	(3)
TA_TYPE	string	rw-r--r--	*string[0...15]*	""
TA_MAXACCESSERS	long	rw-r--r--	1 <= *num* < 32K	(2)
TA_MAXCONV	long	rw-r--r--	0 <= *num* < 32K	(2)
TA_MAXGTT	long	rw-r--r--	0 <= *num* < 32K	(2)
TA_MAXWSCLIENTS	long	rw-r--r--	0 <= *num* < 32K	0
TA_MAXACLCACHE	long	rw-r--r--	10 <= *num* <= 32000	100
TA_TLOGDEVICE	string	rw-r--r--	*string[0...64]*	""
TA_TLOGNAME	string	rw-r--r--	*string[0...30]*	TLOG
TA_TLOGSIZE	long	rw-r--r--	1 <= *num* < 2K+1	100

T_MACHINE Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_BRIDGE	string	rw-r--r--	*string[0...78]*	N/A
TA_NADDR	string	rw-r--r--	*string[0...78]*	N/A
TA_NLSADDR	string	rw-r--r--	*string[0...78]*	N/A
TA_CMPLIMIT	string	rwyr-yr--	"*remote[,local]*"	MAXLONG,MAXLONG
TA_TMNETLOAD	long	rwyr-yr--	0 <= *num* < 32K	0
TA_SPINCOUNT	long	rwyr-yr--	0 <= *num*	0
TA_ROLE	string	r--r--r--	"{MAS-TER|BACKUP|OTHER}"	N/A
TA_MINOR	long	R--R--R--	1 <= *num*	N/A
TA_RELEASE	long	R--R--R--	1 <= *num*	N/A
TA_MINENCRYPTBIT	string	rwxrwx---	{ 0 | 40 | 128 }	0
TA_MAXENCRYPTBIT	string	rwxrwx---	{ 0 | 40 | 128 }	128
TA_MAXPENDINGBYTES	long	rw-r--r--	100000 <= *num* <= MAXLONG	2147483647

(k) - GET key field
(r) - Required field for object creation (SET TA_STATE NEW)
(*) - GET/SET key, one or more required for SET operations

T_MACHINE Class: Local Attributes

Attribute	Type	Permissions	Values	Default
TA_CURACCESSERS	long	R--R--R--	0 <= *num* < 32K	N/A
TA_CURCLIENTS	long	R--R--R--	0 <= *num* < 32K	N/A
TA_CURCONV	long	R--R--R--	0 <= *num* < 32K	N/A
TA_CURGTT	long	R--R--R--	0 <= *num* < 32K	N/A
TA_CURRLOAD	long	R--R--R--	0 <= *num*	N/A
TA_CURWSCLIENTS	long	R--R--R--	0 <= *num* < 32K	N/A
TA_HWACCESSERS	long	R--R--R--	0 <= *num* < 32K	N/A
TA_HWCLIENTS	long	R--R--R--	0 <= *num* < 32K	N/A

T_MACHINE Class: Local Attributes

Attribute	Type	Permissions	Values	Default
TA_HWCONV	long	R--R--R--	0 <= *num* < 32K	N/A
TA_HWGTT	long	R--R--R--	0 <= *num* < 32K	N/A
TA_HWWSCLIENTS	long	R--R--R--	0 <= *num* < 32K	N/A
TA_NUMCONV	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMDEQUEUE	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMENQUEUE	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMPOST	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMREQ	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMSUBSCRIBE	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMTRAN	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMTRANABT	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMTRANCMT	long	R-XR-XR--	0 <= *num*	N/A
TA_PAGESIZE	long	R--R--R--	1 <= *num*	N/A
TA_SWRELEASE	string	R--R--R--	*string[0...78]*	N/A
TA_HWACLCACHE	long	R--R--R--	0 <= *num*	N/A
TA_ACLCACHEHITS	long	R--R--R--	0 <= *num*	N/A
TA_ACLCACHEACCESS	long	R--R--R--	0 <= *num*	N/A
TA_ACLFAIL	long	R--R--R--	0 <= *num*	N/A
TA_WKCOMPLETED	long	R--R--R--	0 <= *num*	N/A
TA_WKINITIATED	long	R--R--R--	0 <= *num*	N/A

[1]TA_LMID and TA_PMID must each be unique within this class. Only one of these fields is required as a key field for a SET operation. If both are specified, then they must match the same object.

[2]Default is same as value set for this attribute in Class T_DOMAIN.

[3]Default is TA_APPDIR for this machine followed by /ULOG.

Attribute Semantics

TA_LMID: *string[1...30]*

Logical machine identifier. This identifier is used within the rest of the TM_MIB definition as the sole means of mapping application resources to T_MACHINE objects.

TA_PMID: *string[1...30]*

Physical machine identifier. This identifier should match the UNIX System nodename as returned by the ‘‘uname -n” command when run on the identified system.

TA_TUXCONFIG: *string[2...64]*

Absolute pathname of the file or device where the binary BEA TUXEDO system configuration file is found on this machine. The administrator need only maintain one such file, namely the one identified by the TA_TUXCONFIG attribute value on the master machine. The information contained in this file is automatically propagated to all other T_MACHINE objects as they are activated. See TA_ENVFILE in this class for a discussion of how this attribute value is used in the environment.

TA_TUXDIR: *string[2...78]*

Absolute pathname of the directory where the BEA TUXEDO system software is found on this machine. See TA_ENVFILE in this class for a discussion of how this attribute value is used in the environment.

TA_APPDIR: *string[2...78]*

Colon separated list of application directory absolute pathnames. The first directory serves as the current directory for all application and administrative servers booted on this machine. All directories in the list are searched when starting application servers. See TA_ENVFILE in this class for a discussion of how this attribute value is used in the environment.

TA_STATE:

GET: {ACTive|INActive|PARtitioned}

A GET operation will retrieve configuration and run-time information for the selected T_MACHINE object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

ACTive	T_MACHINE object defined and active (administrative servers, that is, DBBL, BBL, and BRIDGE).

INActive	T_MACHINE object defined and inactive.
PARtitioned	T_MACHINE object defined, listed in accessible bulletin boards as active, but currently unreachable. This state is ACTive equivalent for the purpose of determining permissions.

SET:{NEW|INValid|ACTive|ReACtivate|INActive|ForceINactive|CLEaning}

A SET operation will update configuration and run-time information for the selected T_MACHINE object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_MACHINE object for application. State change allowed only when in the INValid state. Successful return leaves the object in the INActive state.
unset	Modify an existing T_MACHINE object. This combination is allowed only when in the ACTive or INActive state. Successful return leaves the object state unchanged.
INValid	Delete T_MACHINE object for application. State change allowed only when in the INActive state. Successful return leaves the object in the INValid state.
ACTive	Activate the T_MACHINE object. Necessary administrative servers such as the DBBL, BBL, and BRIDGE are started on the indicated site as well as application servers configured to run on that site (subject to restriction by TA_FLAGS settings). For the purpose of determining permissions for this state transition, the active object permissions are considered (that is, --x--x--x). State change allowed only when in the INActive state. Successful return leaves the object in the ACTive state. The TMIB_NOTIFY TA_FLAG value should be used when activating a machine if status on individual servers is required.

ReACtivate	Activate the T_MACHINE object. Necessary administrative servers such as the DBBL, BBL, and BRIDGE are started on the indicated site as well as application servers configured to run on that site (subject to restriction by TA_FLAGS settings). For the purpose of determining permissions for this state transition, the active object permissions are considered (that is, --x--x--x). State change allowed only when in either the ACTive or INACtive state. Successful return leaves the object in the ACTive state. The TMIB_NOTIFY TA_FLAG value should be used when reactivating a machine if status on individual servers is required.
INACtive	Deactivate the T_MACHINE object. Necessary administrative servers such as the BBL and BRIDGE are stopped on the indicated site as well as application servers running on that site (subject to restriction by TA_FLAGS settings). State change allowed only when in the ACTive state and when no other application resources are active on the indicated machine. Successful return leaves the object in the INACtive state. The TMIB_NOTIFY TA_FLAG value should be used when deactivating a machine if status on individual servers is required.
ForceINactive	Deactivate the T_MACHINE object without regard to attached clients. Necessary administrative servers such as the BBL and BRIDGE are stopped on the indicated site as well as application servers running on that site (subject to restriction by TA_FLAGS settings). State change allowed only when in the ACTive state. Successful return leaves the object in the INACtive state. The TMIB_NOTIFY TA_FLAG value should be used when deactivating a machine if status on individual servers is required.

CLEaning	Initiate cleanup/scanning activities on and relating to the indicated machine. If there are dead clients or servers on the machine, they will be detected at this time. If the machine has been partitioned from the application MASTER site, then global bulletin board entries for that machine will be removed. This combination is allowed when the application is in the ACTive state and the T_MACHINE object is in either the ACTive or PARtitioned state. Successful return for a non-partitioned machine leaves the state unchanged. Successful return for a partitioned machine leaves the object in the INActive state.

Limitation: State change to ForceINactive or INActive allowed only for non-master machines. The master site administrative processes are deactivated via the T_DOMAIN class.

TA_UID: 0 <= *num*

UNIX System user identifier for the BEA TUXEDO system application administrator on this machine. Administrative commands such as tmboot(1), tmshutdown(1), and tmadmin(1) must run as the indicated user on this machine. Application and administrative servers on this machine will be started as this user.

Limitation: This is a UNIX System-specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_GID: 0 <= *num*

UNIX System group identifier for the BEA TUXEDO system application administrator on this machine. Administrative commands such as tmboot(1), tmshutdown(1), and tmadmin(1) must run as part of the indicated group on this machine. Application and administrative servers on this machine will be started as part of this group.

Limitation: This is a UNIX System-specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_ENVFILE: *string[0...78]*

Environment file for clients and servers running on this machine. If the value specifies an invalid file name, no values are added to the environment. the value of *string* is placed into the environment.

When booting servers, local servers inherit the environment of tmboot(1) and remote servers (not on the MASTER) inherit the environment of tlisten(1). TUXCONFIG, TUXDIR, and APPDIR are also put into the environment when a server is booted based on the information in the associated T_MACHINE object. PATH will be set in the environment to

```
APPDIR:TUXDIR/bin:/bin:/usr/bin:<path>
```

where <*path*> is the value of the first PATH= line appearing in the machine environment file, if one exists (subsequent PATH= lines will be ignored). This PATH will be used as a search path for servers that are specified with a simple or relative pathname (i.e., that doesn't begin with slash). LD_LIBRARY_PATH will be set in the environment to

```
APPDIR:TUXDIR/lib:/lib:/usr/lib:<lib>
```

where <*lib*> is the value of the first LD_LIBRARY_PATH= line appearing in the machine environment file, if one exists (subsequent LD_LIBRARY_PATH= lines will be ignored).

As part of server initialization (before tpsvrinit (3c) is called), a server will read and export variables from both the machine and server ENVFILE files. If a variable is set in both the machine and server ENVFILE, the value in the server ENVFILE will override the value in the machine ENVFILE with the exception of PATH which is appended. A client will process only the machine ENVFILE file. When the machine and server ENVFILE files are processed, lines that are not of the form <ident>= will be ignored, where <ident> begins with an underscore or alphabetic character, and contains only underscore or alphanumeric characters. If a PATH= line is encountered, PATH will be set to

```
APPDIR:TUXDIR/bin:/bin:/usr/bin:<path>
```

where <path> is the value of the first PATH= line appearing in the environment file (subsequent PATH= lines are ignored). If PATH appears in both the machine and server files, then <path> is <path1>:<path2> where <path1> is from the machine ENVFILE and <path2> is from the server ENVFILE. If a LD_LIBRARY_PATH= line is encountered, LD_LIBRARY_PATH will be set to

```
APPDIR:TUXDIR/lib:/lib:/usr/lib:<lib>
```

where <lib> is the value of the first LD_LIBRARY_PATH= line appearing in the environment file (subsequent LD_LIBRARY_PATH= lines are ignored). Attempts to reset TUXDIR, APPDIR, or TUXCONFIG will be ignored and a warning will be printed if the value does not match the corresponding

T_MACHINE attribute value. Limitation: Modifications to this attribute for an active object will not affect running servers or clients.

TA_PERM: 0001 <= *num* <= 0777
: UNIX System permissions associated with the shared memory bulletin board created on this machine. Default UNIX System permissions for system and application message queues.

  Limitations: Modifications to this attribute for an active object will not affect running servers or clients.

  This is a UNIX System-specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_ULOGPFX: *string[0...78]*
: Absolute pathname prefix of the path for the userlog (3c) file on this machine. The userlog (3c) file name is formed by appending the string *.mmddyy* to the TA_ULOGPFX attribute value. *mmddyy* represents the month, day, and year that the messages were generated. All application and system userlog (3c) messages generated by clients and servers running on this machine are directed to this file.

  Limitation: Modifications to this attribute for an active object will not affect running servers or clients.

TA_TYPE: *string*[0...15]
: Machine type. Used to group machines into classes of like data representations. Data encoding is not performed when communicating between machines of identical types. This attribute can be given any string value; values are used only for comparison. Distinct TA_TYPE attributes should be set when the application spans a heterogeneous network of machines or when compilers generate dissimilar structure representations. The default for this attribute, a 0-length string, matches any other machine with a 0-length string as its TA_TYPE attribute value.

TA_MAXACCESSERS: 1 <= *num* < 32K
: Maximum number of clients and servers that can have access to the bulletin board on this machine at one time. System administration processes such as the BBL and tmadmin need not be accounted for in this figure, but all application servers and clients and TMS servers should be counted. If the application is booting workstation listeners on this site, then both the listeners and the potential number of workstation handlers that may be booted should be counted.

TA_MAXCONV: 0 <= *num* < 32K

Maximum number of simultaneous conversations in which clients and servers on this machine can be involved.

TA_MAXGTT: 0 <= *num* < 32K

Maximum number of simultaneous global transactions in which this machine can be involved.

TA_MAXWSCLIENTS: 0 <= *num* < 32K

Number of accesser entries on this machine to be reserved for workstation clients. The number specified here takes a portion of the total accesser slots specified with the TA_MAXACCESSERS attribute. The appropriate setting of this parameter helps to conserve IPC resources since workstation client access to the system is multiplexed through a BEA TUXEDO system supplied surrogate, the workstation handler. It is an error to set this number greater than TA_MAXACCESSERS.

TA_MAXACLCACHE: 10 <= *num* <= 32000

Number of entries in the cache used for ACL entries when TA_SECURITY is set to ACL or MANDATORY_ACL. The appropriate setting of this parameter helps to conserve on shared memory resources and yet reduce the number of disk access to do ACL checking.

TA_TLOGDEVICE: *string[0...64]*

The device (raw slice) or UNIX System file containing the BEA TUXEDO filesystem that holds the DTP transaction log for this machine. The DTP transaction log is stored as a BEA TUXEDO system VTOC table on the device. This device or file may be the same as that specified for the TA_TUXCONFIG attribute for this machine.

TA_TLOGNAME: *string[0...30]*

The name of the DTP transaction log for this machine. If more than one DTP transaction log exists on the same TA_TLOGDEVICE, they must have unique names. TA_TLOGNAME must be different from the name of any other table on the TA_TLOGDEVICE where the DTP transaction log table is created.

TA_TLOGSIZE: 1 <= *num* 2K+1

The numeric size, in pages, of the DTP transaction log for this machine. The TA_TLOGSIZE attribute value is subject to limits based on available space in the BEA TUXEDO filesystem identified by the TA_TLOGDEVICE attribute.

TA_BRIDGE: *string[0...78]*

Device name to be used by the BRIDGE process placed on this logical machine to access the network. This is a required value for participation in a networked application via a TLI-based BEA TUXEDO system binary. This attribute is not needed for sockets-based BEA TUXEDO system binaries.

TA_NADDR: *string[0...78]*

Specifies the complete network address to be used by the BRIDGE process placed on the logical machine as its listening address. The listening address for a BRIDGE is the means by which it is contacted by other BRIDGE processes participating in the application. This attribute must be set if the logical machine is to participate in a networked application, that is, if the LAN option is set in the T_DOMAIN:TA_OPTIONS attribute value.

If *string* has the form "0xhex-digits" or "\\xhex-digits", it must contain an even number of valid hex digits. These forms are translated internally into a character array containing the hexadecimal representations of the string specified. For TCP/IP addresses either the

"//*hostname:port*"

or

"//*#.#.#.#:port*"

format is used.

TA_NLSADDR: *string[0...78]*

Network address used by the tlisten(1) process servicing the network on the node identified by this logical machine. This network address is of the same format as that specified for the TA_NADDR attribute above.

This attribute must be set if the logical machine is to participate in a networked application, that is, if the LAN option is set in the T_DOMAIN:TA_OPTIONS attribute value.

TA_CMPLIMIT: *remote[,local]*

Threshold message size at which compression will occur for *remote* traffic and optionally *local* traffic. *remote* and *local* may be either non-negative numeric values or the string "MAXLONG", which is dynamically translated to the maximum long setting for the machine. Setting only the *remote* value will default *local* to MAXLONG.

Limitation: This attribute value is not part of the T_MACHINE object for active sites running BEA TUXEDO system Release 4.2.2 or earlier. However, site release identification is not determined until run-time, so this attribute may be set and accessed for any inactive object. When a BEA TUXEDO system Release 4.2.2 or earlier site is activated, the configured value is not used.

TA_TMNETLOAD: 0 <= *num* < 32K

Service load added to any remote service evaluated during load balancing on this machine.

Limitation: This attribute value is not part of the T_MACHINE object for active sites running BEA TUXEDO System Release 4.2.2 or earlier. However, site release identification is not determined until runtime, so this attribute may be set and accessed for any inactive object. When a BEA TUXEDO System Release 4.2.2 or earlier site is activated, the configured value is not used.

TA_SPINCOUNT: 0 <= *num*

Spincount used on this machine for pre-ticket user level semaphore access. Defaults are built into the BEA TUXEDO system binaries on each machine. These defaults may be overridden at runtime for tuning purposes using this attribute. The spincount may be reset to the default built-in value for the site by resetting this attribute value to 0. There is also a TMSPINCOUNT environment variable, which the system uses if the value is not set here or in the ubbconfig file.

Limitation: This attribute value is not part of the T_MACHINE object for active sites running BEA TUXEDO System Release 4.2.2 or earlier. However, site release identification is not determined until runtime, so this attribute may be set and accessed for any inactive object. When a BEA TUXEDO system Release 4.2.2 or earlier site is activated, the configured value is not used.

TA_ROLE: {MASTER|BACKUP|OTHER}

The role of this machine in the application. MASTER indicates that this machine is the master machine, BACKUP indicates that it is the backup master machine, and OTHER indicates that the machine is neither the master nor backup master machine.

TA_MINOR: 1 <= *num*

The BEA TUXEDO system minor protocol release number for this machine.

TA_RELEASE: 1 <= *num*

The BEA TUXEDO system major protocol release number for this machine. This may be different from the TA_SWRELEASE for the same machine.

MINENCRYPTBITS={0|40|128}
: When establishing a network link to this machine, 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 is "0".

  Limitation: Modifications to this attribute will not effect established network links.

MAXENCRYPTBITS={0|40|128}
: When establishing a network link, negotiate encryption up to this level. "0" means no encryption, whilc "40" and "128" specify the encryption length (in bits). The default is "128".

  Limitation: Modifications to this attribute will not effect established network links.

TA_MAXPENDINGBYTES: 100000 <= *num* <= MAXLONG
: specifies a limit for the amount of space that can be allocated for messages waiting to be transmitted by the BRIDGE process.

TA_CURACCESSERS: 0 <= *num* < 32K
: Number of clients and servers currently accessing the application either directly on this machine or through a workstation handler on this machine.

TA_CURCLIENTS: 0 <= *num* < 32K
: Number of clients, both native and workstation, currently logged in to this machine.

TA_CURCONV: 0 <= *num* < 32K
: Number of active conversations with participants on this machine.

TA_CURGTT: 0 <= *num* < 32K
: Number of in use transaction table entries on this machine.

TA_CURRLOAD: 0 <= *num*
: Current service load enqueued on this machine. Limitation: If the T_DOMAIN:TA_LDBAL attribute is "N" or the T_DOMAIN:TA_MODEL attribute is "MP", then an FML32 NULL value is returned (0).

TA_CURWSCLIENTS: 0 <= *num* < 32K
: Number of workstation clients currently logged in to this machine.

TA_HWACCESSERS: 0 <= *num* < 32K
: High water number of clients and servers accessing the application either directly on this machine or through a workstation handler on this machine.

TA_HWCLIENTS: 0 <= *num* < 32K
: High water number of clients, both native and workstation, logged in to this machine.

TA_HWCONV: 0 <= *num* < 32K
: High water number of active conversations with participants on this machine.

TA_HWGTT: 0 <= *num* < 32K
: High water number of in use transaction table entries on this machine.

TA_HWWSCLIENTS: 0 <= *num* < 32K
: High water number of workstation clients currently logged in to this machine.

TA_NUMCONV: 0 <= *num*
: Number of `tpconnect` (3c) operations performed from this machine.

TA_NUMDEQUEUE: 0 <= *num*
: Number of `tpdequeue` (3c) operations performed from this machine.

TA_NUMENQUEUE: 0 <= *num*
: Number of `tpenqueue` (3c) operations performed from this machine.

TA_NUMPOST: 0 <= *num*
: Number of `tppost` (3c) operations performed from this machine.

TA_NUMREQ: 0 <= *num*
: Number of `tpacall` (3c) or `tpcall` (3c) operations performed from this machine.

TA_NUMSUBSCRIBE: 0 <= *num*
: Number of `tpsubscribe` (3c) operations performed from this machine.

TA_NUMTRAN: 0 <= *num*
: Number of transactions initiated (`tpbegin` (3c)) from this machine.

TA_NUMTRANABT: 0 <= *num*
: Number of transactions aborted (`tpabort` (3c)) from this machine.

TA_NUMTRANCMT: 0 <= *num*
: Number of transactions committed (`tpcommit` (3c)) from this machine.

TA_PAGESIZE: 1 <= *num*
: Disk pagesize used on this machine.

TA_SWRELEASE: *string[0...78]*
: Software release for binary on that machine or a 0-length string if binary is not a BEA TUXEDO system master binary.

TA_HWACLCACHE: 0 <= *num*
: High water number of entries used in the ACL cache.

TA_ACLCACHEHITS: 0 <= *num*
: Number of accesses to the ACL cache that resulted in a "hit" (that is, the entry was already in the cache).

TA_ACLCACHEACCESS: 0 <= *num*
: Number of accesses to the ACL cache.

TA_ACLFAIL: 0 <= *num*
: Number of accesses to the ACL cache that resulted in a access control violation.

TA_WKCOMPLETED: 0 <= *num*
: Total service load dequeued and processed successfully by servers running on this machine. Note that for long running applications this attribute may wraparound, that is, exceed the maximum value for a long, and start back at 0 again.

TA_WKINITIATED: 0 <= *num*
: Total service load enqueued by clients/servers running on this machine. Note that for long running applications this attribute may wraparound, that is, exceed the maximum value for a long, and start back at 0 again.

Limitations

SHM mode (see T_DOMAIN:TA_MODEL) applications can have only one T_MACHINE object. MP mode (see T_DOMAIN:TA_MODEL) applications with the LAN option set (see T_DOMAIN:TA_OPTIONS) may have up to the maximum number of configurable T_MACHINE objects as defined by the T_DOMAIN:TA_MAXMACHINES attribute. Many attributes of this class are tunable only when the application is inactive on the site. Since the master machine at least must be active in a minimumly active application, the use of the ATMI interface routines to administer the application is not possible with respect to the master machine object. The function tpadmcall (3c) is being provided as a means configuring an unbooted application and may be used to set these attributes for the master machine.

T_MSG CLASS

Overview

The T_MSG class represents run-time attributes of the BEA TUXEDO system managed UNIX System message queues.

Attribute Table

T_MSG Class Definition Attribute Table

Attribute[1]	Type	Permissions	Values	Default
TA_LMID(k)	string	R--R--R--	*LMID*	N/A
TA_MSGID(k)	long	R--R--R--	1 <= *num*	N/A
TA_STATE(k)	string	R--R--R--	GET:"{ACT}" SET:N/A	N/A N/A
TA_CURTIME	long	R--R--R--	1 <= *num*	N/A
TA_MSG_CBYTES	long	R--R--R--	1 <= *num*	N/A
TA_MSG_CTIME	long	R--R--R--	1 <= *num*	N/A
TA_MSG_LRPID	long	R--R--R--	1 <= *num*	N/A
TA_MSG_LSPID	long	R--R--R--	1 <= *num*	N/A
TA_MSG_QBYTES	long	R--R--R--	1 <= *num*	N/A
TA_MSG_QNUM	long	R--R--R--	1 <= *num*	N/A
TA_MSG_RTIME	long	R--R--R--	1 <= *num*	N/A
TA_MSG_STIME	long	R--R--R--	1 <= *num*	N/A
(k) - GET key field				

[1]All attributes in Class T_MSG are local attributes.

Attribute Semantics

TA_LMID: *LMID*

Logical machine identifier.

TA_MSGID: 1 <= *num*

UNIX System message queue identifier. Limitation: This is a UNIX System-specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_STATE:

GET: ACTive

A GET operation will retrieve run-time information for the selected T_MSG object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

ACTive	T_MSG object active. This corresponds exactly to the related T_MACHINE object being active.

SET:

SET operations are not permitted on this class.

TA_CURTIME: 1 <= *num*

Current time, in seconds, since 00:00:00 UTC, January 1, 1970, as returned by the time(2) system call on T_MSG:TA_LMID. This attribute can be used to compute elapsed time from the T_MSG:TA_?TIME attribute values.

TA_MSG_CBYTES: 1 <= *num*

Current number of bytes on the queue.

TA_MSG_CTIME: 1 <= *num*

Time of the last msgctl(2) operation that changed a member of the *msqid_ds* structure associated with the queue.

TA_MSG_LRPID: 1 <= *num*

Process identifier of the last process that read from the queue.

TA_MSG_LSPID: 1 <= *num*

Process identifier of the last process that wrote to the queue.

TA_MSG_QBYTES: 1 <= *num*

Maximum number of bytes allowed on the queue.

TA_MSG_QNUM: 1 <= *num*

Number of messages currently on the queue.

`TA_MSG_RTIME`: 1 <= *num*
: Time since the last read from the queue.

`TA_MSG_STIME`: 1 <= *num*
: Time since the last write to the queue.

Limitations This class is UNIX System-specific and may not be supported in non-UNIX implementations of BEA TUXEDO system.

T_NETGROUP CLASS

Overview

The T_NETGROUP class represents application attributes of network groups. Network groups are groups of LMIDs which can communicate over the TA_NADDR network addresses defined in the T_NETMAP class.

Attribute Table

Table 2: T_NETGROUP Class Definition Attribute Table

Attribute[1]	Type	Permissions	Values	Default
TA_NETGROUP(r)(*)	string	rU-------	*string[1 . . . 30]*	DEFAULTNET
TA_NETGRPNO(r)(*)	long	rU-------	1 <= *num* < 8192	N/A
TA_STATE(k)	string	rw-r--r--	GET:"{VAL}" SET:"{NEW|INV}"	N/A N/A
TA_NETPRIO(*)	long	rwyrw----	1 <= *num* < 8192	100

(r) - Required field for object creation (SET TA_STATE NEW)
(*) - GET/SET key, one or more required for SET operation

Attribute Semantics

TA_NETGROUP:*string[1 . . .30]*

Logical name of the network group. A group name is a string of printable characters and cannot contain a point sign, comma, colon, or newline.

TA_NETGRPNO: 1 <= *num* <= 8192

Group identifier associated with network group.

TA_STATE:

GET: {VALid}

A GET operation will retrieve configuration information for the selected T_NETGROUP object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

VALid	T_NETGROUP object is defined and inactive. Note that this is the only valid state for this class. NETGROUPs are never ACTive.

SET: {NEW|INValid}

a SET operation will update configuration information for the selected T_NETGROUP object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_NETGROUP object for application. State change allowed only when in the INValid state. Successful return leaves the object in the VALid state.
unset	Modify an existing T_NETGROUP object. Only allowed in the VALid state. Successful return leaves the object state unchanged.
INVal id	Delete T_NETGROUP object from application. State change allowed only when in the VALid state and only if there are no objects in the T_NETMAP class which have this network group object as a key. Successful return leaves the object in the INValid state.

TA_NETPRIO:1 <= *num* < 8192

The priority band for this network group. All network groups of equivalent band priority will be used in parallel. If all network circuits of a certain priority are torn down by the administrator or by network conditions, the next lower priority circuit is used. Retries of the higher priority are attempted.

Note: In Release 6.4, parallel data circuits are prioritized by network group number (NETGRPNO) within priority group number. In future releases, a different algorithm may be used to prioritize parallel data circuits.

Limitations

None.

T_NETMAP CLASS

Overview

The T_NETMAP class associates TA_LMIDs frp, tje T_MACHINE class in the TM_MIB to a TA_NETGROUP object from the T_NETGROUP class. This class identifies which logical machines belong to which network group. A TA_LMID ,au be om ,amu TA_NETGROUP groups. When one LMID connects to another LMID, the BRIDGE process determines the subset of network groups to which the two LMIDs belong. When the apir of LMIDs are in several common groups, they are sorted in descending TA_NETPRIO order (TA_NETGRPNO is the secondary sort key). The Network groups with the same TA_NETPRIO will flow network data in parallel. Should a networking error prevent data from flowing through all the highest priority group(s), only then the next lower priority network group(s) are used for network traffic (*failover*). All network groups with a higher priority than the ones flowing data are retried periodically. Once a network connection is established with a higher TA_NETPRIO value, no further data is scheduled for the lower priority one. Once the lower priority connection is drained, it is disconnected in an orderly fashion (*failback*).

Attribute Table

T_NETMAP Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_NETGROUP(r)(*)	string	ru-------	*string[1 . . . 30]*	N/A
TA_LMID(r)(*)	string	ru-------	*string[1 . . . 30]*	N/A
TA_STATE	string	RW-------	GET:"{VALl}"	N/A
			SET:"{NEW|INV}"	N/A
TA_NADDR	string	rw-r--r--	*string[1 . . . 78]*	" "
TA_MINENCRYPTBIT	string	rwxrwx---	{0|40|128}	0
TA_MAXENCRYPTBIT	string	rwxrwx---	{0|40|128}	128
(r) - Required field for object creation (SET TA_STATE NEW) (*) - GET/SET key, one or more required for SET operations				

Attribute Semantics

TA_NETGROUP: *string*

This is the name of the associated network group found in the T_NETGROUP class.

TA_LMID: *string*

The logical machine name for the T_MACHINE class (in TM_MIB) for this network mapping.

TA_STATE:

GET: {VALid}

A GET operation will retrieve run-time information for the selected T_NETMAP object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

VALid	T_NETMAP object is defined. Note that this is the only valid state for this class. Network mappings are never ACTive

SET: {NEW|INValid}

A SET operation will update configuration information for the selected T_NETMAP object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed cannot be set.

NEW	Create T_NETMAP object for application. State change allowed only when in the INValid state. Successful return leaves the object in the VALid state.
unset	Modify an existing T_NETMAP object. Successful return leaves the object state unchanged.
INValid	Deletes the given network mapping. If any network links were active as a result of the mapping, they will be disconnected. This disconnection may cause a state change in T_BRIDGE objects (in TM_MIB) associated with the network links.

TA_NADDR: *string*

Specifies the complete network address to be used by the BRIDGE process placed in the logical machines as its listening address. The listening address for a BRIDGE is the mains by which it is contacted by other BRIDGE processes participating in the application, that is, if the LAN option is set in the T_DOMAIN:TA_OPTIONS attribute value. If *string* has the form "0xhexa-digits," it must contain an even number of valid hex digits. These

forms are translated internally into a character array containing the hexadecimal representations of the string specificd. For TCP/IP addresses either the

"//hostname:port"

or

"//#.#.#.#:port"

format is used

TA_MINENCRYPTBITS={0|40|128}

When establishing a network link to this machine, 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 is 0.

Limitation: Modifications to this attribute will not effect established network links.

TA_MAXENCRYPTBITS={0|40|128}

When establishing a network link, negotiate encryption up to this level. 0 means no encryption, whilc 40 and 128 specify the encryption length (in bits). The default is 128.

Limitation: Modifications to this attribute will not effect established network links.

When the 128-bit encryption package is installed, TA_ MAXENCRYPTBITS defaults to 128. When the 40-bit package is installed, the default is 40. When no encryption package is installed, the default is 0 bits. Note that when BRIDGE processes connect, they negotiate to the highest common TA_MAXENCRYPTBITS.

Limitations None.

T_QUEUE CLASS

Overview

The T_QUEUE class represents run-time attributes of queues in an application. These attribute values identify and characterize allocated BEA TUXEDO system request queues associated with servers in a running application. They also track statistics related to application workloads associated with each queue object.

Note that when a GET operation with the MIB_LOCAL flag is performed in a multi-machine application, multiple objects will be returned for each active queue - one object for each logical machine where local attribute values are collected.

Attribute Table

T_QUEUE Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_RQADDR(*)	string	R--R--R--	*string[1...30]*	N/A
TA_SERVERNAME(k)	string	R--R--R--	*string[1...78]*	N/A
TA_STATE(k)	string	R--R--R--	GET:"{ACT|MIG |SUS|PAR}" SET:N/A	N/A N/A
TA_GRACE	long	R--R--R--	0 <= *num*	N/A
TA_MAXGEN	long	R--R--R--	1 <= *num* < 256	N/A
TA_RCMD	string	R--R--R--	*string[0...78]*	N/A
TA_RESTART	string	R--R--R--	"{Y|N}"	N/A
TA_CONV	string	R--R--R--	"{Y|N}"	N/A
TA_LMID(k)	string	R--R--R--	*LMID*	N/A
TA_RQID	long	R--R--R--	1 <= *num*	N/A
TA_SERVERCNT	long	R--R--R--	1 <= *num* < 8K	N/A
T_QUEUE Class:LOCAL Attributes				
TA_TOTNQUEUED	long	R-XR-XR--	0 <= *num*	N/A
TA_TOTWKQUEUED	long	R-XR-XR--	0 <= *num*	N/A
TA_SOURCE(k)	string	R--R--R--	*LMID*	N/A
TA_NQUEUED	long	R--R--R--	0 <= *num*	N/A
TA_WKQUEUED	long	R--R--R--	0 <= *num*	N/A

(k) - GET key field
(*) - GET/SET key, one or more required for SET operations

Attribute Semantics

TA_RQADDR: *string[1...30]*

Symbolic address of the request queue. Servers with the same T_SERVER:TA_RQADDR attribute value are grouped into a Multiple Server Single Queue (MSSQ) set. Attribute values returned with a T_QUEUE object apply to all active servers associated with this symbolic queue address.

TA_SERVERNAME: *string[1...78]*

Full pathname of the server executable file. The server identified by TA_SERVERNAME is running on the machine identified by the T_QUEUE:TA_LMID attribute. When specified as a key field on a GET operation, this attribute may specify a relative pathname; all appropriate full pathnames will be matched.

TA_STATE:

GET: {ACTive|MIGrating|SUSpended|PARtitioned}

A GET operation will retrieve run-time information for the selected T_QUEUE object(s). The T_QUEUE class does not address configuration information directly. Configuration related attributes discussed here must be set as part of the related T_SERVER objects. The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

ACTive	At least one server associated with this T_QUEUE object is active.
MIGrating	The server(s) associated with this T_QUEUE object is currently in the MIGrating state. See the T_SERVER class for more details on this state. This state is ACTive equivalent for the purpose of determining permissions.
SUSpended	The server(s) associated with this T_QUEUE object is currently in the SUSpended state. See the T_SERVER class for more details on this state. This state is ACTive equivalent for the purpose of determining permissions.
PARtitioned	The server(s) associated with this T_QUEUE object is currently in the PARtitioned state. See the T_SERVER class for more details on this state. This state is ACTive equivalent for the purpose of determining permissions.

SET:

A SET operation will update run-time information for the selected T_QUEUE object. State changes are not allowed when updating T_QUEUE object information. Modification of an existing T_QUEUE object is allowed only when the object is in the ACTive state.

TA_GRACE: 0 <= *num*

The period of time, in seconds, over which the T_QUEUE:TA_MAXGEN limit applies. This attribute is meaningful only for restartable servers, that is, if the T_QUEUE:TA_RESTART attribute is set to "Y". A value of 0 for this attribute indicates that a server should always be restarted.

TA_MAXGEN: 1 <= *num* < 256

Number of generations allowed for restartable servers (T_QUEUE:TA_RESTART == "Y") associated with this queue over the specified grace period (T_QUEUE:TA_GRACE). The initial activation of each server counts as one generation and each restart also counts as one.

TA_RCMD: *string[0...78]*

Application specified command to be executed in parallel with the system restart of application servers associated with this queue.

TA_RESTART: {Y|N}

Servers associated with this queue are restartable ("Y") or non-restartable ("N").

TA_CONV: {Y|N}

Servers associated with this queue are conversational-based ("Y") or request/response-based ("N").

TA_LMID: *LMID*

Logical machine on which servers associated with this queue are active.

TA_RQID: 1 <= *num*

UNIX System message queue identifier.
Limitation: This is a UNIX System specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_SERVERCNT: 1 <= *num* < 8K

Number of active servers associated with this queue.

TA_TOTNQUEUED: 0 <= *num*

The sum of the queue lengths of this queue while it has been active. This sum includes requests enqueued to and processed by servers that are no longer active on the queue. Each time a new request is assigned to the queue, the sum

is incremented by the length of the queue immediately before the new request is enqueued.

Limitation: If the T_DOMAIN:TA_LDBAL attribute is "N" or the T_DOMAIN:TA_MODEL attribute is "MP", then TA_TOTNQUEUED is not returned. In the same configuration, updates to this attribute are ignored. Consequently, when this attribute is returned TA_LMID and TA_SOURCE have the same value.

TA_TOTWKQUEUED: 0 <= *num*

The sum of the workloads enqueued to this queue while it has been active. This sum includes requests enqueued to and processed by servers that are no longer active on the queue. Each time a new request is assigned to the queue, the sum is incremented by the workload on the queue immediately before the new request is enqueued.

Limitation: If the T_DOMAIN:TA_LDBAL attribute is "N" or the T_DOMAIN:TA_MODEL attribute is "MP", then TA_TOTWKQUEUED is not returned. In the same configuration, updates to this attribute are ignored. Consequently, when this attribute is returned TA_LMID and TA_SOURCE have the same value.

TA_SOURCE: *LMID*

Logical machine from which local attribute values are retrieved.

TA_NQUEUED: 0 <= *num*

Number of requests currently enqueued to this queue from the TA_SOURCE logical machine. This value is incremented at enqueue time and decremented when the server dequeues the request.

Limitation: If the T_DOMAIN:TA_LDBAL attribute is "N" or the T_DOMAIN:TA_MODEL attribute is "MP", then TA_NQUEUED is not returned. Consequently, when this attribute is returned TA_LMID and TA_SOURCE have the same value.

TA_WKQUEUED: 0 <= *num*

Workload currently enqueued to this queue from the TA_SOURCE logical machine. If the T_DOMAIN:TA_MODEL attribute is set to SHM and the T_DOMAIN:TA_LDBAL attribute is set to "Y" then this attribute reflects the application-wide workload enqueued to this queue. However, if TA_MODEL is set to MP and TA_LDBAL is set to "Y", this attribute reflects the workload enqueued to this queue from the TA_SOURCE logical machine during a recent timespan. This attribute is used for load balancing purposes. So as to not discriminate against newly started servers, this attribute value is zeroed out on each machine periodically by the BBL.

Limitations None.

T_ROUTING CLASS

Overview

The T_ROUTING class represents configuration attributes of routing specifications for an application. These attribute values identify and characterize application data dependent routing criteria with respect to field names, buffer types, and routing definitions.

Attribute Table

T_ROUTING Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_ROUTINGNAME(r)(*)	string	ru-r--r--	*string[1...15]*	N/A
TA_BUFTYPE(r)(*)	string	ru-r--r--	*string[1...256]*	N/A[1]
TA_FIELD(r)(k) (*)	string	ru-r--r--	*string[1...30]*	N/A[1]
TA_RANGES(r)	carray	rw-r--r--	*carray[1...2048]*	N/A
TA_TYPE	string	ru-r--r--	string[1..15]	“SERVICE2
TA_FIELDTYPE (r)	string	rw-r--r--	string[1..30]	N/A
TA_STATE(k)	string	rw-r--r--	GET:"{VAL}" SET:"{NEW|INV}"	N/A N/A

(k) - GET key field
(r) - Required field for object creation (SET TA_STATE NEW)
(*) - GET/SET key, one or more required for SET operations

[1] TA_BUFTYPE applies only to BEA TUXEDO data-dependent routing criteria. TA_FIELDTYPE applies only to BEA WLE factory-based routing criteria. The specified u (uniqueness) permission applies only in the relevant case. In other words, the combination of TA_ROUTINGNAME, TA_TYPE, and TA_BUFTYPE must be unique for TA_TYPE=SERVICE, and TA_ROUTINGNAME, TA_TYPE, and TA_FIELD must be unique for TA_TYPE=FACTORY.

The TA_TYPE attribute determines the permissible attributes for the TA_ROUTING object. TYPE=SERVICE corresponds to BEA TUXEDO data-dependent routing criteria. TYPE=FACTORY corresponds to BEA WLE Factory-Based Routing. The default is

SERVICE. SET operations are assumed to be for data-dependent routing if no TA_TYPE is specified. Specification of TA_FIELDTYPE is invalid for data-dependent routing. Specification of TA_BUFTYPE is invalid for Factory-Based routing.

Attribute Semantics

TA_ROUTINGNAME: *string[1...15]*
Routing criteria name.

TA_BUFTYPE: *type1*[*:subtype1[,subtype2* . . .]][*;type2*[*:subtype3*[, . . .]]] . . .
List of types and subtypes of data buffers for which this routing entry is valid. A maximum of 32 type/subtype combinations are allowed. The types are restricted to be one of FML, VIEW, X_C_TYPE, or X_COMMON. No subtype can be specified for type FML, and subtypes are required for types VIEW, X_C_TYPE, and X_COMMON (''*'' is not allowed). Note that subtype names should not contain semicolon, colon, comma, or asterisk characters. Duplicate type/subtype pairs can not be specified for the same routing criteria name; more than one routing entry can have the same criteria name as long as the type/subtype pairs are unique. If multiple buffer types are specified for a single routing entry, the data types of the routing field for each buffer type must be the same.

TA_FIELD: *string[1...30]*
The routing field name. When TA_TYPE=FACTORY, this is assumed to be a field that is specified in an NVList parameter to PortableServer::POA::create _reference_with_criteria for an interface that has this factory routing criteria associated with it. See section on Factory-based routing for more details. When TA_TYPE=SERVICE this field is assumed to be an FML buffer or view field name that is identified in an FML field table (using the FLDTBLDIR and FIELDTBLS environment variables) or an FML view table (using the VIEWDIR and VIEWFILES environment variables), respectively. This information is used to get the associated field value for data dependent routing during the sending of a message.

TA_FIELDTYPE (Factory-based Routing Only)
Routing field type. This field is only valid if TA_TYPE=FACTORY. Valid types are: SHORT, LONG, FLOAT, DOUBLE, CHAR, STRING. Specification of this attribute is only valid for factory-based routing criteria.

TA_RANGES: *carray[1...2048]*
The ranges and associated server groups for the routing field. The format of *string* is a comma-separated, ordered list of range/group name pairs. A range/group name pair has the following format:

```
lower[-upper]:group
```

lower and *upper* are signed numeric values or character strings in single quotes. *lower* must be less than or equal to *upper*. To embed a single quote in a character string value, it must be preceded by two backslashes (for example, 'O\\'Brien'). The value MIN can be used to indicate the minimum value for the data type of the associated field on the machine. The value MAX can be used to indicate the maximum value for the data type of the associated field on the machine. Thus, "MIN--5" is all numbers less than or equal to -5, and "6-MAX" is all numbers greater than or equal to 6.

The meta-character ''*" (wild-card) in the position of a range indicates any values not covered by the other ranges previously seen in the entry; only one wild-card range is allowed per entry and it should be last (ranges following it will be ignored).

The routing field can be of any data type supported in FML. A numeric routing field must have numeric range values, and a string routing field must have string range values.

String range values for string, carray, and character field types must be placed inside a pair of single quotes and can not be preceded by a sign. Short and long integer values are a string of digits, optionally preceded by a plus or minus sign. Floating point numbers are of the form accepted by the C compiler or atof(3): an optional sign, then a string of digits optionally containing a decimal point, then an optional e or E followed by an optional sign or space, followed by an integer.

The group name indicates the associated group to which the request is routed if the field matches the range. A group name of ''*" indicates that the request can go to any group where a server offers the desired service.

Limitation: Attribute values greater than 256 bytes in length will disable interoperability with BEA TUXEDO Release 4.2.2 and earlier.

TA_STATE:

GET: {VALid}

A GET operation will retrieve configuration information for the selected T_ROUTING object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

VALid	T_ROUTING object is defined. Note that this is the only valid state for this class. Routing criteria are never ACTive; rather, they are associated through the configuration with service names and are acted upon at runtime to provide data dependent routing. This state is INActive equivalent for the purpose of permissions checking.

SET: {NEW|INValid}

A SET operation will update configuration information for the selected T_ROUTING object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_ROUTING object for application. State change allowed only when in the INValid state. Successful return leaves the object in the VALid state.
unset	Modify an existing T_ROUTING object. This combination is not allowed in the INValid state. Successful return leaves the object state unchanged.
INValid	Delete T_ROUTING object for application. State change allowed only when in the VALid state. Successful return leaves the object in the INValid state.

TA_TYPE

Routing criteria type. Valid values are "FACTORY" or "SERVICE". "FACTORY" specifies that the routing criteria applies to factory-based routing for a CORBA interface. The specification of TYPE=FACTORY is mandatory for a factory-based routing criteria. "SERVICE" specifies that the routing criteria applies to data-dependent routing for a BEA TUXEDO service. Default is "SERVICE". Specification of this attribute is optional for data-dependent routing criteria. Note that the type specified affects the validity and possible values for other fields defined for this MIB class. These are noted for each field. TA_TYPE is required for SET operations for Factory-Based Routing criteria.

Limitations None.

T_SERVER CLASS

Overview

The T_SERVER class represents configuration and run-time attributes of servers within an application. These attribute values identify and characterize configured servers as well as provide run-time tracking of statistics and resources associated with each server object.

Attribute Table

T_SERVER Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_SRVGRP(r)(*)	string	ru-r--r--	*string[1...30]*	N/A
TA_SRVID(r)(*)	long	ru-r--r--	1 <= *num* < 30,001	N/A
TA_SERVERNAME(k)(r)	string	rw-r--r--	*string[1...78]*	N/A
TA_GRPNO(k)	long	r--r--r--	1 <= *num* < 30,000	N/A
TA_STATE(k)	string	rwxr-xr--	GET:"{ACT|INA|MIG|CLE|RES|SUS|PAR|DEA}" SET:"{NEW|INV|ACT|INA|DEA}"	N/A N/A
TA_BASESRVID	long	r--r--r--	1 <= *num* < 30,001	N/A
TA_CLOPT	string	rwyr--r--	*string[0...256]*	"-A"
TA_ENVFILE	string	rwyr--r--	*string[0...78]*	""
TA_GRACE	long	rwyr--r--	0 <= *num*	86,400
TA_MAXGEN	long	rwyr--r--	1 <= *num* < 256	1
TA_MAX	long	rwxr--r--	1 <= *num* < 1,001	1
TA_MIN	long	rwyr--r--	1 <= *num* < 1,001	1
TA_RCMD	string	rwyr--r--	*string[0...78]*	""
TA_RESTART	string	rwyr--r--	"{Y|N}"	N
TA_SEQUENCE(k)	long	rwxr--r--	1 <= *num* < 10,000	>= 10,000
TA_SYSTEM_ACCESS	string	rwyr--r--	"{FASTPATH|PROTECTED}"	([1])
TA_CONV(k)	string	rw-r--r--	"{Y|N}"	N
TA_REPLYQ	string	rw-r--r--	"{Y|N}"	N
TA_RPPERM	long	rw-r--r--	0001 <= *num* <= 0777	([1])

T_SERVER Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_RQADDR(k)	string	rw-r--r--	*string[0...30]*	"*GRPNO.SRVID*"
TA_RQPERM	long	rw-r--r--	0001 <= *num* <= 0777	([1])
TA_LMID(k)	string	R--R--R--	*LMID*	N/A
TA_GENERATION	long	R--R--R--	1 <= *num* < 32K	N/A
TA_PID(k)	long	R--R--R--	1 <= *num*	N/A
TA_RPID	long	R--R--R--	1 <= *num*	N/A
TA_RQID	long	R--R--R--	1 <= *num*	N/A
TA_TIMERESTART	long	R--R--R--	1 <= *num*	N/A
TA_TIMESTART	long	R--R--R--	1 <= *num*	N/A
T_SERVER Class: LOCAL Attributes				
TA_NUMCONV	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMDEQUEUE	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMENQUEUE	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMPOST	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMREQ	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMSUBSCRIBE	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMTRAN	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMTRANABT	long	R-XR-XR--	0 <= *num*	N/A
TA_NUMTRANCMT	long	R-XR-XR--	0 <= *num*	N/A
TA_TOTREQC	long	R-XR-XR--	0 <= *num*	N/A
TA_TOTWORKL	long	R-XR-XR--	0 <= *num*	N/A
TA_CLTLMID	string	R--R--R--	*LMID*	N/A
TA_CLTPID	long	R--R--R--	1 <= *num*	N/A
TA_CLTREPLY	string	R--R--R--	"{Y|N}"	N/A
TA_CMTRET	string	R--R--R--	"{COMPLETE|LOGGED}"	N/A
TA_CURCONV	long	R--R--R--	0 <= *num*	N/A
TA_CUROBJECTS	long	R--R--R--	0 <= *num*	N/A
TA_CURINTERFACE	string	R--R--R--	*string*[0..128]	N/A

T_SERVER Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_CURREQ	long	R--R--R--	0 <= *num*	N/A
TA_CURRSERVICE	string	R--R--R--	*string[0...15]*	N/A
TA_CURTIME	long	R--R--R--	1 <= *num*	N/A
TA_LASTGRP	long	R--R--R--	1 <= *num* < 30,000	N/A
TA_SVCTIMEOUT	long	R--R--R--	0 <= *num*	N/A
TA_TIMELEFT	long	R--R--R--	0 <= *num*	N/A
TA_TRANLEV	long	R--R--R--	0 <= *num*	N/A

(k) - GET key field
(r) - Required field for object creation (SET TA_STATE NEW)
(*) - GET/SET key, one or more required for SET operations

[1]Defaults to value set for this attribute in Class T_DOMAIN

Attribute Semantics

TA_SRVGRP: *string[1...30]*
Logical name of the server group. Server group names cannot contain an asterisk (*), comma, or colon.

TA_SRVID: 1 <= *num* < 30,001
Unique (within the server group) server identification number.

TA_SERVERNAME: *string[1...78]*
Name of the server executable file. The server identified by TA_SERVERNAME will run on the machine(s) identified by the T_GROUP:TA_LMID attribute for this server's server group. If a relative pathname is given, then the search for the executable file is done first in TA_APPDIR, then in TA_TUXDIR/bin, then in /bin and /usr/bin, and then in <path>, where <path> is the value of the first PATH= line appearing in the machine environment file, if one exists. Note that the attribute value returned for an active server will always be a full pathname. The values for TA_APPDIR and TA_TUXDIR are taken from the appropriate T_MACHINE object. See discussion of the T_MACHINE:TA_ENVFILE attribute for a more detailed discussion of how environment variables are handled.

TA_GRPNO: 1 <= *num* < 30,000
Group number associated with this server's group.

TA_STATE:

GET:{ACTive|INActive|MIGrating|CLEaning|RESta rting|SUSpended|PARtitioned|DEAd}

A GET operation will retrieve configuration and run-time information for the selected T_SERVER object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

ACTive	T_SERVER object defined and active. This is not an indication of whether the server is idle or busy. An active server with a non 0-length TA_CURRSERVICE attribute should be interpreted as a busy server, that is, one that is processing a service request.
INActive	T_SERVER object defined and inactive.
MIGrating	T_SERVER object defined and currently in a state of migration to the server group's secondary logical machine. The secondary logical machine is the one listed in T_GROUP:TA_LMID attribute that does not match the T_GROUP:TA_CURLMID attribute. This state is ACTive equivalent for the purpose of determining permissions.
CLEaning	T_SERVER object defined and currently being cleaned up by the system after an abnormal death. Note that restartable servers may enter this state if they exceed TA_MAXGEN starts/restarts within their TA_GRACE period. This state is ACTive equivalent for the purpose of determining permissions.
RESta rting	T_SERVER object defined and currently being restarted by the system after an abnormal death. This state is ACTive equivalent for the purpose of determining permissions.
SUSpended	T_SERVER object defined and currently suspended pending shutdown. This state is ACTive equivalent for the purpose of determining permissions.

PARtitioned	T_SERVER object defined and active; however, the machine where the server is running is currently partitioned from the T_DOMAIN:TA_MASTER site. This state is ACTive equivalent for the purpose of determining permissions.
DEAd	T_SERVER object defined, identified as active in the bulletin board, but currently not running due to an abnormal death. This state will exist only until the BBL local to the server notices the death and takes action (RESTarting|CLEaning). Note that this state will only be returned if the MIB_LOCAL TA_FLAGS value is specified and the machine where the server was running is reachable. This state is ACTive equivalent for the purpose of determining permissions.

SET: {NEW|INValid|ACTive|INActive|DEAd}

A SET operation will update configuration and run-time information for the selected T_SERVER object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_SERVER object for application. State change allowed only when in the INValid state. Successful return leaves the object in the INActive state.
unset	Modify an existing T_SERVER object. This combination is allowed only when in the ACTive or INActive state. Successful return leaves the object state unchanged.
INValid	Delete T_SERVER object for application. State change allowed only when in the INActive state. Successful return leaves the object in the INValid state.

ACTive	Activate the T_SERVER object. State change allowed only when in the INActive state. (Servers in the MIGrating state must be restarted by setting the T_GROUP:TA_STATE to ACTive.) For the purpose of determining permissions for this state transition, the active object permissions are considered (that is, --x--x--x). Successful return leaves the object in the ACTive state. The TMIB_NOTIFY TA_FLAG value should be used when activating a server if status on the individual server is required.
INACtive	Deactivate the T_SERVER object. State change allowed only when in the ACTive state. Successful return leaves the object in the INActive state. The TMIB_NOTIFY TA_FLAG value should be used when deactivating a server if status on the individual server is required.
DEAd	Deactivate the T_SERVER object by sending the server a SIGTERM signal followed by a SIGKILL signal if the server is still running after the appropriate timeout interval (see TA_MIBTIMEOUT in MIB(5)). Note that by default, a SIGTERM signal will cause the server to initiate orderly shutdown and the server will become inactive even if it is restartable. If a server is processing a long running service or has chosen to disable the SIGTERM signal, then SIGKILL may be used and will be treated by the system as an abnormal termination. State change allowed only when in the ACTive or SUSpended state. Successful return leaves the object in the INActive, CLEaning or RESterting state.

TA_BASESRVID: 1 <= *num* < 30,001

Base server identifier. For servers with a TA_MAX attribute value of 1, this attribute will always be the same as TA_SRVID. However, for servers with a TA_MAX value greater than 1, this attribute indicates the base server identifier for the set of servers configured identically.

TA_CLOPT: *string[0...256]*

Command line options to be passed to server when it is activated. See the servopts(5) reference page for details. Limitation: Run-time modifications to this attribute will not affect a running server.

TA_ENVFILE: *string[0...78]*

Server specific environment file. See T_MACHINE:TA_ENVFILE for a complete discussion of how this file is used to modify the environment. Limitation: Run-time modifications to this attribute will not affect a running server.

TA_GRACE: 0 <= *num*

The period of time, in seconds, over which the T_SERVER:TA_MAXGEN limit applies. This attribute is meaningful only for restartable servers, that is, if the T_SERVER:TA_RESTART attribute is set to "Y". When a restarting server would exceed the TA_MAXGEN limit but the TA_GRACE period has expired, the system resets the current generation (T_SERVER:TA_GENERATION) to 1 and resets the initial boot time (T_SERVER:TA_TIMESTART) to the current time. A value of 0 for this attribute indicates that a server should always be restarted.

Note that servers sharing a request queue (that is, equal values for T_SERVER:TA_RQADDR) should have equal values for this attribute. If they do not, then the first server activated will establish the run-time value associated with all servers on the queue.

Limitation: Run-time modifications to this attribute will affect a running server and all other active servers with which it is sharing a request queue. However, only the selected server's configuration parameter is modified. Thus, the behavior of the application depends on the order of boot in subsequent activations unless the administrator ensures that all servers sharing a queue have the same value for this attribute.

TA_MAXGEN: 1 <= *num* < 256

Number of generations allowed for a restartable server (T_SERVER:TA_RESTART == "Y") over the specified grace period (T_SERVER:TA_GRACE). The initial activation of the server counts as one generation and each restart also counts as one. Processing after the maximum generations is exceeded is discussed above with respect to TA_GRACE.

Note that servers sharing a request queue (that is, equal values for T_SERVER:TA_RQADDR) should have equal values for this attribute. If they do not, then the first server activated will establish the run-time value associated with all servers on the queue.

Limitation: Run-time modifications to this attribute will affect a running server and all other active servers with which it is sharing a request queue. However, only the selected server's configuration parameter is modified. Thus, the behavior of the application depends on the order of boot in

subsequent activations unless the administrator ensures that all servers sharing a queue have the same value for this attribute.

TA_MAX: 1 <= *num* < 1,001
: Maximum number of occurrences of the server to be booted. Initially, tmboot(3c) boots T_SERVER:TA_MIN objects of the server, and additional objects may be started individually (by starting a particular server id) or through automatic spawning (conversational servers only). Run-time modifications to this attribute will affect all running servers in the set of identically configured servers (see TA_BASESRVID above) as well as the configuration definition of the server.

TA_MIN: 1 <= *num* < 1,001
: Minimum number of occurrences of the server to be booted by. If a T_SERVER:TA_RQADDR is specified and TA_MIN is greater than 1, then the servers will form an MSSQ set. The server identifiers for the servers will be T_SERVER:TA_SRVID up to TA_SRVID + T_SERVER:TA_MAX - 1. All occurrences of the server will have the same sequence number, as well as any other server parameters.

  Limitation: Run-time modifications to this attribute will not affect a running server.

TA_RCMD: *string[0...78]*
: Application specified command to be executed in parallel with the system restart of an application server. This command must be an executable UNIX System file.

  Note that servers sharing a request queue (that is, equal values for T_SERVER:TA_RQADDR) should have equal values for this attribute. If they do not, then the first server activated will establish the run-time value associated with all servers on the queue.

  Limitation: Run-time modifications to this attribute will affect a running server and all other active servers with which it is sharing a request queue. However, only the selected server's configuration parameter is modified. Thus, the behavior of the application depends on the order of boot in subsequent activations unless the administrator ensures that all servers sharing a queue have the same value for this attribute.

TA_RESTART: {Y|N}

Restartable ("Y") or non-restartable ("N") server. If server migration is specified for this server group (T_DOMAIN:TA_OPTIONS/MIGRATE and T_GROUP:TA_LMID with alternate site), then this attribute must be set to "Y". Note that servers sharing a request queue (that is, equal values for T_SERVER:TA_RQADDR) should have equal values for this attribute. If they do not, then the first server activated will establish the run-time value associated with all servers on the queue.

Limitation: Run-time modifications to this attribute will affect a running server and all other active servers with which it is sharing a request queue. However, only the selected server's configuration parameter is modified. Thus, the behavior of the application depends on the order of boot in subsequent activations unless the administrator ensures that all servers sharing a queue have the same value for this attribute.

TA_SEQUENCE: 1 <= *num* < 10,000

Specifies when this server should be booted (tmboot(1)) or shutdown (tmshutdown(1)) relative to other servers. T_SERVER objects added without a TA_SEQUENCE attribute specified or with an invalid value will have one generated for them that is 10,000 or more and is higher than any other automatically selected default. Servers are booted by tmboot(1) in increasing order of sequence number and shutdown by tmshutdown(1) in decreasing order. Run-time modifications to this attribute affect only tmboot(1) and tmshutdown(1) and will affect the order in which running servers may be shutdown by a subsequent invocation of tmshutdown(1).

TA_SYSTEM_ACCESS: {FASTPATH|PROTECTED}

Mode used by BEA TUXEDO system libraries within this server process to gain access to BEA TUXEDO system's internal tables. See T_DOMAIN:TA_SYSTEM_ACCESS for a complete discussion of this attribute.

Limitation: Run-time modifications to this attribute will not affect a running server.

TA_CONV: {Y|N}

Conversational server ("Y") or request/response server ("N").

TA_REPLYQ: {Y|N}

Allocate a separate reply queue for the server (TA_REPLYQ == "Y"). MSSQ servers that expect to receive replies should set this attribute to "Y".

TA_RPPERM: 0001 <= *num* <= 0777

UNIX System permissions for the server's reply queue. If a separate reply queue is not allocated (T_SERVER:TA_REPLYQ == "N"), then this attribute is ignored. Limitation: This is a UNIX System specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_RQADDR: *string[0...30]*

Symbolic address of the request queue for the server. Specifying the same TA_RQADDR attribute value for more than one server is the way multiple server, single queue (MSSQ) sets are defined. Servers with the same TA_RQADDR attribute value must be in the same server group.

TA_RQPERM: 0001 <= *num* <= 0777

UNIX System permissions for the server's request queue.

Limitation: This is a UNIX System specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_LMID: *LMID*

Current logical machine on which the server is running.

TA_GENERATION: 1 <= *num* < 32K

Generation of the server. When a server is initially booted via tmboot(1) or activated through the TM_MIB(5), its generation is set to 1. Each time the server dies abnormally and is restarted, its generation is incremented. Note that when T_SERVER:TA_MAXGEN is exceeded and T_SERVER:TA_GRACE has expired, the server will be restarted with the generation reset to 1.

TA_PID: 1 <= *num*

UNIX System process identifier for the server. Note that this may not be a unique attribute since servers may be located on different machines allowing for duplication of process identifiers.

Limitation: This is a UNIX System specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_RPID: 1 <= *num*

UNIX System message queue identifier for the server's reply queue. If a separate reply queue is not allocated (T_SERVER:TA_REPLYQ == "N"), then this attribute value will be the same as T_SERVER:TA_RQID.

Limitation: This is a UNIX System specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_RQID: 1 <= *num*

UNIX System message queue identifier for the server's request queue. If a separate reply queue is not allocated (T_SERVER:TA_REPLYQ == "N"), then this attribute value will be the same as T_SERVER:TA_RPID.

Limitation: This is a UNIX System specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_TIMERESTART: 1 <= *num*

Time, in seconds, since 00:00:00 UTC, January 1, 1970, as returned by the time(2) system call on T_SERVER:TA_LMID, when the server was last started or restarted.

TA_TIMESTART: 1 <= *num*

Time, in seconds, since 00:00:00 UTC, January 1, 1970, as returned by the time(2) system call on T_SERVER:TA_LMID, when the server was first started. Restarts of the server do not reset this value; however, if T_SERVER:TA_MAXGEN is exceeded and T_SERVER:TA_GRACE is expired, this attribute will be reset to the time of the restart.

TA_NUMCONV: 0 <= *num*

Number of conversations initiated by this server via tpconnect(3c).

TA_NUMDEQUEUE: 0 <= *num*

Number of dequeue operations initiated by this server via tpdequeue(3c).

TA_NUMENQUEUE: 0 <= *num*

Number of enqueue operations initiated by this server via tpenqueue(3c).

TA_NUMPOST: 0 <= *num*

Number of postings initiated by this server via tppost(3c).

TA_NUMREQ: 0 <= *num*

Number of requests made by this server via tpcall(3c) or tpacall(3c).

TA_NUMSUBSCRIBE: 0 <= *num*

Number of subscriptions made by this server via tpsubscribe(3c).

TA_NUMTRAN: 0 <= *num*

Number of transactions begun by this server since its last (re)start.

TA_NUMTRANABT: 0 <= *num*

Number of transactions aborted by this server since its last (re)start.

TA_NUMTRANCMT: 0 <= *num*
: Number of transactions committed by this server since its last (re)start.

TA_TOTREQC: 0 <= *num*
: Total number of requests completed by this server. For conversational servers (T_SERVER:TA_CONV == "Y"), this attribute value indicates the number of completed incoming conversations. This is a run-time attribute that is kept across server restart but is lost at server shutdown.

TA_TOTWORKL: 0 <= *num*
: Total workload completed by this server. For conversational servers (T_SERVER:TA_CONV == "Y"), this attribute value indicates the workload of completed incoming conversations. This is a run-time attribute that is kept across server restart but is lost at server shutdown.

TA_CLTLMID: *LMID*
: Logical machine for the initiating client or server. The initiating client or server is the process that made the service request that the server is currently working on.

TA_CLTPID: 1 <= *num*
: UNIX System process identifier for the initiating client or server. Limitation: This is a UNIX System specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_CLTREPLY: {Y|N}
: The initiating client or server is expecting a reply ("Y") or is not expecting a reply ("N").

TA_CMTRET: {COMPLETE|LOGGED}
: Setting of the TP_COMMIT_CONTROL characteristic for this server. See the description of the BEA TUXEDO system ATMI function tpscmt(3) for details on this characteristic.

TA_CURCONV: 0 <= *num*
: Number of conversations initiated by this server via tpconnect(3c) that are still active.

TA_CUROBJECTS
: The number of entries in use in the bulletin board object table for this server. Scope is local.

TA_CURINTERFACE
: The interface name of the interface currently active in this server. Scope is local.

TA_CURREQ: 0 <= *num*
: Number of requests initiated by this server via tpcall(3c) or tpacall(3c) that are still active.

TA_CURRSERVICE: *string[0...15]*
: Service name that the server is currently working on, if any.

TA_CURTIME: 1 <= *num*
: Current time, in seconds, since 00:00:00 UTC, January 1, 1970, as returned by the time(2) system call on T_SERVER:TA_LMID. This attribute can be used to compute elapsed time from the T_SERVER:TA_TIMESTART and T_SERVER:TA_TIMERESTART attribute values.

TA_LASTGRP: 1 <= *num* < 30,000
: Server group number (T_GROUP:TA_GRPNO) of the last service request made or conversation initiated from this server outward.

TA_SVCTIMEOUT: 0 <= *num*
: Time left, in seconds, for this server to process the current service request, if any. A value of 0 for an active service indicates that no timeout processing is being done. See T_SERVICE:TA_SVCTIMEOUT for more information.

TA_TIMELEFT: 0 <= *num*
: Time left, in seconds, for this server to receive the reply for which it is currently waiting before it will timeout. This timeout may be a transactional timeout or a blocking timeout.

TA_TRANLEV: 0 <= *num*
: Current transaction level for this server. 0 indicates that the server is not currently involved in a transaction.

Limitations

None.

T_SERVICE CLASS

Overview

The T_SERVICE class represents configuration attributes of services within an application. These attribute values identify and characterize configured services. A T_SERVICE object provides activation time configuration attributes for services not specifically configured as part of the T_SVCGRP class. Run-time information about services active in the application is provided solely through the T_SVCGRP class. Run-time updates to the T_SERVICE class are usually not reflected in active T_SVCGRP objects (TA_ROUTINGNAME is the exception).

Both the T_SERVICE class and the T_SVCGRP class define activation time attribute settings for service names within the application. When a new service is activated (advertised), either due to initial activation of a server or due to a call to tpadvertise(3c), the following hierarchy exists for determining the attribute values to be used at service startup time.

1. If a matching configured T_SVCGRP object exists (matching service name and server group), then the attributes defined in that object are used to initially configure the advertised service.
2. Otherwise, if a matching configured T_SERVICE object exists (matching service name), then the attributes defined in that object are used to initially configure the advertised service.
3. Otherwise, if any configured T_SVCGRP objects are found with matching TA_SERVICENAME attribute values, then the first one found is used to initially configure the advertised service.
4. If none of the preceding cases is used, then the system defaults for service attributes are used to initially configure the advertised service.

The specification of configuration attributes for application services is completely optional, that is, services advertised by servers as they are activated will take on the established default service attribute values if configured values are not available (see above for description of how attribute values are identified at service activation time). Service names to be offered by a server are built in at runtime (see buildserver(1)) and may be overridden by the command line options specified for a server object (see T_SERVER:TA_CLOPT and servopts(5)).

Attribute Table

T_SERVICE Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_SERVICENAME(r)(*)	string	ru-r--r--	*string[1...15]*	N/A
TA_STATE(k)	string	rw-r--r--	GET:"{ACT|INA}" SET:"{NEW|INV}"	N/A N/A
TA_AUTOTRAN	string	rwyr--r--	"{Y|N}"	"N"
TA_LOAD	long	rwyr--r--	1 <= *num* < 32K	50
TA_PRIO	long	rwyr--r--	1 <= *num* < 101	50
TA_SVCTIMEOUT	long	rwyr--r--	0 <= *num*	0
TA_TRANTIME	long	rwyr--r--	0 <= *num*	30
TA_BUFTYPE	string	rw-r--r--	*string[1...256]*	"ALL"
TA_ROUTINGNAME	string	rwxr--r--	*string[0...15]*	""

(k) - GET key field
(r) - Required field for object creation (SET TA_STATE NEW)
(*) - GET/SET key, one or more required for SET operations

Attribute Semantics

TA_SERVICENAME: *string[1...15]*

Service name.

TA_STATE:

GET: {ACTive|INActive}

A GET operation will retrieve configuration information for the selected T_SERVICE object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

ACTive	T_SERVICE object is defined and at least one T_SVCGRP object with a matching TA_SERVICENAME value is active.
INActive	T_SERVICE object is defined and no T_SVCGRP object with a matching TA_SERVICENAME value is active.

SET: {NEW|INValid}

A SET operation will update configuration information for the selected T_SERVICE object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_SERVICE object for application. State change allowed only when in the INValid state. Successful return leaves the object in the INActive state. Limitation: Unconfigured services may still be active by virtue of a server advertising them. In this case, the creation of a new T_SERVICE object is not allowed.
unset	Modify an existing T_SERVICE object. This combination is not allowed in the INValid state. Successful return leaves the object state unchanged.
INValid	Delete T_SERVICE object for application. State change allowed only when in the INActive state. Successful return leaves the object in the INValid state.

TA_AUTOTRAN: {Y|N}

Automatically begin a transaction ("Y") when a service request message is received for this service if the request is not already in transaction mode. Limitation: Run-time updates to this attribute are not reflected in active T_SVCGRP objects.

TA_LOAD: 1 <= *num* < 32K

This T_SERVICE object imposes the indicated load on the system. Service loads are used for load balancing purposes, that is, queues with higher enqueued workloads are less likely to be chosen for a new request. Service loads have meaning only if the T_DOMAIN:TA_LDBAL is set to "Y". Limitation: Run-time updates to this attribute are not reflected in active T_SVCGRP objects.

TA_PRIO: 1 <= *num* < 101

This T_SERVICE object has the indicated dequeuing priority. If multiple service requests are waiting on a queue for servicing, the higher priority requests will be serviced first.

Limitation: Run-time updates to this attribute are not reflected in active T_SVCGRP objects.

TA_SVCTIMEOUT: 0 <= *num*

Time limit (in seconds) for processing requests for this service name. Servers processing service requests for this service will be abortively terminated (kill -9) if they exceed the specified time limit in processing the request. A value of 0 for this attribute indicates that the service should not be abortively terminated.

Limitations: Run-time updates to this attribute are not reflected in active T_SVCGRP objects. This attribute value is not enforced on BEA TUXEDO System Release 4.2.2 sites or earlier.

TA_TRANTIME: 0 <= *num*

Transaction timeout value in seconds for transactions automatically started for this T_SERVICE object. Transactions are started automatically when a request not in transaction mode is received and the T_SERVICE:TA_AUTOTRAN attribute value for the service is "Y".

Limitation: Run-time updates to this attribute are not reflected in active T_SVCGRP objects.

TA_BUFTYPE: *type1*[:*subtype1[*,*subtype2* . . .]][;*type2*[:*subtype3*[, . . .]]] . . .

List of types and subtypes of data buffers accepted by this service. Up to 32 type/subtype combinations are allowed. Types of data buffers provided with BEA TUXEDO system are FML (for FML buffers), VIEW, X_C_TYPE, or X_COMMON (for FMLviews), STRING (for NULL terminated character arrays), and CARRAY or X_OCTET (for a character array that is neither encoded nor decoded during transmission). Of these types, only VIEW, X_C_TYPE, and X_COMMON have subtypes. A VIEW subtype gives the name of the particular VIEW expected by the service. Application types and subtypes can also be added (see tuxtypes(5)). For a buffer type that has subtypes, "*" can be specified for the subtype to indicate that the service accepts all subtypes for the associated buffer type.

A single service can only interpret a fixed number of buffer types, namely those found in its buffer type switch (see tuxtypes(5)). If the TA_BUFTYPE attribute value is set to ALL, that service will accept all buffer types found in its buffer type switch.

A type name can be 8 characters or less in length and a subtype name can be 16 characters or less in length. Note that type and subtype names should not contain semicolon, colon, comma, or asterisk characters.

Limitation: This attribute value represents the buffer types that must be supported by each and every instance of an application service with this service name. Since this attribute value is processed at service activation time, updates to this attribute are allowed only when there are no active T_SVCGRP objects with matching service names.

TA_ROUTINGNAME: *string[0...15]*
This T_SERVICE object has the indicated routing criteria name. Active updates to this attribute will be reflected in all associated T_SVCGRP objects.

Limitations None.

T_SVCGRP CLASS

Overview

The T_SVCGRP class represents configuration and run-time attributes of services/groups within an application. These attribute values identify and characterize configured services/groups, and provide run-time tracking of statistics and resources associated with each object.

Both the T_SERVICE class and the T_SVCGRP class define activation time attribute settings for service names within the application. When a new service is activated (advertised), either due to initial activation of a server or due to a call to tpadvertise(3c), the following hierarchy exists for determining the attribute values to be used at service startup time.

1. If a matching configured T_SVCGRP object exists (matching service name and server group), then the attributes defined in that object are used to initially configure the advertised service.
2. Otherwise, if a matching configured T_SERVICE object exists (matching service name), then the attributes defined in that object are used to initially configure the advertised service.
3. Otherwise, if any configured T_SVCGRP objects are found with matching TA_SERVICENAME attribute values, then the first one found is used to initially configure the advertised service.
4. If none of the preceding cases is used, then the system defaults for service attributes are used to initially configure the advertised service.

The specification of configuration attributes for application services is completely optional, that is, services advertised by servers as they are activated will take on the established default service attribute values if configured values are not available (see above for description of how attribute values are identified at service activation time). Service names to be offered by a server are built in at runtime (see buildserver(1)) and may be overridden by the command line options specified for a server object (see T_SERVER:TA_CLOPT and servopts(5).

Once a T_SVCGRP object is active, it is represented solely by the T_SVCGRP class. A particular service name/group name combination may have more than one associated T_SVCGRP class at runtime if there are multiple servers within the group offering the service.

Attribute Table

T_SVCGRP Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_SERVICENAME(r)(*)	string	ru-r--r--	*string[1...15]*	N/A
TA_SRVGRP(r)(*)	string	ru-r--r--	*string[1...30]*	N/A
TA_GRPNO(k)	long	r--r--r--	1 <= *num* < 30,000	N/A
TA_STATE(k)	string	rwxr-xr--	GET:"{ACT|INA |SUS|PAR}" SET:"{NEW|INV |ACT|INA|SUS}"	N/A N/A
TA_AUTOTRAN	string	rwxr-xr--	"{Y|N}"	"N"
TA_LOAD	long	rwxr-xr--	1 <= *num* < 32K	50
TA_PRIO	long	rwxr-xr--	1 <= *num* < 101	50
TA_SVCTIMEOUT	long	rwyr-yr--	0 <= *num*	0
TA_TRANTIME	long	rwxr-xr--	0 <= *num*	30
TA_LMID(k)	string	R--R--R--	*LMID*	N/A
TA_RQADDR(*)	string	R--R--R--	*string[1...30]*	N/A
TA_SRVID(*)	long	R--R--R--	1 <= *num* < 30,001	N/A
TA_SVCRNAM	string	R-XR-XR--	*string[1...15]*	(2)
TA_BUFTYPE	string	r--r--r--	*string[1...256]*	N/A
TA_ROUTINGNAME	string	r--r--r--	*string[0...15]*	N/A
TA_SVCTYPE(k)	string	r--r--r--	"{APP|CALLABLE|SYSTEM}"	"APP"
T_SVCGRP Class: LOCAL Attributes				
TA_NCOMPLETED	long	R-XR-XR--	0 <= *num*	N/A
TA_NQUEUED	long	R--R--R--	0 <= *num* < 32K	N/A

(k) - GET key field
(r) - Required field for object creation (SET TA_STATE NEW)
(*) - GET/SET key, one or more required for SET operations[1]

[1] SET operations on this class must specify sufficient key fields to uniquely identify the object being addressed. If the object is active, then it may be necessary to augment the TA_SERVICENAME and TA_SRVGRP key fields with either TA_RQADDR or TA_SRVID.

Modifications to an active object will affect that object and the related configuration record but not other active objects that may have derived their run-time attributes from the same configuration record.

[2]If nothing is specified for this attribute, it defaults to TA_SERVICENAME.

Attribute Semantics

TA_SERVICENAME: *string[1...15]*
Service name.

TA_SRVGRP: *string[1...30]*
Server group name. Server group names cannot contain an asterisk (*), comma, or colon. The hierarchy of the search for service attributes to be used at service activation time is described in the previous T_SVCGRP OVERVIEW section.

TA_GRPNO: 1 <= *num* < 30,000
Server group number.

TA_STATE:

GET: {ACTive|INActive|SUSpended|PARtitioned}
A GET operation will retrieve configuration and run-time information for the selected T_SVCGRP object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

ACTive	T_SVCGRP object is active within the server identified by the returned values for the TA_SRVGRP and TA_SRVID attributes. Attribute values returned indicate the current run-time instance of the service and may not be reflected in the configuration instance if temporary updates have been performed.
INActive	T_SVCGRP object is defined and inactive.
SUSpended	T_SVCGRP object defined, active, and currently suspended. This service is not available for access by the application in this state. This state is ACTive equivalent for the purpose of determining permissions.

PARtitioned	T_SVCGRP object defined, active, and currently partitioned from the master site of the application. This service is not available for access by the application in this state. This state is ACTive equivalent for the purpose of determining permissions.

SET: {NEW|INValid|ACTive|INActive|SUSpended}

A SET operation will update configuration and run-time information for the selected T_SVCGRP object. Note that run-time modifications to a service object may affect more than one active server. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW	Create T_SVCGRP object for application. State change allowed only when in the INValid state. Successful return leaves the object in the INActive state. ~Limitation: Unconfigured services may still be active by virtue of a server advertising them. In this case, the service class state is ACTive and cannot be updated.
unset	Modify an existing T_SVCGRP object. This combination is not allowed in the INValid state. Successful return leaves the object state unchanged.
INValid	Delete T_SVCGRP object for application. State change allowed only when in the INActive state. Successful return leaves the object in the INValid state.
ACTive	Activate (advertise) the T_SVCGRP object. State change allowed only when in the INActive, SUSpended or INValid states. Either TA_SRVID or TA_RQADDR must be specified with this state change. For the purpose of determining permissions for this state transition, the active object permissions are considered (that is, --x--x--x). Successful return leaves the object in the ACTive state. Limitation: State change not permitted for service names (TA_SERVICENAME) beginning with the reserved string ".".

INActive	Deactivate the T_SVCGRP object. State change allowed only when in the SUSpended state. Successful return leaves the object in either the INActive (configured entries) or INValid (unconfigured entries) state. Limitation: State change not permitted for service names (TA_SERVICENAME) beginning with the reserved string "_".
SUSpended	Suspend the T_SVCGRP object. State change allowed only when in the ACTive state. Successful return leaves the object in the SUSpended state. Limitation: State change not permitted for service names (TA_SERVICENAME) beginning with the reserved string "_".

TA_AUTOTRAN: {Y|N}

Automatically begin a transaction ("Y") when a service request message is received for this service if the request is not already in transaction mode.

TA_LOAD: 1 <= *num* < 32K

This T_SVCGRP object imposes the indicated load on the system. Service loads are used for load balancing purposes, that is, queues with higher enqueued workloads are less likely to be chosen for a new request.

TA_PRIO: 1 <= *num* < 101

This T_SVCGRP object has the indicated dequeuing priority. If multiple service requests are waiting on a queue for servicing, the higher priority requests will be serviced first.

TA_SVCTIMEOUT: 0 <= *num*

Time limit (in seconds) for processing requests for this service name. Servers processing service requests for this service will be abortively terminated (kill -9) if they exceed the specified time limit in processing the request. A value of 0 for this attribute indicates that the service should not be abortively terminated.

Limitation: This attribute value is not enforced on BEA TUXEDO System Release 4.2.2 sites or earlier.

TA_TRANTIME: 0 <= *num*

Transaction timeout value in seconds for transactions automatically started for this T_SVCGRP object. Transactions are started automatically when a

request not in transaction mode is received and the T_SVCGRP:TA_AUTOTRAN attribute value for the service is "Y".

TA_LMID: *LMID*
: Current logical machine on which an active server offering this service is running.

TA_RQADDR: *string[1...30]*
: Symbolic address of the request queue for an active server offering this service. See T_SERVER:TA_RQADDR for more information on this attribute.

TA_SRVID: 1 <= *num* < 30,001
: Unique (within the server group) server identification number for an active server offering this service. See T_SERVER:TA_SRVID for more information on this attribute.

TA_SVCRNAM: *string[1...15]*
: Function name within the associated server assigned to process requests for this service. On a SET request, the server must be able to map the function name to a function using its symbol table to successfully advertise the service. In some situations (for example, direct calls to tpadvertise(3c) by the server), the function name for an ACTive service object will not be known and the string "?" will be returned as the attribute value.

  Limitation: This attribute may only be set along with a state change from INActive to ACTive.

TA_BUFTYPE: *string[1...256]*
: Configured buffer types accepted by this service.

  Limitation: This attribute is settable only via the corresponding T_SERVICE class object.

TA_ROUTINGNAME: *string[0...15]*
: Routing criteria name.

  Limitation: This attribute is settable only via the corresponding T_SERVICE class object.

TA_NCOMPLETED: 0 <= *num*
: Number of service requests completed with respect to the retrieved ACTive or SUSpended object since it was activated (advertised).

Limitation: This attribute is returned only when the T_DOMAIN:TA_LDBAL attribute is set to "Y".

TA_SVCTYPE: {APP|CALLABLE|SYSTEM}

Type of service. APP indicates an application defined service name. CALLABLE indicates a system provided callable service. SYSTEM indicates a system provided and system callable service. SYSTEM services are not available to application clients and servers for direct access. Note that when used as a GET key field, a delimited list ('|' delimiter) may be used to retrieve multiple types of service group entries on one request. By default, only APP services are retrieved.

Number of requests currently enqueued to this service. This attribute is incremented at enqueue time and decremented when the server dequeues the request. Limitation: This attribute is returned only when the T_DOMAIN:TA_MODEL attribute is set to SHM and the T_DOMAIN:TA_LDBAL attribute is set to "Y".

TA_NQUEUED: 0 <= *num*< 32K

Number of requests currently enqueued to this service. This attribute is incremented at enqueue time and decremented when the server dequeues the request.

Limitation: This attribute is returned only when the T_DOMAIN: TA_LDBAL attribute is set to "Y".

Limitations None.

T_TLISTEN CLASS

Overview

The T_TLISTEN class represents run-time attributes of /T listener processes for a distributed application.

Attribute Table

T_TLISTEN Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_LMID(k)	string	R--R--R--	*LMID*	N/A
TA_STATE(k)	string	R--R--R--	GET:"{ACT|INA}"	N/A
			SET:N/A	N/A

(k) - GET key field

Attribute Semantics

TA_LMID: *LMID*

Logical machine identifier.

TA_STATE:

GET: {INActive|ACTive}

A GET operation will retrieve run-time information for the selected T_TLISTEN object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

INActive	T_TLISTEN object not active.
ACTive	T_TLISTEN object active.

SET:

SET operations are not permitted on this class. This attribute is settable only via the corresponding T_SERVICE class object.

Limitations

This class is not available through the tpadmcall(3c) interface.

T_TLOG CLASS

Overview

The T_TLOG class represents configuration and run-time attributes of transaction logs. This class allows the user to manipulate logs within an application, that is, create, destroy, migrate, etc.

Attribute Table

T_TLOG Class Definition Attribute Table

Attribute[1]	Type	Permissions	Values	Default
TA_LMID(*)	string	r--r--r--	*LMID*	N/A
TA_STATE(k)	string	r-xr-xr--	GET:"{ACT|INA|WAR}" SET:"{WAR}"	N/A N/A
TA_TLOGCOUNT	long	r-xr-xr--	1 <= *num*	N/A
TA_TLOGINDEX	long	r-xr-xr--	0 <= *num*	N/A
TA_GRPNO(k)	long	r--r--r--	1 <= *num* < 30,000	Note 2
TA_TLOGDATA	string	r-xr-xr--	*string[1...256]*	Note 2

(k) - GET key field
(*) - GET/SET key, one or more required for SET operations

[1]All attributes in Class T_TLOG are local attributes

One or more TA_GRPNO and TA_TLOGDATA attribute values may be returned with each object of the T_TLOG class. The attribute values for each of these attributes belonging to the particular object are the TA_TLOGCOUNT number of occurrences beginning with the TA_TLOGINDEX.

Attribute Semantics

TA_LMID: *LMID*

Transaction log logical machine identifier.

TA_STATE:

GET: {ACTive|INActive|WARmstart}

A GET operation will retrieve log configuration and run-time information for the selected T_TLOG object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

ACTive	The transaction log exists and is actively logging commit records for transactions coordinated on the site. This corresponds to the associated T_MACHINE object being active.
INActive	The transaction log exists but is currently inactive. This state corresponds to the associated T_MACHINE object being inactive and can only be returned if the site has a tlisten(1) process running; otherwise, the site is unreachable and a object will not be returned.
WARmstart	The transaction log exists, is currently active, and is marked for warmstart processing. Warmstart processing will occur when the next server group is started on the site. This state is ACTive equivalent for the purposes of determining permissions.

SET: {WARmstart}

A SET operation will update log configuration and run-time information for the selected T_TLOG object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

unset	Modify T_TLOG object. Allowed only when in the ACTive state. Successful return leaves the object state unchanged. The only object modifications permitted on this class are additions to the transaction log. In this case, TA_TLOGINDEX and TA_TLOGCOUNT indicate the objects of TA_TLOGDATA to be added.
WARmstart	Initiate warmstart for the T_TLOG object. State change allowed only when in the ACTive state. Successful return leaves the object in the WARmstart state.

TA_TLOGCOUNT: 1 <= *num*

Number of transaction log data records (TA_TLOGDATA) counted, retrieved, or to be added. This attribute is ignored for SET operations with a state change indicated. For valid SET operations with no state change, this attribute indicates the number of log records to be added to an active transaction log. A GET operation with neither TA_GRPNO nor TA_TLOGDATA specified returns a count of in use log records. A GET operation with only TA_GRPNO set will

return a count of in use log records with a coordinator group matching the indicated group. A GET operation with only TA_TLOGDATA set ("") will return a count of in use log records and populate arrays of TA_TLOGDATA and TA_GRPNO attribute values corresponding to the in use log records. A GET operation with both TA_GRPNO and TA_TLOGDATA set ("") will return a count of in use log records with a coordinator group matching the indicated group and populate arrays of TA_TLOGDATA and TA_GRPNO attribute values corresponding to the in use log records.

TA_TLOGINDEX: 0 <= *num*
: Index of the first object specific attribute values (TA_GRPNO and TA_TLOGDATA) corresponding to this object.

TA_GRPNO: 1 <= *num* < 30,000
: Transaction coordinator's group number.

TA_TLOGDATA: *string[1...256]*
: Formatted transaction log entry. This attribute value should not be interpreted directly. Rather, it should be used solely as a means of migrating log records as part of server group migration.

Limitations None

T_TRANSACTION CLASS

Overview

The T_TRANSACTION class represents run-time attributes of active transactions within the application.

Attribute Table

T_TRANSACTION Class Definition Attribute Table

Attribute[1]	Type	Permissions	Values	Default
TA_COORDLMID(k)	string	R--R--R--	*LMID*	N/A
TA_LMID(k)	string	R--R--R--	*LMID*	N/A
TA_TPTRANID(*)	string	R--R--R--	*string[1...78]*	N/A
TA_XID(*)	string	R--R--R--	*string[1...78]*	N/A
TA_STATE(k)	string	R-XR-XR--	GET:"{ACT|ABY|ABD|COM|REA|DEC|SUS}" SET:"{ABD}"	N/A N/A
TA_TIMEOUT	long	R--R--R--	1 <= *num*	N/A
TA_GRPCOUNT	long	R--R--R--	1 <= *num*	N/A
TA_GRPINDEX	long	R--R--R--	0 <= *num*	N/A
TA_GRPNO	long	R--R--R--	1 <= *num* < 30,000	([2])
TA_GSTATE	long	R-XR-XR--	GET: "PREP|PABT|PCOM" SET:"{HCO|HAB}"	N/A N/A

(k) - GET key field
(*) - GET/SET key, one or more required for SET operations

[1]All attributes in Class T_TRANSACTION are local attributes.

[2]One or more TA_GRPNO and TA_GSTATE attribute values may be returned with each object of the T_TRANSACTION class. The attribute values for each of these attributes belonging to the particular object are the TA_GRPCOUNT number of occurrences beginning with the TA_GRPINDEX.

Attribute Semantics

TA_COORDLMID: *LMID*

Logical machine identifier of the server group responsible for coordinating the transaction.

TA_LMID: *LMID*

Retrieval machine logical machine identifier. Note that transaction attributes are primarily kept local to a site and coordinated via common transaction identifiers by transaction management servers (TMSs).

TA_TPTRANID: *string[1...78]*

Transaction identifier as returned from tpsuspend(3c) mapped to a string representation. The data in this field should not be interpreted directly by the user except for equality comparison.

TA_XID: *string[1...78]*

Transaction identifier as returned from tx_info(3c) mapped to a string representation. The data in this field should not be interpreted directly by the user except for equality comparison.

TA_STATE:

GET:{ACTive|ABortonlY|ABorteD|COMcalled|REAdy|DECided|SUSpended}

A GET operation will retrieve run-time information for the selected T_TRANSACTION object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned. Note that distinct objects pertaining to the same global transaction (equivalent transaction identifiers) may indicate differing states. In general, the state indicated on the coordinator's site (TA_COORDLMID) indicates the true state of the transaction. The exception is when a non-coordinator site notices a condition that transitions the transaction state to ABortonlY. This transition will eventually be propagated to the coordinator site and result in the rollback of the transaction, but this change may not be immediately reflected on the coordinator site. All states are ACTive equivalent for the purpose of determining permissions.

PREPrepare	Indicates that the transaction group contains servers that have called xa_end (TMSUSPEND) during the course of transactional work and that commit processing is beginning. This state will exist until either all servers that called xa_end (TMSUSPEND) have caused a call to xa_end (TMSUCESS), at which point the group state will become READy, or until one of the target servers does a rollback of the transaction at which point the group state will become either PostABorT or ABorteD.
PostABorT	Indicates that a server called xa_end (TPFAIL) and that the TMS has not yet called xa_rollback(). (i.e. that other servers that had called xa_end (TMSUSPEND) are being notified by the TMS in order to clean up their associated CORBA objects.
PostCOMmit	Not yet implemented.

SET: {ABorteD}

A SET operation will update run-time information for the selected T_TRANSACTION object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

unset	Modify an existing T_TRANSACTION object. This combination is allowed only when in the READy state and only for the purpose of updating an individual group's state (see TA_GSTATE below). Successful return leaves the object state unchanged.
ABorteD	Abort the T_TRANSACTION object for the application. State change allowed only when in the ACTive, ABortonlY, or COMcalled states. Successful return leaves the object in the ABorteD state.

TA_TIMEOUT: 1 <= *num*

Time left, in seconds, before the transaction will timeout on the retrieval site. Note that this attribute value is returned only when the transaction state (TA_STATE) is ACTive.

TA_GRPCOUNT: 1 <= *num*

Number of groups identified as participants in the transaction by the information returned from the retrieval site.

TA_GRPINDEX: 1 <= *num*

Index of the first group specific attribute values (TA_GRPNO and TA_GSTATE) corresponding to this object.

TA_GRPNO: 1 <= *num* < 30,000

Group number of the participating group.

TA_GSTATE:

GET: {ACTive|ABorteD|ReaDOnly|REAdy|HCOmmit|HABort|DONe}

A GET operation will retrieve runtime information for the selected T_TRANSACTION object(s) pertaining to the indicated group. The following states indicate the meaning of a TA_GSTATE returned in response to a GET request. States not listed will not be returned. Note that distinct objects pertaining to the same global transaction (equivalent transaction identifiers) may indicate differing states for individual groups. In general, the state indicated on the group's site indicates the true state of the group's participation in the transaction. The exception is when the coordinator site determines that the transaction should abort and sets each participant group state to ABorteD. This transition will be propagated to the group's site and result in the rollback of the group's work in the transaction but may not be reflected immediately.

ACTive	The transaction is active in the indicated group.
ABorteD	The transaction has been identified for rollback and rollback has been initiated for the indicated group.
ReaDOnly	The group has successfully completed the first phase of two-phase commit and has performed only read operations on the resource manager, thus making it unnecessary to perform the second phase of commit for this group.
REAdy	The group has successfully completed the first phase of two-phase commit and is ready to be committed.
HCOmmit	The group has been heuristically committed. This may or may not agree with the final resolution of the transaction.

HABort	The group has been heuristically rolled back. This may or may not agree with the final resolution of the transaction.
DONe	This group has completed the second phase of the two-phase commit.

SET: {HCOmmit|HABort}

A SET operation will update runtime information for the first group in the originating request within the selected T_TRANSACTION object. The following states indicate the meaning of a TA_GSTATE set in a SET request. States not listed may not be set. State transitions are allowed only when performed within the object representing the group's site (TA_LMID).

HCOmmit	Heuristically commit the group's work as part of the indicated transaction. State change allowed only when TA_GSTATE is REAdy, TA_STATE is REAdy, and the indicated group is not on the coordinator's site. Successful return leaves the object in the HCOmmit state.
HABort	Heuristically rollback the group's work as part of the indicated transaction. State change allowed only when TA_GSTATE is ACTive or REAdy, TA_STATE is REAdy, and the indicated group is not on the coordinator's site. Successful return leaves the object in the HABort state.

Limitations None.

T_ULOG CLASS

Overview

The T_ULOG class represents run-time attributes of userlog(3c) files within an application.

Attribute Table

T_ULOG Class Definition Attribute Table

Attribute[1]	Type	Permissions	Values	Default
TA_LMID(k)	string	R--R--R--	*LMID*	([2])
TA_PMID(x)	string	R--R--R--	*string[1...30]*	([2])
TA_MMDDYY(k)	long	R--R--R--	*mmddyy*	Current date
TA_STATE	string	R--R--R--	GET:"{ACT}" SET:N/A	N/A N/A
TA_ULOGTIME(k)	long	R--R--R--	*hhmmss*	000000
TA_ENDTIME(k)	long	K--K--K--	*hhmmss*	235959
TA_ULOGLINE(k)	long	R--R--R--	1 <= *num*	1
TA_ULOGMSG(x)	string	R--R--R--	*string[1...256]*	N/A
TA_TPTRANID(k)	string	R--R--R--	*string[1...78]*	N/A
TA_XID(k)	string	R--R--R--	*string[1...78]*	N/A
TA_PID(k)	long	R--R--R--	1 <= *num*	N/A
TA_SEVERITY(x)	string	R--R--R--	*string[1...30]*	N/A
TA_ULOGCAT(x)	string	R--R--R--	*string[1...30]*	N/A
TA_ULOGMSGNUM(k)	long	R--R--R--	1 <= *num*	N/A
TA_ULOGPROCNM(x)	string	R--R--R--	*string[1...30]*	N/A
(k) - GET key field (x) - Regular expression GET key field				

[1]All attributes in Class T_ULOG are local attributes.

[2]TA_LMID is a required field used by the system to determine which application log file should be accessed. It is not used to restrict returned records to only those generated from processes running on the indicated machine. In cases where multiple machines share a log file via a networked file system, multiple TA_LMID values may be returned

even though a specific value has been provided as a *key* field. For the same reasons, TA_PMID is not considered in directing the request to a particular machine, but is used in determining which records should be returned. In this capacity, it may be useful to leverage TA_PMID as a regular expression key field.

Attribute Semantics

TA_LMID: *LMID*
Retrieval machine logical machine identifier.

TA_PMID: *string[1...30]*
Physical machine identifier.

TA_MMDDYY: *mmddyy*
Date of userlog file found or to be accessed.

TA_STATE:

GET: {ACTive}
A GET operation will retrieve run-time information for the selected T_ULOG object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

ACTive	The object returned reflects an existing userlog file on the indicated logical machine.

SET:
SET operations are not permitted on this class.

TA_ULOGTIME: *hhmmss*
The time of the userlog message represented by this object. The value of this attribute is formed by multiplying the hour by 10,000, adding to that the minute multipled by 100, and finally adding in the seconds. When used as a key field, this attribute represents the start of the time range to be accessed for messages.

TA_ENDTIME: *hhmmss*
The latest time to be considered in a GET operation when accessing this userlog file.

TA_ULOGLINE: 1 <= *num*
: The line number of the userlog message returned/requested within the userlog file. When used as a key field for retrieval, this value indicates the starting line within the log file.

TA_ULOGMSG: *string[1...256]*
: The entire text of the userlog message as it appears in the userlog file.

TA_TPTRANID: *string[1...78]*
: Transaction identifier as returned from tpsuspend(3c). The data in this field should not be interpreted directly by the user except for equality comparison. Messages not associated with transactions will retrieve a 0-length string as the value for this attribute.

TA_XID: *string[1...78]*
: Transaction identifier as returned from tx_info(3c). The data in this field should not be interpreted directly by the user except for equality comparison. Messages not associated with transactions will retrieve a 0-length string as the value for this attribute.

TA_PID: 1 <= *num*
: Process identifier of the client or server that generated the userlog message.

TA_SEVERITY: *string[1...30]*
: Severity of message, if any.

TA_ULOGCAT: *string[1...30]*
: Catalog name from which the message was derived, if any.

TA_ULOGMSGNUM: 1 <= *num*
: Catalog message number, if the message was derived from a catalog.

TA_ULOGPROCNM: *string[1...30]*
: Process name of the client or server that generated the userlog message.

Limitations

Retrievals may be done only if the associated T_MACHINE object is also ACTive.

Retrievals for this class must be directed, that is, the TA_LMID attribute must be specified. Retrievals of log records written by workstation clients are available only if the log file used by the client is shared with one of the machines defined in the T_MACHINE class for the application. Otherwise, these log records are unavailable through this class.

Retrievals on this class which cannot be completely satisfied will always return a TA_MORE value of 1 indicating only that more information may be available for the originating request.

Diagnostics

There are two general types of errors that may be returned to the user when interfacing with TM_MIB(5). First, any of the three ATMI verbs (tpcall(3c), tpgetrply(3c), and tpdequeue(3c)) used to retrieve responses to administrative requests may return any error defined for them. These errors should be interpreted as described on the appropriate reference pages.

If, however, the request is successfully routed to a system service capable of satisfying the request and that service determines that there is a problem handling the request, then failure may be returned in the form of an application level service failure. In these cases, tpcall(3c) and tpcall(3c) will return an error with tpgetrply set to TPESVCFAIL and return a reply message containing the original request along with TA_ERROR, TA_STATUS, and TA_BADFLD fields further qualifying the error as described below. When a service failure occurs for a request forwarded to the system through the TMQFORWARD(5) server, the failure reply message will be enqueued to the failure queue identified on the original request (assuming the -d option was specified for TMQFORWARD).

When a service failure occurs during processing of an administrative request, the FML32 field TA_STATUS is set to a textual description of the failure, and the FML32 field TA_ERROR is set to indicate the cause of the failure as indicated below. All error codes are guaranteed to be negative.

[other]
: Other error return codes generic to any component MIB are specified in the MIB(5) reference page. These error codes are guaranteed to be mutually exclusive with any TM_MIB(5) specific error codes defined here.

The following diagnostic codes are returned in TA_ERROR to indicate successful completion of an administrative request. These codes are guaranteed to be non-negative.

[other]
: Other return codes generic to any component MIB are specified in the MIB(5) reference page. These return codes are guaranteed to be mutually exclusive with any TM_MIB(5) specific return codes defined here.

Interoperability

The header files and field tables defined in this reference page are available on BEA TUXEDO system Release 5.0 and later. Fields defined in these headers and tables will not be changed from release to release. New fields may be added which are not defined on the older release site. Access to the /AdminAPI is available from any site with the header files and field tables necessary to build a request.

If sites of differing releases, both greater than or equal to Release 5.0, are interoperating, then information on the older site is available for access and update as defined in the MIB reference page for that release and may be a subset of the information available in the later release.

Portability

The existing FML32 and ATMI functions necessary to support administrative interaction with BEA TUXEDO system MIBs, as well as the header file and field table defined in this reference page, are available on all supported native and workstation platforms.

Examples

This section contains a sequence of code fragments that configure, activate, query, and deactivate a two node application using both `tpadmcall`(3c) and `tpcall`(3c). Variable names are used in places where reasonable values for a local environment are required, for example, `tuxconfig` is a two element array of character pointers with each element identifying the full path name of the `TUXCONFIG` file on that machine.

Field Tables

The field table `tpadm` must be available in the environment to have access to attribute field identifiers. This can be done at the shell level as follows:

```
$ FIELDTBLS=tpadm $ FLDTBLDIR=${TUXDIR}/udataobj
$ export FIELDTBLS FLDTBLDIR
```

Header Files

The following header files are included.

```
#include <atmi.h> #include <fml32.h>
#include <tpadm.h>
```

Initial Configuration

The following code creates and populates an FML32 buffer that is then passed to `tpadmcall`(3c) for processing. This example also shows interpretation of `tpadmcall(3)` return codes. The request shown creates the initial configuration for the application.

```
/* Allocate and initialize the buffer */ ibuf = (FBFR32 *)tpal
loc("FML32", NULL, 4000);
 obuf = (FBFR32 *)tpalloc("FML32", NULL, 4000);
 /* Set MIB(5) attributes defining request type */
 Fchg32(ibuf, TA_OPERATION, 0, "SET", 0);
 Fchg32(ibuf, TA_CLASS, 0, "T_DOMAIN", 0);
 Fchg32(ibuf, TA_STATE, 0, "NEW", 0);
 /* Set TM_MIB(5) attributes to be set in T_DOMAIN class object */
 Fchg32(ibuf, TA_OPTIONS, 0, "LAN,MIGRATE", 0);
 Fchg32(ibuf, TA_IPCKEY, 0, (char *)&ipckey, 0);
 Fchg32(ibuf, TA_MASTER, 0, "LMID1", 0);
 Fchg32(ibuf, TA_MODEL, 0, "MP", 0);
 /* Set TM_MIB(5) attributes for TA_MASTER T_MACHINE class object */
 Fchg32(ibuf, TA_LMID, 0, "LMID1", 0);
 Fchg32(ibuf, TA_PMID, 0, pmid[0], 0);
 Fchg32(ibuf, TA_TUXCONFIG, 0, tuxconfig[0], 0);
 Fchg32(ibuf, TA_TUXDIR, 0, tuxdir[0], 0);
 Fchg32(ibuf, TA_APPDIR, 0, appdir[0], 0);
 Fchg32(ibuf, TA_ENVFILE, 0, envfile[0], 0);
 Fchg32(ibuf, TA_ULOGPFX, 0, ulogpfx[0], 0);
 Fchg32(ibuf, TA_BRIDGE, 0, "/dev/tcp", 0);
 Fchg32(ibuf, TA_NADDR, 0, naddr[0], 0);
 Fchg32(ibuf, TA_NLSADDR, 0, nlsaddr[0], 0);
 /* Perform the action via tpadmcall() */
 if (tpadmcall(ibuf, obuf, 0) 0) {
 fprintf(stderr, "tpadmcall failed: %s\n", tpstrerror(tperrno));
 /* Additional error case processing */
 }
```

Add Second Machine

The following code reuses the buffers allocated in the previous section to build a request buffer. The request shown below adds a second machine to the configuration established earlier.

```
/* Clear the request buffer */ Finit32(ibuf, Fsizeof32(ibuf));
 /* Set MIB(5) attributes defining request type */
 Fchg32(ibuf, TA_OPERATION, 0, "SET", 0);
 Fchg32(ibuf, TA_CLASS, 0, "T_MACHINE", 0);
 Fchg32(ibuf, TA_STATE, 0, "NEW", 0);
 /* Set TM_MIB(5) attributes to be set in T_MACHINE class object */
 Fchg32(ibuf, TA_LMID, 0, "LMID2", 0);
 Fchg32(ibuf, TA_PMID, 0, pmid[1], 0);
 Fchg32(ibuf, TA_TUXCONFIG, 0, tuxconfig[1], 0);
 Fchg32(ibuf, TA_TUXDIR, 0, tuxdir[1], 0);
 Fchg32(ibuf, TA_APPDIR, 0, appdir[1], 0);
 Fchg32(ibuf, TA_ENVFILE, 0, envfile[1], 0);
 Fchg32(ibuf, TA_ULOGPFX, 0, ulogpfx[1], 0);
```

```
 Fchg32(ibuf, TA_BRIDGE, 0, "/dev/tcp", 0);
 Fchg32(ibuf, TA_NADDR, 0, naddr[1], 0);
 Fchg32(ibuf, TA_NLSADDR, 0, nlsaddr[1], 0);

 tpadmcall(...) /* See earlier example for detailed error processing
*/
```

Make Second Machine Backup Master

The existing buffers are again reused to identify the newly configured second machine as the backup master site for this application.

```
/* Clear the request buffer */ Finit32(ibuf, Fsizeof32(ibuf));

 /* Set MIB(5) attributes defining request type */
 Fchg32(ibuf, TA_OPERATION, 0, "SET", 0);
 Fchg32(ibuf, TA_CLASS, 0, "T_DOMAIN", 0);

 /* Set TM_MIB(5) T_DOMAIN attributes changing */
 Fchg32(ibuf, TA_MASTER, 0, "LMID1,LMID2", 0);

 tpadmcall(...); /* See earlier example for detailed error
processing */
```

Add Two Server Groups

Reuse the buffers to generate two requests, each adding one server group to the configured application. Note how the second request simply modifies the necessary fields in the existing input buffer.

```
/* Clear the request buffer */ Finit32(ibuf, Fsizeof32(ibuf));

 /* Set MIB(5) attributes defining request type */
 Fchg32(ibuf, TA_OPERATION, 0, "SET", 0);
 Fchg32(ibuf, TA_CLASS, 0, "T_GROUP", 0);
 Fchg32(ibuf, TA_STATE, 0, "NEW", 0);

 /* Set TM_MIB(5) attributes defining first group */
 Fchg32(ibuf, TA_SRVGRP, 0, "GRP1", 0);
 Fchg32(ibuf, TA_GRPNO, 0, (char *)&grpno[0], 0);
 Fchg32(ibuf, TA_LMID, 0, "LMID1,LMID2", 0);

 tpadmcall(...); /* See earlier example for detailed error
processing */

 /* Set TM_MIB(5) attributes defining second group */
 Fchg32(ibuf, TA_SRVGRP, 0, "GRP2", 0);
```

```
 Fchg32(ibuf, TA_GRPNO, 0, (char *)&grpno[1], 0);
 Fchg32(ibuf, TA_LMID, 0, "LMID2,LMID1", 0);

 tpadmcall(...); /* See earlier example for detailed error
processing */
```

Add One Server Per Group

Reuse the allocated buffers to add one server per group to the configured application.

```
/* Clear the request buffer */ Finit32(ibuf, Fsizeof32(ibuf));

 /* Set MIB(5) attributes defining request type */
 Fchg32(ibuf, TA_OPERATION, 0, "SET", 0);
 Fchg32(ibuf, TA_CLASS, 0, "T_SERVER", 0);
 Fchg32(ibuf, TA_STATE, 0, "NEW", 0);

 /* Set TM_MIB(5) attributes defining first server */
 Fchg32(ibuf, TA_SRVGRP, 0, "GRP1", 0);
 Fchg32(ibuf, TA_SRVID, 0, (char *)&srvid[0], 0);
 Fchg32(ibuf, TA_SERVERNAME, 0, "ECHO", 0);

 tpadmcall(...); /* See earlier example for detailed error
processing */

 /* Set TM_MIB(5) attributes defining second server */
 Fchg32(ibuf, TA_SRVGRP, 0, "GRP2", 0);
 Fchg32(ibuf, TA_SRVID, 0, (char *)&srvid[1], 0);

 tpadmcall(...); /* See earlier example for detailed error
processing */
```

Add Routing Criteria

Add a routing criteria definition. Note that routing criteria may be dynamically added to a running application using a similar operation via the `tpcall`(3c) interface.

```
/* Clear the request buffer */ Finit32(ibuf, Fsizeof32(ibuf));

 /* Set MIB(5) attributes defining request type */
 Fchg32(ibuf, TA_OPERATION, 0, "SET", 0);
 Fchg32(ibuf, TA_CLASS, 0, "T_ROUTING", 0);
 Fchg32(ibuf, TA_STATE, 0, "NEW", 0);

 /* Set TM_MIB(5) attributes defining routing criteria */
 Fchg32(ibuf, TA_ROUTINGNAME, 0, "ECHOROUTE", 0);
 Fchg32(ibuf, TA_BUFTYPE, 0, "FML", 0);
```

```
 Fchg32(ibuf, TA_FIELD, 0, "LONG_DATA", 0);
 Fchg32(ibuf, TA_RANGES, 0, "MIN-100:GRP1,100-MAX:GRP2", 26);

 tpadmcall(...); /* See earlier example for detailed error
processing */
```

Add Service Definition

Define a service object that maps the advertised service name to the routing criteria defined above.

```
/* Clear the request buffer */ Finit32(ibuf, Fsizeof32(ibuf));

 /* Set MIB(5) attributes defining request type */
 Fchg32(ibuf, TA_OPERATION, 0, "SET", 0);
 Fchg32(ibuf, TA_CLASS, 0, "T_SERVICE", 0);
 Fchg32(ibuf, TA_STATE, 0, "NEW", 0);

 /* Set TM_MIB(5) attributes defining service entry */
 Fchg32(ibuf, TA_SERVICENAME, 0, "ECHO", 0);
 Fchg32(ibuf, TA_ROUTINGNAME, 0, "ECHOROUTE", 0);

 tpadmcall(...); /* See earlier example for detailed error
processing */
```

Activate Master Site Admin

Activate the master site administrative processes (DBBL, BBL, BRIDGE) by setting the `T_DOMAIN` class object state to `ACTIVE`.

```
/* Clear the request buffer */ Finit32(ibuf, Fsizeof32(ibuf));

 /* Set MIB(5) attributes defining request type */
 Fchg32(ibuf, TA_OPERATION, 0, "SET", 0);
 Fchg32(ibuf, TA_CLASS, 0, "T_DOMAIN", 0);
 Fchg32(ibuf, TA_STATE, 0, "ACT", 0);

 tpadmcall(...); /* See earlier example for detailed error
processing */
```

Switch to Active Application Administration

Now that the application is active, we need to join the application and make our /AdminAPI requests via the `tpcall`(3c) interface.

```
/* Now that the system is active, join it as the administrator */
tpinfo = (TPINIT *)tpalloc("TPINIT", NULL, TPINITNEED(0));
 sprintf(tpinfo->usrname, "appadmin");
 sprintf(tpinfo->cltname, "tpsysadm");
 if (tpinit(tpinfo) < 0) {
 fprintf(stderr, "tpinit() failed: %s\n", tpstrerror(tperrno));
 /* Additional error case processing */
 }

 /* Reinitialize buffers as typed buffers */
 Finit32(ibuf, Fsizeof32(ibuf));
 Finit32(obuf, Fsizeof32(obuf));
```

Activate Rest of Application

Activate the remaining portions of the application. Note that the administrative user may request unsolicited notification messages be sent just before and just after the attempted boot of each server by setting the `TMIB_NOTIFY` flag in the `TA_FLAGS` attribute of the request. This example shows handling of an error return from `tpcall`(3c).

```
/* Clear the request buffer */ Finit32(ibuf, Fsizeof32(ibuf));

 /* Set MIB(5) attributes defining request type */
 Fchg32(ibuf, TA_OPERATION, 0, "SET", 0);
 Fchg32(ibuf, TA_CLASS, 0, "T_MACHINE", 0);
 Fchg32(ibuf, TA_STATE, 0, "RAC", 0);

 /* Set TM_MIB(5) attributes identifying machine */
 Fchg32(ibuf, TA_LMID, 0, "LMID1", 0);

 /* Invoke the /AdminAPI and interpret results */
 if (tpcall(".TMIB", (char *)ibuf, 0, (char **)&obuf, &olen, 0) <
0) {
 fprintf(stderr, "tpcall failed: %s\n", tpstrerror(tperrno));
 if (tperrno == TPESVCFAIL) {
 Fget32(obuf,TA_ERROR,0,(char *)&ta_error,NULL);
 ta_status = Ffind32(obuf, TA_STATUS, 0, NULL);
 fprintf(stderr, "Failure: %ld, %s\n",
 ta_error, ta_status);
 }
 /* Additional error case processing */
 }
```

Query Server Status

Generate a query on the status of one of the activated servers.

```
/* Clear the request buffer */ Finit32(ibuf, Fsizeof32(ibuf));

 /* Set MIB(5) attributes defining request type */
 Fchg32(ibuf, TA_OPERATION, 0, "GET", 0);
 Fchg32(ibuf, TA_CLASS, 0, "T_SERVER", 0);
 flags = MIB_LOCAL;
 Fchg32(ibuf, TA_FLAGS, 0, (char *)&flags, 0);

 /* Set TM_MIB(5) attributes identifying machine */
 Fchg32(ibuf, TA_SRVGRP, 0, "GRP1", 0);
 Fchg32(ibuf, TA_SRVID, 0, (char *)&srvid[0], 0);

 tpcall(...); /* See earlier example for detailed error processing
*/
```

Deactivate Application

Deactivate the application by setting the state of each machine to `INACTIVE`. Note that the `TMIB_NOTIFY` flag could be used with this operation also.

```
/* Clear the request buffer */ Finit32(ibuf, Fsizeof32(ibuf));

 /* Shutdown Remote Machine First */
 /* Set MIB(5) attributes defining request type */
 Fchg32(ibuf, TA_OPERATION, 0, "SET", 0);
 Fchg32(ibuf, TA_CLASS, 0, "T_MACHINE", 0);
 Fchg32(ibuf, TA_LMID, 0, "LMID2", 0);
 Fchg32(ibuf, TA_STATE, 0, "INA", 0);

 tpcall(....); /* See earlier example for detailed error processing
*/

 /* And now application servers on master machine */
 flags = TMIB_APPONLY;
 Fchg32(ibuf, TA_FLAGS, 0, (char *)&flags, 0);
 Fchg32(ibuf, TA_LMID, 0, "LMID1", 0);

 tpcall(...); /* See earlier example for detailed error processing
*/

 /* Terminate active application access */
 tpterm();

 /* Finally, shutdown the master admin processes */
```

```
 Finit32(ibuf, Fsizeof32(ibuf));
 Fchg32(ibuf, TA_OPERATION, 0, "SET", 0);
 Fchg32(ibuf, TA_CLASS, 0, "T_DOMAIN", 0);
 Fchg32(ibuf, TA_STATE, 0, "INA", 0);

 tpadmcall(...); /* See earlier example for detailed error
processing */
```

Files `${TUXDIR}/include/tpadm.h, ${TUXDIR}/udataobj/tpadm`

See Also `Fintro`(3fml), `Fadd32`(3fml), `Fchg32`(3fml), `Ffind32`(3fml), `tpalloc`(3c), `tprealloc`(3c), `tpcall`(3c), `tpacall`(3c), `tpgetrply`(3c), `tpenqueue`(3c), `tpdequeue`(3c), `MIB`(5), `WS_MIB`(5), *Administering the BEA TUXEDO System*, *BEA TUXEDO Programmer's Guide*

TMQFORWARD(5)

Name BEA TUXEDO system Message Forwarding Server

Synopsis

```
TMQFORWARD SRVGRP="identifier" SRVID="number" REPLYQ=N CLOPT=" [ -A
] [ servopts options ] -- -q queuename[,queuename...] [ -t trantime
] [ -i idletime ] [ -e ] [ -d ] [ -n ] [ -f delay ] "
```

Description The message forwarding server is a BEA TUXEDO system-supplied server that forwards messages that have been stored using `tpenqueue`(3c) for later processing. The application administrator enables automated message processing for the application servers by specifying this server as an application server in the `SERVERS` section.

The location, server group, server identifier and other generic server related parameters are associated with the server using the already defined configuration file mechanisms for servers. The following is a list of additional command line options that are available for customization.

`-q` *queuename[,queuename...]*
: is used to specify the names of one or more queues/services for which this server forwards messages. Queue and service names are strings limited to 15 characters. This option is required.

`-t` *trantime*
: is used to indicate the transaction timeout value used on `tpbegin`(3c) for transactions that dequeue messages and forward them to application servers. If not specified, the default is 60 seconds.

`-i` *idletime*
: is used to indicate the time that the server is idle after draining the queue(s) that it is reading. A value of zero indicates that the server will continually read the queue(s), which can be inefficient if the queues do not continually have messages. If not specified, the default is 30 seconds.

`-e`
: is used to cause the server to exit if it finds no messages on the queue(s). This, combined with the threshold command associated with the queue(s), can be used to start and stop the `TMQFORWARD` server in response to fluctuations of messages that are enqueued.

`-d`

is used to cause messages that result in service failure and have a reply message (non-zero in length) to be deleted from the queue after the transaction is rolled back.

`-n`

is used to cause messages to be sent using the `TPNOTRAN` flag. This flag allows for forwarding to server groups that are not associated with a resource manager.

`-f` *delay*

is used to case the server to forward the message to the service instead of using `tpcall`. The message is sent such that a reply is not expected from the service. The `TMQFORWARD` server does not block waiting for the reply from the service and can continue processing the next message from the queue. To throttle the system such that `TMQFORWARD` does not flood the system with requests, the *delay* numeric value can be used to indicate a delay, in seconds, between processing requests; use zero for no delay.

Messages are sent to a server providing a service whose name matches the queue name from which the message is read. The message priority is the priority specified when the message was enqueued, if that was set. Otherwise, the priority is the priority for the service, as defined in the configuration file, or the default (50).

Messages are dequeued and sent to the server within a transaction. If the service succeeds, the transaction is committed and the message is deleted from the queue. If the message is associated with a reply queue, then any reply from the service is enqueued to the reply queue, along with the returned `tpurcode`. If the reply queue does not exist, the reply is dropped.

If the service fails, then the transaction is rolled back and the message is put back on the queue, up to the number of times specified by the retry limit configured for the queue. When a message is put back on the queue, the rules for ordering and dequeueing that applied when it was first put on the queue are (in effect) suspended for *delay* seconds; this opens up the possibility that a message of a lower priority may be dequeued ahead of the restored message.

If the `-d` option is specified, the message is deleted from the queue if the service fails and a reply message is received from the server, and the reply message (and associated `tpurcode`) are enqueued to the failure queue, if one is associated with the message and the queue exists.

Any configuration condition that prevents TMQFORWARD from dequeuing or forwarding messages will cause the server to fail to boot. These conditions include the following:

- The SRVGRP must have TMSNAME set to TMS_QM.
- OPENINFO must be set to indicate the associated device and queue name.
- The SERVER entry must not be part of an MSSQ set.
- REPLYQ must be set to N.
- The -q option must be specified in the command line options.
- The server must not advertise any services (that is, the -s option must not be specified).

Handling Application Buffer Types

As delivered, TMQFORWARD handles the standard buffer types provided with the BEA TUXEDO system. If additional application buffer types are needed, then a customized version of TMQFORWARD needs to be built using buildserver(1) with a customized type switch.

If your application uses shared libraries, it is not necessary to go through the compile and link process described in the previous paragraph. See the description in the Chapter "Buffer Types" of the *BEA TUXEDO Administrator's Guide.*

The files included by the caller should include only the application buffer type switch and any required supporting routines. buildserver is used to combine the server object file, $TUXDIR/lib/TMQFORWARD.o, with the application type switch file(s), and link it with the needed BEA TUXEDO system libraries. The following example provides a sample for further discussion.

```
buildserver -v -o TMQFORWARD -r TUXEDO/QM -f
${TUXDIR}/lib/TMQFORWARD.o -f apptypsw.o
```

The buildserver options are as follows:

-v
: specifies that buildserver should work in verbose mode. In particular, it writes the cc command to its standard output.

-o *name*
: specifies the file name of the output load module. The name specified here must also be specified in the SERVERS section of the configuration file. It is recommended that the name TMQFORWARD be used for consistency. The application specific version of the command can be installed in $APPDIR it is booted instead of the version in $TUXDIR/bin.

`-r TUXEDO/QM`
: specifies the resource manager associated with this server. The value `TUXEDO/QM` appears in the resource manager table located in `$TUXDIR/udataobj/RM` and includes the library for the BEA TUXEDO system queue manager.

`-f $TUXDIR/lib/TMQFORWARD.o`
: specifies the object file that contains the `TMQFORWARD` service and should be specified as the first argument to the `-f` option.

`-f` *firstfiles*
: specifies one or more user files to be included in the compilation and/or link edit phases of `buildserver`. Source files are compiled using the either the `cc` command or the compilation command specified through the CC environment variable. These files must be specified after including the `TMQFORWARD`.o object file. If more than one file is specified, file names must be separated by white space and the entire list must be enclosed in quotation marks. This option can be specified multiple times.

The `-s` option must not be specified to advertise services.

Portability `TMQFORWARD` is supported as a BEA TUXEDO system-supplied server on UNIX operating systems.

Interoperability `TMQFORWARD` may be run in an interoperating application, but it must run on a Release 4.2 or later node.

Examples

```
*GROUPS # For NT/Netware, :myqueue becomes ;myqueue
TMQUEUEGRP LMID=lmid GRPNO=1 TMSNAME=TMS_QM
   OPENINFO="TUXEDO/QM:/dev/device:myqueue"
# no CLOSEINFO is required

*SERVERS # recommended values RESTART=Y GRACE=0
TMQFORWARD SRVGRP="TMQUEUEGRP" SRVID=1001 RESTART=Y GRACE=0
   CLOPT=" -- -qservice1,service2" REPLYQ=N
TMQUEUE SRVGRP="TMQUEUEGRP" SRVID=1000 RESTART=Y GRACE=0
   CLOPT="-s ACCOUNTING:TMQUEUE"
```

See Also `servopts`(5), `buildserver`(1), `tpenqueue`(3c), `tpdequeue`(3c), `ubbconfig`(5), `TMQUEUE`(5), `TMQFORWARD`(5), *Administering the BEA TUXEDO System*, *BEA TUXEDO Programmer's Guide*

TMQUEUE(5)

Name TMQUEUE-BEA TUXEDO system Message Queue Manager

Synopsis

```
TMQUEUE
SRVGRP="identifier"
SRVID="number" CLOPT=" [ -A ][servopts options ] -- [-t trantime]"
```

Description The message queue manager is a BEA TUXEDO system-supplied server that enqueues and dequeues messages on behalf of programs calling tpenqueue(3) and tpdequeue(3), respectively. The application administrator enables message enqueuing and dequeuing for the application by specifying this server as an application server in the SERVERS section.

The location, server group, server identifier and other generic server related parameters are associated with the server using the already defined configuration file mechanisms for servers. The following is a list of additional command line options that are available for customization.

`-t` *trantime*
: is used to indicate the transaction timeout value used on tpbegin(3) for enqueue and dequeue requests not in transaction mode (e.g., tpenqueue(3) or tpdequeue(3) are called when the caller is not in transaction mode or with the TPNOTRAN flag). This value also has an impact on dequeue requests with the TPQWAIT option since the transaction will timeout and an error will be sent back to the requester based on this value. If not specified, the default is 30 seconds.

A TMQUEUE server is booted as part of an application to facilitate application access to its associated queue space; a queue space is a collection of queues.

Any configuration condition that prevents the TMQUEUE from enqueuing or dequeuing messages will cause the TMQUEUE to fail at boot time. The SRVGRP must have TMSNAME set to TMS_QM, and must have OPENINFO set to indicate the associated device and queue space name.

Queue Name for Message Submission The tpenqueue() and tpdequeue() functions take a queue space name as their first argument. This name must be the name of a service advertised by TMQUEUE. By default, TMQUEUE only offers the service "TMQUEUE". While this may be sufficient for applications with only a single queue space, applications with multiple queue spaces may need to have different queue space names. Additionally, applications may wish to provide more descriptive service names that match the queue space names.

Advertising additional service names can be done using the standard server command line option, -s, as shown below in EXAMPLES. An alternative is to hard-code the service when generating a custom TMQUEUE program, as discussed in the following section.

While these methods (the server command line option or a customized server) may be used for static routing of messages to a queue space, dynamic routing may be accomplished using data dependent routing. In this case, each TMQUEUE server would advertise the same service name(s) but a ROUTING field in the configuration file would be used to specify routing criteria based on the application data in the queued message. The routing function returns a GROUP based on the service name and application typed buffer data, which is used to direct the message to the service at the specified group (note that there can be only one queue space per GROUP, based on the OPENINFO string).

Handling Application Buffer Types

As delivered, TMQUEUE handles the standard buffer types provided with BEA TUXEDO system. If additional application buffer types are needed, then a customized version of TMQUEUE needs to be built using buildserver(1).

If your application uses shared libraries, it is not necessary to go through this compile and link process. See the description in the *Buffers* chapter of the *BEA TUXEDO Administrator's Guide*.

The customization described in buildserver can also be used to hard-code service names for the server.

The files included by the caller should include only the application buffer type switch and any required supporting routines. buildserver is used to combine the server object file, $TUXDIR/lib/TMQUEUE.o, with the application type switch file(s), and link it with the needed BEA TUXEDO system libraries. The following example provides a sample for further discussion.

```
buildserver -v -o TMQUEUE -s qspacename:TMQUEUE -r TUXEDO/QM \e -f
${TUXDIR}/lib/TMQUEUE.o -f apptypsw.o
```

The buildserver options are as follows:

-v
: specifies that buildserver should work in verbose mode. In particular, it writes the cc command to its standard output.

-o *name*
: specifies the file name of the output load module. The name specified here must also be specified in the SERVERS section of the configuration file. It is recommended that TMQUEUE be used for consistency.

`-s` *qspacename,qspacename* `:TMQUEUE`
: specifies the names of services that can be advertised when the server is booted (see `servopts`(5)). For this server, they will be used as the aliases for the queue space name to which requests may be submitted. Spaces are not allowed between commas. The function name, `TMQUEUE`, is preceded by a colon. The `-s` option may appear several times.

`-r TUXEDO/QM`
: specifies the resource manager associated with this server. The value `TUXEDO/QM` appears in the resource manager table located in `$TUXDIR/udataobj/RM` and includes the library for the BEA TUXEDO system queue manager.

`-f $TUXDIR/lib/TMQUEUE.o`
: specifies the object file that contains the `TMQUEUE` service and should be specified as the first argument to the `-f` option.

`-f` *firstfiles*
: specifies one or more user files to be included in the compilation and/or link edit phases of `buildserver`. Source files are compiled using the either the `cc` command or the compilation command specified through the CC environment variable. These files must be specified after including the `TMQUEUE.o` object file. If more than one file is specified, file names must be separated by white space and the entire list must be enclosed in quotation marks. This option can be specified multiple times.

Portability `TMQUEUE` is supported as a BEA TUXEDO system-supplied server on UNIX operating systems.

Interoperability `TMQUEUE` may be run in an interoperating application, but it must run on a Release 4.2 or later node.

Examples

```
*GROUPS
# For NT/Netware, :myqueue becomes ;myqueue
TMQUEUEGRP1 GRPNO=1 TMSNAME=TMS_QM
   OPENINFO="TUXEDO/QM:/dev/device1:myqueue"
# For NT/Netware, :myqueue becomes ;myqueue
TMQUEUEGRP2 GRPNO=2 TMSNAME=TMS_QM
   OPENINFO="TUXEDO/QM:/dev/device2:myqueue"

*SERVERS
# The queue space name, myqueue, is aliased as ACCOUNTING in this example
TMQUEUE SRVGRP="TMQUEUEGRP1" SRVID=1000 RESTART=Y GRACE=0
```

```
    CLOPT="-s ACCOUNTING:TMQUEUE"
TMQUEUE SRVGRP="TMQUEUEGRP2" SRVID=1000 RESTART=Y GRACE=0
    CLOPT="-s ACCOUNTING:TMQUEUE"
TMQFORWARD SRVGRP="TMQUEUEGRP1" SRVID=1001 RESTART=Y GRACE=0 REPLYQ=N
    CLOPT=" -- -qservice1"
TMQFORWARD SRVGRP="TMQUEUEGRP2" SRVID=1001 RESTART=Y GRACE=0 REPLYQ=N
    CLOPT=" -- -qservice1"
*SERVICES
ACCOUNTING ROUTING="MYROUTING"
*ROUTING
MYROUTING FIELD=ACCOUNT BUFTYPE="FML"
RANGES="MIN - 60000:TMQUEUEGRP1,60001-MAX:TMQUEUEGRP2"
```

In this example, two queues spaces are available. Both `TMQUEUE` servers offer the same services and routing is done via the `ACCOUNT` field in the application typed buffer.

See Also `ubbconfig`(5), `servopts`(5), `buildserver`(1), `tpenqueue`(3), `tpdequeue`(3), `TMQFORWARD`(5), *Administering the BEA TUXEDO System*, *BEA TUXEDO Programmer's Guide*

TMSYSEVT(5)

Name TMSYSEVT-system event reporting process

Synopsis

```
TMSYSEVT SRVGRP="identifier" SRVID="number"
  [CLOPT="[-A] [servopts options]
  [-- [-S] [-p poll-seconds] [-f control-file]]"]
```

Description TMSYSEVT is a BEA TUXEDO system provided server that processes event reports related to system failure or potential failure conditions. The event reports are filtered, and may trigger one or more notification actions.

Filtering and notification rules are stored in *control-file*, which defaults to ${APPDIR}/tmsysevt.dat. Control file syntax is defined in EVENT_MIB(5); specifically, the attributes of the classes in EVENT_MIB can be set to activate subscriptions under the full range of notification rules.

It is possible to boot one or more secondary TMSYSEVT processes for increased availability. Additional servers must be booted with the -S command line option, which indicates a "secondary" server.

When the EVENT_MIB(5) configuration is updated, the primary TMSYSEVT server writes to its control file. Secondary servers poll the primary server for changes and update their local control file if necessary. The polling interval is controlled by the -p option, and is 30 seconds by default.

Interoperability TMSYSEVT must run on a Release 6.0 or later machine.

Notices To migrate the primary TMSYSEVT server to another machine, the system administrator must provide a current copy of *control-file*. Each secondary TMSYSEVT server automatically maintains a recent copy.

TMSYSEVT needs access to the system's FML32 field table definitions for system events. FLDTBLDIR32 should include $TUXDIR/udataobj, and FIELDTBLS32 should include evt_mib. These environment variables may be set in the machine's or server's environment file.

Example

```
*SERVERS
TMSYSEVT SRVGRP=ADMIN1 SRVID=100 RESTART=Y GRACE=900 MAXGEN=5
  CLOPT="-A --"
TMSYSEVT SRVGRP=ADMIN2 SRVID=100 RESTART=Y GRACE=900 MAXGEN=5
  CLOPT="-A -- -S -p 90"
```

See Also tpsubscribe(3), EVENTS(5), EVENT_MIB(5), TMUSREVT(5)

tmtrace(5)

Name

`tmtrace`-BEA TUXEDO run-time tracing facility

Description

The run-time tracing facility allows application administrators and developers to trace the execution of a BEA TUXEDO application.

Run-time tracing is based on the notion of a *trace point*, which marks an interesting condition or transition during the execution of an application. Examples of trace points are the entry to an ATMI function such as `tpcall`, the arrival of a BEA TUXEDO message, or the start of a transaction.

When a trace point is reached, the following things happen. First, a *filter* is applied to determine if the trace point is of interest. If so, a *trace record* is emitted to a *receiver*, which is a file or (in the future) a buffer. Finally, an *action* is triggered, such as aborting the process. Both the emission to a receiver and the trigger are optional, and neither takes place if the trace point does not pass the filter.

The filter, receiver, and trigger are specified in the *trace specification*, whose syntax is described below. The trace specification is initialized from the `TMTRACE` environment variable. The trace specification of a running process may be changed either as a trigger action or by using the `changetrace` command of `tmadmin`(1).

Trace points are classified into *trace categories*, enumerated below. Each trace point belongs to a single category. The filter describes the trace categories of interest, and minimal processing occurs for trace points that do not pass the filter.

Run-time tracing also provides the capability to *dye* the messages sent by a client to a server, and transitively by that server to other servers. If a process chooses to dye its messages, the dye is automatically passed by the originating process to all processes that directly or indirectly receive messages from the originating process. When a process receives a dyed message, it automatically turns on the `atmi` trace category and starts emitting trace records to the userlog, if this was not being done already.

Dyeing can be explicitly turned on or off by the `dye` and `undye` triggers in the trace specification. Dyeing is also implicitly turned on when a dyed message is received, and implicitly turned off by `tpreturn` and `tpforward`. When it is implicitly turned off, the tracing specification in effect when dyeing was turned on is restored.

Trace Categories

The trace categories are

`atmi`

trace points for explicit application calls to the ATMI and TX interfaces, that is, calls to the `tp` and `tx_` functions, and the invocation of application services

There are a few exceptions. Implicit calls are printed in this category where some TX interfaces directly call ATMI interfaces, for the implicit call to `tpinit` when an ATMI call is done with first calling `tpinit`, and for cases where `tpreturn` is called on error (to aid in debugging).

`iatmi`

trace points for `I`mplicit calls to the ATMI and TX interface. These trace points indicate all internal calls made while processing application requests and for administration. Setting this level implies the `atmi` level, that is, every call to an ATMI or TX interface is traced (both explicit and implicit).

`xa`

trace points for every call to the XA interface (the interface between the Transaction Manager and a Resource Manager, e.g., a database).

`trace`

trace points related to the tracing feature itself, including message dyeing

Trace Specification

The trace specification is a string with the syntax *filter-spec* `:` *receiver-spec* [`:` *trigger-spec*] where *filter-spec* describes the trace categories to be examined or ignored, *receiver-spec* is the receiver of trace records, and the optional *trigger-spec* describes the action to be performed.

The null string is also a legal trace specification. It is the default for all BEA TUXEDO processes if no other specification is supplied.

The strings `on` and `off` are also accepted: `on` is an alias for `atmi:ulog:dye`, and `off` is equivalent to `::undye`.

Filter Specification

The filter specification, which is the first component of the trace specification, has the syntax

[{ + | - }] [*category*] ...

where *category* is one of the categories listed above. The symbol * can be used in place of *category* to denote all categories. The prefix + or - specifies that the following category is to be added or subtracted from the set of categories currently in effect. If no category follows a + or -, then the categories currently in effect are not modified.

An empty filter means that no categories are to be selected, which effectively disables tracing.

When a trace point occurs, its category is compared with the filter specification. If the category is included, then the trace point is processed further -- according to the receiver and trigger specifications. If the category is not included, no further processing of the trace point occurs.

Receiver Specification

A receiver is the entity to which a trace record is sent. There is at most one receiver of each trace record.

The receiver specification, which is the second component of the trace specification, has the syntax

[/ *regular-expression* /] *receiver*

where the optional regular expression may be used to select a subset of the trace points that pass the filter. The regular expression is matched with the trace record. An empty receiver specification is also legal, in which case no trace records are emitted.

Currently, the only legal value for *receiver* is

`ulog`
: emit the trace record to the userlog

Trigger Specification

A trigger is an optional action performed after a trace record is emitted. At most one action is executed for each trace record that passes the filter.

The trigger specification, which is the optional third part of the trace specification, has the syntax

[/ *regular-expression* /] *action*

where the optional regular expression may be used to restrict the trigger so that it is executed only for a subset of the trace points that pass the filter. The regular expression is matched with the trace record.

The available actions are

`abort`
: terminate the process by calling abort().

ulog(*message*)
: write the *message* to the userlog.

system(*command*)
: execute the *command* using system(3) (this is not supported for Windows, OS/2, or MAC Workstation clients); occurrences of %A are expanded to the value of trace record.

trace(*trace-spec*)
: reset the trace specification to the supplied *trace-spec*.

`dye`
: turn on message dyeing.

`undye`
: turn off message dyeing.

sleep(*seconds*)
: sleep the specified number of seconds (this is not supported for DOS, Windows, or MAC Workstation clients).

Trace Records

A trace record is a string with the format

cc`:`*data*

where *cc* is the first two characters of the trace category and *data* contains additional information about the trace point.

When a trace record appears in the userlog, the line looks like this:

hhmmss.system-name`!`*process-name.pid*`: TRACE:`*cc*`:`*data*

Notices

Match patterns cannot be specified for the receiver and trigger for Workstation clients running on MAC platforms; the regular expressions will be ignored.

The `tmadmin changetrace` command cannot be used to affect the tracing level for Workstation clients.

Examples

To trace a client, as well as to trace all ATMI calls made by an application server on behalf of that client, set and export `TMTRACE=on` in the environment of the client. This specification will cause all explicit ATMI trace points in the client to be logged and message dyeing to be turned on. Any application server process that performs a service on behalf of the client will automatically log all explicit ATMI trace points.

To see all client trace points, both explicit and implicit, for the previous example, set and export

```
TMTRACE="*:ulog:dye:"
```

To trace service requests from a client as in the previous example, but restrict the tracing output from the client to the bare minimum of information about `tpcall` requests, set and export

```
TMTRACE=atmi:/tpacall/ulog:dye
```

in the environment of the client. This specification will cause all `tpacall` invocations in the client to be logged and message dyeing to be turned on. Any application server process that performs a service on behalf of the client will automatically log all ATMI trace points. The client's identifier, which is included in the `tpacall` trace record, can be correlated with the value of the `TPSVCINFO` parameter passed to any service routine invoked on the client's behalf.

To trace the invocations of all service requests performed by application servers, set

```
TMTRACE=atmi:/tpservice/ulog
```

in the server *ENVFILE*s on all participating machines.

To enable run-time tracing of all trace categories throughout an application, with message dyeing turned on, set and export

```
TMTRACE=*:ulog:dye
```

in the environment of all clients and in the machine *ENVFILE*s on all participating machines. This setting will probably produce an unmanageable amount of output because all processes, including the `BBL` and `DBBL`, will emit trace records.

To turn on ATMI tracing in all running servers in group `GROUP1` *after* they are booted, invoke the `changetrace` command of `tmadmin` as follows:

```
changetrace -g GROUP1 on
```

Note that `changetrace` affects only currently-existing processes; it does not change the trace configuration of servers in group `GROUP1` that have not yet been booted. (To set the default trace configuration of a server, set `TMTRACE` in its *ENVFILE*.)

To turn off tracing in all currently-running application processes, use `changetrace` as follows:

```
changetrace -m all off
```

To cause the running server process whose identifier is 1 in group `GROUP1` to abort when it executes `tpreturn`, specify the following to `tmadmin`:

```
changetrace -i 1 -g GROUP1 "atmi::/tpreturn/abort"
```

See Also `tmadmin`(1), `userlog`(3)

TMUSREVT(5)

Name TMUSREVT-user event reporting process

Synopsis

```
TMUSREVT SRVGRP="identifier" SRVID="number"
  [CLOPT="[-A] [servopts options]
  [-- [-S] [-p poll-seconds] [-f control-file]]"]
```

Description TMUSREVT is a BEA TUXEDO system provided server that processes event report message buffers from tppost(3), and acts as an Event Broker to filter and distribute them.

Filtering and notification rules are stored in *control-file*, which defaults to ${APPDIR}/tmusrevt.dat. Control file syntax is defined in EVENT_MIB(5); specifically, the attributes of the classes in EVENT_MIB can be set to activate subscriptions under the full range of notification rules.

It is possible to boot one or more secondary TMUSREVT processes for increased availability. Additional servers must be booted with the -S command line option, which indicates a "secondary" server.

When the EVENT_MIB(5) configuration is updated, the primary TMUSREVT server writes to its control file. Secondary servers poll the primary server for changes and update their local control file if necessary. The polling interval is controlled by the -p option, and is 30 seconds by default.

Interoperability TMUSREVT must run on a Release 6.0 or later machine.

Notices To migrate the primary TMUSREVT server to another machine, the system administrator must provide a current copy of *control-file*. Each secondary TMUSREVT server automatically maintains a recent copy.

If tppost(3) will be called in transaction mode, all TMUSREVT server groups must have transactional capability (a TMS process).

The TMUSREVT server's environment variables must be set so that FML field tables and VIEW files needed for message filtering and formatting are available. They could be set in the machine's or server's environment file.

Example

```
*SERVERS
TMUSREVT SRVGRP=ADMIN1 SRVID=100 RESTART=Y MAXGEN=5 GRACE=3600
 CLOPT="-A --"
TMUSREVT SRVGRP=ADMIN2 SRVID=100 RESTART=Y MAXGEN=5 GRACE=3600
 CLOPT="-A -- -S -p 120"
```

See Also tppost(3), tpsubscribe(3), EVENTS(5), EVENT_MIB(5), TMSYSEVT(5)

tperrno(5)

Name

BEA TUXEDO system error codes

Synopsis

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

Description

The numerical value represented by the symbolic name of an error condition is assigned to `tperrno` for errors that occur when executing a BEA TUXEDO system library routine.

The name `tperrno` expands to a modifiable *lvalue* that has type `int`, the value of which is set to a positive error number by several BEA TUXEDO system library routines. `tperrno` need not be the identifier of an object; it might expand to a modifiable *lvalue* resulting from a function call. It is unspecified whether `tperrno` is a macro or an identifier declared with external linkage. If a `tperrno` macro definition is suppressed to access an actual object, or if a program defines an identifier with the name `tperrno`, the behavior is undefined.

The reference pages for BEA TUXEDO system library routines list possible error conditions for each routine and the meaning of the error in that context. The order in which possible errors are listed is not significant and does not imply precedence. The value of `tperrno` should be checked only after an error has been indicated; that is, when the return value of the component indicates an error and the component definition specifies that `tperrno` is set on error. An application that checks the value of `tperrno` must include the `<atmi.h>` header file.

The following list describes the general meaning of each error:

`TPEABORT`
: A transaction could not commit because either the work performed by the initiator or by one or more of its participants could not commit.

`TPEBADDESC`
: A call descriptor is invalid or is not the descriptor with which a conversational service was invoked.

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

`TPEINVAL`
: An invalid argument was detected.

TPELIMIT
: The caller's request was not sent because the maximum number of outstanding requests or connections has been reached.

TPENOENT
: Can not send to *svc* because it does not exist or is not the correct type of service.

TPEOS
: An operating system error has occurred.

TPEPERM
: A client cannot join an application because it does not have permission to do so or because it has not supplied the correct application password.

TPEPROTO
: A library routine was called in an improper context.

TPESVCERR
: A service routine encountered an error either in `tpreturn`() or `tpforward`() (for example, bad arguments were passed).

TPESVCFAIL
: The service routine sending the caller's reply called `tpreturn`() with `TPFAIL`. This is an application-level failure.

TPESYSTEM
: A BEA TUXEDO system error has occurred.

TPETIME
: A timeout occurred.

TPETRAN
: The caller cannot be placed in transaction mode.

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

TPERMERR
: A resource manager failed to open or close correctly.

TPEITYPE
: The type and sub-type of the input buffer is not one of the types and sub-types that the service accepts.

`TPEOTYPE`
: The type and sub-type of the reply are not known to the caller.

`TPEHAZARD`
: Due to some failure, the work done on behalf of the transaction can have been heuristically completed.

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

`TPEEVENT`
: An event occurred; the event type is returned in *revent*.

`TPEMATCH`
: *svcname* is already advertised for the server but with a function other than *func*.

Usage Some routines do not have an error return value. Because no routine sets `tperrno` to zero, an application can set `tperrno` to zero, call a routine and then check `tperrno` again to see if an error has occurred.

See Also See the `ERRORS` section of the individual BEA TUXEDO library routines for a more detailed description of the meaning of the error codes returned by each routine.

tpurcode(5)

Name BEA TUXEDO system global variable for an application-specified return code

Synopsis

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

Description `tpurcode` is a global variable defined in `atmi.h`. Its value is the same long integer used as the value of the `rcode` argument of `tpreturn`(3c). `tpurcode` may be used by the application to return additional information to the process that calls an application service. For details, see `tpreturn`(3c).

Assigning meanings to values in `tpurcode` is the responsibility of the application.

Examples Following are examples showing the use of `tpurcode`:

If you return the value `myval` through `rcode` in an application service:

```
.
.
.
tpreturn(TPSUCCESS, myval, rqst->data, 0L, 0);
.
.
.
```

Then the code in the client module might be as follows:

```
.
.
.
ret = tpcall("TOUPPER", (char *)sendbuf, 0, (char **)&rcvbuf, \
&rcvlen, (long)0);
.
.
.
(void) fprintf(stdout, "Returned string is: %s\n", rcvbuf);
(void) fprintf(stdout, "Returned tpurcode is: %d\n", tpurcode);
```

If we call the sample client, `simpcl`, with the value of "My String," the output will look like this:

```
%simpcl "My String"
Returned string is: MY STRING
Returned tpurcode is: myval
```

The significance of `myval` must be defined by the application.

See Also `tpreturn`(3c)

tuxenv(5)

Name A list of environment variables in the BEA TUXEDO system

Description In order to compile application clients and servers, and run the BEA TUXEDO system, it is important that the proper environment variables be set and exported. This reference page provides a list of the most frequently used variables.

The environment variables are grouped in the following sections:

- Operating System Variables
- Key BEA TUXEDO system Variables
- Variables for Field Table Files, View Files and MIO
- File System and `TLOG` Variables
- Workstation Variables
- System/Q Variables
- COBOL Variables
- DEBUG Variables
- Additional Miscellaneous Variables

Operating System Variables

`CC`
: standard C compiler for use by `buildserver` and other BEA TUXEDO commands.

`CFLAGS`
: contains flags to be used by the C compiler.

`EDITOR`
: specifies the editor to be invoked by the BEA TUXEDO system.

`LANG`
: used to set the locale for language specification. See `nl_types`(5).

`LOGNAME`
: specifies the user name for use in error messages.

`LD_LIBRARY_PATH`
: must be set to the pathname for run-time shared libraries.

`NLSPATH`
: specifies the pathname for the message catalog. If not specified, a default path is used. See `nl_paths`(5).

`PAGER`
: specifies the paging command used for paging output in `qmadmin`(1), `tmadmin`(1). This overrides the system default (pg(1) on UNIX operating systems).

`PATH`
: contains pathnames to be searched for executables.

`SHELL`
: the shell program to be invoked by the BEA TUXEDO system.

`TERM`
: specifies terminal type, if a terminal is used.

`TMPDIR`
: the pathname of the directory where all temporary files should be written.

`TZ`
: on systems where the ANSI C `mktime` functions does not exist, `TZ` must be set to use the BEA TUXEDO `gp_mktime`(3) function.

More information on these variables is available in the UNIX System reference page `environ`(5).

Key BEA TUXEDO System Variables

In general, the following environment variables should be set and exported:

`APPDIR`
: full pathname of the base directory for application files.

`APP_PW`
: may be used to specify a password for system clients that prompt for an application password (when security is on). Setting the password in a variable allows the password to be provided from a script, rather than demanding manual entry.

`ENVFILE`
: this variable is used by `tmloadcf`(1). It customarily contains setting for other BEA TUXEDO system environment variable, which are set automatically by the system.

`TLOGDEVICE`
: the pathname for the transaction log. This should be the same as the `TLOGDEVICE` specified in the configuration file for the application.

`TUXCONFIG`
: the pathname of the binary configuration file to be loaded by `tmloadcf`(1).

`ULOGPFX`
: prefix of the filename of the central event log; default, `ULOG`.

`TUXDIR`
: specifies the base directory where the BEA TUXEDO system software is installed.

More information about these variables can be found in the *BEA TUXEDO Programmer's Guide* and the *BEA TUXEDO Administrator's Guide.*

Variables for Field Table Files, View Files and MIO

The following environment variables are used by FML and `VIEWS` and by `mio`(1) clients:

`FIELDTBLS`
: comma-separated list of field table files.

`VIEWFILES`
: comma-separated list of binary view files.

`FLDTBLDIR`
: colon-separated list of directories to search for `FIELDTBLS` files.

`VIEWDIR`
: colon-separated list of directories to search for `VIEWFILES` files.

`MASKPATH`
: list of directories for `mio`(1) to search for binary masks.

`MSKIPCKEY`
: the IPC key for shared memory for a mask cache.

`NGXACTS`
: a comma-separated list of masks the current user is not permitted to use.

`OKXACTS`
: a comma-separated list of masks the current user is permitted to use.

More information about these variables can be found in the *BEA TUXEDO Administrator's Guide*, the *BEA TUXEDO Programmer's Guide*, and the *BEA TUXEDO FML Programmer's Guide*.

File System and TLOG Variables

The following variables are used by the BEA TUXEDO system file system and the transaction log.

`FSCONFIG`
: the pathname for the Universal Device List.

`FSMAXCOMMIT`
: sets the maximum size of the commit buffer.

`FSMAXUPDATE`
: sets the size of the update list and the maximum number of updates.

`FSMSGREP`
: sets the message repetition interval.

`FSOFFSET`
: specifies an offset into the Universal Device List.

Workstation Variables

The following variables are used on Workstation client machines:

`WSBUFFERS`
: the number of packets per application.

`WSDEVICE`
: the network device to be used for network access. For workstation clients in Release 6.4 and higher, this variable is no longer required.

`WSENVFILE`
: pathname of a file containing Workstation client environment variables.

`WSNADDR`
: the network address of the native site network listener.

`WSRPLYMAX`
: the maximum message size before a message is dumped to a file for transfer.

`WSTYPE`
: the machine type of the workstation machine.

More information on these variable can be found in the *BEA TUXEDO Workstation Guide*.

System/Q Variables

The following environment variable is used by BEA TUXEDO System/Q:

`QMCONFIG`
: sets the device where queue space is available to System/Q.

There is more information on this in the *BEA TUXEDO /Q Guide*.

COBOL Variables

The following environment variables are used with COBOL:

`ALTCC`
: specifies the compiler for use with COBOL compilations.

`ALTCFLAGS`
: flags to be passed to the COBOL compiler.

`COBCPY`
: directories to be searched for COBOL Copy files.

`COBDIR`
: specifies the directory where COBOL compiler software is located.

`COBOPT`
: contains command line arguments for the COBOL compiler.

There is more information on these variables in the *BEA TUXEDO COBOL Guide*.

Additonal Miscellaneous Variables

The following additional environment variables may be of use:

`MHSCACHE`
: specifies the number of message catalog handles to keep open (BEA TUXEDO system messages only). The default is 3.

`PMID`
: in MP mode, can be used to specify the physical machine id.

`TAGENTLOG`
: used to set the pathname for the `tlisten`(1) log.

`TMCMPLIMIT`
: used to specify whether compression should be used on messages and to set thresholds for both local and remote messages. The syntax of the variable is:

```
TMCMPLIMIT=[remote_threshold[,local_threshold]]
```

A threshold is a number in the range 0 to MAXLONG. It sets the minimum byte size of a message on which data compression will be performed.

TMCMPPRFM
: This variable sets the compression level for any process that picks it up. Valid values are the integers 1 through 9; 1 results in womewhat less compression that the higher levels, but takes place faster. An informational ULOG message is written when a process reads TMCMPPRFM.

TMNETLOAD
: used to establish load balancing over a network. The value is an arbitrary number of units to be added to the load factor of remote services. Use of this variable tends to force the use of a local service.

UIMMEDSIGS
: to override deferal of signals, set this variable to "Y".

USPOOLDIR
: names a directory to be used by the FRMPRT(5) server for temporary files. It defaults to /tmp.

See Also buildclient(1), buildserver(1), viewc(1) cc(1), environ(5) in a UNIX System reference manual

tuxtypes(5)

Name buffer type switch-buffer types provided by BEA TUXEDO system

Synopsis Default Buffer Type Switch

```
/*
 * The following definitions are specified in
 * $TUXDIR/lib/tmtypesw.c
 */

#include "tmtypes.h"

/*
 * Initialization of the buffer type switch.
 */

struct tmtype_sw_t tm_typesw[] = {
{
"CARRAY", "", 0
},
{
"STRING", "", 512, NULL, NULL, NULL,
_strpresend, NULL, NULL, _strencdec, NULL, _sfilter, _sformat
},
{
"FML", "", 1024, _finit, _freinit, _funinit, _fpresend,
_fpostsend, _fpostrecv, _fencdec, _froute, _ffilter, _fformat
},
{
"VIEW", "*", 1024, _vinit, _vreinit, NULL, _vpresend, NULL,
NULL, _vencdec, _vroute, _vfilter, _vformat
},
{    /* XATMI - identical to CARRAY */
"X_OCTET", "", 0
},
{    /* XATMI - identical to VIEW */
{'X','_','C','_','T','Y','P','E'}, "*", 1024, _vinit, _vreinit,
NULL, _vpresend, NULL, NULL, _vencdec, _vroute, _vfilter, _vformat
},
{    /* XATMI - identical to VIEW */
{'X','_','C','O','M','M','O','N'}, "*", 1024, _vinit, _vreinit,
NULL, _vpresend, NULL, NULL, _vencdec, _vroute, _vfilter, _vformat
},
{
"FML32", "", 1024, _finit32, _freinit32, _funinit32, _fpresend32,
_fpostsend32, _fpostrecv32, _fencdec32, _froute32, _ffilter32,
_fformat32
```

```
},
{
"VIEW32", "*", 1024, _vinit32, _vreinit32, NULL, _vpresend32, NULL,
NULL, _vencdec32, _vroute32, _vfilter32, _vformat32
},
{
""
}
};

struct tmtype_sw_t _TM_FAR *
_TMDLLENTRY
_tmtypeswaddr(void)
{
  return(tm_typesw);
}
```

Description

Shown in this page are the nine buffer types provided by the BEA TUXEDO system:

- character array (possibly containing `NULL` characters) that is neither encoded nor decoded during transmission
- `NULL`-terminated character array
- FML fielded buffer
- a C structure or an FML view
- equivalent to `CARRAY`; provided for XATMI compatibility
- equivalent to `VIEW`; provided for XATMI compatibility
- equivalent to `VIEW`; provided for XATMI compatibility
- FML32 fielded buffer, using 32-bit identifiers and offsets
- a C structure or an FML32 view, using 32-bit identifiers, counter variables, and size variables

Note that all `VIEW`, `X_C_TYPE`, and `X_COMMON` buffers are handled by the same set of routines; the name of a particular view is its sub-type name.

An application wishing to supply their own buffer type can do so by adding an instance to the `tm_typesw` array shown above. Whenever a new buffer type is added or one is deleted, care should be taken to leave a `NULL` entry at the end of the array as shown above. Note that a buffer type with a `NULL` name is not permitted.

A copy of the default array is delivered in `$TUXDIR/lib/tmtypesw.c`, and may be used as a starting point. The recommended procedure for installing a new buffer type switch is to compile `tmtypesw.c` and store it as the only element in a library named `libbuft`.

On systems with shared object capability, build and install a new instance of `libbuft.so.` under `$TUXDIR/lib`. All processes, including BEA TUXEDO system processes like `WSH`, will then automatically have access to the new type switch without recompilation. On a Windows workstation, the shared object for the buffer type switch is named `WBUFT.DLL`. It should be stored in `$TUXDIR\bin`. See the "Buffers" chapter in the *BEA TUXEDO Administrator's Guide*.

On systems without shared object capability, build and install a new instance of `libbuft.a` under `$TUXDIR/lib`. All processes needing to know about the new types must then be rebuilt, using `buildclient`(1) or `buildserver`(1). System processes like `WSH`(5) may need to be rebuilt using special commands such as `buildwsh`(1).

See `buffer`(3) for a description of the elements and routines in the buffer type switch. Also found there is a description of built in routines provided by the BEA TUXEDO system (for example, `_finit`()) that applications can use when changing the system-provided buffer types.

The two system-provided routing functions, `_froute`() and `_vroute`(), are used for data dependent routing of `FML` buffers and `VIEW` buffers, respectively. See `ubbconfig`(5) for a description of how to define routing criteria to be used by these two functions.

Files

```
$TUXDIR/tuxedo/include/tmtypes.h - the type switch definition
$TUXDIR/lib/tmtypesw.c - the default type switch instantiation
$TUXDIR/lib/libbuft.so. - type switch shared object
$TUXDIR/lib/libbuft.a - type switch archive library
```

See Also `buffer`(3), `typesw`(5), `ubbconfig`(5)

typesw(5)

Name buffer type switch structure-parameters and routines needed for each buffer type

Synopsis Buffer Type Structure

```
/*
 * The following definitions are in $TUXDIR/tuxedo/include/tmtypes.h
 */
#define TMTYPELEN  8
#define TMSTYPELEN 16
struct tmtype_sw_t {
 char type[TMTYPELEN];  /* type of buffer */
 char subtype[TMSTYPELEN];/* sub-type of buffer */
 long dfltsize;     /* default size of buffer */
 int (*initbuf)();  /* initialization function pointer */
 int (*reinitbuf)(); /* re-initialization function pointer */
 int (*uninitbuf)(); /* un-initialization function pointer */
 long (*presend)(); /* pre-send manipulation function pointer */
 void (*postsend)(); /* post-send manipulation function pointer */
 long (*postrecv)(); /* post-receive manipulation function pointer */
 long (*encdec)();  /* encode/decode function pointer */
 int (*route)();   /* data dependent routing function pointer */
 int (*filter)();   /* buffer filtering function pointer */
 int (*format)();   /* buffer format string function pointer */
 void (*reserved[10])();   /* reserved space for new function pointers */
 };

/*
 * application types switch pointer
 * always use this pointer when accessing the table
 */
extern struct tmtype_sw_t *tm_typeswp;
```

Description Each buffer type and sub-type must have an entry in the `tm_typesw` array such that when a buffer is manipulated the appropriate routines are called. For the buffer types provided by the BEA TUXEDO system. see `tuxtypes`(5).

An application wishing to supply their own buffer type can do so by adding an instance to the `tm_typesw` array in `$TUXDIR/lib/tmtypesw.c` (`tuxtypes`(5) shows how this can be done). The semantics of the routines which must be supplied when adding a new type are specified in `buffer`(3).

Files
```
$TUXDIR/tuxedo/include/tmtypes.h - the type switch definition
$TUXDIR/lib/tmtypesw.c - the type switch instantiation
```

See Also `buffer`(3), `tuxtypes`(5)

ubbconfig(5)

Name BEA TUXEDO System ASCII configuration file

Description A binary configuration file, called the TUXCONFIG file, contains information used by tmboot(1) to start the servers and initialize the bulletin board of a BEA TUXEDO system bulletin board instantiation in an orderly sequence. The binary TUXCONFIG file cannot be created directly (although an existing TUXCONFIG file can be dynamically modified through tmconfig(1)). Initially, a UBBCONFIG file of the format described on this reference page must be created. That file is parsed and loaded into the TUXCONFIG file using tmloadcf(1). tmadmin(1) uses the configuration file (or a copy of it) in its monitoring activity. tmshutdown(1) references the configuration file for information needed to shut the application down.

Definitions A server is a process that accepts requests and sends replies for clients and other servers. A client originates requests and gets replies.

A resource manager is an interface and associated software providing access to a collection of information and/or processes. An example of a resource manager is a database management system; a resource manager instance is a particular instantiation of a database controlled by a DBMS. A distributed transaction is a transaction that spans multiple resource manager instances, is started with tpbegin, and ended with tpcommit or tpabort.

A server group is a resource manager instance and the collection of servers and/or services providing access to that resource manager instance on a particular machine. The XA interface associated with the group is used for transaction management. If a server does not access a resource manager instance or does not access it as part of a distributed transaction, it must be in a server group with a null XA interface. Similarly, clients run in a special client group that does not have to be specified in the GROUPS section. The client group is not associated with a resource manager.

A remote domain is defined to be an environment for which this configuration's BEA TUXEDO system bulletin board is not available. Remote domains are not specified in the UBBCONFIG file, but rather through host-specific environment variables that are specified in host-specific reference pages.

Configuration File Format The format of a UBBCONFIG file is as follows:

The file is made up of up to none specification sections. Lines beginning with an asterisk (*) indicate the beginning of a specification section. Each such line contains the name of the section immediately following the *. Allowable section names are: RESOURCES, MACHINES, GROUPS, NETGROUPS, NETWORK, SERVERS, SERVICES,

INTERFACES, and ROUTING. The RESOURCES and MACHINES sections must be the first two sections in that order; the GROUPS section must be ahead of SERVERS, SERVICES, and ROUTING. The NETGROUPS section must ber ahead of the NETWORK section.

Parameters (except in the RESOURCES section) are generally specified by: *KEYWORD* = *value*. This sets *KEYWORD* to *value*. Valid keywords are described within each section. *KEYWORDs* are reserved; they can not be used as *values*.

Lines beginning with the reserved word, DEFAULT:, contain parameter specifications that apply to any lines that follow them in the section in which they appear. Default specifications can be used in all sections other than the RESOURCES section. They can appear more than once in the same section. The format for these lines is:

```
DEFAULT:  [optional KEYWORD=value pairs]
```

The values set on this line remain in effect until reset by another DEFAULT: line, or until the end of the section is reached. These values can also be overridden on non-DEFAULT: lines by placing the optional parameter setting on the line. If on a non-DEFAULT: line, the parameter setting is valid for that line only; lines that follow revert to the default setting. If DEFAULT: appears on a line by itself, all previously set defaults are cleared and their values revert to the system defaults.

If a value is *numeric*, standard C notation is used to denote the base (that is, 0x prefix for base 16 (hexadecimal), 0 prefix for base 8 (octal), and no prefix for base 10 (decimal)). The range of acceptable values for a numeric parameter is given under the description of that parameter.

If a value is an *identifier*, standard C rules are used. An *identifier* must start with an alphabetic character or underscore and contain only alphanumeric characters or underscores. The maximum allowable length of an identifier is 30 (not including the terminating null). An identifier cannot be the same as a KEYWORD.

A value that is neither an integer number or an identifier must be enclosed in double quotes. This value is called a *string*. The maximum allowable length of a string is 78 (not including the terminating null). Exceptions to this are the CLOPT, BUFTYPE, OPENINFO, and CLOSEINFO parameters, which can be 256 characters in length, and the RANGES parameter, which can be 2048 characters in length (except in Domain, where it can be no more than 1024 characters). In the RANGES parameter of the ROUTING section, certain special characters can be escaped inside a string using a backslash.

“\\” translates to a single backslash.
“\"” translates to a double quote.
“\n” translates to a newline.

"\t" translates to a tab.
"\f" translates to a formfeed.
"\O+" translates to a character whose octal value is O+

where `O+` is one, two, or three octal characters. "`\0`" translates to an embedded null character. "`\xH+`" or "`\XH+`" translates to a character whose hexadecimal value is `H+` where `H+` is one or more hexadecimal characters. "`\y`" (where 'y' is any character other than one of the previously mentioned characters) translates to 'y'; this produces a warning.

Some values are required to be an identifier. Other values that can be either an identifier or a string are indicated as a *`string_value`*. The maximum allowable length of a *`string_value`* is 78 characters if it is a string (not including the terminating null) and 30 characters if it is an identifier.

"#" (pound sign) introduces a comment. A newline ends a comment.

An identifier or a numeric constant must always be followed by white space (space, tab, or newline) or a punctuation character (pound sign, equals sign, asterisk, colon, comma, backslash, or period).

Blank lines and comments are ignored.

Comments can be freely attached to the end of any line.

Lines are continued by placing at least one tab after the newline. Comments can not be continued.

The RESOURCES Section

This section provides for user specification of the system-wide resources, such as the number of servers, and services which can exist within a service area. Lines in the `RESOURCES` section are of the form: *`KEYWORD value`* where *`KEYWORD`* is the name of the parameter, and *`value`* its associated value. Valid *`KEYWORDs`* are as follows:

`IPCKEY` *`numeric_value`*

specifies the numeric key for the well-known address in a BEA TUXEDO system bulletin board. In a single processor environment, this key "names" the bulletin board. In a multiple processor environment, this key names the message queue of the DBBL. In addition, this key is used as a basis for deriving the names of resources other than the well-known address, such as the names for bulletin boards throughout a multiprocessor. `IPCKEY` must be greater than 32,768 and less than 262,143. This parameter is required.

MASTER *string_value1[,string_value2]*

specifies the machine on which the master copy of the TUXCONFIG is found. Also, if the application is being run in MP mode, MASTER names the machine on which the DBBL should be run. *string_value2* names an alternate LMID location used during process relocation and booting. If the primary location is not available, the DBBL is booted at the alternate location and the alternate TUXCONFIG file found there is used. Both LMID values must name machines found in the MACHINES section and must be less than or equal to 30 characters in length. This parameter is required (even in SHM mode).

In an application that supports multiple release levels of the BEA TUXEDO system on different machines, MASTER and BACKUP must always have a release with a number greater than or equal to all other machines in the application. This rule is not enforced during a "Hot Upgrade."

DOMAINID *string_value1*

specifies the domain identification string. If not specified, the value "" is used. If the value of DOMAINID is a character string, it may contain a maximum of 30 characters (including the trailing null). If the value of DOMAINID is a string of hexadecimal digits, it may contain a maximum of 30 octets. If DOMAINID is specified, its value is included, as a parameter (-C dom=*domainid*), in any command output that reports on the processes associated with a particular domain, such as the output of the ps command. This comment is useful for an administrator managing multiple domains, who may have some difficulty, without this comment, in interpreting a single output stream that refers to several domains.

UID *numeric_value*

specifies the numeric user id to be associated with the IPC structures created for the bulletin board. This value should be a UNIX System user id on the local system. If not specified, the value is taken to be the effective user id of the user executing tmloadcf(1). The *RESOURCES* value for this parameter can be overridden in the MACHINES section on a per-processor basis.

GID *numeric_value*

specifies the numeric group id to be associated with the IPC structures created for the bulletin board. This value should be a valid UNIX System group id on the local system. If GID is not specified, the effective group id of the user executing tmloadcf(1) is used. The RESOURCES value for this parameter can be overridden in the MACHINES section on a per-processor basis.

PERM *numeric_value*

specifies the numeric permissions associated with the IPC structures that implement the bulletin board. It is used to specify the read/write permissions for processes in the usual UNIX system fashion (that is, with an octal number such a 0600). If not specified, the permissions on the IPC structures default to 0666 (read/write access by same user, same group, and any other). The value can be between 0001 and 0777, inclusive. The RESOURCES value for this parameter can be overridden in the MACHINES section on a per-processor basis.

MAXACCESSERS *numeric_value*

specifies the default maximum number of processes that can have access to a bulletin board on a particular processor at any one time. System administration processes, such as the BBL and tmadmin, need not be accounted for in this figure. This value must be greater than 0 and less than 32,768. If not specified, the default is 50. The RESOURCES value for this parameter can be overridden in the MACHINES section on a per-processor basis.

MAXSERVERS *numeric_value*

specifies the maximum number of servers to be accommodated in the server table of the bulletin board. This value must be greater than 0 and less than 8192. If not specified, the default is 50.

MAXSERVICES *numeric_value*

specifies the maximum total number of services to be accommodated in the services table of the bulletin board. This value must be greater than 0 and less than 32,768. If not specified, the default is 100.

MAXGROUPS *numeric_value*

specifies the maximum number of configured server groups to be accommodated in the group table of the bulletin board. This value must greater than or equal to 100 and less than 32,768. If not specified, the default is 100.

MAXNETGROUPS *numeric_value*

specifies the maximum number of configured network groups to be accommodated in the NETWORK section of the TUXCONFIGfile. This value must be greater than or equal to 1 and less than 8192. If not specified, the default is 8.

MAXMACHINES *numeric_value*
: specifies the maximum number of configured machines to be accommodated in the machine tables of the bulletin board. This value must greater than or equal to 256 and less than 8,191. If not specified, the default is 256.

MAXQUEUES *numeric_value*
: specifies the maximum number of server request queues to be accommodated in the queue table of the bulletin board. This value must greater than or equal to 1 and less than 8,192. If not specified, the value is set to the configured value for MAXSERVERS. Interoperability with releases prior to 5.0 requires that this value be equal to the configured value for MAXSERVERS.

MAXACLGROUPS *numeric_value*
: specifies the maximum number of group identifiers that can be used for ACL permissions checking. The maximum group identifier that can be defined is TA_MAXACLGROUPS - 1. This value must be greater than or equal to 1 and less than or equal to 16K. If not specified, the default is 16K.

MODEL { SHM | MP }
: specifies the configuration type. This parameter is required and only one of the two settings can be specified. SHM specifies a single machine configuration; only one machine may be specified in the MACHINES section. MP specifies a multi-machine configuration; MP must be specified if a networked application is being defined. Note: to change *value* without relinking, servers must be built to support the models needed (see buildserver(1)).

LDBAL { Y | N }
: specifies whether or not load balancing should be performed. If LDBAL is not specified, the default is Y. It is recommended that if each service maps to one and only one queue, then LDBAL should be set to N, since load balancing is automatic.

  If you set LDBAL to Y, server load balancing is performed automatically. Each interface request is routed to the server with the smallest total load. The routing of a request to a server causes the server's total to be increased by the LOAD factor of the CORBA interface requested.

  When load balancing is not activated and multiple servers offer the same CORBA interface, the first available queue receives the request.

CMTRET { COMPLETE | LOGGED }
: specifies the initial setting of the TP_COMMIT_CONTROL characteristic for all client and server processes in a System/T application. If *value* is LOGGED, then the TP_COMMIT_CONTROL characteristic is initialized to TP_CMT_LOGGED; otherwise, it is initialized to TP_CMT_COMPLETE. If CMTRET is not specified, the default is COMPLETE. See the description of the BEA TUXEDO system ATMI function, tpscmt, for details on the setting of this characteristic.

OPTIONS *identifier[,identifier . . .]*
: specifies options that are used. If more than one option is given, they are separated by commas. The following are the options that can be specified. The identifier LAN indicates that this is a networked application. The identifier MIGRATE indicates that server group migration can be done. If MIGRATE is specified, LAN should also be specified, (except for the case where the configuration runs on a single multiprocessor computer). This parameter is optional and the default is no options.

SYSTEM_ACCESS *identifier[,identifier]*
: specifies the default mode used by BEA TUXEDO system libraries within application processes to gain access to BEA TUXEDO system's internal tables. Valid access types are FASTPATH or PROTECTED. FASTPATH specifies that the internal tables should be accessible by the libraries via shared memory for fast access. PROTECTED specifies that while the internal tables are accessible by BEA TUXEDO system libraries via shared memory, the shared memory for these tables is not accessible outside of the BEA TUXEDO system libraries. NO_OVERRIDE can be specified (either alone or in conjunction with FASTPATH or PROTECTED) to indicate that the mode selected cannot be overridden by an application process. If SYSTEM_ACCESS is not specified, the default mode is FASTPATH.

SECURITY *string_value1*
: specifies the type of application security to be enforced. The possible string values are NONE, APP_PW, USER_AUTH, ACL, or MANDATORY_ACL. This parameter defaults to NONE. The value APP_PW indicates that application password security is to be enforced (clients must provide the application password during initialization). Setting APP_PW causes tmloadcf to prompt for an application password. The value USER_AUTH is similar to APP_PW but, in addition, indicates that per-user authentication will be done during client initialization. The value ACL is similar to USER_AUTH but, in addition, indicates that access control checks will be done on service names, queue names, and event names. If an associated ACL is not found for a name, it is

assumed that permission is granted. The value MANDATORY_ACL is similar to ACL but permission is denied if an associated ACL is not found for the name.

AUTHSVC *string_value*
: specifies the name of an application authentication service that is invoked by the system for each client joining the system. This parameter requires that the SECURITY identifier be set to USER_AUTH, ACL, or MANDATORY_ACL. (For upward compatibility, setting both SECURITY APP_PW and AUTHSVC implies SECURITY USER_AUTH.) The parameter value must be 15 characters or less in length. For SECURITY level USER_AUTH, the default service name, if not specified, is AUTHSVC. For SECURITY level ACL or MANDATORY_ACL, the service name must be ..AUTHSVC. (This will be silently enforced if the administrator tries to set it to anything else.)

MAXGTT *numeric_value*
: specifies the maximum number of simultaneous global transactions in which a particular machine can be involved. It must be greater than or equal to 0 and less than 32768. If not specified, the default is 100. The RESOURCES value for this parameter can be overridden in the MACHINES section on a per-processor basis.

MAXCONV *numeric_value*
: specifies the maximum number of simultaneous conversations in which processes on a particular machine can be involved. It must be greater than 0 and less than 32,768. If not specified, the default is 10 if any conversational servers are defined in the SERVERS section and 1 otherwise. The RESOURCES value for this parameter can be overridden in the MACHINES section on a per-processor basis. The maximum number of simultaneous conversations per server is 64.

MAXBUFTYPE *numeric_value*
: specifies the maximum number of buffer types that can be accommodated in the buffer type table in the bulletin board. It must be greater than 0 and less than 32,768. If not specified, the default is 16.

MAXBUFSTYPE *numeric_value*
: specifies the maximum number of buffer subtypes that can be accommodated in the buffer subtype table in the bulletin board. It must be greater than 0 and less than 32,768. If not specified, the default is 32.

MAXDRT *numeric_value*
: specifies the maximum number of configured data dependent routing criteria entries. It must be greater than or equal to 0 and less than 32,768. If not

specified, the default is determined from the configured ROUTING section entries.

MAXRFT *numeric_value*
: specifies the maximum number of data dependent routing range field table entries. It must be greater than or equal to 0 and less than 32,768. If not specified, the default is determined from the configured ROUTING section entries.

MAXRTDATA *numeric_value*
: specifies the maximum string pool size for data dependent routing range strings. It must be greater than or equal to 0 and less than 32,761. If not specified, the default is determined from the configured ROUTING section entries.

SCANUNIT *numeric_value*
: is the interval of time (in seconds) between which periodic scans are done by the BBL to find old transactions and timed-out blocking calls within service requests. This value is used as the basic unit of scanning by the BBL. It affects the granularity with which transaction timeout values can be specified on tpbegin(3c) and the blocking timeout value specified with the BLOCKTIME parameter. The SANITYSCAN, BBLQUERY, DBBLWAIT, and BLOCKTIME parameters are multipliers of this unit for other timed operations within the system. SCANUNIT must be a multiple of 5 greater than 0 and less than or equal to 60 seconds. The default is 10 seconds.

SANITYSCAN *numeric_value*
: sets a multiplier of the basic SCANUNIT between sanity checks of the system. The value SCANUNIT must be greater than 0. If this parameter is not specified, the defaultdefault is set so that (SCANUNIT * SANITYSCAN) is approximately 120 seconds. Sanity checks include checking servers as well as the bulletin board data structure itself. Each BBL checks that all servers on its machine are viable; that is, the server hasn't terminated abnormally and is not looping. Processes deemed not viable are either cleaned up, or restarted depending on the options with which they were started. Following that, the BBL sends a message (without reply) to the DBBL to indicate it is okay.

DBBLWAIT *numeric_value*
: sets a multiplier of the basic SCANUNIT for the maximum amount of wall time a DBBL should wait for replies from all its BBLs before timing out. Every time the DBBL forwards a request to its BBLs, it waits for all of them to reply with a positive acknowledgment before replying to the requester. This option

can be used for noticing dead or insane BBLs in a timely manner. The value of DBBLWAIT must be greater than 0. If this parameter is not specified, the default is set so that (SCANUNIT * DBBLWAIT) is the greater of SCANUNIT or 20 seconds.

BBLQUERY *numeric_value*
: sets a multiplier of the basic SCANUNIT between status checks by the DBBL of all BBLs. The DBBL checks to ensure that all BBLs have reported in within the BBLQUERY cycle. If a BBL has not been heard from, the DBBL sends a message to that BBL asking for status. If no reply is received, the BBL is partitioned. The value of BBLQUERY must be greater than 0. If this parameter is not specified, the default is set so that (SCANUNIT * BBLQUERY) is approximately 300 seconds.

BLOCKTIME *numeric_value*
: sets a multiplier of the basic SCANUNIT after which a blocking call (for example, receiving a reply) times out. The value of BLOCKTIME must be greater than 0. If this parameter is not specified, the default is set so that (SCANUNIT * BLOCKTIME) is approximately 60 seconds.

NOTIFY { DIPIN | SIGNAL | IGNORE }
: specifies the default notification detection method to be used by the system for unsolicited messages sent to client processes. This default can be overridden on a per-client basis using the appropriate tpinit(3c) flag value. Note that once unsolicited messages are detected, they are made available to the application through the application defined unsolicited message handling routine identified via the tpsetunsol function (tpnotify(3c)). The value DIPIN specifies that dip-in-based notification detection should be used. This means that the system will only detect notification messages on behalf of a client process while within ATMI calls. The point of detection within any particular ATMI call is not defined by the system and dip-in detection will not interrupt blocking system calls. DIPIN is the default notification detection method. The value SIGNAL specifies that signal-based notification detection should be used. This means that the system sends a signal to the target client process after the notification message has been made available. The system installs a signal catching routine on behalf of clients selecting this method of notification. All signaling of client processes is done by administrative system processes and not by application processes. Therefore, only clients running with the same UNIX System user identifier can be notified using the SIGNAL method. The value IGNORE specifies that by default, notification messages are to be ignored by application clients. This would be appropriate

in applications where only clients that request notification at tpinit time should receive unsolicited messages.

`USIGNAL { SIGUSR1 | SIGUSR2 }`
: specifies the signal to be used if `SIGNAL`-based notification is used. The legal values for this parameter are `SIGUSR1` and `SIGUSR2`. `SIGUSR2` is the default for this parameter. `USIGNAL` may be specified even if `SIGNAL`-based notification is not selected with the `NOTIFY` parameter, because callers of `tpinit` may choose signal-based notification.

The MACHINES Section

The `MACHINES` section specifies the logical names for physical machines for the configuration. It also specifies parameters specific to a given machine. The `MACHINES` section must contain an entry for each physical processor used by the application. Entries have the form:

`ADDRESS  required parameters [optional parameters]`

where *`ADDRESS`* is the physical name of a processor, for example, the value produced by the UNIX system `uname -n` command. The length of the entire `ADDRESS` must be 30 characters or less. If the name is not an identifier, it must be enclosed in double quotes. If the `LAN` option is not specified, only one machine name can appear in this section. One of the required *`KEYWORD`*s is `LMID`, which is the logical machine *`string_value`* assigned to the physical machine. An `LMID` *`string_value`* must be unique within the `MACHINES` section of the configuration file.

`LMID` = *`string_value`*
: specifies that *`string_value`* is to be used in other sections as the symbolic name for *`ADDRESS`*. This name cannot contain a comma, and must be 30 characters or less. This parameter is required. There must be an `LMID` line for every machine used in a configuration.

These parameters are required:

`TUXCONFIG` = *`string_value`*
: This is the absolute pathname of the file or device where the binary `TUXCONFIG` file is found on this machine. The maximum string value length is 64 characters. The administrator need only maintain one `TUXCONFIG` file, namely the one that is pointed to by the `TUXCONFIG` environment variable on the `MASTER` machine. Copies on other machines of this master `TUXCONFIG` file are synchronized with the `MASTER` machine automatically when the system is booted. This parameter must be specified for each machine. If `TUXOFFSET` is specified, then the BEA TUXEDO file system starts at that number of blocks from the beginning of the `TUXCONFIG` device (see `TUXOFFSET` below). See

ENVFILE in the MACHINES section for a discussion of how this value is used in the environment.

TUXDIR = *string_value*

This is the absolute pathname of the directory where the BEA TUXEDO system software is found on this machine. This parameter must be specified for each machine and the pathname should be local to each machine; in other words, TUXDIR should not be on a remote file system. If the machines of a multiprocessor application have different BEA TUXEDO system releases installed, check the discussion of "Interoperability" in the *BEA TUXEDO Release Notes* for the higher level release to make sure you will get the functionality you expect. See ENVFILE in the MACHINES section for a discussion of how this value is used in the environment.

APPDIR = *string_value*

The value specified for this parameter is the absolute pathname of the application directory and is the current directory for all application and administrative servers booted on this machine. The absolute pathname can optionally be followed by a colon-separated list of other pathnames. In a configuration where SECURITY is set, each application must have its own distinct APPDIR. See ENVFILE in the MACHINES section for a discussion of how this value is used in the environment.

Optional parameters are:

UID = *number*

specifies the numeric user id to be associated with the IPC structures created for the bulletin board. The valid range is 0-2147483647. If not specified, the default is the value specified in the RESOURCES section.

GID = *number*

specifies the numeric group id to be associated with the IPC structures created for the bulletin board. The valid range is 0-2147483647. If not specified, the default is the value specified in the RESOURCES section.

PERM = *number*

specifies the numeric permissions associated with the IPC structures that implement the bulletin board. It is used to specify the read/write permissions for processes in the usual UNIX system fashion (that is, with an octal number such as 0600). The value can be between 0001 and 0777, inclusive. If not specified, the default is the value specified in the RESOURCES section.

MAXACCESSERS = *number*

specifies the maximum number of processes that can have access to the bulletin board on this processor at any one time. System administration processes, such as the BBL and `tmadmin`, need not be accounted for in this figure, but all application servers and clients and `TMS` servers are counted. This value must be greater than 0 and less than 32,768. If not specified, the default is the value specified in the `RESOURCES` section.

MAXWSCLIENTS = *number*

specifies the number of accesser entries on this processor to be reserved for workstation clients only. The parameter is only used when the BEA TUXEDO system Workstation feature is used. The number specified here takes a portion of the total accesser slots specified with `MAXACCESSERS`. The appropriate setting of this parameter helps to conserve IPC resources since workstation client access to the system is multiplexed through a BEA TUXEDO system-supplied surrogate, the workstation handler. This value must be greater than or equal to 0 and less than 32,768. The default is 0. It is an error to set this number greater than `MAXACCESSERS`.

MAXACLCACHE = *number*

specifies the number of entries in the cache used for ACL entries when `SECURITY` is set to `ACL` or `MANDATORY_ACL`. The appropriate setting of this parameter helps to conserve on shared memory resources and yet reduce the number of disk access to do ACL checking. This value must be greater than or equal to 10 and less than or equal to 30,000. The default is 100.

MAXCONV = *number*

specifies the maximum number of simultaneous conversations in which processes on a particular machine can be involved. It must be greater than 0 and less than 32,768. If not specified, the default is the value specified in the `RESOURCES` section. The maximum number of simultaneous conversations per server is 64.

MAXPENDINGBYTES = *number*

specifies a limit for the amount of space that can be allocated for messages waiting to be transmitted by the bridge process. *number* must be between 100,000 and `MAXLONG`.

MAXGTT = *number*

specifies the maximum number of simultaneous global transactions in which a particular machine can be involved. It must be greater than or equal to 0 and

less than 32,768. If not specified, the default is the value specified in the RESOURCES section.

TYPE = *string_value*
: is used for grouping machines into classes. TYPE can be set to any string value that is 15 characters or less. If two machines have the same TYPE value, data encoding/decoding is bypassed when sending data between the machines. TYPE can be given any string value. It is used simply for comparison. The TYPE parameter should be used when the application involves a heterogeneous network of machines or when different compilers are used on the machines in the network. If not specified, the default is the null string, which matches any other entry that does not have a value specified.

CMPLIMIT = *string_value1[,string_value2]*
: specifies the threshold message size for messages bound to remote processes (*string_value1*) and local processes (*string_value2*) respectively, at which automatic data compression will take place. Both values must be either a non-negative numeric value or the string MAXLONG. If not specified, the default for this parameter is MAXLONG,MAXLONG.

NETLOAD = *numeric_value*
: specifies the additional load to be added when computing the cost of sending a service request from this machine to another machine. It must be greater than or equal to 0 and less than 32,768. If not specified, the default is 0.

SPINCOUNT = *numeric_value*
: specifies the number of attempts that should be made at user level to lock the bulletin board before blocking processes on a UNIX semaphore. This value must be greater than or equal to 0. A value of 0 indicates that the spincount built into the delivered binary should be used. If set, this parameter causes the TMSPINCOUNT environment variable to be ignored. This varies from platform to platform. The default for this parameter is 0.

TLOGDEVICE = *string_value*
: specifies the BEA TUXEDO file system that contains the DTP transaction log (TLOG) for this machine. The TLOG is stored as a BEA TUXEDO system VTOC table on the device. If this parameter is not specified, then the machine is assumed to not have a TLOG. The maximum string value length is 64 characters.

TLOGOFFSET = *offset*
: specifies the numeric offset in pages (from the beginning of the device) to the start of the BEA TUXEDO file system that contains the DTP transaction log

for this machine. The offset must be greater than or equal to 0 and less than the number of pages on the device. The default is 0.

`TLOGNAME` = *`string_value`*

specifies the name of the DTP transaction log for this machine. If not specified, the default is `TLOG`. If more than one `TLOG` exists on the same `TLOGDEVICE`, they must have unique names. `TLOGNAME` must be different from the name of any other table on the configuration where the `TLOG` table is created. It must be 30 characters or less.

`TLOGSIZE` = *`size`*

specifies the numeric size, in pages, of the DTP transaction log for this machine. It must be greater than 0 and less than or equal to 2048, subject to the amount of available space on the BEA TUXEDO file system. If not specified, the default is 100 pages.

`ULOGPFX` = *`string_value`*

specifies the absolute pathname prefix of the path for the `userlog`(3c) message file on this machine. The value of `ULOGPFX` for a given machine is used to create the `userlog`(3c) message file for all servers, clients, and administrative processes executed on that machine. If this parameter is not specified, `$APPDIR/ULOG` is used. "`mmddyy`" (month, day, year) is appended to the prefix to get the actual log file name.

`TUXOFFSET` = *`offset`*

specifies the numeric offset in pages (from the beginning of the device) to the start of the BEA TUXEDO file system that contains the `TUXCONFIG` for this machine. The offset must be greater than or equal to 0 and less than the number of pages on the device. The default offset is 0. The value of `TUXOFFSET`, if non-zero, is placed in the environment of all servers booted on a machine. See `ENVFILE` in the `MACHINES` section for a discussion of how this value is used in the environment.

`ENVFILE` = *`string_value`*

specifies that all clients and servers on the machine are to be executed with the environment specified in the named file. If the value specifies an invalid file name, no values are added to the environment. Lines must be of the form *`ident=value`* where *`ident`* begins with an underscore or alphabetic character, and contains only underscore or alphanumeric characters. Within the *`value`*, strings of the form ${*`env`*} are expanded when the file is processed using variables already in the environment. (Forward referencing is not supported and if a value is not set, the variable is replaced with the empty

string). Backslash (\) may be used to escape the dollar sign and itself. All other shell quoting and escape mechanisms are ignored and the expanded *value* is placed into the environment.

Client programs process only the MACHINES ENVFILE during tpinit()

When booting servers, local servers inherit the environment of tmboot(1) and remote servers (not on the MASTER) inherit the environment of tlisten(1). TUXCONFIG, TUXDIR, and APPDIR are also put into the environment when a server is booted based on the information in the associated MACHINES entry. An attempt to reset these three variables to another value will not be allowed and will result in a warning. tmboot and tlisten process the machine ENVFILE before starting the server, allowing for the environment to indicate necessary pathnames for finding executable and dynamically loaded files. Once the server is running,as part of server initialization (before the application gets control in tpsvrinit), a server will read and export variables from both the machine and server ENVFILE files. If a variable is set in both the machine and server ENVFILE, the value in the server ENVFILE will override the value in the machine ENVFILE.

PATH and LD_LIBRARY_PATH are treated specially. Before a server is activated, the machine ENVFILE is scanned to find the first occurrence of a PATH or LD_LIBRARY_PATH variable; embedded environment variables within either PATH variable are not expanded. PATH and LD_LIBRARY_PATH are used to find pathnames for executable and dynamically loaded files. PATH will always be prefixed with

```
${APPDIR}:${TUXDIR}/bin:/bin:
```

if the value doesn't already begin with this string. This PATH will be used as a search path for servers that are specified with a simple or relative pathname. LD_LIBRARY_PATH will always be prefixed with

```
${APPDIR}:${TUXDIR}/lib:/lib:/usr/lib:
```

if the value doesn't already begin with this string. SHLIB_PATH is set on HPUX and LIBPATH is set on AIX instead of LD_LIBRARY_PATH.

The GROUPS Section

This section provides information about server groups. This section must have at least one server group defined in it (which can be added via tmconfig(1) after the TUXCONFIG file has been created). A server group entry provides a logical name for a collection of servers and/or services on a machine. The logical name is used as the value of the SRVGRP parameter in the SERVERS section to identify a server as part of this group. SRVGRP is also used in the SERVICES section to identify a particular

instance of a service with its occurrences in the group. Other GROUPS parameters associate this group with a specific resource manager instance (for example, the employee database). Lines within the GROUPS section have the form:

GROUPNAME required parameters [optional parameters]

where *GROUPNAME* specifies the logical name (*string_value*) of the group. The group name must be unique within all group names in the GROUPS section and LMID values in the MACHINES section and cannot contain an asterisk (*), comma, or colon. It must be 30 characters or less.

Required parameters are:

LMID = *string_value1 [, string_value2]*
: specifies that this group of servers resides on the machine symbolically named by *string_value1* in the MACHINES section (or the default in SHM mode). Each LMID value must be 30 characters or less. Up to two logical machine names can be specified. The second logical name, if given and if server group migration is enabled, indicates the machine to which the server group can be migrated.

GRPNO = *number*
: specifies the numeric group number associated with this server group. This number must be greater than 0 and less than 30000, and must be unique among all entries in the GROUPS section.

Optional parameters are:

TMSNAME = *string_value*
: specifies the name of the transaction manager server a.out associated with this group. This parameter must be specified for any group entry whose servers will participate in distributed transactions (transactions across multiple resource managers--and possibly machines--that are started with tpbegin(3c), and ended with tpcommit(3c)/ tpabort(3c)). It specifies the file (*string_value*) to be executed by tmboot(1) when booting the server group. The value TMS is reserved to indicate use of the null XA interface. If a non-empty value other than TMS is specified, then a TLOGDEVICE must be specified for the machine(s) associated with the LMID value(s) for this entry. A unique server identifier is selected automatically for each TM server, and the servers will be restartable an unlimited number of times.

ENVFILE = *string_value*

specifies that all servers in the group are to be executed with the environment specified in the named file. If the value specifies an invalid file name, no values are added to the environment. Lines must be of the form *ident=value* where *ident* contains only underscore or alphanumeric characters. Within the *value*, strings of the form ${*env*} are expanded when the file is processed using variables already in the environment. (Forward referencing is not supported and if a value is not set, the variable is replaced with an empty string.) A backslash (\) may be used to escape the dollar sign and itself. All other shell quoting and escape mechanisms are ignored and the expanded *value* is placed in the environment.

The ENVFILE is read after the MACHINES section ENVFILE (if one exists) and before the SERVERS section ENVFILE (if one is specified).

TMSCOUNT = *number*

specifies the number of transaction manager servers to start for the associated group, if TMSNAME is specified. This parameter is optional and the default is 3. If specified and the value is non-zero, the minimum value is 2 and the maximum value is 10. The servers are set up in an MSSQ set automatically.

OPENINFO = "*string*"

specifies the resource manager instance-dependent information needed when opening the resource manager. The value must be enclosed in double quotes and must be less than or equal to 256 characters in length. This value is ignored if TMSNAME is not set or is set to TMS. The format of the OPENINFO string is dependent on the requirements of the vendor providing the underlying resource manager. The information required by the vendor must be prefixed with "*rm_name*:," which is the published name of the vendor's transaction (XA) interface followed immediately by a colon (:).
For BEA TUXEDO system databases, the format is:

OPENINFO="TUXEDO/D:*fsconfig*:*dbname*:*openmode*"

where "TUXEDO/D" is the published name of the BEA TUXEDO XA interface, *fsconfig* is the name of the FSCONFIG on which the database resides, *dbname* is the name of the database, and *openmode* is one of "readonly" or "readwrite". For NT and NetWare, the colon separator after *fsconfig* and *dbname* must be a semi-colon.

For BEA TUXEDO system/SQL databases, the format is:

OPENINFO="TUXEDO/SQL:*fsconfig*:*dbname*:*openmode*"

where "TUXEDO/SQL" is the published name of the BEA TUXEDO system/SQL XA interface, *fsconfig* is the name of the FSCONFIG on which the database resides, *dbname* is the name of the database, and *openmode* is one of "readonly" or "readwrite". For NT and NetWare, the colon separator after *fsconfig* and *dbname* must be a semi-colon.

For BEA TUXEDO /Q databases, the format is:

OPENINFO="TUXEDO/QM:*qmconfig*:*qspace*"

where "TUXEDO/QM" is the published name of the BEA TUXEDO /Q XA interface, *qmconfig* is the name of the QMCONFIG on which the queue space resides and *qspace* is the name of the queue space. For NT and NetWare, the colon separator after *qmconfig* must be a semi-colon.

If TMSNAME is set but the OPENINFO string is set to the null string ("") or this parameter does not appear on the entry, it means that a resource manager exists for the group but does not require any information for executing an open operation.

CLOSEINFO = "*string*"

specifies the resource manager instance-dependent information needed when closing the resource manager. The value must be enclosed in double quotes and must be less than or equal to 256 characters in length. This value is ignored if TMSNAME is not set or is set to TMS. The format of the CLOSEINFO string is dependent on the requirements of the vendor providing the underlying resource manager. The information required by the vendor must be prefixed with "*rm_name*:," which is the published name of the vendor's transaction (XA) interface followed immediately by a colon (:). For BEA TUXEDO system/SQL databases, a CLOSEINFO string is not used. If TMSNAME is set but the CLOSEINFO string is set to the null string ("") or this parameter does not appear on the entry, it means that a resource manager exists for the group but does not require any information for executing a close operation.

The NETGROUPS Section

The NETGROUPS section describes the network groups available to the application in the LAN environment. Any pair of machines may be in any number of network groups. Two communicating nodes use the priority mechanism in order to determine how to communicate between elements of its group.

Every LMID must be a member of the default network group, DEFAULTNET. Machines running BEA TUXEDO releases earlier than Release 6.4 (in which NETGROUPS became available) can belong only to the DEFAULTNET network group. The network group number (NETGRPNO) for DEFAULTNET. is 0 (zero), and may not be changed. The default priority of DEFAULTNET, however, may be modified.

The general format for entries in this section is:

```
NETGROUP required_parameters [ optional_parameters ]
```

where *NETGROUP* is the network group name. If *NETGROUP* is equal to DEFAULTNET then the entry describes the default network group.

Required parameters are:

NETGRPNO = *numeric_value*
: This is a unique network group number which must be assigned by the administrator for use in failover and failback situations. If this entry describes DEFAULTNET, then the numeric value must be 0 (zero).

Optional parameters are:

NETPRIO = *numeric_value*
: Specifies the priority of this network group. A pair of machines in multiple network groups of the same priority will communicate in parallel over the priority band as long as no network group of a higher priority is available. If all the network links of a certain priority band have been torn down by the administrator or by network conditions, then the next lowest priority band is used. Retries of the higher priority bands will be attempted. For more information, see the chapter on BEA TUXEDO networks in the *BEA TUXEDO Administrator's Guide* This value must be greater than zero and less than 8192. If not specified, the default is 100. Note that this is the only parameter of the DEFAULTNET that can be altered.
Note: In Release 6.4, parallel data circuits are prioritized by network group number (NETGRPNO) within priority group number. In future releases, a different algorithm may be used to prioritize parallel data circuits.

The NETWORK Section

The NETWORK section describes the network configuration for a LAN environment. For each processor on which a bridge server is located, an entry must be placed in the NETWORK section giving the network address of the bridge process. An error is generated if this section exists and LAN is not specified for the OPTIONS parameter of the RESOURCES section.

The general format for entries in this section is:

LMID required parameters [optional parameters]

where *LMID* is the logical machine where the bridge process is placed. *LMID* must have direct access to the network device to be used (as given in the BRIDGE parameter).

Required parameters are:

NADDR = *string_value*

Specifies the complete network address to be used by the bridge process placed on the *LMID* as its listening address. The listening address for a bridge is the means by which it is contacted by other bridge processes participating in the application. If *string_value* has the form "0xhex-digits" or "\\xhex-digits", it must contain an even number of valid hex digits. These forms are translated internally into a character array containing TCP/IP addresses may also be in either of the following two forms:

"//host.name:port_number"
"//#.#.#.#:port_number"

In the first of these formats, *hostname* is resolved to a TCP/IP host address at the time the address is bound using the locally configured name resolution facilities accessed via an operating system command. The "*#.#.#.#*" is the dotted decimal format where each *#* represents a decimal number in the range 0 to 255. *Port_number* is a decimal number in the range 0 to 65535. the hexadecimal representations of the string specified.

Optional parameters are:

BRIDGE = *string_value*

specifies the device name to be used by the bridge process placed on that *LMID* to access the network. The BRIDGE parameter is not required. In prior releases, for networks that were TLI-based, an absolute pathname for a device was required as the value of BRIDGE. The network transport endpoint file path has the form: /dev/*provider_name*. For StarLAN this is: /dev/starlan.

NLSADDR = *string_value*

specifies the network address used by the tlisten(1) process servicing the network on the node identified by the *LMID*. The network address used for NLSADDR is of the same format as that specified for the NADDR parameter above. If the address has the form "0xhex-digits" or "\\xhex-digits", it must contain an even number of valid hex digits. TCP/IP addresses may be in the "//*#.#.#.#:port*" format. tmloadcf(1) prints an error if NLSADDR is missing on any entry but the MASTER LMID, for which it prints a warning. However, if NLSADDR is missing on the MASTER LMID, tmadmin(1) will not

be able to run in administrator mode on remote machines; it will be limited to read-only operations. This also means that the backup site will be unable to to reboot the master site after failure.

`MINENCRYPTBITS={0|40|128}`
: When establishing a network link to this machine, 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 is `0`.

`MAXENCRYPTBITS={0|40|128}`
: When establishing a network link, negotiate encryption up to this level. `0` means no encryption, whilc `40` and `128` specify the encryption length (in bits). The default is `128`

`NETGROUP = string_value`
: *`string_value`* is the network group associated with this network entry. If unspecified, then the default, `DEFAULTNET`, is assumed. The `NETGROUP` parameter, if not set to `DEFAULTNET`, must have previously appeared as a group name in the `NETGROUPS` section of the file. All network entries with a`NETGROUP DEFAULTNET` are represented in the `T_MACHINE` class of the `TM_MIB`, while `NETWORK` entries associated with any other `NETGROUP` are represented in the `T_NETMAP` class of the `TM_MIB` to interoperate with previous releases.

The SERVERS Section

This section provides information on the initial conditions for servers started in the system. The notion of a server as a process that continually runs and waits for a server group's service requests to process, may or may not apply to a particular remote environment. For many environments, the operating system or perhaps a remote gateway will be the sole dispatcher of services; when either of these is the case, only `SERVICE` table entries (see next section) and no `SERVER` table entries need be specified for remote program entry points; BEA TUXEDO system gateway servers would advertise and queue remote domain service requests. Host-specific reference pages must indicate whether or not `UBBCONFIG` server table entries apply in their particular environments, and if so, the corresponding semantics. Lines within the `SERVERS` section have the form:

`AOUT` `required parameters [optional parameters]`

where *`AOUT`* specifies the file (*`string_value`*) to be executed by `tmboot`(1). `tmboot` executes *`AOUT`* on the machine specified for the server group to which the server belongs. `tmboot` searches for the *`AOUT`* file on its target machine. Thus, *`AOUT`* must exist in a filesystem on that machine. (Of course, the path to *`AOUT`* can include RFS

connections to filesystems on other machines.) If a relative pathname for a server is given, the search for *AOUT* is done first in APPDIR, then in TUXDIR/bin, then in /bin, and then in path> where path> is the value of the last PATH= line appearing in the machine environment file, if one exists. The values for APPDIR and TUXDIR are taken from the appropriate machine entry in the TUXCONFIG file. See ENVFILE in the MACHINES section for a more detailed discussion.

Required parameters are:

SRVGRP = *string_value*
: specifies the group name for the group in which the server is to run. *string_value* must be the logical name associated with a server group in the GROUPS section. It must be 30 characters or less. This association with an entry in the GROUPS section means that *AOUT* is executed on the machine with the LMID specified for the server group. It also specifies the GRPNO for the server group and parameters to pass when the associated resource manager is opened. All server entries must have a server group parameter specified.

SRVID = *number*
: specifies an integer that uniquely identifies a server within a group. Identifiers must be between 1 and 30,000 inclusive. This parameter must be present on every server entry.

The optional parameters are divided into two categories: boot options and run-time options. Boot options are used by tmboot(1) when it executes a server. Once running, a server reads its entry from the configuration file to determine its run-time options. The unique server id is used to find the right entry.

Optional boot parameters are:

CLOPT = *string_value*
: specifies servopts(5) options to be passed to *AOUT* when booted. If none is specified, the default is -A. *string_value* can be up to 256 characters in length.

SEQUENCE = *number*
: specifies when this server should be booted or shutdown relative to other servers. If the SEQUENCE parameter is not specified, servers are booted in the order found in the SERVERS section (and shut down in the reverse order). If a mixture of servers with and without sequence numbers is given, all servers with sequence numbers are booted first from low to high sequence number, then all servers without sequence numbers are booted in the order they appear

in the configuration file. Sequence numbers must be in the range between 1 and 9999.

`MIN` = *`number`*
: specifies the minimum number of occurrences of the server to boot by `tmboot`. If an `RQADDR` is specified and `MIN` is greater than 1, then the servers will form an MSSQ set. The server identifiers for the servers will be `SRVID` up to `SRVID + MAX - 1`. All occurrences of the server will have the same sequence number, as well as any other server parameters. The value range for `MIN` is 0 to 1000. If not specified, the default is 1.

`MAX` = *`number`*
: specifies the maximum number of occurrences of the server that can be booted. Initially, `tmboot` boots `MIN` servers, and additional servers can be booted up to `MAX` occurrences using the `-i` option of `tmboot` to specify the associated server identifier. The value range for `MAX` is 0 to 1000. If not specified, the default is the same value as `MIN`.

Optional run-time parameters are:

`ENVFILE` = *`string_value`*
: requests the addition of the values in this file to the environment of the server during its initialization. If a server is associated with a server group that can be migrated to a second machine, the `ENVFILE` must be in the same location on both machines.
Note that this file is processed after the server starts. Therefore, it cannot be used to set the pathnames used to find executable or dynamically loaded files needed to execute the server; use the machine `ENVFILE` instead. See `ENVFILE` in the `MACHINES` section for a discussion of how this file is used to modify the environment.

`CONV` = `{ Y | N }`
: specifies whether or not the server is a conversational server. Connections can only be made to conversational servers, and rpc requests (via `tpacall`(3c) or `tpcall`(3c)) can only be made to non-conversational servers. The default is `N`.

`RQADDR` = *`string_value`*
: specifies the symbolic name of the request queue for *`AOUT`*. It must be 30 characters or less. If not specified, a unique key (`GRPNO.SRVID`) is chosen for a queue that *`AOUT`* accesses. Specifying the same `RQADDR` for more than one server is the way multiple server, single queue (MSSQ) sets are achieved. If

two servers are given an RQADDR with the same queue name, they must be in the same server group.

RQPERM = *number*
: specifies the numeric permissions on the request queue. *number* is specified in the usual UNIX fashion (for example, 0600). If RQPERM is not specified, and a PERM is specified in the RESOURCES section, then that value is used. Otherwise, a value 0666 is used. The value can be between 0001 and 0777, inclusive.

REPLYQ = { Y | N }
: specifies whether a reply queue should be established for the AOUT. If Y is specified, the reply queue is created on the same LMID as the AOUT. The default is N. For servers in an MSSQ set, servers that expect replies should have REPLYQ set to Y.

RPPERM = *number*
: specifies the numeric permissions on the reply queue. number is specified in the usual UNIX fashion (for example, 0600). If RPPERM is not specified, the default value 0666 is used. If requests and replies are both read from the same queue, only RQPERM need be specified; RPPERM is ignored. The value can be between 0001 and 0777, inclusive.

RCMD = *string_value*
: If AOUT is restartable, this parameter specifies the command that should be executed when AOUT abnormally terminates. The string, up to the first space or tab, must be the name of an executable UNIX file, either a full pathname or relative to APPDIR (don't attempt to set a shell variable at the beginning of the command). The command name may be optionally followed by command line arguments. Two additional arguments are appended to the command line: the GRPNO and SRVID associated with the restarting server. string_value is executed in parallel with restarting the server.

MAXGEN = *number*
: If *AOUT* is restartable, this parameter specifies that it can be restarted at most *number* - 1 times within the period specified by GRACE. The value must be greater than 0 and less than 256. If not specified, the default is 1 (which means that the server can be started once, but not restarted).

GRACE = *number*
: If *AOUT* is restartable, this parameter specifies that it can have up to MAXGEN lives within the specified number of seconds. The value must be greater than or equal to 0 and less than 2147483648. If 0, the *AOUT* can be restarted an

unlimited number of times. If GRACE is not specified, the default is 86,400 seconds (24 hours).

RESTART = { Y | N }
: specifies whether or not *AOUT* is restartable. The default is N. If server migration is specified, RESTART must be set to Y. Note that a server terminated with a SIGTERM signal cannot be restarted; it must be rebooted.

SYSTEM_ACCESS *identifier[,identifier]*
: specifies the default mode used by BEA TUXEDO system libraries within application processes to gain access to BEA TUXEDO system's internal tables. Valid access types are FASTPATH or PROTECTED. FASTPATH specifies that the internal tables should be accessible by the libraries via shared memory for fast access. PROTECTED specifies that while the internal tables are accessible by BEA TUXEDO system libraries via shared memory, the shared memory for these tables is not accessible outside of the BEA TUXEDO system libraries. NO_OVERRIDE can be specified (either alone or in conjunction with FASTPATH or PROTECTED) to indicate that the mode selected cannot be overridden by an application process. If SYSTEM_ACCESS is not specified, the default mode is determined by the setting of the SYSTEM_ACCESS keyword in the RESOURCES section.

The SERVICES Section

This section provides information on services used by the application. Lines within the SERVICES section have the form:

SVCNM [optional parameters]

where *SVCNM* is the (*string_value*) name of the service. *SVCNM* must be 15 characters or fewer in length.

There are no required parameters. Services need not be listed if no optional parameters need to be set. Optional parameters are:

LOAD = *number*
: specifies that *SVCNM* imposes a load on the system of *number*. *number* can be between 1 and 32767 inclusive. If not specified, the default is 50. A higher number indicates a greater load.

PRIO = *number*
: specifies that *SVCNM* has a dequeueing priority of the specified number. The value must be greater than 0 and less than or equal to 100, with 100 being the highest priority. The default is 50.

SRVGRP = *string_value*

This parameter says that any parameters specified apply to *SVCNM* within server group *string_value*. The use of SRVGRP allows the same service to have different parameter settings within different server groups. It must be 30 characters or less.

BUFTYPE = "*type1*[*:subtype1[,subtype2 . . .*]][*;type2*[*:subtype3*[, . . .]]] . . .

is a list of types and subtypes of data buffers accepted by this service. This parameter can be up to 256 characters in length and a maximum of 32 type/subtype combinations are allowed. Types of data buffers provided with BEA TUXEDO system are FML (for FML buffers), VIEW, X_C_TYPE, or X_COMMON (for FML views), STRING (for NULL terminated character arrays) and CARRAY or X_OCTET (for a character array that is neither encoded nor decoded during transmission). Of these types, only VIEW, X_C_TYPE, and X_COMMON have subtypes. A view subtype gives the name of the particular view expected by the service. Application types and subtypes can also be added (see tuxtypes(5)). For a TYPE that has subtypes, "*" can be specified for the subtype to indicate that the service accepts all subtypes for the associated type.

A single service can only interpret a fixed number of buffer types, namely those found in its buffer type switch (see tuxtypes(5)). If the BUFTYPE parameter is set to ALL, that service will accept all buffer types found in its buffer type switch. Omitting the BUFTYPE parameter is equivalent to setting it to ALL. If multiple entries exist for the same service name but with different SRVGRP parameters, the BUFTYPE parameter must be the same for all of these entries.

A type name can be 8 characters or less in length and a subtype name can be 16 characters or less in length. Note that type and subtype names should not contain semicolon, colon, comma, or asterisk characters (this will make it hard to see where type and subtype values end).

Some examples of valid BUFTYPE specifications are:

```
BUFTYPE=FML implies that the service takes FML buffers.
BUFTYPE=VIEW:* implies that the service takes all subtypes
of FML views.
```

ROUTING = *string_value*

specifies the name of the routing criteria used for this service when doing data dependent routing. If not specified, data dependent routing is not done for this

service. *string_value* must be 15 characters or less in length. If multiple entries exist for the same service name but with different SRVGRP parameters, the ROUTING parameter must be the same for all of these entries.

SVCTIMEOUT = *number*
: specifies the amount of time, in seconds, that is allowed for processing of the indicated service. The value must be greater than or equal to 0. A value of 0 indicates that the service will not be timed out. A timed-out service will cause the server processing the service request to be terminated with a SIGKILL signal. The default for this parameter is 0.

The following parameters are for DTP applications only:

AUTOTRAN = { Y | N }
: specifies whether or not a transaction should automatically be started if a request message is received that is not already in transaction mode. The default is N.

TRANTIME = *number*
: specifies the default timeout value in seconds for a transaction automatically started for the associated service. The value must be greater than or equal to 0 and less than 2147483648. The default is 30 seconds. A value of 0 implies the maximum timeout value for the machine.

The INTERFACES Section

This section provides information for defining application-wide default parameters for CORBA interfaces used by the application. There are no required parameters for CORBA interfaces unless you are implementing factory-based routing, a BEA WLE feature that allows you to distribute processing to specific server groups. If you are implementing factory-based routing, you must specify the parameters listed in the following table.

Factory-based Routing Parameters

In this section . . .	You must specify . . .
INTERFACES	♦ Names of the interfaces being used ♦ Names of the routing criteria that the BEA WLE system should apply to each interface
ROUTING	Routing criteria
GROUPS	Names of the server groups

For details about factory-based routing and the parameters associated with it, see "The ROUTING Section" after this section.

You do not need to list any CORBA interfaces if you do not want to specify any parameters.

The following optional parameters are available.

`AUTOTRAN = {Y | N }`
: indicates that you want the system to automatically initiate a transaction on every operation invocation and end it upon return from the invocation. The AUTOTRAN parameter is only honored for interfaces that have the `optional` transaction policy. Otherwise, this parameter is ignored. The default is `N`.

  The transactional policy is specified in an implementation configuration file. This transactional policy will become the transactional policy attribute of the associated `T_IFQUEUE MIB` object at run time.

  Before setting the `AUTOTRAN` value, the system administrator must know the value of the transactional policy assigned to the interface by the programmer. Without knowing the policy, the administrator's expectations of run-time `AUTOTRAN` functionality may be wrong.

  If `AUTOTRAN` is set to `Y`, you must also set the `TRANTIME` parameter.

`FACTORYROUTING = criteria-name`
: is required if you want to use a routing criteria when creating object references for this interface. The routing criteria is specified in the `ROUTING` section of the `UBBCONFIG` file.

`LOAD = number`
: is an arbitrary number between 1 and 100 that represents the relative load that the CORBA interface is expected to impose on the system. The numbering scheme is relative to the `LOAD` numbers assigned to other CORBA interfaces used by this application. The default is 50. The value of `LOAD` is used by the BEA WLE system to select the best machine to `enqueue` a request. The routing of the request causes the server's total load to be increased by the `LOAD` factor of the CORBA interface requested.

`PRIO = number`
: specifies the `dequeuing` priority number for all methods of the CORBA interface. The value must be greater than 0 and less than or equal to 100. 100 is the highest priority. The default is 50.

SRVGRP = *server-group-name*
: indicates that any parameter defined in this portion of the INTERFACES section applies to the interface within the specified server group. This feature lets you define, for a given CORBA interface, different parameter values in different server groups.

TRANTIME = *number*
: is the length of the time out (in seconds) for the transactions to be computed. If AUTOTRAN is set to Y, you must set the TRANTIME parameter. The value must be greater than or equal to zero and must not exceed 2,147,483,647 (2^{31} - 1), or about 70 years. A value of 0 implies there is no time out for the transaction. (The default is 30 seconds.)

TIMEOUT = *number*
: indicates the amount of time (in seconds) to allow for processing of a method for this CORBA interface. The value must be greater than or equal to 0. A value of 0 indicates that the interface cannot time out. A timed-out method causes the server processing the method for the interface to terminate with a SIGKILL event. We recommend specifying a time out value for the longest-running method for the interface.

The ROUTING Section

This section provides information for data dependent routing of service requests using FML buffers and views. The routing criteria specified here are used only if the default routing functions, _froute and _vroute, are being used (see tuxtypes(5)). Lines within the ROUTING section have the form:

CRITERION_NAME required parameters

where *CRITERION_NAME* is the (*string_value*) name of the routing entry that was specified on the services entry. *CRITERION_NAME* must be 15 characters or less in length.

Required parameters are:

FIELD = *string_value*
: specifies the name of the routing field. It must be 30 characters or less. This field is assumed to be an FML buffer or view field name that is identified in an FML field table (using the FLDTBLDIR and FIELDTBLS environment variables) or an FML view table (using the VIEWDIR and VIEWFILES environment variables), respectively. This information is used to get the associated field value for data-dependent routing during the sending of a message. If a field in an FML32 buffer will be used for routing, it must have a field number less than or equal to 8191.

RANGES = "*string*"

specifies the ranges and associated server groups for the routing field. *string* must be enclosed in double quotes. *string* can be up to 2048 characters in length (except in Domains, where *string* can be up to 1024 characters). The format of string is a comma-separated ordered list of range/group_name pairs (see "EXAMPLE" below).

A range is either a single value (signed numeric value or character string in single quotes), or a range of the form "lower - upper" (where lower and upper are both signed numeric values or character strings in single quotes). Note that "lower" must be less than or equal to "upper". To embed a single quote in a character string value (as in O'Brien, for example), it must be preceded by two backslashes ('O\\'Brien'). The value MIN can be used to indicate the minimum value for the data type of the associated FIELD on the machine. The value MAX can be used to indicate the maximum value for the data type of the associated FIELD on the machine. Thus, "MIN - -5" is all numbers less than or equal to -5 and "6 - MAX" is all numbers greater than or equal to 6. The meta-character "*" (wild-card) in the position of a range indicates any values not covered by the other ranges previously seen in the entry; only one wild-card range is allowed per entry and it should be last (ranges following it will be ignored).

The routing field can be of any data type supported in FML. A numeric routing field must have numeric range values, and a string routing field must have string range values.

String range values for string, carray, and character field types must be placed inside a pair of single quotes and can not be preceded by a sign. Short and long integer values are strings of digits, optionally preceded by a plus or minus sign. Floating point numbers are of the form accepted by the C compiler or atof: an optional sign, then a string of digits optionally containing a decimal point, then an optional e or E followed by an optional sign or space, followed by an integer.

The group name indicates the associated group to which the request is routed if the field matches the range. A group name of "*" indicates that the request can go to any group where a server offers the desired service.

Within a range/group pair, range is separated from the group name by a ":".

BUFTYPE = "*type1[:subtype1[,subtype2...]][;type2[:subtype3[,...]]]...*"

is a list of types and subtypes of data buffers for which this routing entry is valid. This parameter can be up to 256 characters in length and a maximum

of 32 type/subtype combinations are allowed. The types are restricted to be either FML, VIEW, X_C_TYPE, or X_COMMON. No subtype can be specified for type FML, and subtypes are required for type VIEW, X_C_TYPE, and X_COMMON ("*" is not allowed). Note that subtype names should not contain semicolon, colon, comma, or asterisk characters. Duplicate type/subtype pairs can not be specified for the same routing criteria name; more than one routing entry can have the same criteria name as long as the type/subtype pairs are unique. This parameter is required. If multiple buffer types are specified for a single routing entry, the data types of the routing field for each buffer type must be the same.

An example of a routing entry is:

```
BRNCH FIELD=B_FLD RANGES="0-2:DBG1,3-5:DBG2,6-9:DBG3"
BUFTYPE="FML"
```

which sends buffers with field B_FLD values 0-2 to server group DBG1, values 3-5 to server group DBG2, and values 6-9 to DBG3; no other values are allowed.

If the field value is not set (for FML buffers), or does not match any specific range and a wild-card range has not been specified, an error is returned to the application.

Files

The TUXCONFIG and TUXOFFSET environment variables are used to find the TUXCONFIG configuration file on the MASTER machine.

Example

```
# The following configuration file defines a 2-site
# configuration with two machine types. Data dependent
# routing is used.
*RESOURCES
IPCKEY    80952  # key for well known address
DOMAINID   My_Domain_Name
UID      4196  # user id for ipc structures
GID      601   # group id for ipc structures
PERM     0660  # permissions for ipc access
MAXSERVERS  20   # at most 20 simultaneous servers
MAXSERVICES  40   # offering at most 40 services
MAXGTT    20   # at most 20 simultaneous global transactions
MASTER    SITE1
SCANUNIT   10
SANITYSCAN  12
BBLQUERY   180
BLOCKTIME   30
DBBLWAIT   6
NOTIFY    DIPIN
OPTIONS    LAN,MIGRATE
SECURITY   USER_AUTH
AUTHSVC    AUTHSVC
```

```
     MP  # a multiprocessor based bulletin board
LDBAL     Y  # perform load balancing
#
*MACHINES
mach1     LMID=SITE1 TUXDIR="/usr4/tuxbin"
       MAXACCESSERS=25
       APPDIR="/usr2/apps/bank"
       ENVFILE="/usr2/apps/bank/ENVFILE"
       TLOGDEVICE="/usr2/apps/bank/TLOG" TLOGNAME=TLOG
       TUXCONFIG="/usr2/apps/bank/tuxconfig" TYPE="3B2"
       ULOGPFX="/usr2/apps/bank/ULOG"
       SPINCOUNT=5
mach386    LMID=SITE2 TUXDIR="/usr5/tuxbin"
       MAXACCESSERS=100
       MAXWSCLIENTS=50
       APPDIR="/usr4/apps/bank"
       ENVFILE="/usr4/apps/bank/ENVFILE"
       TLOGDEVICE="/usr4/apps/bank/TLOG" TLOGNAME=TLOG
       TUXCONFIG="/usr4/apps/bank/tuxconfig" TYPE="386"
       ULOGPFX="/usr4/apps/bank/ULOG"
#
*GROUPS

DEFAULT:   TMSNAME=TMS_SQL  TMSCOUNT=2
# For NT/NetWare, :bankdb: becomes ;bankdb;
BANKB1    LMID=SITE1  GRPNO=1
  OPENINFO="TUXEDO/SQL:/usr2/apps/bank/bankdl1:bankdb:readwrite"
# For NT/NetWare, :bankdb: becomes ;bankdb;
BANKB2    LMID=SITE2  GRPNO=2
  OPENINFO="TUXEDO/SQL:/usr4/apps/bank/bankdl2:bankdb:readwrite"
DEFAULT:
AUTHGRP    LMID=SITE1  GRPNO=3
#
*NETWORK
SITE1     NADDR="mach1.80952" BRIDGE="/dev/starlan"
       NLSADDR="mach1.serve"
#
SITE2     NADDR="mach386.80952" BRIDGE="/dev/starlan"
       NLSADDR="mach386.serve"
*SERVERS
#
DEFAULT: RESTART=Y MAXGEN=5 REPLYQ=Y CLOPT="-A"

TLR      SRVGRP=BANKB1  SRVID=1  RQADDR=tlr1
       CLOPT="-A -- -T 100"
```

```
TLR      SRVGRP=BANKB1  SRVID=2  RQADDR=tlr1
       CLOPT="-A -- -T 200"
TLR      SRVGRP=BANKB2  SRVID=3  RQADDR=tlr2
       CLOPT="-A -- -T 600"
TLR      SRVGRP=BANKB2  SRVID=4  RQADDR=tlr2
       CLOPT="-A -- -T 700"
XFER     SRVGRP=BANKB1  SRVID=5
XFER     SRVGRP=BANKB2  SRVID=6
ACCT     SRVGRP=BANKB1  SRVID=7
ACCT     SRVGRP=BANKB2  SRVID=8
BAL      SRVGRP=BANKB1  SRVID=9
BAL      SRVGRP=BANKB2  SRVID=10
BTADD     SRVGRP=BANKB1  SRVID=11
BTADD     SRVGRP=BANKB2  SRVID=12
AUTHSVR    SRVGRP=AUTHGRP SRVID=20
#
*SERVICES
DEFAULT:   LOAD=50     AUTOTRAN=N
WITHDRAWAL  PRIO=50     ROUTING=ACCOUNT_ID
DEPOSIT    PRIO=50     ROUTING=ACCOUNT_ID
TRANSFER   PRIO=50     ROUTING=ACCOUNT_ID
INQUIRY    PRIO=50     ROUTING=ACCOUNT_ID
CLOSE_ACCT  PRIO=40     ROUTING=ACCOUNT_ID
OPEN_ACCT   PRIO=40     ROUTING=BRANCH_ID
BR_ADD    PRIO=20     ROUTING=BRANCH_ID
TLR_ADD    PRIO=20     ROUTING=BRANCH_ID
ABAL     PRIO=30     ROUTING=b_id
TBAL     PRIO=30     ROUTING=b_id
ABAL_BID   PRIO=30     ROUTING=b_id
TBAL_BID   PRIO=30     ROUTING=b_id SVCTIMEOUT=300
#
#
*ROUTING
ACCOUNT_ID  FIELD=ACCOUNT_ID  BUFTYPE="FML"
 RANGES="MIN - 9999:*,10000-59999:BANKB1,60000-109999:BANKB2,*:*"
BRANCH_ID   FIELD=BRANCH_ID   BUFTYPE="FML"
 RANGES="MIN - 0:*,1-5:BANKB1,6-10:BANKB2,*:*"
b_id     FIELD=b_id  BUFTYPE="VIEW:aud"
 RANGES="MIN - 0:*,1-5:BANKB1,6-10:BANKB2,*:*"
```

Interoperability In an interoperating application, the master site must be the latest release available. Parameter values for `PMID` (machine `ADDRESS`), `LMID`, `TLOGNAME`, group names, `RQADDR`, service names, and `ROUTING` (routing criteria names) must be identifiers (valid C identifiers that are not `UBBCONFIG` keywords) when interoperating with BEA TUXEDO system.

Network Addresses

Suppose the local machine on which the bridge is being run is using TCP/IP addressing and is named `backus.company.com`, with address `155.2.193.18`. Further suppose that the port number at which the bridge should accept requests is `2334`. Assume that port number `2334` has been added to the network services database under the name `bankapp-naddr`. The address could be represented in the following ways:

```
//155.2.193.18:bankapp-naddr//155.2.193.18:2334
//backus.company.com:bankapp-naddr
//backus.company.com:2334
0x0002091E9B02C112
```

The last of these representations is hexadecimal format. The `0002` is the first part of a TCP/IP address. The `091E` is the port number `2334` translated into a hexadecimal number. After that each element of the IP address `155.2.193.1` is translated into a hexadecimal number. Thus the `155` becomes `9B`, `2` becomes `02` and so

See Also

`buildserver`(1), `tmadmin`(1), `tmboot`(1), `tmshutdown`(1), `tmloadcf`(1), `tmunloadcf`(1), `tpinit`(3c), `buffer`(3c), , `servopts`(5), *Administering the BEA TUXEDO System*, *BEA TUXEDO Programmer's Guide*

udfk(5)

Name

udfk-description file for form user-defined function keys

```
mio -u file
export UDFK=file
 ...
do_form("formname", NULL)
```

Description

Predefined character sequences constitute mio's function key set. This same set of sequences is used by do_form(3). These sequences can be changed by listing their replacements in the udfk file, and passing the udfk file name to mio in the -u option, or setting the UDFK environment variable for do_form(3).

The format of the udfk file is described and demonstrated below. Blank lines and lines beginning with an asterisk in column one are comments, and are ignored. All other lines consist of two white space separated sequences, optionally followed by a comment. The first sequence or string should be a special keyword describing the command key. The keywords are listed below. The second sequence should consist of a string containing the new character sequence for the command key. Unprintable characters may be represented using an escape sequence. The escape sequence consists of a backslash followed by exactly two hexadecimal digits.

The table below lists the keyword names and their defaults. It is in proper description file format.

```
HP  \01  help
PP  \02  previous page
IC  \03  insert character
QT  \04  quit
NP  \06  Next page
BS  \08  backspace
FF  \09  forward field
DL  \0a  down line
UL  \0b  up line
FS  \0c  forward space
CR  \0d  carriage return
EM  \0e  print error message
BF  \0f  back field
RF  \10  repaint form
HC  \14  home cursor
DC  \15  delete character
F11  \16  leave key
DD  \17  display defaults
CL  \18  clear
PR  \19  print
```

```
ES  \1b!  shell escape
CF  \1b$  delete from cursor to end of field
DF  \1b@  delete all characters from field
F0  \1b0  leave key
F1  \1b1  leave key
F2  \1b2  leave key
F3  \1b3  leave key
F4  \1b4  leave key
F5  \1b5  leave key
F6  \1b6  leave key
F7  \1b7  leave key
F8  \1b8  leave key
F9  \1b9  leave key
```

When the description file contains only a partial list of keywords, keywords not appearing in the file keep their previous values.

Notices

`mio` and `do_form`() perform the first command that matches the user input. Hence, when the character sequence for one command is a leading substring of the sequence for another command, the latter is never performed.

See Also

`mio`(1), `udfk_test`(1), `do_form`(3), *Administering the BEA TUXEDO System*

viewfile(5)

Name

`viewfile`-source file for view descriptions

Description

Viewfiles are source files for descriptions of one or more C data structures, or ‘‘views.” When used as input to the `viewc`(1) command, the viewfile forms the basis for a binary file (filename *`view_filename`*.`V`) and a header file (*`view_filename`*.`h`).

The binary `.V` files are used two ways in the BEA TUXEDO system:

- for programs that use `Fvftos`(3) and `Fvstof`(3), the `.V` file is interpreted at runtime to effect the mapping between FML buffers and C structures
- for programs allocating typed buffers of type `VIEW` the `.V` file is searched for a structure of the name provided in the *`subtype`* argument of `tpalloc`(3)

The `.h` file must be included in all programs using the view so that structure members can be referenced by their logical names.

VIEW Descriptions

Each view description in a source viewfile consists of three parts:

- a line beginning with the keyword “`VIEW`”, followed by the name of the view description; the name can have a maximum of 33 characters and must be a valid C identifier (that is, it must start with an underscore or an alphabetic character and contain only alphanumeric or underscore characters); when used with tpalloc(3), the name can only have a maximum of 16 characters
- a list of member descriptions, each line containing 7 fields
- a line beginning with the keyword “`END`”

The first line of each view description must begin with the keyword “`VIEW`” followed by the name of the view description. A member description (or mapping entry) is a line with information about a member in the C structure. A line with the keyword “`END`” must be the last line in a view description. Lines beginning with a # are treated as comments and ignored.

Thus, a source view description has this general structure:

```
VIEW vname
# type  cname  fbname  count  flag  size  null
# ----  -----  ------  -----  ----  ----  ----
--------------member descriptions-------------------
.
.
```

```
.
END
```

In the view description, the variable fields have the following meaning:

vname
: is the name of the view description, and should be a valid C identifier name, since it is also used as the name of a C structure.

type
: is the type of the member, and is specified as one of the following: int, short, long, char, float, double, string, carray or dec_t; if type is '-', the type of the member is defaulted to the type of *fbname* if the view is mapped to FML buffers.

cname
: is the identifier for the structure member, and should be a valid C identifier name, since it is the name of a C structure member. If the view is mapped to FML buffers, it can not be a valid *fbname*.

fbname
: is the name of the field in the fielded buffer; this name must appear in either a field table file or a field header file. For views not mapped to FML buffers, this field is ignored but must contain a place holder value such as a dash ( ).

count
: is the number of elements to be allocated (that is, the maximum number of occurrences to be stored for this member); must be less than or equal to 65535

flag
: is a list of options, optionally separated by commas, or '-' meaning no options are set; see below for a discussion of *flag* options. For views not mapped to FML buffers, this field may contain the `C` and/or `L` options, or must contain a dash ( ) place holder value

size
: is the size of the member if the type is either string or carray; must be less than or equal to 65535. For 32-bit FML, the max size is 2 to the 32nd or several gazillion. For the dec_t type, *size* is two numbers separated by a comma, the first being the number of bytes in the decimal value (it must be greater than 0 and less than 10) and the second being the number of decimal places to the right of the decimal point (it must be greater than 0 and less than two times the number of bytes minus one). For other field types, '-' should be specified, and the view compiler will compute the size.

null

is the user-specified null value or '-' to indicate the default null value for that field; see below for a discussion of null values.

Flag Options

The following is a list of the options that can be specified as the `flag` element of a member description in a view description. Note that the `L` and `C` options generate additional structure members even for views that are not FML-based.

C

This option specifies that an additional structure member, called the associated count member (ACM), be generated, in addition to the structure member described in the member description (even for views that are not FML-based). When transferring data from a fielded buffer to a structure, each ACM in the structure is set to the number of occurrences transferred to the associated structure member. A value of 0 in an ACM indicates that no fields were transferred to the associated structure member; a positive value indicates the number of fields actually transferred to the structure member array; a negative value indicates that there were more fields in the buffer than could be transferred to the structure member array (the absolute value of the ACM equals the number of fields not transferred to the structure). During a transfer of data from a structure member array to a fielded buffer, the ACM is used to indicate the number of array elements that should be transferred. For example, if a member's ACM is set to N, then the first N non-null fields are transferred to the fielded buffer. If N is greater than the dimension of the array, it then defaults to the dimension of the array. In either event, after the transfer takes place, the ACM is set to the actual number of array members transferred to the fielded buffer. The type of an ACM is declared to be short (32-bit long integer for VIEW32), and its name is generated as "C_*cname*", where *cname* is the `cname` entry for which the ACM is declared. For example, an ACM for a member named `parts` would be declared as follows:

```
short C_parts;
```

It is possible for the generated ACM name to conflict with structure members whose names begin with a "C_" prefix. Such conflicts will be reported by the view compiler, and are considered fatal errors by the compiler. For example, if a structure member has the name "C_parts", it would conflict with the name of an ACM generated for the member "parts". Note also that the view compiler will generate structured record definitions for ACM and ALM (see the L option, below) members when you specify the `-r` command line option.

F

Specifies one-way mapping from structure to fielded buffer (this option is ignored for views that are not FML-based). The mapping of a member with this option is effective only when transferring data from structures to fielded buffers.

L

This option is used only for member descriptions of type carray or string to indicate the number of bytes transferred for these possibly variable length fields. If a string or carray field is always used as a fixed length data item, then this option provides no benefit. The `L` option generates an associated length member (`ALM`) for a structure member of type carray or string (even for views that are not FML-based). When transferring data from a fielded buffer to a structure, the `ALM` is set to the length of the corresponding transferred fields. If a field's length in the fielded buffer exceeds the space allocated in the mapped structure member, only the allocated number of bytes is transferred. The corresponding `ALM` is set to the size of the fielded buffer item. Therefore, if the `ALM` is greater than the dimension of the structure member array, the fielded buffer information was truncated on transfer. When transferring data from a structure member to a field in a fielded buffer, the `ALM` is used to indicate the number of bytes to transfer to the fielded buffer, if it is a carray type field. For strings, the `ALM` is ignored on transfer, but is set afterwards to the number of bytes transferred. Note that since carray fields may be of zero length, an `ALM` of 0 indicates that a zero length field should be transferred to the fielded buffer, unless the value in the associated structure member is the null value. An `ALM` is defined to be an unsigned short (32-bit unsigned long integer for `VIEW32`), and has a generated name of "`L_`*cname*", where *cname* is the name of the structure for which the `ALM` is declared. If the number of occurrences of the member for which the `ALM` is declared is 1 (or defaults to 1), then the `ALM` is declared as:

```
unsigned short L_cname;
```

whereas if the number of occurrences is greater than 1, say N, the ALM is declared as:

```
unsigned short L_cname[N];
```

and is referred to as an `ALM` Array. In this case, each element in the ALM array refers to a corresponding occurrence of the structure member (or field). It is possible for the generated `ALM` name to conflict with structure members whose names begin with a "`L_`" prefix. Such conflicts will be reported by the view compiler, and are considered fatal errors by the compiler. For example,

if a structure member has the name "`L_parts`", it would conflict with the name of an `ALM` generated for the member "parts". Note also that the view compiler will generate structured record definitions for `ACM` and `ALM` (see the C option, above) members when you specify the `-r` command line option.

N

Specifies zero-way mapping, that is, no fielded buffer is mapped to the C structure (this option is ignored for views that are not FML-based). This can be used to allocate fillers in C structures.

P

This option can be used to affect what value is interpreted as a null value for string and carray type structure members (this option is ignored for views that are not FML-based). If this option is not used, a structure member is null if its value is equal to the user-specified null value (without considering any trailing null characters). If this option is set, however, a member is null if its value is equal to the user-specified null value with the last character propagated to full length (without considering any trailing null character). Note that a member whose value is null will not be transferred to the destination buffer when data is transferred from the C structure to the fielded buffer. For example, a structure member TEST is of type carray[25] and a user-specified null value "abcde" is established for it. If the P option is not set, TEST is considered null if the first five characters are a, b, c, d, and e, respectively. If the P option is set, TEST is null if the first four characters are a, b, c, and d, respectively, and the rest of the carray must contain the character 'e' (21 e's).

S

Specifies one-way mapping from fielded buffer to structure (this option is ignored for views that are not FML-based). The mapping of a member with this option is effective only when transferring data from fielded buffers to structures.

Null Values

Null values are used in views to indicate empty C structure members. Default null values are provided, and you may also define your own.

The default null value for all numeric types is 0 (0.0 for dec_t); for char types, it is "\"; and for string and carray types, it is "".

Escape convention constants can also be used to specify a null value. The view compiler recognizes the following escape constants: *ddd* (where *d* is an octal digit), 0, n, t, v, b, r, f, , ', and ".

String, carray, and char null values may be enclosed in double or single quotes. Unescaped quotes within a user-defined null value are not accepted by the view compiler.

Alternatively, an element is null if its value is the same as the null value for that element, except in the following cases:

- if the P option is set for the structure member, and the structure member is of string or carray type; see above for details on the P option flag
- if a member is of type string, its value must be the same string as the null value
- if a member is of type carray, and the null value is of length N, then the first N characters in the carray must be the same as the null value

You can also specify the keyword "NONE" in the null field of a view member description, which means there is no null value for the member.

The maximum size of defaults for string and character array members is 2660 characters.

Note that for string members, which usually end with a "0", a "0" is not required as the last character of a user-defined null value.

Environment Variables

`VIEWFILES`
: should contain a comma separated list of object viewfiles for the application. Files given as full pathnames are used as is; files listed as relative path names are searched for through the list of directories specified by the `VIEWDIR` variable (see below).

`VIEWDIR`
: specifies a colon-separated list of directories where view object files can be found. If `VIEWDIR` is not set, then its value is taken to be the current directory.

For `VIEW32`, the environment variable `VIEWFILES32` and `VIEWDIR32` are used.

Examples

```
# BEGINNING OF AN FML-BASED VIEWFILE
VIEW custdb
$/* This is a comment */
#
#type    cname   fbname count flag  size  null
#
carray  bug   BUG_CURS 4    -   12   "no bugs"
long   custid CUSTID  2    -   -   -1
short   super  SUPER_NUM 1    -   -   999
long   youid  ID    1    -   -   -1
```

```
float   tape  TAPE_SENT 1    -   -   -.001
char   ch   CHR    1    -   -   "0"
string  action ACTION  4    -   20   "no action"
END
# BEGINNING OF AN INDEPENDENT VIEWFILE
VIEW viewx
$ /* View structure for viewx information */
#
#type    cname   fbname count flag  size  null
#
int     in    -   1   -   -   -
short    sh    -   2   -   -   -
long     lo    -   3   -   -   -
char     ch    -   1   -   -   -
float    fl    -   1   -   -   -
double    db    -   1   -   -   -
string    st    -   1   -   15   -
carray    ca    -   1   -   15   -
END
```

See Also `viewc`(1), `tpalloc`(3), `Fvftos`(3), `Fvstof`(3), *BEA TUXEDO FML Programmer's Guide*

WS_MIB(5)

Name

WS_MIB-BEA TUXEDO system Workstation Management Information Base

Synopsis

```
#include <fml32.h>
#include <tpadm.h>
```

Description

The BEA TUXEDO system MIB defines the set of classes through which a Workstation group (one WSL and its associated WSH processes) may be managed.

WS_MIB(5) should be used in combination with the generic MIB reference page MIB(5) to format administrative requests and interpret administrative replies. Requests formatted as described in MIB(5) using classes and attributes described in this reference page may be used to request an administrative service using any one of a number of existing ATMI interfaces in an active application. WS_MIB(5) consists of the following classes:

WS_MIB Classes

Class Name	Attributes
T_WSH	Workstation Handler
T_WSL	Workstation Listener

Each class description section has four subsections:

Overview
: High level description of the attributes associated with the class.

Attribute Table
: A table that lists the name, type, permissions, values and default for each attribute in the class. The format of the attribute table is described below.

Attribute Semantics
: Tells how each attribute should be interpreted.

Limitations
: Limitations in the access to and interpretation of this class.

Attribute Table Format

As described above, each class that is a part of this MIB is defined below in four parts. One of these parts is the attribute table. The attribute table is a one-page reference guide to the attributes within a class and how they may used by administrator's, operator's and general user's to interface with an application. There are five components to each attribute description in the attribute tables; name, type, permissions, values and default. Each of these components is discussed in MIB(5).

TA_FLAGS Values

MIB(5) defines the generic `TA_FLAGS` attribute which is a long valued field containing both generic and component MIB specific flag values. At this time, there are no `WS_MIB`(5) specific flag values defined.

FML32 Field Tables

The field tables for the attributes described in this reference page are found in the file `udataobj/tpadm` relative to the root directory of the BEA TUXEDO system software installed on the system. The directory `${TUXDIR}/udataobj` should be included by the application in the colon separated list specified by the `FLDTBLDIR` environment variable and the field table name `tpadm` should be included in the comma separated list specified by the `FIELDTBLS` environment variable.

Limitations

Access to the header files and field tables for this MIB is being provided only on BEA TUXEDO system 6.0 sites and later, both native and Workstation.

T_WSH Class Definition

Overview

The T_WSH class represents run-time attributes of WSH client processes. These attribute values characterize Workstation statistics specific to a particular WSH client process. This class is linked to the T_WSL class by the common key fields, TA_SRVGRP and TA_SRVID. It is also linked to the T_CLIENT class (see

(5)) by the common key field TA_WSHCLIENTID.

Attribute Table

T_WSL Class Definition attribute Table

Attribute[1]	Type	Permissions	Values	Default
TA_CLIENTID(*)	string	R--R--R--	*string[1..78]*	N/A
TA_WSHCLIENTID(*)	string	R--R--R--	*string[1..78]*	N/A
TA_SRVGRP(*)	string	R--R--R--	*string[1..30]*	N/A
TA_SRVID(*)	long	R--R--R--	1 <= *num* < 30,001	N/A
TA_GRPNO(*)	long	R--R--R--	1 <= *num* < 30,000	N/A
TA_STATE(k)	string	R-XR-XR--	See T_CLIENT Class in TM_MIB(5)	
TA_LMID(*)	string	R--R--R--	*LMID*	N/A
TA_PID(*)	long	R--R--R--	1 <= *num*	N/A
TA_NADDR	string	R--R--R--	*string[1..78]*	N/A
TA_HWCLIENTS	long	R--R--R--	1 <= *num* < 32,767	N/A
TA_MULTIPLEX	long	R--R--R--	1 <= *num* < 32,767	N/A
TA_CURCLIENTS	long	R--R--R--	1 <= *num* < 32,767	N/A
TA_TIMELEFT	long	R--R--R--	0 <= *num*	N/A
TA_ACTIVE	string	R--R--R--	"{Y|N}"	N/A

T_WSL Class Definition attribute Table

Attribute[1]	Type	Permissions	Values	Default
TA_TOTACTTIME	long	R--R--R--	0 <= num	N/A
TA_TOTIDLTIME	long	R--R--R--	0 <= num	N/A
TA_CURWORK	long	R--R--R--	0 <= num	N/A
TA_FLOWCNT	long	R--R--R--	0 <= num	N/A
TA_NUMBLOCKQ	long	R--R--R--	0 <= num	N/A
TA_RCVDBYT	long	R--R--R--	0 <= num	N/A
TA_RCVDNUM	long	R--R--R--	0 <= num	N/A
TA_SENTBYT	long	R--R--R--	0 <= num	N/A
TA_SENTNUM	long	R--R--R--	0 <= num	N/A

(k) - GET key field
(*) - GET/SET key, one or more required for SET operations

[1]All attributes in Class T_WSH are local attributes.

Attribute Semantics

TA_CLIENTID : *string[1..78]*
Client identifier for this WSH. The data in this field should not be interpreted directly by the end user except for equality comparison.

TA_WSHCLIENTID : *string[1..78]*
Client identifier for this WSH. The data in this field should not be interpreted directly by the end user except for equality comparison. This field can be used to link the WSH to its associated Workstation client T_CLIENT objects. This field value is always equal to the value for the TA_CLIENTID attribute for this class.

TA_SRVGRP : *string[1..30]*
Logical name of the server group for the associated WSL.

`TA_SRVID` : 1 = *num* 30,001
: Unique (within the server group) server identification number for the associated WSL.

`TA_STATE` :
: State for the WSH client within the application. Any state defined for the `T_CLIENT` class in `TM_MIB`(5) may be returned or set as indicated on that reference page. State changes to the SUSpended state are transitive to all clients associated with this WSH as is the resetting of a SUSpended WSH to ACTive. Additionally, SUSpended WSH clients will not be assigned any additional incoming clients by the WSL. Note that the state of a WSH client may not be set to `DEAD` when accessing the `T_CLIENT` class; however, the state transition to `DEAD` is allowed via the `T_WSH` class and will result in all connections being handled by the targetted WSH to be dropped abortively.

`TA_LMID` : *LMID*
: Current logical machine on which the WSH is running.

`TA_PID` : 1 = *num*
: Native operating system process identifier for the WSH client. Note that this may not be a unique attribute since clients may be located on different machines allowing for duplication of process identifiers.

`TA_NADDR` : *string[1..78]*
: Network address of workstation handler. Hexadecimal addresses are converted to an ascii format with a leading "0x". TCP/IP addresses are reported in the "//*#.#.#.#:port*" format.

`TA_HWCLIENTS` : 1 = *num* 32,767
: High water number of clients accessing application via this WSH.

`TA_MULTIPLEX` : 1 = *num* 32,767
: Maximum number of clients that may access the application via this WSH.

`TA_CURCLIENTS` : 1 = *num* 32,767
: Current number of clients accessing application via this WSH.

`TA_TIMELEFT` : 0 = *num*
: A non-0 value for this attribute indicates that the WSH has been assigned a newly connecting workstation client that has the indicated amount of time, in seconds, to complete the initialization process with the WSH.

TA_ACTIVE : {Y|N}
: A value of Y indicates that the WSH is currently performing work on behalf of one of its associated workstation clients. A value of N indicates that the WSH is currently waiting for work to perform on behalf of one of its associated workstation clients.

TA_TOTACTTIME : 0 = *num*
: Time, in seconds, that the WSH has been active since it started processing.

TA_TOTIDLTIME : 0 = *num*
: Time, in seconds, that the WSH has been idle since it started processing.

TA_CURWORK : 0 = *num*
: Amount of work processed by this WSH since the last WSH assignment by the WSL. This value is used by the WSL to load balance new incoming connections amongst a set of WSH processes.

TA_FLOWCNT : 0 = *num*
: Number of times flow control has been encountered by this WSH. This attribute should be considered only in relation to recent past values as it may wrap around during the lifetime of the WSH.

TA_NUMBLOCKQ : 0 = *num*
: Number of times this WSH has been unable to enqueue a message to a local UNIX System message queue due to queue blocking conditions. This attribute should be considered only in relation to recent past values as it may wrap around during the lifetime of the WSH.

TA_RCVDBYT : 0 = *num*
: Number of bytes received from the network by this WSH from all of its present and past workstation clients. This attribute should be considered only in relation to recent past values as it may wrap around during the lifetime of the WSH.

TA_RCVDNUM : 0 = *num*
: Number of BEA TUXEDO system messages received from the network by this WSH from all of its present and past workstation clients. This attribute should be considered only in relation to recent past values as it may wrap around during the lifetime of the WSH.

`TA_SENTBYT` : 0 = *num*

Number of bytes sent to the network by this WSH to all of its present and past workstation clients. This attribute should be considered only in relation to recent past values as it may wrap around during the lifetime of the WSH.

`TA_SENTNUM` : 0 = *num*

Number of BEA TUXEDO system messages sent to the network by this WSH to all of its present and past workstation clients. This attribute should be considered only in relation to recent past values as it may wrap around during the lifetime of the WSH.

Limitations

This class represents a specialization of the `T_CLIENT` class and as such represents certain attributes that are duplicated in the corresponding `T_CLIENT` objects. Attributes not listed that are included in the `T_CLIENT` class must be accessed via that class and are not available through the `T_WSH` class.

T_WSL Class Definition

Overview

The T_WSL class represents configuration and run-time attributes of WSL server processes configured to manage Workstation groups. These attribute values identify and characterize Workstation specific configuration attributes for WSL T_SERVER objects within the application. This class is linked to the T_WSH class by the common key fields, TA_SRVGRP and TA_SRVID.

Attribute Table

T_WSL Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_SRVGRP(r)(*)	string	ru-r--r--	*string[1..30]*	N/A
TA_SRVID(r)(*)	long	ru-r--r--	1 = *num* 30,001	N/A
TA_GRPNO(k)	long	r--r--r--	1 = *num* 30,001	N/A
TA_STATE(k)	string	rwxr-xr--	See T_SERVER Class in TM_MIB(5)	
TA_LMID(k)	string	R--R--R--	*LMID*	N/A
TA_PID(k)	long	R--R--R--	1 = *num*	N/A
TA_DEVICE	string	rw-r--r--	*string[0..78]*	N/A
TA_NADDR(r)	string	rw-r--r--	*string[1..78]*	N/A
TA_EXT_NADDR	string	rw-r--r--	*string[0..78]*	""
TA_WSHNAME	string	rw-r--r--	*string[1..78]*	"WSH"
TA_MINHANDLERS	long	rwxr-xr--	0 = *num* 256	0
TA_MAXHANDLERS	long	rw-r--r--	0 = *num* 32,767	See note 1.
TA_MULTIPLEX	long	rw-r--r--	1 = *num* 32,767	10
TA_MINENCRYPTBIT	string	rwxrwx---	{ 0 | 40 | 128 }	0
TA_MAXENCRYPTBIT	string	rwxrwx---	{ 0 | 40 | 128 }	128
TA_MINWSHPORT	long	rwxr-xr--	0 = *num* 65,535	2048
TA_MAXWSHPORT	long	rw-r--r--	0 = *num* 65,535	65,535.
TA_MAXIDLETIME	long	rwxr-xr--	0 = *num* 35,204,650	35,204,649
TA_MAXINITTIME	long	rwxr-xr--	1 = *num* 32,767	60
TA_CMPLIMIT	string	rwxr-xr--	*threshold*	MAXLONG

T_WSL Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_CLOPT	string	rwxr--r--	*string[0..128]*	"-A"
TA_ENVFILE	string	rwxr--r--	*string[0..78]*	""
TA_GRACE	long	rwxr--r--	0 = *num*	0
TA_KEEPALIVE	string	rwxr-xr--	"{client | handler | both | none}"	"none"
TA_MAXGEN	long	rwxr--r--	0 = *num* 256	1
TA_NETTIMEOUT	long	rwxr-xr--	*0 <= num <= MAXLONG*	0
TA_RCMD	string	rwxr--r--	*string[0..78]*	""
TA_RESTART	string	rwxr--r--	"{Y|N}"	"Y"
TA_SEQUENCE(k)	long	rwxr--r--	1 = *num* 10,000	>= 10,000
T_WSL Class: Local Attributes				
TA_CURHANDLERS	long	R--R--R--	0 = *num*	N/A
TA_HWHANDLERS	long	R--R--R--	0 = *num*	N/A
TA_WSPROTO	long	R--R--R--	0 = *num*	N/A
TA_SUSPENDED	string	R-XR-XR--	"{NEW|ALL|NONE}"	N/A
TA_VIEWREFRESH	string	--X--X---	"Y"	N/A
(k) - GET key field (r) - Required field for object creation (SET TA_STATE NEW) (*) - GET/SET key, one or more required for SET operations				

[1]If a value for this attribute is not specified at the time the object is created, then a value of 0 will be assigned. A value of 0 for this attribute indicates that the effective value is determined at activation time from the current setting for TA_MAXHANDLERS and the T_MACHINE class setting for TA_MAXWSCLIENTS. Note that a GET operation with the MIB_LOCAL flag set will get the effective value for objects with an activation time default setting.

Attribute Semantics

TA_SRVGRP: *string[1..30]*

Logical name of the server group. Server group names cannot contain an asterisk (*), comma, or colon.

TA_SRVID: 1 = *num* 30,001

Unique (within the server group) server identification number.

TA_GRPNO: 1 = *num* 30,000

Group number associated with this servers group.

TA_STATE:

State for the WSL server within the application. Any state defined for the T_SERVER class in TM_MIB(5) may be returned or set as indicated on that reference page.

TA_LMID: *LMID*

Current logical machine on which the server is running.

TA_PID: 1 = *num*

Native operating system process identifier for the WSL server. Note that this may not be a unique attribute since servers may be located on different machines allowing for duplication of process identifiers.

TA_DEVICE: *string[0..78]*

Device name to be used by the WSL process to access the network. This attribute is optional.

TA_NADDR: *string[1..78]*

Specifies the complete network address to be used by the WSL process as its listening address. The listening address for a WSL is the means by which it is contacted by workstation client processes participating in the application. If *string* has the form "0xhex-digits" or "\\xhex-digits", it must contain an even number of valid hex digits. These forms are translated internally into a character array containing TCP/IP addresses may also be in either of the following two forms:

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

```
//#.#.#.#:port_number
```

In the first of these formats, *hostname* is resolved to a TCP/IP host address at the time the address is bound using the locally configured name resolution facilities accessed via gethostbyname(3c). The *#.#.#.#* is the dotted decimal format where each # represents a decimal number in the range 0 to 255.

Port_number is a decimal number in the range 0 to 65535. the hexadecimal representations of the string specified.

TA_EXT_NADDR: *string[0..78]*

Specifies the complete network address to be used as a well known address template of the WSH process. The address will be combined with a WSH network address to generate a well known network address used by the workstation client to connect to a WSH process. It has the same format as the TA_NADDR except that it substitutes the port number with same length of character M to indicate the position of the combined network address will be copied from the WSH network address. For example when Address template is 0x0002MMMMdddddddd and WSH network address is 0x00021111ffffffff then the well known network address will be 0x00021111dddddddd. When address template starts with "//" network address type assumes to be IP based and the TCP/IP port number of WSH network address will be copied into the address template to form the combined network address. This feature is useful when workstation client needs to connect to a WSH through a router which performs Network Address Translation. Empty TA_EXT_NADDR string in a SET operation on an existing T_WSL object will eliminate the -H entry from the TA_CLOPT attribute.

TA_WSHNAME: *string[1..78]*

The name of the executable providing workstation handler services for this workstation listener. The default for this is WSH which corresponds to the system provided workstation handler. Workstation handlers may be customized using the command buildwsh(1). See the Customization section and the buildwsh(1) reference page for more details.

TA_MINHANDLERS: 0 = *num* 256

The minimum number of handlers that should be available in conjunction with this WSL at any given time. The WSL will start this many WSHs immediately upon being activated and will not deplete the supply of WSHs below this number until the administrator issues a shutdown to the WSL. Modifications to this attribute for a running WSL may cause additional handlers to be activated.

TA_MAXHANDLERS: 0 = *num* 32,767

The maximum number of handlers that should be available in conjunction with this WSL at any given time. Handlers are started as necessary to meet the demand of workstation clients attempting to access the system. This

attribute must be greater than or equal to the setting for the minumum number of handlers.

TA_MULTIPLEX: 1 = *num* 32,767
: Maximum number of clients that are to be supported by any one handler process concurrently.

TA_MINENCRYPTBITS={0|40|128}
: When connecting to the BEA TUXEDO system, 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 is "0".

TA_ MAXENCRYPTBITS={0|40|128}
: When connecting to the BEA TUXEDO system, negotiate encryption up to this level. "0" means no encryption, whilc "40" and "128" specify the encryption length (in bits). The default is "128"

TA_MINWSHPORT: 0 = *num* 65,535
: The lower end of the range of available port numbers that may be allocated to WSH processes by this listener.

TA_MAXWSHPORT: 0 = *num* 65,535
: The upper end of the range of available port numbers that may be allocated to WSH processes by this listener.

TA_MAXIDLETIME: 0 = *num* 35,204,650
: Maximum amount of time, in minutes, that a workstation client is permitted to be idle before it will be abortively disconnected from the application by the handler. A value of 35,204,650 allows clients to be idle as long as they wish without being timed out. A value of 0 indicates clients may be terminated after any period of inactivity greater than 1 second.

TA_MAXINITTIME: 1 = *num* 32,767
: The minimum number of seconds that should be allowed for a workstation client to complete initialization processing through the WSH before being timed out by the WSL.

TA_CMPLIMIT: *threshold*
: Threshold message size at which compression will occur for traffic to and from workstation clients. >*threshold* may be either non-negative numeric values of the string "MAXLONG," which is dynamically translated to the maximum long setting for the machine. Limitation: This attribute value is not

used for workstation clients running BEA TUXEDO system Workstation Release 6.1 or earlier.

TA_CLOPT: *string[0..128]*

Command line options to be passed to WSL server when it is activated. See the servopts(5) reference page for details. Limitations: Run-time modifications to this attribute will not affect a running WSL server. Server specific options (i.e., those after a double-dash "--") may not be set and will not be returned.

TA_ENVFILE: *string[0..78]*

WSL server specific environment file. See T_MACHINE:TA_ENVFILE for a complete discussion of how this file is used to modify the environment. Limitation: Run-time modifications to this attribute will not affect a running WSL server.

TA_GRACE: 0 = *num*

The period of time, in seconds, over which the T_WSL:TA_MAXGEN limit applies. This attribute is meaningful only for restartable WSL servers, i.e., if the T_WSL:TA_RESTART attribute is set to "Y". When a restarting server would exceed the TA_MAXGEN limit but the TA_GRACE period has expired, the system resets the current generation (T_SERVER:TA_GENERATION) to 1 and resets the initial boot time (T_SERVER:TA_TIMESTART) to the current time. A value of 0 for this attribute indicates that the WSL server should always be restarted.

TA_KEEPALIVE:{client | handler | both | none}

Here you can turn on the network keep-alive operation for the client, the handler, or both. You may also turn off this operation for both the client and handler by specifying none.

Changes to the value of this attribute affect only new connections.

TA_MAXGEN: 1 = *num* 256

Number of generations allowed for a restartable WSL server (T_WSL:TA_RESTART == "Y") over the specified grace period (T_WSL:TA_GRACE). The initial activation of the WSL server counts as one generation and each restart also counts as one. Processing after the maximum generations is exceeded is discussed above with respect to TA_GRACE.

TA_NETTIMEOUT:0 <= *num* <= MAXLONG

The value of TA_NETTIMEOUT is the minimum number of seconds that a workstation client is allowed to wait to receive a response from the WSL/WSH. A value of 0 indicates no network time-out.

Changes to the value of this attribute affect only new connections.

TA_RCMD: *string[0..78]*

Application specified command to be executed in parallel with the system restart of an application server. This command must be an executable file in the native operating system.

TA_RESTART: {Y|N}

Restartable ("Y") or non-restartable ("N") WSL server. If server migration is specified for this server group (T_RESOURCE:TA_OPTIONS/MIGRATE T_GROUP:TA_LMID w/ alternate site), then this attribute must be set to "Y".

TA_SEQUENCE: 1 = *num* 10,000

Specifies when this server should be booted (tmboot(1)) or shutdown (tmshutdown(1)) relative to other servers. T_WSL objects added without a TA_SEQUENCE attribute specified or with an invalid value will have one generated for them that is 10,000 or more and is higher than any other automatically selected default. Servers are booted by tmboot(1) in increasing order of sequence number and shutdown by tmshutdown(1) in decreasing order. Run-time modifications to this attribute affect only tmboot(1) and tmshutdown(1) and will affect the order in which running servers may be shutdown by a subsequent invocation of tmshutdown(1).

TA_CURHANDLERS: 0 = *num*

Number of currently active handlers associated with this WSL.

TA_HWHANDLERS: 0 = *num*

Maximum number of currently active handlers associated with this WSL at any one time.

TA_WSPROTO: 0 = *num*

The BEA TUXEDO system Workstation protocol version number for this Workstation group. Note that Workstation clients connecting to this group may themselves have a different protocol version number associated with them.

TA_SUSPENDED: {NEW|ALL|NONE}

A value of NEW indicates that new incoming clients may not connect through this WSL object. A value of ALL indicates that workstation clients already connected to the application through this WSL have been suspended (see TM_MIB(5)) in addition to disallowing new incoming connections. A value of NONE indicates that no suspension characteristics are in effect.

TA_VIEWREFRESH: Y
: Setting a value of Y will cause all active WSHs in the Workstation group to refresh their VIEW buffer type cache.

Limitations

This class represents a specialization of the T_SERVER class and as such represents certain attributes that are duplicated in the corresponding T_SERVER objects. Attributes not listed that are included in the T_SERVER class must be accessed via that class and are not available through the T_WSL class.

Diagnostics

There are two general types of errors that may be returned to the user when interfacing with WS_MIB(5). First, any of the three ATMI verbs (tpcall(3), tpgetrply(3) and tpdequeue(3)) used to retrieve responses to administrative requests may return any error defined for them. These errors should be interpreted as described on the appropriate reference pages.

If, however, the request is successfully routed to a system service capable of satisfying the request and that service determines that there is a problem handling the request, then failure may be returned in the form of an application level service failure. In these cases, tpcall(3) and tpgetrply(3) will return an error with tperrno set to TPESVCFAIL and return a reply message containing the original request along with TA_ERROR, TA_STATUS and TA_BADFLD fields further qualifying the error as described below. When a service failure occurs for a request forwarded to the system through the TMQFORWARD(5) server, the failure reply message will be enqueued to the failure queue identified on the original request (assuming the -d option was specified for TMQFORWARD).

When a service failure occurs during processing of an administrative request, the FML32 field TA_STATUS is set to a textual description of the failure, the FML32 field TA_ERROR is set to indicate the cause of the failure as indicated below. All error codes specified below are guaranteed to be negative.

[other]
: Other error return codes generic to any component MIB are specified in the MIB(5) reference page. These error codes are guaranteed to be mutually exclusive with any WS_MIB(5) specific error codes defined here.

The following diagnostic codes are returned in TA_ERROR to indicate successful completion of an administrative request. These codes are guaranteed to be non-negative.

[other]

Other return codes generic to any component MIB are specified in the MIB(5) reference page. These return codes are guaranteed to be mutually exclusive with any WS_MIB(5) specific return codes defined here.

Interoperability

The header files and field tables defined in this reference page are available on BEA TUXEDO system release 5.0 and later. Fields defined in these headers and tables will not be changed from release to release. New fields may be added which are not defined on the older release site. Access to the /AdminAPI is available from any site with the header files and field tables necessary to build a request. The T_WSL and T_WSH classes are new with BEA TUXEDO system release 6.0; therefore, local administration of WSL and WSH processes on earlier release sites via the /AdminAPI is not available. However, many of the administrative actions defined in this reference page are available for pre-release 6.0 sites if they are interoperating with a release 6.0 site. If sites of differing releases, both greater than or equal to release 6.0, are interoperating, then information on the older site is available for access and update as defined in the MIB reference page for that release and may be a subset of the information available in the later release.

Portability

The existing FML32 and ATMI functions necessary to support administrative interaction with BEA TUXEDO system MIBs, as well as the header file and field table defined in this reference page, are available on all supported native and workstation platforms.

Example

Following is a sequence of code fragments that deactivate a Workstation group in an orderly fashion using a combination of TM_MIB(5) and WS_MIB(5).

Field Tables

The field table *tpadm* must be available in the environment to have access to attribute field identifiers. This can be done at the shell level as follows:

```
$ FIELDTBLS=tpadm
$ FLDTBLDIR=${TUXDIR}/udataobj
$ export FIELDTBLS FLDTBLDIR
```

Header Files

The following header files are included.

```
#include <atmi.h>
#include <fml32.h>
#include <tpadm.h>
```

Suspend Workstation Group

The following code fragment sets the state of the Workstation group to SUSpended. This disables the Workstation group from accepting new connections from workstation clients and suspends all workstation clients that are currently part of the

group. This code fragment and those that follow assume that the local variables ta_srvgrp and ta_srvid are already set to identify the Workstation group with which we are working.

```
/* Allocate input and output buffers */ ibuf = tpalloc("FML32",
NULL, 1000);
 obuf = tpalloc("FML32", NULL, 1000);
 /* Set MIB(5) attributes defining request type */
 Fchg32(ibuf, TA_OPERATION, 0, "SET", 0);
 Fchg32(ibuf, TA_CLASS, 0, "T_WSL", 0);
 /* Set WS_MIB(5) attributes */
 Fchg32(ibuf, TA_SRVGRP, 0, ta_srvgrp, 0);
 Fchg32(ibuf, TA_SRVID, 0, (char *)ta_srvid, 0);
 Fchg32(ibuf, TA_SUSPENDED, 0, "ALL", 0);
 /* Make the request */
 if (tpcall(".TMIB", (char *)ibuf, 0, (char **)obuf, olen, 0) 0) {
 fprintf(stderr, "tpcall failed: %s\en", tpstrerror(tperrno));
 if (tperrno == TPESVCFAIL) {
 Fget32(obuf, TA_ERROR, 0,(char *)ta_error, NULL);
 ta_status = Ffind32(obuf, TA_STATUS, 0, NULL);
 fprintf(stderr, "Failure: %ld, %s\en",
 ta_error, ta_status);
 }
 /* Additional error case processing */
 }
 /* Copy the logical machine identifier for later use */
 strcpy(ta_lmid, Ffind32(obuf, TA_LMID, 0, NULL));
```

Get List of WSH Objects

Using the existing input buffer, simply change the class and operation and make a new request. We'll retrieve all T_WSH objects associated with the given T_WSL object key fields, ta_srvgrp and ta_srvid. Set the TA_FILTER attribute to limit the retrieval for efficiency.

```
/* Set MIB(5) attributes defining request type */ Fchg32(ibuf,
TA_CLASS, 0, "T_WSH", 0);
 Fchg32(ibuf, TA_OPERATION, 0, "GET", 0);
 longval = TA_WSHCLIENTID;
 Fchg32(ibuf, TA_FILTER, 0, (char *)longval, 0);
 /* Set WS_MIB(5) attributes */
 Fchg32(ibuf, TA_LMID, 0, ta_lmid, 0);
 /* Allocate a separate output buffer to save the TA_WSHCLIENTID
values */
 wshcltids = tpalloc("FML32", NULL, 1000);
 /* Make the request */
 tpcall(".TMIB", (char *)ibuf, 0, (char **)wshcltids, olen, 0);
 /* See how many we got */
 Fget32(wshcltids, TA_OCCURS, 0,(char *)wshcltcnt, NULL);
```

Get T_CLIENT Objects

Use the retrieved TA_WSHCLIENTID values to get a list of associated TA_CLIENTID values for workstation clients in this Workstation group.

```
/* Initialize request buffer */ Finit32(ibuf, Fsizeof32(ibuf));
 /* Set MIB(5) attributes defining request type */
 Fchg32(ibuf, TA_OPERATION, 0, "GET", 0);
 Fchg32(ibuf, TA_CLASS, 0, "T_CLIENT", 0);
 longval = TA_CLIENTID;
 Fchg32(ibuf, TA_FILTER, 0, (char *)longval, 0);
 longval = TA_WSHCLIENTID;
 Fchg32(ibuf, TA_FILTER, 1, (char *)longval, 0);
 /* Set WS_MIB(5) attributes */
 Fchg32(ibuf, TA_LMID, 0, ta_lmid, 0);
 Fchg32(ibuf, TA_WSC, 0, "Y", 0);
 if (wshcltcnt == 1) {
 /* Since only 1, use it as key field. */
 Fchg32(ibuf, TA_WSHCLIENTID, 0,
 Ffind32(wshcltids, TA_WSHCLIENTID, 0, NULL));
 }
 /* Allocate output buffer to save TA_CLIENTID/TA_WSHCLIENTID
values */
 cltids = tpalloc("FML32", NULL, 1000);
 /* Make the request */
 tpcall(".TMIB", (char *)ibuf, 0, (char **)cltids, olen, 0);
 /* See how many we got */
 Fget32(cltids, TA_OCCURS, 0,(char *)cltcnt, NULL);
 /* Eliminate unassociated clients if necessary */
 if (wshcltcnt > 1) {
 for (i=(cltcnt-1); i >= 0 ;i--) {
 p = Ffind32(cltids, TA_WSHCLIENTID, i, NULL);
 for (j=0; j wshcltcnt ;j++) {
 q = Ffind32(wshcltids, TA_WSHCLIENTID, j, NULL);
 if (strcmp(p, q) == 0) {
 break; /* This client is in our group */
 }
 }
 if (j >= wshcltcnt) {
 /* Client not found, delete it from list */
 Fdel32(cltids, TA_CLIENTID, i);
 Fdel32(cltids, TA_WSHCLIENTID, i);
 cltcnt--;
 }
 }
 }
```

Notify T_CLIENT Objects

Use the retrieved TA_CLIENTID values to notify workstation clients in this Workstation group that they should logoff.

```
notstr = tpalloc("STRING", NULL, 100);
(void)strcpy(notstr, "Please logoff now!");

/* Now loop through affected clients and suspend/notify them */
for (i=0; i cltcnt ;i++) {
  p = Ffind32(cltids, TA_CLIENTID, i, NULL);

  /* Notify the client to logoff */
  tpconvert(p, (char *)ci, TPCONVCLTID);
  tpnotify(ci, notptr, 0, 0);
}
```

Deactivate Remaining T_CLIENT Objects

Use the retrieved TA_CLIENTID values to deactivate any remaining workstation clients in this Workstation group. Note that those that are already gone will return an error on the SET that we will ignore.

```
/* Initialize request buffer */
Finit32(ibuf, Fsizeof32(ibuf));
/* Set MIB(5) attributes defining request type */
Fchg32(ibuf, TA_OPERATION, 0, "SET", 0);
Fchg32(ibuf, TA_CLASS, 0, "T_CLIENT", 0);
Fchg32(ibuf, TA_STATE, 0, "DEAd", 0);

/* Now loop through affected clients and deactivate them */
for (i=0; i cltcnt ;i++) {
  p = Ffind32(cltids, TA_CLIENTID, i, NULL);
  Fchg32(ibuf, TA_CLIENTID, 0, p);

  /* Make the request */
  tpcall(".TMIB", (char *)ibuf, 0, (char **)obuf, olen, 0);
}
```

Deactivate T_WSL Object

Now deactivate the T_WSL object. This will automatically deactivate any associated active T_WSH objects.

```
/* Set MIB(5) attributes defining request type */
Fchg32(ibuf, TA_OPERATION, 0, "SET", 0);
Fchg32(ibuf, TA_CLASS, 0, "T_WSL", 0);
Fchg32(ibuf, TA_STATE, 0, "INActive", 0);

/* Set WS_MIB(5) attributes */
Fchg32(ibuf, TA_SRVGRP, 0, ta_srvgrp, 0);
Fchg32(ibuf, TA_SRVID, 0, (char *)ta_srvid, 0);

/* Make the request */
tpcall(".TMIB", (char *)ibuf, 0, (char **)obuf, olen, 0);
}
```

Files

${TUXDIR}/include/tpadm.h, ${TUXDIR}/udataobj/tpadm

See Also `tpacall`(3c), `tpalloc`(3c), `tpcall`(3c), `tpdequeue`(3c), `tpenqueue`(3c), `tpgetrply`(3c), `tprealloc`(3c), `Fadd32`(3fml), `Fchg32`(3fml), `Ffind32`(3fml), `Fintro`(3fml), `MIB`(5), `TM_MIB`(5), *Administering the BEA TUXEDO System*, *BEA TUXEDO Programmer's Guide*

WSL(5)

Name WSL-BEA TUXEDO system Workstation Listener Server

Synopsis

```
WSL SRVGRP="identifier"
  SRVID="number"
  CLOPT="[ -A ] [ servopts options ] -- -n netaddr [ -d device ]
      [ -w WSHname ] [ -t timeout-factor ] [ -T Client-timeout ]
      [ -m minh ] [ -M maxh ] [ -x mpx-factor ]
      [ -p minwshport ] [ -P maxwshport ] [ -I init-timeout ]
      [-c compression-threshold] [-k compression-threshold]
      [-K {client|handler|both|none}]
      [ -z bits ] [ -Z bits ] [-H external netaddr ]"
```

Description The workstation listener is a BEA TUXEDO system-supplied server that enables access to native services by workstation clients. The application administrator enables workstation access to the application by specifying the workstation listener server as an application server in the SERVERS section. The associated command line options are used to parameterize the processing of the workstation listener and workstation handlers.

The location, server group, server ID, and other generic server related parameters are associated with the workstation listener using the already defined configuration file mechanisms for servers. Workstation listener specific command line options are specified to allow for customization.

Each WSL booted as part of an application facilitates application access for a large number of workstation clients by providing access via a single well known network address to a set of workstation handlers (WSHs) acting as surrogate clients for the users running on the workstations. The WSHs are started and stopped dynamically by the WSL as necessary to meet the incoming load from the application workstations. The advantages to the application administrator are that a small number of native site processes (WSHs) can support a much larger number of clients, thus reducing the process count on the native site, and that the native site does not need to incur the overhead of maintaining bulletin board information on the workstation sites, which may be quite numerous.

The following WSL-specific command line options are available and may be listed after the double-dash (--) in the CLOPT parameter.

-n *netaddr*

Specifies the complete network address to be used by the WSL process as its listening address. The listening address for a WSL is the means by which it is contacted by workstation client processes participating in the application. If *string* (which can be from 1 to 78 characters in length) has the form `0xhex-digits` or `\\xhex-digits`, it must contain an even number of valid hex digits. These forms are translated internally into a character array containing TCP/IP addresses may also be in either of the following two forms:

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

The *#.#.#.#* is the dotted decimal format where each # represents a decimal number in the range 0 to 255. *Port_number* is a decimal number in the range 0 to 65535. the hexadecimal representations of the string specified. This is the only required parameter.

[-d *device*]

The name of the device file used for network access by the workstation listener and its workstation handlers. This parameter is optional. There is no default.

[-w *WSHname*]

The name of the executable providing workstation handler services for this workstation listener. The default for this is WSH, which corresponds to the system provided workstation handler. Workstation handlers may be customized using the command buildwsh(1). See the `buildwsh`(1) reference page for more details.

[-t *timeout-factor*]

This option is being replaced by the -I option and is being supported for upward compatibility in release 6.0 but may be removed in future releases. The number, when multiplied by `SCANUNIT`, results in the amount of time in seconds that should be allowed for a workstation client to complete initialization processing through the WSH before being timed out by the WSL. The default for this parameter is 3 in a non-security application and 6 in a security application. The legal range is between 1 and 255.

[-T *client-timeout*]

Client-timeout is the amount of time (in minutes) a client is allowed to stay idle. If a client does not make any requests within this time period, the WSH disconnects the client. The option can be used for client platforms that are

unstable (such as a personal computer that might be turned off without calling `tpterm()`). Note that the option also affects clients that get unsolicited message notifications and do not follow up on them. If `-T` is specified without an argument, there is no timeout.

[`-m` *minh*]

The minimum number of handlers that should be available in conjunction with this WSL at any given time. The WSL will start this many WSHs immediately upon being booted and will not deplete the supply of WSHs below this number until the administrator issues a shutdown to the WSL. The default for this parameter is 0. The legal range is between 0 and 255.

[`-M` *maxh*]

The maximum number of handlers that should be available in conjunction with this WSL at any given time. Handlers are started as necessary to meet the demand of workstation clients attempting to access the system. The default for this parameter is equal to the setting for `MAXWSCLIENTS` on the logical machine divided by the multiplexing factor for this WSL (see `-x` option below) rounded up by one. The legal range for this parameter is between 1 and 4096. The value must be equal to or greater than *minh.*

[`-x` *mpx-factor*]

An optional parameter used to control the degree of multiplexing desired within each workstation handler. The value for this parameter indicates the number of workstation clients that can be supported simultaneously by each workstation handler. The workstation listener ensures that new handlers are started as necessary to handle new workstation clients. This value must be greater than or equal to 1 and less than or equal to 4096. The default for this parameter is 10.

[`-p` *minwshport*]

[`-P` *maxwshport*] This pair of command line options can be used to specify the number range for port numbers available for use by WSHs associated with this listener server. The port numbers must be in the range between 0 and 65535. The default is 2048 for *minwshport* and 65535 for *maxwshport*.

[`-I` *init-timeout*]

This option is replacing the -t option and is the recommended method for setting client initialization timeout intervals. The time, in seconds that should be allowed for a workstation client to complete initialization processing through the WSH before being timed out by the WSL. The default for this parameter is 60. The legal range is between 1 and 32,767.

[-c *compression-threshold*]

This option determines the compression threshold to be used by workstation clients and handlers. Any buffers sent between workstation clients and handlers will be compressed if they are larger than the given value. The default for this parameter is 2,147,483,647, which means no compression is done since the legal range is between 0 and 2,147,483,647.

[-k *compression-threshold*]

This is a special compression option for BEA TUXEDO releases prior to 6.2 with clients from USL France or ITI. If this situation applies to you, it is acceptable to have multiple WSLWorkstationH pairs, some controlling compression threshold with the `-c` option, others using the `-k` option. The `-k` works exactly like `-c`.

`[-K { client | handler | both | none }]`

The `-K` option turns on the network keep-alive feature for the `client`, the `handler`, or `both`. You can turn off this option for both the client and handler by specifying `none`.

`[-z [0|40|128]]`

When establishing a network link between a Workstation client and the Workstation Handler, require at least this minimum level of encryption. `0` means no encryption, while `40` and `128` specify the length (in bits) of the encryption key. If this minimum level of encryption cannot be met, link establishment will fail. The default is `0`.

`[-Z [0|40|128]]`

When establishing a network link between a Workstation client and the Workstation Handler, allow encryption up to this level. `0` means no encryption, while `40` and `128` specify the length (in bits) of the encryption key. The default is `28`. The `-z` or `-Z` options are available only if either the International or Domestic BEA TUXEDO Security Add-on Package is installed.

[-H *external netadder*]

Specifies the complete network address to be used as a well known address template of the WSH process. The address will be combined with a WSH network address to generate a well known network address used by the workstation client to connect to a WSH process. It has the same format as the -n option except that it substitutes the port number with same length of character M to indicate the position of the combined network address will be copied from the WSH network address. For example when address template

is 0x0002MMMMddddddddd and WSH network address is 0x00021111ffffffff then the well known network address will be 0x00021111dddd dddd. When address template starts with "//" network address type assumes to be IP based and the TCP/IP port number of WSH network address will be copied into the address template to form the combined network address. This feature is useful when workstation client needs to connect to a WSH through a router which performs Network Address Translation.

Any configuration that prevents the WSL from supporting workstation clients will cause the WSL to fail at boot time, for example, if the `MAXWSCLIENTS` value for the site is 0.

Portability `WSL` is supported as a BEA TUXEDO-supplied server on UNIX operating systems.

Interoperability `WSL` may be run in an interoperating application, but it must run on a Release 4.2 or later node.

Examples

```
*SERVERS
WSL SRVGRP="WSLGRP" SRVID=1000 RESTART=Y GRACE=0
  CLOPT="-A -- -n 0x0002ffffaaaaaaaa -d /dev/tcp"
WSL SRVGRP="WSLGRP" SRVID=1001 RESTART=Y GRACE=0
  CLOPT="-A -- -n 0x0002aaaaffffffff -d /dev/tcp -H 0x0002MMMMddddddddd"
WSL SRVGRP="WSLGRP" SRVID=1002 RESTART=Y GRACE=0
  CLOPT="-A -- -n //hostname:aaaa -d /dev/tcp -H //external_hostname:MMMM"
```

See Also `buildwsh`(1), `servopts`(5), `ubbconfig`(5), *Administering the BEA TUXEDO System*, *BEA TUXEDO Programmer's Guide*