



BEA Tuxedo

File Formats and Data Descriptions

BEA Tuxedo Release 7.1
Document Edition 7.1
May 2000

Copyright

Copyright © 2000 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

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

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

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

Trademarks or Service Marks

BEA, ObjectBroker, TOP END, and Tuxedo are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Connect, BEA Manager, BEA MessageQ, BEA Jolt, M3, eSolutions, eLink, WebLogic, and WebLogic Enterprise are trademarks of BEA Systems, Inc.

All other company names may be trademarks of the respective companies with which they are associated.

BEA Tuxedo File Formats and Data Descriptions

Document Edition	Date	Software Version
7.1	May 2000	BEA Tuxedo Release 7.1

Contents

Section 5 - File Formats and Data Descriptions

Introduction to Tables and Files	3
ACL_MIB(5)	4
T_ACLGROUP Class Definition	6
T_ACLPERM Class Definition	8
T_ACLPRINCIPAL Class Definition	10
APPQ_MIB(5)	15
T_APPQ Class Definition	18
T_APPQMSG Class Definition	26
T_APPQSPACE Class Definition	32
T_APPQTRANS Class Definition	45
AUTHSVR(5)	52
SECURITY USER_AUTH	53
SECURITY ACL or MANDATORY_ACL	55
compilation(5)	57
DMADM(5)	62
DMCONFIG(5)	64
DM_LOCAL_DOMAINS Section	67
DM_REMOTE_DOMAINS Section	72
DM_TDOMAIN Section	75
DM_ACCESS_CONTROL Section	78
DM_LOCAL_SERVICES Section	79
DM_REMOTE_SERVICES Section	81
DM_RESOURCES	83
DM_ROUTING Section	84
DMCONFIG for GWTOPEND(5)	91

DM_LOCAL_DOMAINS Section	94
DM_REMOTE_DOMAINS Section	99
DM_TOPEND Section.....	101
DM_ACCESS_CONTROL Section	104
DM_LOCAL_SERVICES Section	105
DM_REMOTE_SERVICES Section	109
DM_RESOURCES	114
DM_ROUTING Section	115
DM_MIB(5).....	122
T_DM_ACL Class Definition.....	127
T_DM_CONNECTION Class Definition.....	129
T_DM_EXPORT Class Definition	132
T_DM_IMPORT Class Definition.....	137
T_DM_LOCAL Class Definition.....	142
T_DM_OSITP Class Definition.....	149
T_DM_PASSWORD Class Definition	153
T_DM_PRINCIPAL_MAP Class Definition.....	155
T_DM_REMOTE Class Definition	157
T_DM_RESOURCES Class Definition.....	161
T_DM_ROUTING Class Definition.....	162
T_DM_RPRINCIPAL Class Definition.....	166
T_DM_SNACRM Class Definition	168
T_DM_SNALINK Class Definition	170
T_DM_SNASTACK Class Definition.....	174
T_DM_TDOMAIN Class Definition	177
T_DM_TOPEND Class Definition	180
T_DM_TRANSACTION Class Definition.....	183
EVENTS(5)	189
EVENT_MIB(5).....	195
T_EVENT_CLIENT Class Definition.....	197
T_EVENT_COMMAND Class Definition.....	199
T_EVENT_QUEUE Class Definition.....	201
T_EVENT_SERVICE Class Definition.....	205
T_EVENT_USERLOG Class Definition.....	207
Error, Error32(5).....	209

field_tables(5).....	211
GWADM(5)	214
GWTDOMAIN(5).....	216
GWTOPEND(5)	218
GWTUX2TE, GWTE2TUX(5).....	220
langinfo(5)	228
MIB(5).....	232
Usage.....	241
T_CLASS Class Definition.....	252
T_CLASSATT Class Definition.....	254
nl_types(5).....	261
servopts(5)	262
TM_MIB(5).....	267
T_BRIDGE Class Definition	270
T_CLIENT Class Definition	275
T_CONN Class Definition.....	284
T_DEVICE Class Definition	287
T_DOMAIN Class Definition.....	290
T_GROUP Class Definition.....	309
T_MACHINE Class Definition	319
T_MSG Class Definition	339
T_NETGROUP Class Definition.....	342
T_NETMAP Class Definition.....	344
T_QUEUE Class Definition.....	348
T_ROUTING Class Definition	353
T_SERVER Class Definition.....	357
T_SERVERCTXT Class Definition	375
T_SERVICE Class Definition	378
T_SVCGRP Class Definition.....	383
T_TLISTEN Class Definition.....	390
T_TLOG Class Definition	391
T_TRANSACTION Class Definition.....	394
T_ULOG Class Definition	398
TMQFORWARD(5).....	410
TMQUEUE(5).....	415

TMSYSEVT(5)	419
tmtrace(5).....	420
TMUSREVT(5).....	425
tperrno(5).....	427
tpurcode(5).....	430
tuxenv(5).....	431
tuxtypes(5).....	437
typesw(5).....	442
UBBCONFIG(5)	444
RESOURCES Section.....	447
MACHINES Section.....	459
GROUPS Section.....	468
NETGROUPS Section	474
NETWORK Section.....	475
SERVERS Section	478
SERVICES Section.....	484
ROUTING Section.....	487
viewfile(5)	494
WS_MIB(5).....	501
T_WSH Class Definition	503
T_WSL Class Definition	508
WSL(5).....	522

Section 5 - File Formats and Data Descriptions

BEA Tuxedo File Formats and Data Descriptions

Name	Description
Introduction to Tables and Files	Overview of this document
ACL_MIB(5)	Management Information Base for ACLs
APPQ_MIB(5)	Management Information Base for /Q
AUTHSVR(5)	Server providing per-user authentication
compilation(5)	Instructions for compilation of BEA Tuxedo system application components
DMADM(5)	Domains administrative server
DMCONFIG(5)	Text version of a Domains configuration file
DMCONFIG for GWTOPEND(5)	Text version of a Domains configuration file for a TOP END Domain Gateway
DM_MIB(5)	Management Information Base for Domains
EVENTS(5)	List of system-generated events
EVENT_MIB(5)	Management Information Base for Event Broker
Error, Ferror, Ferror32(5)	FML error codes
field_tables(5)	FML mapping files for field names
GWADM(5)	Domains gateway administrative server

Section 5 - File Formats and Data Descriptions

BEA Tuxedo File Formats and Data Descriptions

Name	Description
GWTDOMAIN(5)	TDomain gateway process
GWTOPEND(5)	TOP END Domain Gateway process
GWTUX2TE, GWTE2TUX(5)	BEA Tuxedo / BEA TOP END gateway servers
langinfo(5)	Language information constants
MIB(5)	Management Information Base
nl_types(5)	Native language data types
servopts(5)	Run-time options for server processes
TM_MIB(5)	Management Information Base for core BEA Tuxedo System
TMQFORWARD(5)	Message Forwarding Server
TMQUEUE(5)	Message Queue Manager
TMSYSEVT(5)	System event reporting process
tmtrace(5)	Run-time tracing facility
TMUSREVT(5)	User event reporting process
tperrno(5)	BEA Tuxedo system error codes
tpurcode(5)	BEA Tuxedo system global variable for an application-specified return code
tuxenv(5)	List of environment variables in the BEA Tuxedo system
tuxtypes(5)	Buffer type switch; descriptions of buffer types provided by the BEA Tuxedo system
typesw(5)	Buffer type switch structure; parameters and routines needed for each buffer type
UBBCONFIG(5)	Text version of a BEA Tuxedo configuration file
viewfile(5)	Source file for view descriptions
WS_MIB(5)	Management Information Base for Workstation
WSL(5)	Workstation Listener server

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)`
- `DM_MIB(5)`
- `EVENT_MIB(5)`
- `TM_MIB(5)`
- `WS_MIB(5)`

ACL_MIB(5)

Name ACL_MIB - Management Information Base for ACLs

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 be 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 <code>udataobj/tpadm</code> relative to the root directory of the BEA Tuxedo system software installed on the system. The directory <code>#{TUXDIR}/udataobj</code> should be included by the application in the colon separated list specified by the <code>FLDTBLDIR</code> environment variable and the field table name <code>tpadm()</code> should be included in the comma separated list specified by the <code>FIELDTBLS</code> environment variable.
Limitations	Access to the header files and field tables for this MIB is provided only at sites running BEA Tuxedo Release 6.0 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.

VALid	T_ACLGROUP object is defined and inactive. Note that this is the only valid state for this class. ACL groups are never <i>active</i> .
-------	--

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-----	<i>string</i> [1..30]	N/A
TA_ACLTYPE(r)(*)	string	rw-----	"ENQ DEQ SERVICE POSTEVENT"	N/A
TA_ACLGROUPIDS	string	rw-----	<i>string</i>	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.

VALid	T_ACLPERM object is defined and inactive. Note that this is the only valid state for this class. ACL permissions are never <i>active</i> .
-------	--

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.
<i>unset</i>	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-----	<i>string</i> [1..30]	N/A
TA_PRINCLTNAME(k)	string	rw-----	<i>string</i> [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-----	<i>string</i>	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*

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.

VALid	T_ACLPRINCIPAL object is defined and inactive. Note that this is the only valid state for this class. ACL principals are never <i>active</i> .
-------	--

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.
-----	--

<i>unset</i>	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()`, `tpgetrply()` and `tpdequeue()`) 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()` and `tpgetrply()` 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 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 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 = tmalloc("FML32", NULL, 1000);

obuf = tmalloc("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\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 */
}
```

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

See Also `tpacall(3c)`, `tpalloc(3c)`, `tpcall(3c)`, `tpdequeue(3c)`, `tpenqueue(3c)`,
`tpgetrply(3c)`, `tprealloc(3c)`, **Introduction to FML Functions**, `Fadd`,
`Fadd32(3fml)`, `Fchg`, `Fchg32(3fml)`, `Ffind`, `Ffind32(3fml)`, `MIB(5)`,
`TM_MIB(5)`

Setting Up a BEA Tuxedo Application

Programming a BEA Tuxedo Application Using C

Programming a BEA Tuxedo Application Using FML

APPQ_MIB(5)

- Name** APPQ_MIB - Management Information Base for /Q
- 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()` 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 persistent (reliable disk-based) and non-persistent (in memory) 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 be 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 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 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 (that is, when 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()`, then all `T_APPQ` objects within the specified queue space will be retrieved.

Attribute Table

APPQ_MIB(5): T_APPQ Class Definition Attribute Table

Attribute ^a	Type	Permissions	Values	Default
<code>TA_APPQNAME(k)(r)(*)</code>	string	ru-r--r--	<i>string</i> [1..15]	N/A
<code>TA_APPQSPACENAME(k)(r)(*)</code>	string	ru-r--r--	<i>string</i> [1..15]	N/A
<code>TA_QMCONFIG(k)(r)(*)</code>	string	ru-r--r--	<i>string</i> [1..78]	N/A
<code>TA_LMID(k)(r)(*)^b</code>	string	ru-r--r--	<i>string</i> [1..30]	N/A
<code>TA_STATE</code> (Note 3) ^c	string	rw-r--r--	GET: "VAL" SET: "{NEW INV}"	N/A N/A
<code>TA_APPQORDER^d</code>	string	rw-r--r--	{PRIO TIME LIFO FIFO EXPIR}	FIFO
<code>TA_DEFEXPIRATIONTIME</code>	string	rw-r--r--	{+seconds NONE}	N/A
<code>TA_DEFDELIVERYPOLICY</code>	string	rw-r--r--	{PERSIST NONPERSIST}	PERSIST
<code>TA_CMD</code>	string	rw-r--r--	<i>shell-command</i> <i>-string</i> [0..78]	" "
<code>TA_CMDHW</code>	string	rw-r--r--	0 <= num [bBm%]	100%

APPQ_MIB(5): T_APPQ Class Definition Attribute Table

Attribute ^a	Type	Permissions	Values	Default
TA_CMDLW	string	rw-r--r--	0 <= num [bBm%]	0%
TA_CMDNONPERSIST	string	rw-r--r--	shell-command-string [0-78]	""
TA_CMDNONPERSISTHW	string	rw-r--r--	0 <= num[bB%]	100%
TA_CMDNONPERSISTLW	string	rw-r--r--	0 <= num[bB%]	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
TA_CURNONPERSISTBYTES	long	r--r--r--	0 <= num	N/A
TA_CURNONPERSISTMSG	long	r--r--r--	0 <= num	N/A

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

^aAll attributes of class T_APPQ are local attributes.

^bTA_LMID must be specified as a key field except when the application is unconfigured (that is, the TUXCONFIG environment variable is not set).

^cAll operations on T_APPQ objects—both GET and SET—silently open the associated queue space (that is, 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.

^dTA_APPQORDER cannot be modified after the application queue is created.

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

Section 5 - File Formats and Data Descriptions

Attribute	TA_APPQNAME: <i>string</i> [1..15]
Semantics	Name of the application queue.
	TA_APPQSPACENAME: <i>string</i> [1..15]
	Name of the application queue space containing the application queue.
	TA_QMCONFIG: <i>string</i> [1..78]
	Absolute pathname of the file or device where the application queue space is located.
	TA_LMID: <i>string</i> [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.
	<hr/>
	VALid The specified queue exists. This state is INActive equivalent for purposes of permissions checking.
	<hr/>
	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 cannot be set.
	<hr/>
	NEW Create a new queue in the specified queue space. The queue is left in state VALid following successful creation.

<code>INValid</code>	Delete the specified queue. The queue must be in state <code>VALid</code> to be deleted. If the queue space has processes attached to it (that is, it is in the <code>ACTive</code> state), the queue will not be deleted unless the <code>TA_FLAGS</code> attribute includes the <code>QMIB_FORCEDELETE</code> flag. In addition, if the queue has messages in it, it will not be deleted unless <code>QMIB_FORCEPURGE</code> is specified. Successful return leaves the object in the <code>INValid</code> state.
<code>unset</code>	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`, `TIME`, or `EXPIR`. A combination of sort criteria may be specified with the most significant criterion specified first, followed by other criteria, and optionally followed by either `LIFO` or `FIFO`, which are mutually exclusive. If `EXPIR` is specified, messages with no expiration time are dequeued after all messages with an expiration time. If neither `FIFO` nor `LIFO` is specified, `FIFO` is assumed. If no order is specified when a queue is created, the default order is `FIFO`. For example, the following are settings are legal:

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

`TA_CMD: shell-command-string[0..78]`

The command to be automatically executed when the high water mark for persistent (disk-based) messages, `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. Both `TA_CMDHW` and `TA_CMDLW` must be followed by one of the following keyletters and 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 persistent (disk based) messages in the queue.

B

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

m

The high and low water marks pertain to the number of messages (both persistent and non-persistent) in the queue.

%

The high and low water marks are expressed in terms of a percentage of queue capacity. This pertains only to persistent messages.

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_CMDNONPERSIST`: *shell-command-string*[0..78]

This attribute specifies the command to be executed automatically when the high-water mark for non-persistent (memory-based delivery) messages, `TA_CMDNONPERSISTHW`, is reached. The command is re-executed when the high-water mark is reached again after the low-water mark for non-persistent (memory-based delivery) messages, `TA_CMDNONPERSISTLW`, has been reached.

`TA_CMDNONPERSISTHW`: `0 <= num[bB%]`

`TA_CMDNONPERSISTLW`: `0 <= num[bB%]`

These attributes specify the high- and low-water marks that control the automatic execution of the command specified in the `TA_CMDNONPERSIST` attribute. Each is an integer greater than or equal to zero followed by one of the following keyletters. The keyletters must be consistent for `TA_CMDNONPERSISTHW` and `TA_CMDNONPERSISTLW`.

- b The high- and low-water marks are expressed as the number of bytes used by non-persistent (in memory) messages in the queue.
- B The high- and low-water marks are expressed as the number of blocks used by non-persistent (in memory) messages in the queue.
- % The high- and low-water marks are expressed as a percentage of the shared memory capacity reserved for non-persistent messages in the queue space used by the queue.

The messages threshold type specified via the `TA_CMDHW` and `TA_CMDLW` attributes (when followed by an `m`) applies to all messages in a queue, including both persistent and non-persistent messages, and therefore is not available as a threshold type for `TA_CMDNONPERSISTHW` and `TA_CMDNONPERSISTLW`.

`TA_CURBLOCKS: 0 <= num`

The number of disk pages currently consumed by the queue.

`TA_CURMSG: 0 <= num`

The number of persistent messages currently in the queue. To determine the total number of messages in the queue, add `TA_CURMEMMSG` to this value.

`TA_DEFAULTEXPIRATIONTIME:`

This attribute specifies an expiration time for messages enqueued with no explicit expiration time. The expiration time may be either a relative expiration time or `NONE`. The relative expiration time is determined by associating a fixed amount of time with a message after the message arrives at the queue manager process. When a message's expiration time is reached and the message has not been dequeued or administratively deleted, all resources associated with the message are reclaimed by the system and statistics are updated. If a message expires during a transaction, the expiration does not cause the transaction to fail. Messages that expire while being enqueued or dequeued within a transaction are removed from the queue when the transaction ends. There is no notification that the message has expired. If no default expiration time is specified for a queue, then messages without an explicit expiration time do not expire. When the queue's expiration time is modified, the expiration times of messages that were in the queue before the modification are not changed.

The format is `+seconds` where `seconds` is the number of seconds allowed to lapse between the time that the queue manager successfully completes the operation and the time that the message is to expire. If `seconds` is set to zero (0) the message expires immediately.

The value of this attribute may also be set to the string `NONE`. The `NONE` string indicates that messages enqueued to the queue with no explicit expiration time do not expire. You may change the expiration time for messages already in a queue with the `TA_EXPIRETIME` attribute of the `T_APPQMSG` class in the `APPQ_MIB`.

`TA_DEFDELIVERYPOLICY:`

This attribute specifies the default delivery policy for the queue when no delivery mode is specified for a message enqueued to the queue. When the value is `PERSIST`, messages enqueued to the queue without an explicitly specified delivery mode are delivered using the persistent (disk-based) delivery method. When the value is `NONPERSIST`, messages enqueued to the queue without an explicitly specified delivery mode are delivered using the non-persistent (in memory) delivery method. When a queue's default delivery policy is modified, the delivery quality of service of messages that are in the queue before the modification are not changed. If the queue being modified is the reply queue named for any messages currently in the queue space, the reply quality of service is not changed for those messages as a result of changing the default delivery policy of the queue.

For non-persistent delivery, if the memory area is exhausted or fragmented such that a message cannot be enqueued, the enqueueing operation fails, even if there is sufficient persistent storage for the message. Similarly, if the persistent storage area is exhausted or fragmented such that a message cannot be enqueued, the enqueueing operation fails, even if there is sufficient non-persistent storage for the message. If the `TA_MEMNONPERSIST` attribute of the `T_APPQSPACE` class is zero (0) for a queue space, no space is reserved for non-persistent messages. In such a case, any attempt to enqueue a non-persistent message fails. This type of failure results, for example, when no delivery quality of service has been specified for a message and the `TA_DEFDELIVERYPOLICY` attribute for the target queue has been set to `NONPERSIST`.

`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: {NONE | TOP | MSGID}

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.

TA_CURNONPERSISTBYTES: 0 <= *num*

This attribute specifies the number of shared memory bytes currently consumed by the non-persistent messages on the queue.

TA_CURNONPERSISTMSG: 0 <= *num*

This attribute specifies the number of non-persistent messages currently in the queue. To determine the total number of messages in the queue, add TA_CURMSG to this value.

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()`. A message can be destroyed either by a call to `tpdequeue()` 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 (that is, 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()`, 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
<code>TA_APPQMSGID(k)(*)</code>	string	r--r--r--	<i>string</i> [1..32]	N/A
<code>TA_APPQNAME(k)(*)</code>	string	r--r--r--	<i>string</i> [1..15]	N/A
<code>TA_APPQSPACENAME(k) (*)</code>	string	r--r--r--	<i>string</i> [1..15]	N/A
<code>TA_QMCONFIG(k)(*)</code>	string	r--r--r--	<i>string</i> [1..78]	N/A
<code>TA_LMID(k)(*)^b</code>	string	r--r--r--	<i>string</i> [1..30]	N/A
<code>TA_STATE^c</code>	string	rw-r--r--	GET: "VAL" SET: "INV"	N/A N/A
<code>TA_NEWAPPQNAME</code>	string	-w--w----	<i>string</i> [1..15]	N/A
<code>TA_PRIORITY</code>	long	rw-rw-r--	{ 1 <= num <= 100 -1 }	N/A

APPQ_MIB(5): T_APPQMSG Class Definition Attribute Table

Attribute ^a	Type	Permissions	Values	Default
TA_TIME	string	rw-rw-r--	{ YY[MM[DD[hh[mm[ss]]]]] +seconds}	N/A
TA_EXPIRETIME	string	rw-rw-r--	{ YY[MM[DD[hh[mm[ss]]]]] +seconds}	N/A
TA_CORRID(k)	string	r--r--r--	string[0..32]	N/A
TA_PERSISTENCE (k)	string	r--r--r--	{ PERSIST NONPERSIST}	N/A
TA_REPLYPERSISTENCE	string	r--r--r--	{ PERSIST NONPERSIST DEFAULT}	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_MSGEXPIREENDTIME(k)	string	k--k--k--	{ YY[MM[DD[hh[mm[ss]]]]] +seconds/NONE}	MAXLONG
TA_MSGEXPIRESTARTTIME(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

^aAll attributes of class T_APPQMSG are local attributes.

^bTA_LMID must be specified as a key field except when the application is unconfigured (that is, the TUXCONFIG environment variable is not set).

^cAll operations on T_APPQMSG objects—both GET and SET—silently open the associated queue space (that is, 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.

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

Attribute Semantics	TA_APPQMSGID: <i>string</i> [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: <i>string</i> [1..15]	Name of the application queue in which the message is stored.		
	TA_APPQSPACENAME: <i>string</i> [1..15]	Name of the application queue space containing the message.		
	TA_QMCONFIG: <i>string</i> [1..78]	Absolute pathname of the file or device where the application queue space is located.		
	TA_LMID: <i>string</i> [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.		
		<hr/> <table><tr><td>VALid</td><td>The message exists. This state is INActive equivalent for purposes of permissions checking.</td></tr></table> <hr/>	VALid	The message exists. This state is INActive equivalent for purposes of permissions checking.
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 cannot be set.		

<code>INValid</code>	The message is deleted from its queue space. The message must be in state <code>VALid</code> before attempting this operation. Successful return leaves the object in the <code>INValid</code> state.
<code>unset</code>	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 `topenqueue(3c)` request. The empty string indicates that a correlation identifier is not present.

`TA_EXPIRETIME:`

This attribute specifies the time at which a message expires (that is, the time at which the message should be removed from the queue if it has not already been dequeued or administratively deleted). When a message expires, all resources used by it are reclaimed by the system and statistics are updated. If a message expires during a transaction, the expiration does not cause the transaction to fail. Messages that expire while being enqueued or dequeued within a transaction are removed from the queue when the transaction ends. There is no notification that the message has expired.

Expiration times cannot be added to messages enqueued by versions of the BEA Tuxedo system that do not support message expiration, even when the queue manager responsible for changing this value supports message expiration. Attempts to add an expiration time fail.

The empty string is returned by a `GET` operation if the expiration time is not set. The `TA_EXPIRETIME` format is one of the following.

`+seconds`

Specifies that the message will be removed after the specified number of seconds. If the value of `seconds` is set to zero (0), the message is removed immediately from the queue. Relative expiration time is calculated on the basis of the time at which the MIB request arrives and has been processed by the corresponding queue manager.

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

Specifies the year, month, day, hour, minute, and second when the message will be removed if it has not been dequeued or administratively deleted already. 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 2037, 70 through 99 are treated as 1970 through 1999, and 38 through 69 are invalid. An absolute expiration time is determined by the clock on the machine where the queue manager process resides.

NONE

Specifies that the message will never expire.

`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.

`TA_MSGEXPIRESTARTTIME:`

`TA_MSGEXPIREENDTIME:`

The expiration start and end times within which to search for occurrences of `T_APPQMSG` objects. The range is inclusive. A start or end time must be specified as an absolute time value; see [TA_EXPIRETIME](#) for the format. These attributes may only be used as key fields with `GET` operations.

`TA_MSGSIZE: 0 <= num`

The size of the message, in bytes.

`TA_MSGSTARTTIME:`

`TA_MSGENDTIME:`

The start and end time within which to search for occurrences of `T_APPQMSG` objects. The range is inclusive. A start or end time must be specified as an absolute time value; see [TA_TIME](#) for the format. These attributes may only be used as key fields with `GET` operations.

`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. The delivery quality of service of messages that are moved will not be changed as a result of the default delivery policy of the new queue. When messages with an expiration time are moved, the expiration time is

considered an absolute expiration time in the new queue, even if it was previously specified as a relative expiration time.

TA_PERSISTENCE:

The quality of service with which the message is being delivered. This read-only state is set to `NONPERSIST` for non-persistent messages and `PERSIST` for persistent messages.

TA_PRIORITY: $1 \leq num \leq 100$

The priority of the message.

TA_REPLYPERSISTENCE:

The quality of service with which replies to the message should be delivered. This read-only state is set to `NONPERSIST` for non-persistent, `PERSIST` for persistent, and `DEFAULT` when the reply is to use the default persistence established for the queue where the reply is to be enqueued.

Note that the default delivery policy is determined when the reply to a message is enqueued. That is, if the default delivery policy of the reply queue is modified between the time that the original message is enqueued and the reply to the message is enqueued, the policy used is the one in effect when the reply is finally enqueued.

TA_TIME:

The time when the message will be made available. The format is one of the following:

+seconds

Specifies that the message will be processed *seconds* in the future. The value zero (0) 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 2037, 70 through 99 are treated as 1970 through 1999, and 38 through 69 are invalid.

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()` in the context of an unconfigured application (that is, 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
<code>TA_APPQSPACENAME(k)(r)(*)</code>	string	ru-r--r--	<i>string</i> [1..15]	N/A
<code>TA_QMCONFIG(k)(r)(*)</code>	string	ru-r--r--	<i>string</i> [1..78]	N/A
<code>TA_LMID(k)(r)(*)b</code>	string	ru-r--r--	<i>string</i> [1..30]	N/A
<code>TA_STATE(k)c</code>	string	rwrxrwxr--	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_MAXACTIONS	long	rw-r--r--	0 <= num	0
TA_MAXHANDLES	long	rw-r--r--	0 <= num	0
TA_MAXOWNERS	long	rw-r--r--	0 <= num	0
TA_MAXTMPQUEUES	long	rw-r--r--	0 <= num	0
TA_MAXCURSORS	long	rw-r--r--	0 <= num	0
TA_MEMNONPERSIST	string	rw-r--r--	0 <= num[bB]	0
TA_MEMFILTERS	long	rw-r--r--	0 <= num	0
TA_MEMOVERFLOW	long	rw-r--r--	0 <= num	0
TA_MEMSYSTEMRESERVED	long	r--r--r--	0 <= num	N/A
TA_MEMTOTALALLOCATED	long	r--r--r--	0 <= num	N/A

Section 5 - File Formats and Data Descriptions

APPQ_MIB(5): T_APPQSPACE Class Definition Attribute Table

Attribute ^a	Type	Permissions	Values	Default
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_CURACTIONS	long	r--r--r--	0 <= num	N/A
TA_CURHANDLES	long	r--r--r--	0 <= num	N/A
TA_CUOWNERS	long	r--r--r--	0 <= num	N/A
TA_CURTMPQUEUES	long	r--r--r--	0 <= num	N/A
TA_CURCURSORS	long	r--r--r--	0 <= num	N/A
TA_CURMEMNONPERSIST	long	r--r--r--	0 <= num	N/A
TA_CURMEMFILTERS	long	r--r--r--	0 <= num	N/A
TA_CURMEMOVERFLOW	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_HWACTIONS	long	R--R--R--	0 <= num <= 100	N/A
TA_HWHANDLES	long	R--R--R--	0 <= num	N/A
TA_HWOWNERS	long	R--R--R--	0 <= num	N/A
TA_HWTMPQUEUES	long	R--R--R--	0 <= num	N/A
TA_HWCURSORS	long	R--R--R--	0 <= num	N/A
TA_HWMEMNONPERSIST	long	R--R--R--	0 <= num	N/A
TA_HWMEMFILTERS	long	R--R--R--	0 <= num	N/A
TA_HWMEMOVERFLOW	long	R--R--R--	0 <= num	N/A
TA_PERCENTINIT	long	r--r--r--	0 <= num	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 (that is, 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 (that is, 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 cannot 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: <i>string</i> [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.

INActive	The queue space exists; that is, 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 cannot 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. When successful, all non-persistent messages are permanently lost.
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. When successful, all non-persistent messages are permanenently lost.
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_CURACTIONS: 0 <= *num*

This attribute specifies the current number of actions in use in the queue space. This number can be determined 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_CURCURSORS: 0 <= *num*

This attribute specifies the current number of cursors in use in the queue space. This number can be determined 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_CUREXTENT: 0 <= *num* <= 100

The current number of extents used by the queue space. The largest number allowed is 100. Each time the value of the `TA_MAXPAGES` attribute is increased, a new extent is allocated. When this attribute is modified, all non-persistent messages in the queue space are permanently lost.

TA_CURHANDLES: 0 <= *num*

This attribute specifies the current number of handles in use in the queue space. This number can be determined 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_CURMEMFILTERS: 0 <= *num*

This attribute specifies the current number of bytes in use for filters in the queue space. This number can be determined 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_CURMEMNONPERSIST: 0 <= *num*

The current amount of memory in bytes consumed by non-persistent messages in the queue space. This number can be determined 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_CURMEMOVERFLOW: 0 <= *num*

This attribute specifies the current number of bytes in use of the overflow memory in the queue space. This number can be determined 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_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 these conditions are met, the value `-1` is returned.

`TA_CUOWNERS`: `0 <= num`

This attribute specifies the current number of owners in use in the queue space. This number can be determined 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 are met, the value `-1` is returned.

`TA_CURTMPQUEUES`: `0 <= num`

This attribute specifies the current number of temporary queues in use in the queue space. This number can be determined 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_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 (for example, regular file or raw slice), initialization can be done even if it is not requested.

TA_HWACTIONS: 0 <= *num*

This attribute specifies the highest number of concurrent actions reached 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_HWCURSORS: 0 <= *num*

This attribute specifies the highest number of concurrent cursors created 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_HWHANDLES: 0 <= *num*

This attribute specifies the highest number of concurrent handles opened 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_HWMEMFILTERS: 0 <= *num*

This attribute specifies the highest number of bytes used for filters 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_HWMEMNONPERSIST: 0 <= *num*

The largest amount of memory in bytes consumed by non-persistent messages since the queue space was last opened. The number is reset to 0 when the queue space state is set to `CLEaning`.

TA_HWMEMOVERFLOW: 0 <= *num*

This attribute specifies the highest number of bytes used in the overflow memory 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_HWMSG: 0 <= *num*

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

TA_HWOWNERS: 0 <= *num*

This attribute specifies the highest number of concurrent owners reached 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 at a given time since the queue space was last opened. The number is reset to 0 when the queue space state is set to `CLEaning`.

TA_HWTMPQUEUES: 0 <= *num*

This attribute specifies the highest number of concurrent temporary queues opened 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 at a given time 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, 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_MAXACTIONS: 0 <= *num*

This attribute specifies the number of additional actions that the Queuing Services component of the BEA Engine can handle concurrently. When a blocking operation is encountered and additional actions are available, the blocking operation is set aside until it can be satisfied. After setting aside the blocking operation, another operation request can be handled. When the blocking operation completes, the action associated with the operation is made available for a subsequent operation. The system reserves actions equivalent to the number of processes that can attach to a queue space, so that each queue manager process may have at least one blocking action. Beyond the system-reserved number of blocking actions, the administrator may configure the system to be able to accommodate additional blocking actions beyond the reserve. An operation fails if a blocking operation is requested and cannot be immediately satisfied and there are no actions available.

TA_MAXCURSORS: 0 <= *num*

This attribute specifies the number of cursors that users of that the Queuing Services component of the BEA Engine may use concurrently. Cursors are

used to navigate a queue. When a cursor is destroyed, the cursor resources are made available for subsequent cursor creation operations. When the cursors are used by an application, the administrator must configure the system to accommodate the maximum number of cursors that will be allocated concurrently. An operation fails if a user attempts to create a cursor and there are no cursor resources available. BEA Tuxedo applications need not adjust this value. Adjusting this value has no effect on BEA Tuxedo applications other than unnecessarily consuming shared memory resources.

TA_MAXHANDLES: 0 <= *num*

This attribute specifies the number of handles that users of that the Queuing Services component of the BEA Engine may use concurrently. Objects manipulated by the queuing services API require handles to access the objects. When an object is opened by a call to the Queuing Services API, a new handle is created and returned to the user. When an object handle is closed, the handle is made available for subsequent open object operations. When the Queuing Services API is used by an application, the administrator must configure the system to accommodate the maximum number of handles that will be opened concurrently. An operation fails if a user attempts to open a queuing services object and there are no handles available. Adjusting this value has no effect on BEA Tuxedo applications other than unnecessarily consuming shared memory resources.

TA_MAXMSG: 0 <= *num*

The maximum number of messages that the queue space can contain at a given time.

TA_MAXOWNERS: 0 <= *num*

This attribute specifies the number of additional BEA Engine authenticated users that may concurrently use Queuing Services resources. There is one owner record per user, regardless of the number of open handles for the user. When there are no open handles for a user, the owner record is made available for subsequent users. The system reserves owners equivalent to the number of actions so that each action may be initiated by a different owner. Beyond the system-reserved number of owners that may be concurrently using queuing services resources, the administrator may configure the system to accommodate additional owners beyond the reserve. An operation fails if a user attempts to open a handle when they currently do not have any open handles, and there are no owners available. Adjusting this value has no effect on BEA Tuxedo applications other than unnecessarily consuming shared memory resources.

`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 at a given time.

`TA_MAXTMPQUEUES: 0 <= num`

This attribute specifies the number of temporary queues that may be opened concurrently in the Queuing Services component of the BEA Engine. Temporary queues reduce the need for administrators to configure each queue used by an application. They are used by dynamic self-configuring applications. Messages enqueued to temporary queues are not persistent. When all handles to a temporary queue are closed, the temporary queue resources are made available for subsequent temporary queue creation. When the temporary queues are used by an application, the administrator must configure the system to accommodate the maximum number of temporary queues that will be active concurrently. An open operation fails if a user attempts to open a temporary queue and there are no temporary queue resources available. Adjusting this value has no effect on BEA Tuxedo applications other than unnecessarily consuming shared memory resources.

`TA_MAXTRANS: 0 <= num`

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

`TA_MEMFILTERS: 0 <= num`

This attribute specifies the size of the memory area to reserve in shared memory to hold the compiled representation of user defined filters. The memory size is specified in bytes. Filters are used by the Queuing Services component of the BEA Engine for message selection in dequeuing and cursor operations. Filters may be specified using various grammars but are compiled into an engine normal form and stored in shared memory. Filters are referenced by a handle returned when they are compiled. When a filter is destroyed, the memory used by the filter is made available for subsequent compiled filters. When the filters are defined by an application, the

administrator must configure the system to accommodate the maximum number of filters that will be concurrently compiled. An operation fails if a user attempts to create a new filter and there is not enough memory allocated for the compiled version of the filter. Adjusting this value has no effect on BEA Tuxedo applications other than unnecessarily consuming shared memory resources.

TA_MEMNONPERSIST: 0 <= num [bB]

This attribute specifies the size of the area reserved in shared memory to hold non-persistent messages for all queues in the queue space. The memory size may be specified in bytes (b) or blocks (B). (The size of a block, in this context, is equivalent to the size of a disk block.) The [bB] suffix is optional and, if not specified, the default is blocks (B).

If the value is specified in bytes (b) for this attribute, the system divides the specified value by the number of bytes per *page* (page size is equivalent to the disk page size), rounds down the result to the nearest integer, and allocates that number of pages of memory. For example, assuming a page size of 1024 bytes (1KB), a requested value of 2000b results in a memory allocation of 1 page (1024 bytes), and a requested value of 2048b results in a memory allocation of 2 pages (2048 bytes). Requesting a value less than the number of bytes per page results in an allocation of 0 pages (0 bytes).

If the value is specified in blocks (B) for this attribute and assuming that one block of memory is equivalent to one page of memory, the system allocates the same value of pages. For example, a requested value of 50B results in a memory allocation of 50 pages.

All non-persistent messages in the specified queue space are permanently lost when TA_MEMNONPERSIST is successfully changed.

If TA_MEMNONPERSIST for a queue space is zero (0) for a queue space, no space is reserved for non-persistent messages. In this case, any attempt to enqueue a non-persistent message fails. This type of failure results, for example, when no delivery quality of service has been specified for a message and the TA_DEFDELIVERYPOLICY attribute of the T_APPQ class for the target queue has been set to NONPERSIST. For non-persistent delivery, if the memory area is exhausted or fragmented such that a message cannot be enqueued, the enqueueing operation fails, even if there is sufficient persistent storage for the message. Similarly, if the persistent storage area is exhausted or fragmented such that a message cannot be enqueued, the enqueueing

operation fails, even if there is sufficient non-persistent storage for the message.

`TA_MEMOVERFLOW: 0 <= num`

This attribute specifies the size of the memory area to reserve in shared memory to accommodate peek load situations where some or all of the allocated shared memory resources are exhausted. The memory size is specified in bytes. Additional objects are allocated from this additional memory on a first-come-first-served basis. When an object created in the additional memory is closed or destroyed, the memory is released for subsequent overflow situations. This additional memory space may yield more objects than the configured number, but there is no guarantee that additional memory is available for any particular object at any given point in time. Currently, only actions, handles, cursors, owners, temporary queues, timers, and filters use the overflow memory.

`TA_MEMSYSTEMRESERVED: 0 <= num`

This attribute specifies the total amount of memory (in bytes) reserved from shared memory for queuing services system use.

`TA_MEMTOTALALLOCATED: 0 <= num`

This attribute specifies the total amount of memory (in bytes) allocated from shared for all queuing services objects.

`TA_PERCENTINIT: 0 <= num <= 100`

The percentage 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 a request using `tpcall()`, 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
<code>TA_XID(k)(*)</code>	string	R--R--R--	<i>string</i> [1..78]	N/A
<code>TA_APPQSPACENAME(k)(*)</code>	string	r--r--r--	<i>string</i> [1..15]	N/A
<code>TA_QMCONFIG(k)(*)</code>	string	r--r--r--	<i>string</i> [1..78]	N/A
<code>TA_LMID(k)(*)</code>	string	r--r--r--	<i>string</i> [1..30]	N/A
<code>TA_STATE^b</code>	string	R-XR-XR--	GET: "{ACT ABY ABD COM REA DEC HAB HCO}" SET: "{HAB HCO}"	N/A 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—both GET and SET—silently open the associated queue space (that is, 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.

Section 5 - File Formats and Data Descriptions

Attribute Semantics	<p>TA_XID: <i>string</i>[1..78] Transaction identifier as returned by <code>tx_info()</code> and mapped to a string representation. The data in this field should not be interpreted directly by the user except for equality comparison.</p> <p>TA_APPQSPACENAME: <i>string</i>[1..15] Name of the application queue space associated with the transaction.</p> <p>TA_QMCONFIG: <i>string</i>[1..78] Absolute pathname of the file or device where the application queue space is located.</p> <p>TA_LMID: <i>string</i>[1..30] (no comma) Identifier of the logical machine where the application queue space is located.</p> <p>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. All states are ACTIVE equivalent for purposes of permissions checking.</p>
---------------------	---

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 <code>tpcommit()</code> 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 cannot 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 6.0 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 = tmalloc("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", rqbbuf, 0, rpbbuf, 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>
```

Libraries `${TUXDIR}/lib/libtmib.a`, `${TUXDIR}/lib/libqm.a`,
`${TUXDIR}/lib/libtmib.so.<rel>`, `${TUXDIR}/lib/libqm.so.<rel>`,
`${TUXDIR}/lib/libqm.lib`

The libraries must be linked manually when using `buildclient`. The user must use:
`-L${TUXDIR}/lib -ltmib -lqm`

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(rqbbuf, TA_OPERATION, 0, "SET", 0);
Fchg32(rqbbuf, TA_CLASS, 0, "T_DEVICE", 0);
Fchg32(rqbbuf, TA_STATE, 0, "NEW", 0);
Fchg32(rqbbuf, TA_CFGDEVICE, 0, "/dev/q/dsk001", 0);
Fchg32(rqbbuf, TA_LMID, 0, "SITE1", 0);
size = 500;
Fchg32(rqbbuf, TA_DEVSZ, 0, (char *)size, 0);

/* Make the request; see above */

/* Reinitialize the same buffer for reuse */
Finit32(rqbbuf, (FLDLN) Fsizeof32(rqbbuf));

/* 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 (that is, server migration is not used).

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, (FLDLLEN) 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 <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 ";" /* separator for PATH */
#else
#define pathsep ":" /* separator for PATH */
#endif
    appqspacename = strchr(qmconfig, pathsep);
    appqspacename[0] = '\0'; /* 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, rdbuf2, 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 (that is, `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

`tpacall(3c)`, `tpadmcall(3c)`, `tpalloc(3c)`, `tpcall(3c)`, `tpdequeue(3c)`, `topenqueue(3c)`, `tpgetrply(3c)`, `tprealloc(3c)`, **Introduction to FML Functions**, `Fadd`, `Fadd32(3fml)`, `Fchg`, `Fchg32(3fml)`, `Ffind`, `Ffind32(3fml)`, `MIB(5)`, `TM_MIB(5)`

Setting Up a BEA Tuxedo Application

Administering a BEA Tuxedo Application at Run Time

Programming a BEA Tuxedo Application Using C

Programming a BEA Tuxedo Application Using FML

AUTHSVR(5)

- Name** AUTHSVR - 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.
- The rcode parameter of tpreturn(3c) is used to set the application key. It is returned (in tpurcode) to the code that has called tpinit(3c) upon either successful validation or permission failure.

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 using the `AUTHSVC` parameter in the `RESOURCES` section of the `UBBCONFIG` file. For example, the following `AUTHSVC` parameter setting specifies the authentication service (`AUTHSVC`) advertised by `AUTHSVR` when `SECURITY` is set to `USER_AUTH`.

```
*RESOURCES
SECURITY    USER_AUTH
AUTHSVC     AUTHSVC
```

If the `AUTHSVC` parameter is not specified, the authentication service defaults to `AUTHSVC`.

By default, the file `tpusr` in the directory referenced by the first path name defined in the application's `APPDIR` variable 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()`, `tpdelusr()`, and `tpmodusr()`. 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()`, `tpusrdel()`, and `tpusrmod()` programs to maintain the user file.

Note: To use `tpusradd()`, `tpusrdel()`, and `tpusrmod()`, `SECURITY` for the target application must be set to `USER_AUTH`, `ACL`, or `MANDATORY_ACL`. Otherwise, the system returns an error when you attempt to use these programs.

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 (for example, 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 can be configured for the application using the `AUTHSVC` parameter in the `RESOURCES` section of the `UBBCONFIG` file. For example, the following `AUTHSVC` parameter setting specifies the authentication service (`..AUTHSVC`) advertised by `AUTHSVR` when `SECURITY` is set to `ACL` or `MANDATORY_ACL`.

```
*RESOURCES
SECURITY    ACL
AUTHSVC     ..AUTHSVC
```

If the `AUTHSVC` parameter is not specified, the authentication service defaults to `..AUTHSVC`.

Note: `AUTHSVR` advertises the authentication service as `AUTHSVC` when `SECURITY` is set to `USER_AUTH`, and as `..AUTHSVC` when `SECURITY` is set to `ACL` or `MANDATORY_ACL`. `AUTHSVC` and `..AUTHSVC` point to the same authentication service.

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()`, `tpusrdel()`, and `tpusrmod()` 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.

Section 5 - File Formats and Data Descriptions

For SECURITY_ACL or MANDATORY_ACL, you must use the standard AUTHSVR shipped as part of the system in `${TUXDIR}/bin/AUTHSVR`.

- 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 USER_AUTH
*RESOURCES
SECURITY USER_AUTH
AUTHSVC AUTHSVC

*SERVERS
AUTHSVR SRVGRP="AUTH" CLOPT="-A -- -f /usr/tuxedo/users" \
 SRVID=100 RESTART=Y GRACE=0 MAXGEN=2
#
#
Using ACLs
*RESOURCES
SECURITY ACL
AUTHSVC ..AUTHSVC

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

*SERVERS
KERBEROSSVR SRVGRP="AUTH1" SRVID=100 RESTART=Y GRACE=0 MAXGEN=2
```
- See Also**    `tpadduser(1)`, `tpusradd(1)`, `UBBCONFIG(5)`
- Setting Up a BEA Tuxedo Application*
- Administering a BEA Tuxedo Application at Run Time*
- Programming a BEA Tuxedo Application Using C*

# compilation(5)

**Name** `compilation` - instructions for compilation of 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, programmers need to know:

- Which header files to include, and the order in which to specify them
- Which environment variables to set and export
- Which utilities are used to compile the application modules

A programmer who has finished writing code modules and is ready to build an executable program must:

- Compile the source files
- Link edit the executables with the required libraries

The BEA Tuxedo system provides two commands that perform both of these operations for client and server modules: `buildclient()` and `buildserver()`, respectively. If you run one of these commands to perform both operations, be sure to specify, on the command line, the libraries with which your files need to be link edited. (For details, see `buildclient(1)` or `buildserver(1)` in the *BEA Tuxedo Command Reference*.)

Link editing must be done by running `buildclient` or `buildserver`, but the system allows more flexibility about how compiling is done. If you prefer, you can use the compile command of your choice to compile your files, and then run `buildclient` or `buildserver` to perform the link editing.

This rest of this reference page specifies the header files and environment variables required for various types of programs.

**Basic BEA  
Tuxedo System**

**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`.

## Section 5 - File Formats and Data Descriptions

---

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| Environment Variables | The following environment variables should be set and exported:                                      |
|                       | TUXDIR<br>Specifies the topmost directory in which the BEA Tuxedo system software resides            |
|                       | PATH<br>Should include \$TUXDIR/bin                                                                  |
|                       | ULOGPFX<br>Prefix of the filename of the central event log; by default, the value of ULOGPFX is ULOG |

---

| <b>If . . .</b>                                                                                                                                                                                                                                                                                                                                   | <b>Then you must first set and export the following environment variables . . .</b>                                                                                                                                                                            |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| You want to run <ul style="list-style-type: none"><li>■ buildclient(1)</li><li>■ buildserver(1)</li></ul>                                                                                                                                                                                                                                         | <ul style="list-style-type: none"><li>■ TUXDIR - always required for servers; also required for native clients</li><li>■ CC - if you want to use a non-default compiler</li><li>■ CFLAGS - if you want to specify flags to be passed to the compiler</li></ul> |
| A default or validation routine references FML fields                                                                                                                                                                                                                                                                                             | <ul style="list-style-type: none"><li>■ FIELDTBLS - a comma-separated list of field table files</li><li>■ FLDTBLDIR - a colon-separated list of directories to search for the FIELDTBLS</li></ul>                                                              |
| You want to execute a server                                                                                                                                                                                                                                                                                                                      | TUXCONFIG - full path name of the binary configuration file (default is the current directory)                                                                                                                                                                 |
| <ul style="list-style-type: none"><li>■ Security is turned on in your application</li><li>■ You are going to supply input indirectly (that is, from a source other than standard input) for any of the following system-supplied clients: tmadmin(1), tmconfig or wtmconfig (see tmconfig, wtmconfig(1)), or ud or wud (see ud, wud(1))</li></ul> | <ul style="list-style-type: none"><li>■ APP_PW - application password</li><li>■ USR_PW - user password</li></ul>                                                                                                                                               |

---

|               |                                                                                   |
|---------------|-----------------------------------------------------------------------------------|
| <b>If ...</b> | <b>Then you must first set and export the following environment variables ...</b> |
|---------------|-----------------------------------------------------------------------------------|

---

- |                                          |                                                                                                                                                                                                                       |
|------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| You want to execute a Workstation client | <ul style="list-style-type: none"> <li>■ WSENVFILE - file containing environment variable settings</li> <li>■ WSDEVICE - network device to use for connection</li> <li>■ WSTYPE - workstation machine type</li> </ul> |
|------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
- 

**Note:** More information about these variables can be found in *Programming a BEA Tuxedo Application Using C*, *Programming a BEA Tuxedo Application Using COBOL*, and *Setting Up a BEA Tuxedo Application*.

After the system has been built with shared libraries and before you execute a client, you must set a variable that defines the location of the shared libraries.

---

|                             |                                                   |
|-----------------------------|---------------------------------------------------|
| <b>On this platform ...</b> | <b>Set the following environment variable ...</b> |
|-----------------------------|---------------------------------------------------|

---

|                                    |                              |
|------------------------------------|------------------------------|
| All platforms except HP-UX and AIX | LD_LIBRARY_PATH=\$TUXDIR/lib |
| HP-UX                              | SHLIB_PATH=\$TUXDIR/lib      |
| AIX                                | LIBPATH=\$TUXDIR/lib         |

---

**Note:** 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 following order.

```
#include <UNIX_header_files> (if needed by the application)
#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 -lengine -o pgm
where pgm is the name of the executable file.
```

## Section 5 - File Formats and Data Descriptions

---

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/libengine.a -o pgm
```

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

Compiling FML VIEWS To use the FML view compiler, execute the following.

```
viewc view_file
```

Here *view\_file* is a set of 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`

A comma-separated list of field table files

`FLDTBLDIR`

A colon-separated list of directories to search for the `FIELDTBLS`

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

`FIELDTBLS`

A comma-separated list of field table files

`FLDTBLDIR`

A colon-separated list of directories to search for the `FIELDTBLS`

`VIEWDIR`

A directory containing view files; the default is the current directory

Portability The `buildclt(1)` command must be used in place of the `buildclient(1)` command on Windows 3.1 and AS/400 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)`, `viewc`, `viewc32(1)`  
`cc(1)`, `mc(1)` in a UNIX system reference manual

# DMADM(5)

|                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name             | DMADM - Domains administrative server                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Synopsis         | DMADM SRVGRP = " <i>identifier</i> "<br>SRVID = " <i>number</i> "<br>REPLYQ = "N"                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Description      | <p>The Domains administrative server (DMADM) is a BEA Tuxedo system-supplied server that provides run-time access to the BDMCONFIG file.</p> <p>DMADM is described in the SERVERS section of the UBBCONFIG file as a server running within a group, for example, 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").</p> <p>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.</p> <p>The BDMCONFIG environment variable should be set to the path name of the file containing the binary version of the DMCONFIG file.</p> |
| Portability      | DMADM is supported as a BEA Tuxedo system-supplied server on UNIX System platforms.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Interoperability | DMADM must be installed on BEA Tuxedo Release 5.0 or later; other machines in the same domain with a Release 5.0 gateway may be Release 4.1 or later.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Examples         | <p>The following example illustrates the definition of the administrative server and a gateway group in the UBBCONFIG file. This example uses the GWTDOMAIN gateway process to provide connectivity with another BEA Tuxedo domain. To provide interoperability with a BEA TOP END system, use the GWTOPEND gateway process. For details on the GWTOPEND gateway process and an example using GWTOPEND, see GWTOPEND(5).</p>                                                                                                                                                                                                                                                                                                                 |

```
#
*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)`, `DMCONFIG` for `GWTOPEND(5)`, `GWADM(5)`, `GWTOPEND(5)`, `servopts(5)`, `UBBCONFIG(5)`

*Setting Up a BEA Tuxedo Application*

*Administering a BEA Tuxedo Application at Run Time*

*Using the BEA Tuxedo TOP END Domain Gateway*

# DMCONFIG(5)

Name DCONFIG - text version of a Domains configuration file

Description A Domains configuration is a set of two or more domains (or applications) that can communicate and share services with the help of the BEA Tuxedo Domains feature. How multiple domains are connected and which services they make accessible to each other are defined in a Domains configuration file on each domain. The text version of a Domains configuration file is known as the DCONFIG file (after the environment variable used to hold the name of the actual file used).

A DCONFIG file defines the following:

- The remote domains with which the local domain can communicate
- The local resources (such as services and queues) accessible to remote domains
- The remote resources accessible to the local domain
- Which local and remote resources are accessible through which gateways

The DCONFIG file is parsed and loaded into a binary version, called BDCONFIG, by the `dmloadcf(1)` utility. The `dmadmin(1)` command uses BDCONFIG (or a copy of it) for monitoring the run-time application.

One BDCONFIG file is required on each domain in a multi-domain configuration in which the Domains feature is being used.

The DCONFIG and BDCONFIG files are analogous to the UBBCONFIG and TUXCONFIG files used to define a BEA Tuxedo application.

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_RESOURCES`, `DM_ROUTING`, `DM_ACCESS_CONTROL`, and `DM_domtype`, where *domtype* is one of the following: OSITP, SNAX, TDOMAIN, or TOPEND. The `DM_LOCAL_DOMAINS` section must precede the `DM_REMOTE_DOMAINS` section.

**Note:** This reference page describes how to configure a domain of only one type: TDOMAIN. See BEA eLink documentation for information about how to configure an OSITP or an SNAX domain. For information about how to configure a TOP END Domain Gateway, see `DMCONFIG` for `GWTOPEND(5)` and *Using the BEA Tuxedo TOP END Domain Gateway*.

Parameters are generally specified by: `KEYWORD = value`; white space (space or tab character) is allowed on either side of the equal sign (=). This format sets `KEYWORD` to `value`. Valid keywords are described below within each section.

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* (a string value already known to the BEA Tuxedo Domains feature such as TDOMAIN for the TYPE parameter), standard C rules are typically used. A standard C *identifier* starts with an alphabetic character or underscore and contains only alphanumeric characters or underscores. The maximum allowable length of an identifier is 30 (not including the terminating null).

There is no need to enclose an identifier in double quotes. A value that is neither an integer number nor 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 cannot be continued.

### Domains Terminology Improvements

In this release, some of the domains terminology is changing. The Domains MIB uses improved class and attribute terminology to describe the interaction between local and remote domains. While this improved terminology is more accurate than previous domains terminology, the scope of changes to domains-related documentation and error messages is limited in this release. The improved terminology has been applied to the DM\_MIB classes, reference page, and error messages, the DMCONFIG file syntax, and various DMCONFIG error messages.

For backwards compatibility, aliases are provided between the DMCONFIG terminology used prior to this release and the improved Domains MIB terminology. In this release, DMCONFIG accepts both versions of the terminology. For details, see “Domains Terminology Improvements” on page -122 in the DM\_MIB(5) reference page.

## 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, OSITP or TOPEND. 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. The TOPEND value indicates that this local domain can communicate only with a BEA TOP END system. 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 `DMmmddy.y.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\_PRINCIPAL\_NAME = *string*[0..511]

specifies the connection principal name identifier, which is the principal name for verifying the identity of this local domain when establishing a connection to a remote domain. This parameter only applies to domains of type `TDOMAIN` that are running BEA Tuxedo 7.1 or later software.

The `CONNECTION_PRINCIPAL_NAME` parameter may contain a maximum of 511 characters (excluding the terminating null character). If this parameter is not specified, the connection principal name defaults to the `DOMAINID` string for this local domain.

For default authentication plug-ins, if a value is assigned to the `CONNECTION_PRINCIPAL_NAME` parameter for this local domain, it must be the same as the value assigned to the `DOMAINID` parameter for this local domain. If these values do not match, the local domain gateway process will *not* boot, and the system will generate the following `userlog(3c)` message: `ERROR: Unable to acquire credentials.`

`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.

## DM\_REMOTE\_DOMAINS Section

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

Entries have the form:

```
RDOM required_parameters [optional_parameters]
```

where *RDOM* is an *identifier* value used to identify each remote domain known to this configuration. *RDOM* must be unique within the configuration.

The following are required parameters:

```
TYPE = identifier
```

is used for grouping remote domains into classes. TYPE can be set to one of the following values: TDOMAIN, SNAX, OSITP, or TOPEND. The TDOMAIN value indicates that this remote domain can communicate only 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. The TOPEND value indicates that this remote domain is a BEA TOP END system and can communicate only with a local domain of type TOPEND.

```
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".

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

```
ACL_POLICY = {LOCAL | GLOBAL}
```

specifies the access control list (ACL) policy for this remote domain. This parameter only applies to domains of type TDOMAIN that are running BEA Tuxedo 7.1 or later software.

LOCAL means that the local domain modifies the identity of service requests received from this remote domain to the principal name specified in the LOCAL\_PRINCIPAL\_NAME parameter for this remote domain. GLOBAL means that the local domain passes the service requests with no change. If not specified, the default is LOCAL.

`CONNECTION_PRINCIPAL_NAME = string[0..511]`

specifies the connection principal name identifier, which is the principal name for verifying the identity of this remote domain when establishing a connection to the local domain. This parameter only applies to domains of type `TDOMAIN` that are running BEA Tuxedo 7.1 or later software.

The `CONNECTION_PRINCIPAL_NAME` parameter may contain a maximum of 511 characters (excluding the terminating null character). If this parameter is not specified, the connection principal name defaults to the `DOMAINID` string for this remote domain.

For default authentication plug-ins, if a value is assigned to the `CONNECTION_PRINCIPAL_NAME` parameter for this remote domain, it must be the same as the value assigned to the `DOMAINID` parameter for this remote domain. If these values do not match, any attempt to set up a connection between the local domain gateway and the remote domain gateway will fail, and the system will generate the following `userlog(3c)` message: `ERROR: Unable to initialize administration key for domain domain_name.`

`LOCAL_PRINCIPAL_NAME = string[0..511]`

The local principal name identifier, which is the identity assigned by the local domain to service requests received from this remote domain. This parameter only applies to domains of type `TDOMAIN` that are running BEA Tuxedo 7.1 or later software.

The `LOCAL_PRINCIPAL_NAME` parameter is valid only if the `ACL_POLICY` parameter for this remote domain is set (or defaulted) to `LOCAL`. The `LOCAL_PRINCIPAL_NAME` parameter may contain a maximum of 511 characters (excluding the terminating null character). If this parameter is not specified, the local principal name defaults to the `DOMAINID` string for this remote domain.

`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.

## 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 associated 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. The value of *string* may also be represented in either of the following 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 string *#.#.#.#* 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.

**Note:** Some port numbers may be reserved for the underlying transport protocols (such as TCP/IP) used by your system. Check the documentation for your transport protocols to find out which numbers, if any, are reserved on your system.

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 parameter defaults to 2,147,483,647.

MINENCRYPTBITS = { 0 | 40 | 56 | 128 }

Specifies the minimum level of encryption required when a link for this domain is being established. 0 means no encryption, while 40, 56, and 128 specify the encryption key length (in bits). If this minimum level of encryption cannot be met, link establishment fails. The default is 0.

**Note:** The link-level encryption value of 40 bits is provided for backward compatibility.

MAXENCRYPTBITS = { 0 | 40 | 56 | 128 }

Specifies the maximum level of encryption allowed when a link for this domain is being established. 0 means no encryption, while 40, 56, and 128 specify the encryption length (in bits). The default is 128.

**Note:** The link-level encryption value of 40 bits is provided for backward compatibility.

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.

## 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* used to specify an access control list; it may contain no more than 15 characters.

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.

## 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 value used to specify the 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.

## 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.

## DM\_RESOURCES

This optional section is used for defining global Domains configuration information, specifically a user-supplied configuration version string.

The only parameter in this section is

`VERSION = string`

where *string* is a field in which users can enter a version number for the current Domains configuration file. This field is not checked by the software.

## DM\_ROUTING Section

This section provides information for data-dependent routing of service requests using FML, FML32, XML, VIEW, VIEW32, 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 in 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. It is assumed that the value of *identifier* is one of the following: a field name that is identified in an FML field table (for FML and FML32 buffers); an XML element or element attribute (for XML buffers); or an FML view table (for VIEW, X\_C\_TYPE, or X\_COMMON buffers). Two environment variables — FLDTBLDIR and FIELDTBLS, or FLDTBLDIR32 and FIELDTBLS32 are used to locate FML field tables. Similarly, two environment variables VIEWDIR and VIEWFILES, or VIEWDIR32 and VIEWFILES32 are used to locate FML view tables. If a field in an FML or FML32 buffer is used for routing, the value of that field must be a number less than or equal to 8191.

When XML documents are being routed on the basis of element content or element attribute, the `FIELD` parameter must be defined with the following syntax.

```
FIELD = "root_element[/child_element][/child_element][/. . .][/@attribute_name]"
```

The value of `FIELD` specifies the name of a routing element or an element attribute. It is assumed that the value of *root\_element* is an element type (or name) or an element attribute name for an XML document or datagram. This information is used to identify the element content or element attribute value for data-dependent routing while sending a document or datagram. The element name and attribute name combined may contain no more than 30 characters. Because indexing is not supported, the BEA Tuxedo system recognizes only the first occurrence of a given element type when processing an XML buffer for data-dependent routing.

XML strictly defines the set of characters that may be used in an attribute name. An attribute name must be a string consisting of a single letter,

underscore, or colon, followed by one or more name characters. Both element names and attribute names are case-sensitive.

You can find more information about XML on the World Wide Web Consortium Web site at <http://www.w3c.org/XML>.

`FIELDTYPE = type`

indicates the type of routing field specified in the `FIELD` parameter. This parameter is used only for routing XML buffers. The value `type` can be set to one of the following: `CHAR`, `SHORT`, `LONG`, `FLOAT`, `DOUBLE`, or `STRING`. The default type of the routing field is `STRING`.

`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 arrays, 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 arrays, 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).

A numeric routing field must have numeric range values and a string routing field must have string range values. String range values for string, array, and

character field types must be placed inside a pair of single quotes and cannot 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 ":".

An XML element content and attribute value must be encoded in UTF-8 and can be used for routing if it can be converted to the data type specified by the FIELDTYPE parameter.

When used for routing, the element content cannot contain character references, entity references, or CDATA sections.

An XML attribute value (encoded in UTF-8) can be used for routing if the element to which the attribute belongs is defined.

```
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 either FML, FML32, XML, VIEW, VIEW32, X\_C\_TYPE, or X\_COMMON. No subtype can be specified for types FML, FML32, or XML; subtypes are required for types VIEW, VIEW32, X\_C\_TYPE, and X\_COMMON ("\*" is not allowed). Duplicate type/subtype pairs cannot 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 or FML32 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 Domains configuration. The example shows four Bank Branch domains communicating with a Central Bank Branch. Three of the Bank Branches run within other BEA Tuxedo domains. The fourth Branch runs under the control of another TP Domain. OSI TP is used for communication between that domain and the Central Bank. The example shows the Domains configuration file from the Central Bank point of view.

```
BEA 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.CENTRAL02"
 DMTLOGDEV = "/usr/apps/bank/DMTLOG"
 DMTLOGNAME = "DMTLG_C02"

#
*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"
```

## Section 5 - File Formats and Data Descriptions

---

```
*DM_TDOMAIN
#
<local or remote domain name> <network address> [<nwdevice>]
#
Local network addresses
c01 NWADDR = "//newyork.acme.com:65432" 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 = b02, b03 LDOM = c02
```

```
*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 Domains configuration file for one of the Bank Branches (BANK01).

```
#
#BEA 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"
```

## Section 5 - File Formats and Data Descriptions

---

```
*DM_ACCESS_CONTROL
#
central ACLIST = c01
```

**Network Addresses** Suppose the local machine on which a TDomain 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 TDomain 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 `155.2.193.18` is translated into a hexadecimal number. Thus the `155` becomes `9B`, `2` becomes `02` and so on.

**See Also** `dmadmin(1)`, `dmloadcf(1)`, `dmunloadcf(1)`, `tmboot(1)`, `tmshutdown(1)`, `DMADM(5)`, `DMCONFIG` for `GWTOPEND(5)`, `GWADM(5)`, `GWTDOMAIN(5)`

*Setting Up a BEA Tuxedo Application*

*Administering a BEA Tuxedo Application at Run Time*

*Using the BEA Tuxedo Domains Component*

*Programming a BEA Tuxedo Application Using C*

# DMCONFIG for GWTOPEND(5)

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name        | DMCONFIG for GWTOPEND - text version of a Domains configuration file for a TOP END Domain Gateway                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Description | <p>A Domains configuration is a set of two or more domains (or applications) that can communicate and share services with the help of the BEA Tuxedo Domains feature. How multiple domains are connected and which services they make accessible to each other are defined in a Domains configuration file on each domain. The text version of a Domains configuration file is known as the DMCONFIG file (after the environment variable used to hold the name of the actual file used).</p> <p>A DMCONFIG file defines the following:</p> <ul style="list-style-type: none"> <li>■ The remote domains with which the local domain can communicate</li> <li>■ The local resources (such as services and queues) accessible to remote domains</li> <li>■ The remote resources accessible to the local domain</li> <li>■ Which local and remote resources are accessible through which gateways</li> </ul> <p>The DMCONFIG file is parsed and loaded into a binary version, called BDMCONFIG, by the <code>dmloadcf(1)</code> utility. The <code>dmadmin(1)</code> command uses BDMCONFIG (or a copy of it) for monitoring the run-time application.</p> <p>One BDMCONFIG file is required on each domain in a multi-domain configuration in which the Domains feature is being used.</p> <p>The DMCONFIG and BDMCONFIG files are analogous to the UBBCONFIG and TUXCONFIG files used to define a BEA Tuxedo application.</p> |
| Definitions | <p>A BEA Tuxedo system domain <i>Application</i> 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.</p> <p>A <i>Gateway Group</i> is a collection of domain gateway processes that provide communication services with a specific type of TP Domain.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

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 several possible specification sections. Allowable section names are: `DM_LOCAL_DOMAINS`, `DM_REMOTE_DOMAINS`, `DM_LOCAL_SERVICES`, `DM_REMOTE_SERVICES`, `DM_TOPEND`, `DM_RESOURCES`, `DM_ROUTING`, and `DM_ACCESS_CONTROL`. Additional section names applicable only to other gateway types are: `DM_TDOMAINS`, `DM_OSITP`, `DM_SNACRM`, `DM_SNASTACKS`, and `DM_SNALINKS`. The `DM_LOCAL_DOMAINS` section must precede the `DM_REMOTE_DOMAINS` section.

**Note:** This reference page describes how to configure a domain gateway of only one type: `TOPEND`. For information about how to configure a domain of type `TDOMAIN`, see `DMCONFIG(5)`. See BEA eLink documentation for information about how to configure an `OSITP` or an `SNAX` domain.

Parameters are generally specified by: `KEYWORD = value`; white space (space or tab character) is allowed on either side of the equal sign (=). This format sets `KEYWORD` to `value`. Valid keywords are described below within each section.

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 is given under the description of that parameter.

If a value is an *identifier* (a string value already known to the BEA Tuxedo Domains feature such as `TOPEND` for the `TYPE` parameter), standard C rules are typically used. A standard C *identifier* starts with an alphabetic character or underscore and contains only alphanumeric characters or underscores. The maximum allowable length of an identifier is 30 (not including the terminating null).

There is no need to enclose an identifier in double quotes. A value that is neither an integer number nor 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 cannot be continued.

#### Domains Terminology Improvements

In this release, some of the domains terminology is changing. The Domains MIB uses improved class and attribute terminology to describe the interaction between local and remote domains. While this improved terminology is more accurate than previous domains terminology, the scope of changes to domains-related documentation and error messages is limited in this release. The improved terminology has been applied to the `DM_MIB` classes, reference page, and error messages, the `DMCONFIG` file syntax, and various `DMCONFIG` error messages.

For backwards compatibility, aliases are provided between the `DMCONFIG` terminology used prior to this release and the improved Domains MIB terminology. In this release, `DMCONFIG` accepts both versions of the terminology. For details, see "Domains Terminology Improvements" on page -122 in the `DM_MIB(5)` reference page.

## 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. The entry defines a GWTOPEND instance (with its associated GWADM) that is associated with a single BEA TOP END system. The local domain communicates with remote domains of type TOPEND that are part of the same BEA TOP END system. The BEA TOP END system name is defined in the DM\_TOPEND section.

Entries have the form:

*L*DOM *required\_parameters* [*optional\_parameters*]

where *L*DOM is an *identifier* value used to name the local domain. *L*DOM must be unique within a particular configuration. As described in the DM\_LOCAL\_SERVICES section, *L*DOM is the identifier that associates the local and remote 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: TOPEND, TDOMAIN, SNAX, or OSITP. The TOPEND value indicates that this local domain can communicate only with a BEA TOP END system. The TDOMAIN value indicates that this local domain can communicate only 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. For a local domain of type TOPEND, this value is used as the BEA TOP END user ID for requests made to a BEA TOP END system. The associated password can be entered using the `dmadmin(1)`

subcommand `topendpasswd`. The BEA TOPEND user ID is 1-12 characters excluding any trailing null. ASCII characters ranging from “ ”(32) through “~” (126), excluding “/” (47) are valid for this string.

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` file are always 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`, and `INCOMING_ONLY`. A connection policy of `ON_DEMAND` means that a connection is 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`. Multiple entries for a remote domain may be specified in the `DM_TOPEND` section if you want to configure multiple network addresses to be tried serially to connect to the remote domain. Connection retry processing is not allowed when the connection policy is `ON_DEMAND`.

A connection policy of `ON_STARTUP` means that a domain gateway attempts to establish a connection with its remote domains at gateway server initialization time. Multiple entries for a remote domain may be specified in the `DM_TOPEND` section if you want to configure multiple network addresses

to be tried serially to connect to the remote domain. If `CONNECTION_POLICY` is set to `ON_STARTUP`, then remote services (that is, services advertised by the local domain gateway) are 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 are suspended. By default, this connection policy retries failed connections (all network addresses tried) 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 does not attempt an initial connection to remote domains upon starting and remote services are initially suspended. The domain gateway is available for incoming connections from remote domains or a connection can be attempted when requested due to an administrative “connect” command. Remote services are advertised when the local domain gateway receives an incoming connection or an administrative connection is made. Multiple entries for a remote domain may be specified in the `DM_TOPEND` section if you want to configure multiple network addresses to be tried serially only on an administrative connect to the remote domain. Connection retry processing is not allowed when the connection policy is `INCOMING_ONLY`.

`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.

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 tries to establish connections to remote domains. The minimum value is 0 and the maximum is MAXLONG. MAXLONG indicates that retry processing is 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.

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.

RETRY\_INTERVAL = *numeric*

specifies the number of seconds between automatic attempts (all network addresses tried) 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.

SECURITY = *value*

specifies the type of application security to be enforced. The SECURITY parameter currently has four valid values for domains of type TOPEND: NONE, CLEAR, SAFE, or PRIVATE. The value NONE is the default and specifies that BEA TOP END security is not used by the system for authentication, authorization, or protection of inter-node messages. A value other than NONE specifies that BEA TOP END authentication and authorization are used by the BEA TOP END system and the gateway. In addition, the value CLEAR specifies that no protection is required for inter-node messages. The value SAFE indicates that messages should be sent using the Kerberos SAFE

message checksum. The value `PRIVATE` indicates that messages should be encrypted using the Kerberos 4 implementation of DES. The value of `SECURITY` must be consistent with the corresponding BEA TOP END Node Manager configuration in the `nm_config (4T)` file on each BEA TOP END node. This is validated when a connection is established with a remote BEA TOP END node.

## DM\_REMOTE\_DOMAINS Section

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

For TOP END Domain Gateway (TEDG) definitions, this section defines connections to Network Interface components on nodes of remote BEA TOP END systems.

Entries have the form:

```
RDOM required_parameters
```

where *RDOM* is an *identifier* value used to identify each remote domain known to this configuration. *RDOM* must be unique within the configuration.

Each *RDOM* defines a node in the BEA TOP END system to which a BEA TOP END *LDOM* may have a connection. The *LDOM* communicates with remote domains of type TOPEND that are part of the same BEA TOP END system as the *LDOM*. (The BEA TOP END system name is defined in the DM\_TOPEND section.) Because of the BEA TOP END adjacent node routing topology, the services for the BEA TOP END system may reside on several different nodes. Therefore, a TEDG *LDOM* may need several *RDOM* entries to define connections to the BEA TOP END nodes where the desired BEA TOP END services reside.

The TYPE and DOMAINID parameters are required; MTYPE is not used for remote domains of type TOPEND:

```
TYPE = identifier
```

is used for grouping remote domains into classes. TYPE can be set to one of the following values: TOPEND, TDOMAIN, SNAX or OSITP. The TOPEND value indicates that this remote domain is a BEA TOP END system and can communicate only with a local domain of type TOPEND. The TDOMAIN value indicates that this remote domain can communicate only 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". The DOMAINID value is used by the BEA TOP END

gateway as the BEA Tuxedo user ID for requests made to the BEA Tuxedo system by the BEA TOP END system on this *RDOM* connection.

## DM\_TOPEND Section

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

Local and remote domains and their network addresses must be configured such that no more than one BEA TOP END gateway connection to a BEA TOP END node is activated for a particular TP\_SYSTEM name at runtime. The BEA TOP END network interface protocol does not support multiple gateway connections. If an attempt is made to activate more than one connection, runtime errors occur in the TEDG or on the BEA TOP END node, and all connections except one are rejected. Due to variations in how network addresses can be specified, this type of configuration cannot be fully validated in this configuration file.

The following parameters are required:

```
NWADDR = string
```

specifies the network address associated with a local domain or a remote domain. If the association is with a local domain, the NWADDR is used to accept connections from BEA TOP END systems. 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 Network Interface components of the BEA TOP END system.

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. They may also be written in either of the following two forms:

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

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

In the first of these formats, *host.name* 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.

The value of *port\_number* is a decimal number in the range 0 to 65535, the hexadecimal representations of the string specified.

**Note:** Some port numbers may be reserved for the underlying transport protocols (such as TCP/IP) used by your system. Check the documentation for your transport protocols to find out which numbers, if any, are reserved on your system.

This parameter specifies the network port address used by a local or a remote domain to accept connections. If *string* has the form “0xhex-digits”, it must contain an even number of valid hexadecimal digits.

If the administrator wishes to specify `INADDR_ANY` for an *LDOM* listening address such as the one used by the BEA TOP END Network Interface, the format should be “//0.0.0.0:port\_number”. When an address is specified in this format, the TEDG (`GWTOPEND`) process can listen on *port\_number* for all available IP addresses on the machine.

**Note:** Care should be taken when specifying the host address portion of the `NWADDR` parameter. When a BEA TOP END NI accepts a connection request that was issued from a TEDG, it resolves the network address of the TEDG to a name. The resolved name must match the defined hostname of the TEDG. If the defined hostname of the TEDG and the resolved name differ, including case, the NI connection fails. Such a failure may not be evident from either the `GWTOPEND` log file or the remote BEA TOP END NI log file. As a general rule, ensure that the hostname definitions match in the `DMCONFIG` file, the TOP END NI configuration file, the TOP END nodemap file, the TOP END `tp_alias` file, and the locally configured name resolution facilities. For further information on NI name resolution, refer to the `tp_alias(4T)` reference page in the *BEA TOP END Programmer's Reference Manual*.

`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.

`TP_SYSTEM = string`

defines the BEA TOP END system associated with the *LDOM* or *RDOM* defined in the `DM_LOCAL_DOMAINS` and `DM_REMOTE_DOMAINS` sections of the configuration file. The parameter accepts a string that corresponds to the BEA TOP END system name. The BEA TOP END system name may contain from 1 to 8 characters, excluding any trailing null. ASCII characters ranging from “ ”(32) through “~” (126), excluding “/” (47) are valid for this string. The value of *string* must match the value of the `TP_SYSTEM` environment variable, which is defined in the `nm_script (4T)` file on the BEA TOP END system.

If the entry is for a BEA TOP END local domain (that is, if *DOM* matches a previously specified *LDOM*), then `NWADDR` is the network address to be used to listen for incoming connections. Only one entry may be configured for a local domain.

Multiple entries may be configured for a remote domain to specify network addresses to be tried serially on a connection attempt. The first one specified is considered 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. Each subsequent entry is used if all previous entries have failed. A connection attempt fails when all configured network addresses have been tried. Entries associated with a remote domain can be specified an unlimited number of times. Configuring too many network addresses or addresses that may not be operational can degrade performance.

If the entry is for a secondary remote domain (that is, if *DOM* matches a previously specified *RDOM*), then the entry is used only when a network connection cannot be established using the `NWADDR` of the primary entry (and any prior secondary entries). For every secondary entry:

- The value of `TP_SYSTEM` must match the value of `TP_SYSTEM` for the primary remote gateway entry.
- The entry must include a reference to an alternate network connection to the same node to which the primary remote domain is connected.

Secondary remote gateway definitions are not recommended for use with TOP END Domain Gateways.

## DM\_ACCESS\_CONTROL Section

This section specifies the access control lists used by the local domain.

Lines in this section are of the form:

*ACL\_NAME* *required\_parameters*

where *ACL\_NAME* is an *identifier* used to specify an access control list; it may contain no more than 15 characters.

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.

## DM\_LOCAL\_SERVICES Section

This section defines the mapping information required to make BEA Tuxedo services and /Q queue spaces available to BEA TOP END systems. In DMCNFIG files written for domain gateways other than the TEDG, the purpose of entries in this section is to map local services to remote names for those services. In DMCNFIG files written for the TEDG, however, the type entry is used to define request/reply and conversational service mapping. Additionally, similar entries are accepted in this section for defining BEA Tuxedo queue space mapping and queue name mapping. Entries may be for a SERVICE, QSPACE or QNAME and are identified by the TYPE parameter. This section is required for TOP END domain gateways.

Lines within this section have one of these forms:

```
service [TYPE=SERVICE]required_parameters [optional_parameters]
qspace TYPE=QSPACE required_parameters [optional_parameters]
qname TYPE=QNAME required_parameters [optional_parameters]
```

where *service* is the name of an exported BEA Tuxedo service, *qspace* is the name of an exported BEA Tuxedo queue space, and *qname* is the name of a queue name defined within a BEA Tuxedo queue space. Each of these names may contain no more than 15 characters.

SERVICE entries define BEA Tuxedo services that are advertised (product, function, target) to the BEA TOP END system by the TEDG. Entries for BEA Tuxedo services that are advertised to a BEA TOP END system must include a mapping from BEA TOP END service identifiers (product, function, target, qualifier) to BEA Tuxedo service names. These service identifiers are used with the BEA TOP END `tp_client_send(3T)` and `tp_client_signon(3T)` routine calls.

QSPACE entries in this section define BEA Tuxedo queue spaces that are made available to BEA TOP END as RTQ queues (limitations apply). RTQ queues are made available in BEA TOP END by advertising the RTQ Group name, RTQ Queue name, and target name as a BEA TOP END service name. The BEA TOP END gateway handles `tp_rtq_put(3T)` requests sent to its RTQ queue names in a manner similar to that used by the RTQ server. Each request is then mapped to the BEA Tuxedo queue space identified in this QSPACE entry. Both QSPACE entries and QNAME entries are required for message queuing.

QNAME entries define the mapping of a BEA TOP END service request to a BEA Tuxedo queue name for requests enqueued to the BEA Tuxedo system via RTQ. QNAME entries are not advertised as services to the BEA TOP END system. QSPACE and

QNAME entries are independent. Any combination of QSPACE and QNAME identifiers may be used by an application by supplying the associated BEA TOP END identifiers with a `tp_rtq_put(3T)` routine call. A run-time error results if the combination does not exist in the local BEA Tuxedo domain.

QNAME entries should be unique with respect to their product, function, target, and qualifier combination for a particular LDOM. If multiple entries of the same combination are configured, the TEDG uses only the first one.

Any SERVICE or QNAME entry that includes the `TE_PRODUCT` parameter, or any QSPACE entry that includes the `TE_RTQGROUP` parameter, is applicable to all local domains of type `TOPEND` if the entry is not configured for a particular local domain using the `LDOM` parameter. Entries configured for a specific LDOM are applicable only to the gateway for that domain.

Because SERVICE and QSPACE entries configure BEA TOP END service identifiers that are advertised as BEA TOP END services, these identifiers must not overlap for a particular LDOM. For a SERVICE entry, the `TE_PRODUCT`, `TE_FUNCTION`, and `TE_TARGET` are advertised. For a QSPACE entry, the `TE_RTQGROUP`, `TE_RTQNAME`, and `TE_TARGET` are advertised as product, function, and target identifiers. Therefore if a SERVICE entry product, function, and target match a QSPACE entry RTQ Group, RTQ Queue name and target, the TEDG cannot route the request. Note that, as in the BEA TOP END system, the default value for the target is the truncated node name.

If the configuration includes `LDOMS` for more than one BEA TOP END system, or if it includes multiple gateway types, the `LDOM` parameter should be specified in the local service entry. Mixed configurations that do not specify `LDOM` should not be created; they may prevent a gateway from initializing properly. If in doubt, explicitly set `LDOM`.

The following are the required and optional parameters for each entry type:

| Entry TYPE | Required Parameters                                                       | Optional Parameters                                                                                                                                                    |
|------------|---------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SERVICE    | <code>TE_PRODUCT</code> ,<br><code>TE_FUNCTION</code>                     | <code>TYPE</code> , <code>ACL</code> , <code>LDOM</code> , <code>INBUFTYPE</code> ,<br><code>OUTBUFTYPE</code> , <code>TE_TARGET</code> ,<br><code>TE_QUALIFIER</code> |
| QSPACE     | <code>TYPE</code> , <code>TE_RTQGROUP</code> ,<br><code>TE_RTQNAME</code> | <code>ACL</code> , <code>LDOM</code> , <code>TE_TARGET</code>                                                                                                          |
| QNAME      | <code>TYPE</code> , <code>TE_PRODUCT</code> ,<br><code>TE_FUNCTION</code> | <code>LDOM</code> , <code>INBUFTYPE</code> , <code>TE_TARGET</code> ,<br><code>TE_QUALIFIER</code>                                                                     |

The following are descriptions of both required and optional parameters.

`TYPE = SERVICE | QSPACE | QNAME`

Specifies the type of entry being defined. The value `SERVICE` specifies that the entry defines the mapping parameters applicable to a local BEA Tuxedo service being exported to the BEA TOP END system. The value `QSPACE` specifies that the entry defines the mapping parameters applicable to a local BEA Tuxedo queue space being made available to the BEA TOP END system as an RTQ queue. The value `QNAME` specifies that the entry defines the parameters applicable to mapping a BEA TOP END service name to a BEA Tuxedo queue name for requests enqueued to the BEA Tuxedo system via RTQ. The default value is `SERVICE`.

`TE_PRODUCT = string`

specifies the BEA TOP END product name, which may contain up to 32 characters, excluding the trailing null. Only the following characters are considered valid: a-z, A-Z, 0-9, \_, -, and . (period).

`TE_FUNCTION = string`

specifies the BEA TOP END function name, which may contain up to 8 characters, excluding the trailing null. Only the following characters are considered valid: a-z, A-Z, 0-9, \_, -, and . (period).

`TE_TARGET = string`

specifies the BEA TOP END Message Sensitive Routing (MSR) target. The value of *string* may contain up to 8 characters, excluding the trailing null. Only the following characters are considered valid: a-z, A-Z, 0-9, \_, -, and . (period). For `SERVICE` and `QSPACE` space entries, an asterisk is allowed as the last non-space character. The default value of the `TE_TARGET` parameter in the `DMCONFIG` file is spaces, indicating that it is not set. For `SERVICE` and `QSPACE` entries, the value of this parameter is changed at run time to default to the truncated node name of the TEDG. These values match the convention followed by the BEA TOP END system for default target names.

`TE_QUALIFIER = integer`

specifies the BEA TOP END function qualifier. Values in the following range are valid: 0-2147483647. The default is 0.

`TE_RTQGROUP = string`

specifies the BEA TOP END RTQ Group name. The value of *string* may contain up to 32 characters, excluding the trailing null. Only the following characters are considered valid: a-z, A-Z, 0-9, \_, -, and . (period).

`TE_RTQNAME = string`

specifies the BEA TOP END RTQ Queue name. The value of *string* may contain up to 8 characters, excluding the trailing null. Only the following characters are considered valid: a-z, A-Z, 0-9, \_, -, and .(period).

`ACL = identifier`

specifies the name of the access control list (ACL) to be used by the TEDG to restrict requests made to this `SERVICE` or `QSPACE` by BEA TOP END systems. The ACL is defined in the `DM_ACCESS_CONTROL` section of the `DMCONFIG` file. If this parameter is not specified, then access control is not performed for requests to this entry.

`LDOM = identifier`

specifies the name of the local domain that exports this service or queue space or to which the queue name entry applies. If this keyword is not specified then the entry is applicable to all the local domains of type `TOPEND` defined in the `DM_LOCAL_DOMAINS` section.

`INBUFTYPE = type[:subtype]`

restricts the input buffer type allowed for this service or queue name to a single buffer type. For BEA TOP END service and queue name entries, the valid values for *type* are: `FML32`, `CARRAY`, and `X_OCTET`.

`OUTBUFTYPE = type[:subtype]`

restricts the output buffer type accepted from this service to a single buffer type. For BEA TOP END service entries the valid values for *type* are: `FML32`, `CARRAY`, and `X_OCTET`.

## DM\_REMOTE\_SERVICES Section

This section defines the mapping information required to make BEA TOP END services, RTQ queues, and services accessed via RTQ available to BEA Tuxedo applications. In DMCONFIG files written for domain gateways other than the TEDG, the purpose of entries in this section is to map local services to remote names for those services.

In a DMCONFIG file written for the TEDG, however, the purpose of entries in this section is different.

- The `TYPE` parameter is used to define request/reply and conversational service mapping.
- Similar entries are used in this section for mapping BEA Tuxedo queue spaces and queue names.

Entries may be included for a `SERVICE`, `QSPACE`, or `QNAME`, and are identified by the `TYPE` parameter. This section is required for TOP END domain gateways.

Lines within this section have one of these forms:

```

service [TYPE=SERVICE]required_parameters [optional_parameters]
qspace TYPE=QSPACE required_parameters [optional_parameters]
qname TYPE=QNAME required_parameters [optional_parameters]

```

where *service* is the BEA Tuxedo service name assigned to the BEA TOP END service, *qspace* is the BEA Tuxedo queue space name assigned to the RTQ Queue, and *qname* is the BEA Tuxedo queue name assigned to a BEA TOP END service accessed through RTQ. Each of these names may contain 15 characters or fewer.

`SERVICE` entries define BEA TOP END services that are advertised to the BEA Tuxedo domain by the TEDG. Entries for BEA TOP END services that are advertised to a BEA Tuxedo application must map BEA Tuxedo service names to BEA TOP END service identifiers. These service names are used with the XATMI `tpcall(3c)` and `tpacall(3c)` functions.

`QSPACE` entries in this section define BEA TOP END RTQ queues that are made available in the BEA Tuxedo domain by the TEDG as if they were BEA Tuxedo queue spaces (limitations apply). A queue space is made available in the BEA Tuxedo system by advertising the queue space name as a BEA Tuxedo service name. The gateway handles a `tpenqueue` request sent to its queue space name in a manner similar to that

used by the `TMQUEUE` server. The request is then mapped to the `RTQ` queue identified in this `qspace` entry. Both `QSPACE` and `QNAME` entries are required for message queuing.

`QNAME` entries define the mapping of a BEA Tuxedo queue name to a BEA TOP END service name for requests enqueued to the BEA TOP END system. `QNAME` entries are not advertised as services to BEA Tuxedo systems. Note that `QSPACE` and `QNAME` entries are independent; any combination of `QSPACE` and `QNAME` identifiers may be used by an application with the `tpenqueue(3c)` function.

`QNAME` entries should be unique with respect to the queue name identifier for a particular `LDOM`. If multiple entries for the same identifier value are configured, the `TEDG` uses only the first one.

Any `SERVICE` or `QNAME` entry that includes the `TE_PRODUCT` parameter, or any `QSPACE` entry that includes the `TE_RTQGROUP` parameter, is applicable to each local domain of type `TOPEND` if the entry is not configured for a particular local domain using the `LDOM` parameter. Entries configured for a specific `LDOM` are also applicable to the gateway for that domain.

Because `SERVICE` and `QSPACE` entries configure service identifiers and queue space identifiers that are advertised as BEA Tuxedo services, these identifiers must not overlap for a particular `LDOM`. However, multiple entries of the same type and identifier are permitted for load balancing. All entries for the same service identifier must have the same value for the `CONV` parameter.

If the configuration includes `LDOMS` for more than one BEA TOP END system, or if it includes multiple gateway types, the `LDOM` parameter should be specified in the `DM_REMOTE_SERVICES` section of the `DMCONFIG` file. If an `RDOM` is specified in a remote services entry, or in a referenced routing entry, its value should match the value of the `LDOM` type (`TOPEND`) and `TP_SYSTEM`. Mixed configurations that do not specify `LDOM`, or that reference `RDOMS` of mixed types or mixed `TP_SYSTEMS` should not be created; they may prevent gateways from initializing properly. If in doubt, explicitly set `LDOM` and specify a remote domain (via the `RDOM` parameter or `ROUTING`). A “wildcard” specification for a remote domain should be used only when a single gateway type is defined.

The following are the required and optional parameters for each entry type.

| Entry TYPE | Required Parameters              | Optional Parameters                                                                                   |
|------------|----------------------------------|-------------------------------------------------------------------------------------------------------|
| SERVICE    | TE_PRODUCT,<br>TE_FUNCTION       | TYPE, LDOM, RDOM,<br>INBUFTYPE, OUTBUFTYPE,<br>CONV, TE_TARGET,<br>TE_QUALIFIER, TRANTIME,<br>ROUTING |
| QSPACE     | TYPE, TE_RTQGROUP,<br>TE_RTQNAME | LDOM, RDOM, TE_TARGET,<br>TRANTIME, ROUTING                                                           |
| QNAME      | TYPE, TE_PRODUCT,<br>TE_FUNCTION | LDOM, INBUFTYPE,<br>TE_TARGET, TE_QUALIFIER                                                           |

The following are descriptions of both required and optional parameters.

TYPE = SERVICE | QSPACE | QNAME

specifies the type of entry being defined. The value `SERVICE` specifies that the entry defines the mapping parameters needed to make a BEA TOP END service available as a local BEA Tuxedo service. The value `QSPACE` specifies that the entry defines the mapping parameters needed to make a BEA TOP END RTQ queue available as a local BEA Tuxedo queue space. The value `QNAME` specifies that the entry defines the parameters needed to map a BEA Tuxedo queue name to a BEA TOP END service name for requests enqueued to the BEA TOP END system via /Q. The default value is `SERVICE`.

TE\_PRODUCT = *string*

specifies the BEA TOP END product name, which may contain up to 32 characters, excluding the trailing null. Only the following characters are considered valid: a-z, A-Z, 0-9, \_, -, and . (period).

TE\_FUNCTION = *string*

specifies the BEA TOP END function name, which may contain up to 8 characters, excluding the trailing null. Only the following characters are considered valid: a-z, A-Z, 0-9, \_, -, and . (period).

TE\_TARGET = *string*

specifies the BEA TOP END Message Sensitive Routing (MSR) target, which may contain up to 8 characters, excluding the trailing null. Only the

following characters are considered valid: a-z, A-Z, 0-9, \_, -, and . (period). The default is spaces.

`TE_QUALIFIER = integer`

specifies the BEA TOP END function qualifier. Values in the following range are valid: 0-2147483647 (unsigned long max). The default is 0.

`TE_RTQGROUP = string`

specifies the BEA TOP END RTQ Group name, which may contain up to 32 characters, excluding the trailing null. Only the following characters are considered valid: a-z, A-Z, 0-9, \_, -, and . (period).

`TE_RTQNAME = string`

specifies the BEA TOP END RTQ Queue name, which may contain up to 8 characters, excluding the trailing null. Only the following characters are considered valid: a-z, A-Z, 0-9, \_, -, and . (period).

`LDOM = identifier`

specifies the name of the local domain that imports this service or RTQ queue, or to which a queue name entry applies. If this keyword is not specified then the entry is applicable to all the local domains of type `TOPEND` defined in the `DM_LOCAL_DOMAINS` section.

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

specifies the name of the remote domain providing this service or RTQ queue. The remote domain must be of type `TOPEND` and it must be part of the same `TP_SYSTEM` as the local domain to which this entry applies. If this parameter is not specified and a routing criteria is not specified, the local domain assumes that any remote domain of type `TOPEND` with the same `TP_SYSTEM` value as the local domain provides the service or RTQ queue.

If you want to configure alternate remote domains with the `identifier2` and `identifier3` arguments, then you must specify `ON_STARTUP` as the value of the `CONNECTION_POLICY` parameter. 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 domains specified by `identifier1` and `identifier2` are unavailable, the remote domain specified by `identifier3` is used.) Note that both load balancing (multiple remote service entries) and domains failover (using alternate remote domains specified with this parameter) may be used for a remote 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 this parameter is not specified, data-dependent routing is not used for this service. The value of *identifier* may contain up to 15 characters. If multiple entries for the same service or queue space name are included with different *RDOM* parameters, the value of the *ROUTING* parameter should be the same for all of these entries. Additionally, the remote domains configured (with *RDOM*) in the referenced routing criteria must be part of the same *TP\_SYSTEM* as the local domain to which this entry applies.

INBUFTYPE = *type[:subtype]*

restricts the input buffer type allowed for this service or queue name to a single buffer type. For BEA TOP END service and queue name entries, the valid values for *type* are: *FML32*, *CARRAY*, and *X\_OCTET*.

OUTBUFTYPE = *type[:subtype]*

restricts the output buffer type accepted from this service to a single buffer type. For BEA TOP END service entries, the valid values for *type* are: *FML32*, *CARRAY*, and *X\_OCTET*.

CONV = {*Y* | *N*}

must be set to *Y* if a pseudo-conversation is to be managed by a BEA TOP END server application that may or may not maintain application context. If *CONV* is set to *Y*, the XATMI functions for conversation (*tpconnect*, *tpsend*, and *tpdiscon*) must be used. If *CONV* is set to *N*, the request/reply XATMI functions (*tpcall*, *tpacall*) must be used with this service. The default is *N*.

TRANTIME = *integer*

specifies (in seconds) the default interval (or timeout) 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 indicates the maximum timeout value for the machine, or 2147483648-1.

## DM\_RESOURCES

This optional section is used for defining global Domains configuration information, specifically a user-supplied configuration version string.

The only parameter in this section is

`VERSION = string`

where *string* is a field in which users can enter a version number for the current Domains configuration file. This field is not checked by the software.

## DM\_ROUTING Section

This section provides information for data-dependent routing of service requests using FML32. This description only applies to gateways of type TOPEND. For routing information for gateways of type TDOMAIN, refer to the DMCNFIG(5) reference page.

Lines within the DM\_ROUTING section have the following form.

```
CRITERION_NAME required_parameters
```

where *CRITERION\_NAME* is the *identifier* name of the routing entry that was specified in the services entry. The value of *CRITERION\_NAME* may contain up to 15 characters.

The following parameters are required.

```
FIELD = identifier
```

specifies the name of the routing field, which may contain up to 30 characters. It is assumed that the value of this parameter is a field name identified in an FML field table (for FML32 buffers). The FLDTBLDIR32 and FIELDTBLS32 environment variables are used to locate FML field tables. If a field in an FML32 buffer is used for routing, the number of the field must be 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 value of *string* must be a comma-separated ordered list of range/RDOM pairs (see the “Example” section later in this reference page).

A range is either a single value (a signed numeric value or a character string enclosed 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 the value of *lower* must be less than or equal to the value of *upper*.

To embed a single quote in a character string value (as in O'Brien, for example), you must precede it with 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 arrays, 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 arrays, 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, the value of the string “`MIN - -5`” is the set of all numbers less than or equal to -5 and the value of the string “`6 - MAX`” is the set of all numbers greater than or equal to 6. The meta-character \* (wildcard) 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 listed last (ranges following it are 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, array, and character field types must be placed inside a pair of single quotes and cannot 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. An `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 colon (:).

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

is a list of types and subtypes of data buffers for which this routing entry is valid. For `TOP END` domain gateways, the type is restricted to `FML32`. No subtype can be specified for type `FML32`. This parameter is required.

If the field value is not set (for `FML32` 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.

**Example** The following configuration file example is based on Example 1 in the core BEA Tuxedo DMCONFIG(5) reference page. The example has been extended to include a TOP END domain gateway with a single connection to a BEA TOP END system. In this scenario the BEA TOP END system is also running a banking application that offers services needed by users of a BEA Tuxedo application. Conversely, certain BEA Tuxedo services need to be available to the BEA TOP END system for use by modified client programs. A simple queuing example is also included.

The DMCONFIG file for this configuration is shown below. Changes from the original example in the core DMCONFIG(5) file are shown in **bold**.

```
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>]
[<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.CENTRAL02"
 DMTLOGDEV = "/usr/apps/bank/DMTLOG"
 DMTLOGNAME = "DMTLG_C02"

c03 GWGRP = bankg3
 TYPE = TOPEND
 DOMAINID = "CENTRALBKGW"
 DMTLOGDEV = "/usr/apps/bank/DMTLOG"
 DMTLOGNAME = "DMTLG_C03"
 SECURITY = CLEAR

#
*DM_REMOTE_DOMAINS
```

## Section 5 - File Formats and Data Descriptions

---

```
#<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"

b05 TYPE = TOPEND
 DOMAINID = "BANK05"

*DM_TDOMAIN
#
<local or remote domain name> <network address> [<nwdevice>]
#
Local network addresses
c01 NWADDR = "//newyork.acme.com:65432" 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.CENTRAL02"
 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_TOPEND
#Local network addresses
c03 NWADDR = "//newyork.acme.com:65434"
 TP_SYSTEM = "BANKSYS"
#Remote network addresses
```

```

b05 NWADDR = "//sandiego.acme.com:65434"
 TP_SYSTEM = "BANKSYS"

*DM_LOCAL_SERVICES
#<service_name> [<Local Domain name>] [<access control>] [<exported svcname>]
[<inbuftype>] [<outbuftype>]
#
#Not available to TOP END, no mapping
open_act ACL = branch LDOM=c01
close_act ACL = branch LDOM=c01
credit LDOM=c01
debit LDOM=c01
loan LDOM = c02 ACL = loans

#Services exported to TOP END and other domains
balance TYPE=SERVICE TE_PRODUCT="TUX" TE_FUNCTION="BALANCE" LDOM=c03

#Queues available to TOP END
qspace TYPE=QSPACE TE_RTQGROUP="TUXQUEUE" TE_RTQNAME="TUXQ" LDOM=c03
qname TYPE=QNAME TE_PRODUCT="TUX" TE_FUNCTION="QSERV" LDOM=c03

*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"

#
New TOP END services available to TUXEDO
DEFAULT: LDOM=c03 RDOM=b05
 TYPE=SERVICE TE_PRODUCT="EBANK"
te_start TE_FUNCTION="START"
te_end TE_FUNCTION="END"
te_login TE_FUNCTION="LOGIN"
te_listacct TE_FUNCTION="LISTACCT"
te_getpayees TE_FUNCTION="GETPAYES"
te_elecpay TE_FUNCTION="ELECPAY"
te_bal TE_FUNCTION="BAL"
te_transfer TE_FUNCTION="TRANSFER"
te_withdrawl TE_FUNCTION="WITHDRAW"
te_deposit TE_FUNCTION="DEPOSIT"

#
#TOP END RTQ queues available to Tuxedo
DEFAULT: LDOM=c03 RDOM=b05 TYPE=QSPACE

```

## Section 5 - File Formats and Data Descriptions

---

```
tuxqspace TE_RTQGROUP="TEQGROUP" TE_RTQNAME="TEQNAME"

#
#TOP END services available to TUXEDO via tpenqueue and RTQ.
DEFAULT: LDOM=c03 RDOM=b05 TYPE=QNAME
te_report TE_PRODUCT="EBANK" TE_FUNCTION="REPORT"
te_update TE_PRODUCT="EBANK" TE_FUNCTION="UPDATE"

*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
```

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

```
//155.2.193.18:bankapp-gwaddr
//155.2.193.18:2334
//backus.company.com:bankapp-gwaddr
//backus.company.com:2334
0x0002091E9B02C112
```

The last of these representations is written in hexadecimal format. The string `0002` is the first part of a TCP/IP address. `091E` is the port number `2334` translated into a hexadecimal number. The rest of the address (`9B02C112`) consists of hexadecimal translations of each element of the IP address (`155.2.193.12`): `9B` is translated from `155`, `02` is translated from `2`, and so on.

See Also `dmadmin(1)`, `dmloadcf(1)`, `dmunloadcf(1)`, `tmboot(1)`, `tmshutdown(1)`,  
`DMADM(5)`, `GWADM(5)`, `GWTOPEND(5)`

*BEA TOP END Programmer's Reference Manual: `tp_intro(3T)`, `ni_config(4T)`,  
`nm_config(4T)`, `nm_script(4T)`*

*Administering a BEA Tuxedo Application at Run Time*

*Setting Up a BEA Tuxedo Application*

*Programming a BEA Tuxedo Application Using C*

*Using the BEA Tuxedo Domains Component*

*Using the BEA Tuxedo TOP END Domain Gateway*

# DM\_MIB(5)

Name DM\_MIB - Management Information Base for Domains

Synopsis 

```
#include <fml32.h>
#include <tpadm.h> /* MIB Header, includes DOMAINS */
```

Domains Terminology Improvements The Domains MIB uses improved class and attribute terminology to describe the interaction between domains. This improved terminology has also been applied to DMCONFIG file syntax.

These terminology improvements eliminate multiple uses of the term “domain” and introduce terms that more clearly describe the actions that occur. For example, the term *access point* defines an object through which you gain access to another object. Therefore, you access a remote domain through a remote domain access point, and remote domains gain access to a local domain through a local domain access point. The following table reflects the DMCONFIG section name changes that result from eliminating multiple uses of the term “domain.”

| This DMCONFIG section name. . . | Has changed to. . . |
|---------------------------------|---------------------|
| DM_LOCAL_DOMAINS                | DM_LOCAL            |
| DM_REMOTE_DOMAINS               | DM_REMOTE           |

Within these sections, the following parameter names have changed.

| This parameter name. . . | Has changed to. . . |
|--------------------------|---------------------|
| DOMAINID                 | ACCESSPOINTID       |
| MAXRDOM                  | MAXACCESSPOINT      |
| MAXRDTRAN                | MAXRAPTRAN          |

The equivalent DM\_MIB classes for these DMCONFIG sections are T\_DM\_LOCAL and T\_DM\_REMOTE, respectively.

In certain configurations, both available services and resources, such as queue spaces and queue names, need to be imported and exported. As such, the `DMCONFIG` section names `DM_LOCAL_SERVICES` and `DM_REMOTE_SERVICES` no longer accurately describe the necessary activity. Replacing these section names with `DM_EXPORT` and `DM_IMPORT`, respectively, clearly describes the actions that occur; that is, from the perspective of a single BEA Tuxedo domain, resources are exported from the domain through local access points and imported into the domain through remote access points. These `DMCONFIG` section name changes are shown in the following table.

| <b>This <code>DMCONFIG</code> section name. . .</b> | <b>Has changed to. . .</b> |
|-----------------------------------------------------|----------------------------|
| <code>DM_LOCAL_SERVICES</code>                      | <code>DM_EXPORT</code>     |
| <code>DM_REMOTE_SERVICES</code>                     | <code>DM_IMPORT</code>     |

Within these sections, the following parameter names have changed.

| <b>This parameter name. . .</b> | <b>Has changed to. . .</b> |
|---------------------------------|----------------------------|
| <code>LDOM</code>               | <code>LACCESSPOINT</code>  |
| <code>RDOM</code>               | <code>RACCESSPOINT</code>  |

The equivalent `DM_MIB` classes for these `DMCONFIG` sections are `T_DM_EXPORT` and `T_DM_IMPORT`, respectively.

## Backwards Compatibility

While the improved domains terminology is more accurate than previous terminology, the scope of changes to domains-related documentation and error messages is limited in this release. The improved terminology has been applied to the `DM_MIB` classes, reference page, and error messages, the `DMCONFIG` file syntax, and various `DMCONFIG` error messages.

For backwards compatibility, aliases are provided between the `DMCONFIG` terminology used prior to this release and the improved Domains MIB terminology. In this release, `dmloadcf` accepts both versions of the `DMCONFIG` terminology. `dmunloadcf`, however, generates a `DMCONFIG` file that uses the improved domains terminology by default. Use the `-c` option of `dmunloadcf` to generate a `DMCONFIG` file that uses the previous domains terminology.

**Description** The Domains MIB defines the set of classes through which a domain may import or export services using domain gateways and domain gateway administrative servers. This reference page assumes the reader is familiar with the BEA Tuxedo System Domains feature.

Use `DM_MIB(5)` 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 `DM_MIB` may be used to request an administrative service using existing ATMI interfaces in an active application.

`DM_MIB(5)` consists of the following classes:

### DM\_MIB Classes

| <b>Class Name</b>               | <b>Attributes</b>                                       |
|---------------------------------|---------------------------------------------------------|
| <code>T_DM_ACL</code>           | Domain access control list                              |
| <code>T_DM_CONNECTION</code>    | Connection status between two domains                   |
| <code>T_DM_EXPORT</code>        | Exported resource                                       |
| <code>T_DM_IMPORT</code>        | Imported resource                                       |
| <code>T_DM_LOCAL</code>         | Local access point                                      |
| <code>T_DM_OSITP</code>         | OSI TP specific configuration for an access point       |
| <code>T_DM_PASSWORD</code>      | Domain password entry                                   |
| <code>T_DM_PRINCIPAL_MAP</code> | Principal mapping entry                                 |
| <code>T_DM_REMOTE</code>        | Remote access point                                     |
| <code>T_DM_RESOURCES</code>     | Global Domains configuration information                |
| <code>T_DM_ROUTING</code>       | Access point routing criteria                           |
| <code>T_DM_RPRINCIPAL</code>    | Remote principal entry                                  |
| <code>T_DM_SNACRM</code>        | SNA-CRM-specific configuration for a local access point |
| <code>T_DM_SNALINK</code>       | SNAX-specific configuration for a remote access point   |
| <code>T_DM_SNASTACK</code>      | SNA stack to be used by a specific SNA CRM              |

**DM\_MIB Classes**

| Class Name                       | Attributes                                             |
|----------------------------------|--------------------------------------------------------|
| <a href="#">T_DM_TDOMAIN</a>     | TDomain-specific configuration for an access point     |
| <a href="#">T_DM_TOPEND</a>      | BEA TOP END-specific configuration for an access point |
| <a href="#">T_DM_TRANSACTION</a> | Transaction entry associated with a local access point |

Each class description consists of four sections:

- 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 - 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** The attribute table is a reference guide to the attributes within a class and how they may be used by administrators, operators, and general users to interface with an application.

There are five components to each attribute description 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-valued field containing both generic and component MIB-specific flag values. At this time, there are no DM\_MIB-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. The field table name `tpadm` should be included in the comma-separated list specified by the `FIELDTBLS` environment variable.

- Interoperability** Access to the header files and field tables for this MIB is provided only on BEA Tuxedo Release 7.1 sites and later, both native and Workstation. If a Release 5.0 or earlier site is active in the application, then global information updates ("SET" operations) are not allowed to gateway groups on those sites.
- Local information access for Release 5.0 and earlier sites is not available. If the class accessed also has global information, then only the global information is returned. Otherwise, an error is returned.
- 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 tables defined in this reference page, are available on all supported native and Workstation platforms.

## T\_DM\_ACL Class Definition

**Overview** The T\_DM\_ACL class represents access control information for domains.

### Attribute Table

**DM\_MIB(5): T\_DM\_ACL Class Definition Attribute Table**

| Attribute                 | Type   | Permissions | Values                         | Default    |
|---------------------------|--------|-------------|--------------------------------|------------|
| TA_DMACLNAME (r) (k) (*)  | string | rw-r--r--   | string [1..15]                 | N/A        |
| TA_DMRACCESSPOINTLIST (*) | string | rw-r--r--   | string [0..1550]               | " "        |
| TA_STATE(r)               | string | rw-r--r--   | GET: "VAL"<br>SET: "{NEW INV}" | N/A<br>N/A |

(r) - required when a new object is created

(k) - a key field for object retrieval

(\*) - a required key field for all SET operations on the class

**Attribute Semantics** TA\_DMACLNAME: *string* [1..15]  
The access control list name, unique within the scope of the T\_DM\_ACL entry names in the domain configuration.

TA\_DMRACCESSPOINTLIST: *string* [0..1550]  
The list of remote domain access points associated with this access control list. TA\_DMRACCESSPOINTLIST is a comma-separated list of remote access point names (that is, the value of the TA\_DMRACCESSPOINT attribute of a valid T\_DM\_REMOTE object). The list can contain up to 50 remote access point identifier elements. Setting this attribute to " " means that all the remote domains in the configuration are associated with this entry. " " means no remote access points are associated with this entry. The default is " ".

TA\_STATE:

GET: "{VALid}"

A GET operation retrieves configuration information for the T\_DM\_ACL object. The following state indicates the meaning of a TA\_STATE attribute value returned in response to a GET request. States not listed are not returned.

---

|         |                                                                                                               |
|---------|---------------------------------------------------------------------------------------------------------------|
| "VALid" | The object is defined and inactive. This is the only valid state for this class. ACL groups are never active. |
|---------|---------------------------------------------------------------------------------------------------------------|

---

SET: "{NEW | INVALid}"

A SET operation updates configuration information for the selected T\_DM\_ACL object. The following states indicate the meaning of a TA\_STATE set in a SET request. States not listed may not be set.

---

|       |                                                                                                                                                  |
|-------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| "NEW" | A new object is created. A state change is allowed only when in the "INVALid" state. A successful return leaves the object in the "VALid" state. |
|-------|--------------------------------------------------------------------------------------------------------------------------------------------------|

---

|              |                                                                                                                                           |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>unset</i> | Modify an existing object. This combination is not allowed in the "INVALid" state. A successful return leaves the object state unchanged. |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------|

---

|           |                                                                                                                                                |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------|
| "INVALid" | The object is deleted. A state change is allowed only when in the "VALid" state. A successful return leaves the object in the "INVALid" state. |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------|

---

Limitations None.

## T\_DM\_CONNECTION Class Definition

**Overview** The T\_DM\_CONNECTION class represents the status of connections between domain access points.

### Attribute Table

#### DM\_MIB(5): T\_DM\_CONNECTION Class Definition Attribute Table

| Attribute                                           | Type   | Permissions | Values                                           | Default    |
|-----------------------------------------------------|--------|-------------|--------------------------------------------------|------------|
| TA_DMLACCESSPOINT(k)(*)                             | string | rw-r--r--   | string[1..30]                                    | N/A        |
| TA_DMRAccessPOINT(k)                                | string | rw-r--r--   | string[1..30]                                    | N/A        |
| TA_DMTYPE                                           | string | r--r--r--   | "{TDOMAIN   TOPEND}"                             | N/A        |
| TA_STATE(k)(*)                                      | string | rwxr-xr--   | GET: "{ACT SUS INI INA UNK}"<br>SET: "{ACT INA}" | N/A<br>N/A |
| <b>Attributes available when TA_DMTYPE=TDOMAIN:</b> |        |             |                                                  |            |
| TA_DMCURENCRYPTBITS                                 | string | r-----      | "{0   40   56   128}" <sup>Note 1</sup>          | "0"        |

(k) - a key field for object retrieval  
 (\*) - a required key field for all SET operations on the class

<sup>Note 1</sup>The link-level encryption value of 40 bits is provided for backward compatibility.

|                            |                                                                                                                                                                                                                                                                                                                            |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Attribute Semantics</b> | <p>TA_DMLACCESSPOINT: <i>string</i>[1..30]</p> <p>The name of the local domain access point identifying the connection between the domains.</p> <p>On GET and SET operations, a specific local domain access point must be specified for this attribute.</p>                                                               |
|                            | <p>TA_DMRAccessPOINT: <i>string</i>[1..30]</p> <p>The name of the remote domain access point identifying the connection between the domains.</p> <p>On GET and SET operations, if TA_DMRAccessPOINT is absent, all the T_DM_CONNECTION entries for the local access point specified by TA_DMLACCESSPOINT are selected.</p> |

## Section 5 - File Formats and Data Descriptions

---

TA\_DMTYPE: "{TDOMAIN | TOPEND}"

The type of domain, which can be either "TDOMAIN" or "TOPEND".

TA\_STATE:

GET: "{ACTIVE | SUSPENDED | INITIALIZING | INACTIVE | UNKNOWN}"

A GET operation retrieves run-time information for the connection. The following states indicate the meaning of a TA\_STATE attribute value returned in response to a GET request. States not listed are not returned.

|                |                                                                                                                                         |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| "ACTIVE"       | The connection is active.                                                                                                               |
| "SUSPENDED"    | The connection is awaiting retry.                                                                                                       |
| "INITIALIZING" | The connection is initializing.                                                                                                         |
| "INACTIVE"     | The specified domain access points are disconnected. (This state is only returned by gateways running BEA Tuxedo Release 7.1 or later.) |
| "UNKNOWN"      | The connection state of the specified domain access points cannot be determined.                                                        |

SET: "{ACTIVE | INACTIVE}"

A SET operation updates run-time information for the connection. The following states indicate the meaning of a TA\_STATE in a SET request. States not listed may not be set.

|            |                                                                                                                                                                                                 |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| "ACTIVE"   | Connect the specified domain access points. If the current state is "SUSPENDED" or "INACTIVE", SET: "ACTIVE" places the connection into the state "INITIALIZING", otherwise there is no change. |
| "INACTIVE" | Disconnect the specified domain access points and destroy the object.                                                                                                                           |

## Attributes available when TA\_DMTYPE=TDOMAIN

TA\_DMCURENCRYPTBITS: "{0 | 40 | 56 | 128}"

The level of encryption in use on this connection. "0" means no encryption, while "40", "56", and "128" specify the encryption length (in bits). This attribute is valid only for gateways running BEA Tuxedo Release 7.1 or higher. For all other gateways, this value is set to "0".

**Note:** The link-level encryption value of 40 bits is provided for backward compatibility.

**Limitations** The Domain gateway administration (GWADM) server and the domain gateway supporting the local domain access point specified in the TA\_DMLACCESSPOINT attribute must be active in order to perform GET or SET operations on connections to that access point.

## T\_DM\_EXPORT Class Definition

**Overview** The T\_DM\_EXPORT class represents local resources that are exported to one or more remote domains through a local access point.

Attribute Table

**DM\_MIB(5): T\_DM\_EXPORT Class Definition Attribute Table**

| Attribute                                                                                 | Type   | Permissions | Values                           | Default        |
|-------------------------------------------------------------------------------------------|--------|-------------|----------------------------------|----------------|
| TA_DMRESOURCE(r)(k)(*)                                                                    | string | rw-r--r--   | string[1..15]                    | N/A            |
| TA_DMLACCESSPOINT(k)(*)                                                                   | string | rw-r--r--   | string[1..30]                    | *(meaning all) |
| TA_STATE(r)                                                                               | string | rw-r--r--   | GET: "VAL"<br>SET: "{NEW   INV}" | N/A<br>N/A     |
| TA_DMACLNAME                                                                              | string | rw-r--r--   | string[1..15]                    | N/A            |
| TA_DMCONV                                                                                 | string | rw-r--r--   | "{Y   N}"                        | "N"            |
| TA_DMRESOURCETYPE                                                                         | string | rw-r--r--   | "{SERVICE   QSPACE   QNAME}"     | "SERVICE"      |
| TA_DMREMOTENAME                                                                           | string | rw-r--r--   | string[1..30]                    | N/A            |
| <b>Attributes available from remote access points of TA_DMTYPE=SNAX   OSITP   TOPEND:</b> |        |             |                                  |                |
| TA_DMINBUFTYPE                                                                            | string | rw-r--r--   | string[0..513]                   | N/A            |
| TA_DMOUTBUFTYPE                                                                           | string | rw-r--r--   | string[0..513]                   | N/A            |
| <b>Attributes available from remote access points of TA_DMTYPE=TOPEND:</b>                |        |             |                                  |                |
| TA_DMTE_PRODUCT                                                                           | string | rw-r--r--   | string[1..32]                    |                |
| TA_DMTE_FUNCTION                                                                          | string | rw-r--r--   | string[1..8]                     |                |
| TA_DMTE_TARGET                                                                            | string | rw-r--r--   | string[1..8]                     | Spaces         |
| TA_DMTE_QUALIFIER                                                                         | long   | rw-r--r--   | 0 <= num <= MAXLONG              | 0 (Zero)       |
| TA_DMTE_RTQGROUP                                                                          | string | rw-r--r--   | string[1..32]                    |                |
| TA_DMTE_RTQNAME                                                                           | string | rw-r--r--   | string[1..8]                     |                |

## DM\_MIB(5): T\_DM\_EXPORT Class Definition Attribute Table

| Attribute                                                                                                                                                                              | Type                                                                                                                                                                                                                                                                                                                                                                                                                              | Permissions | Values                                                                                                                                    | Default |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|-------------------------------------------------------------------------------------------------------------------------------------------|---------|
| <p>(r) - required when a new object is created<br/>           (k) - a key field for object retrieval<br/>           (*) - a required key field for all SET operations on the class</p> |                                                                                                                                                                                                                                                                                                                                                                                                                                   |             |                                                                                                                                           |         |
| Attribute                                                                                                                                                                              | TA_DMRESOURCENAME: <i>string</i> [1..15]                                                                                                                                                                                                                                                                                                                                                                                          |             |                                                                                                                                           |         |
| Semantics                                                                                                                                                                              | <p>The local resource name for entries of resource type SERVICE (the service name), QSPACE (the queue space name), and QNAME (the queue name). For a SERVICE entry, the value of this attribute corresponds to the value of the TA_SERVICENAME attribute of an active T_SVCGRP object. This resource is exported to other domains with the same name or with the alias defined in the TA_DMREMOTENAME or TA_DMTE* attributes.</p> |             |                                                                                                                                           |         |
|                                                                                                                                                                                        | TA_DMLACCESSPOINT: <i>string</i> [1..30]                                                                                                                                                                                                                                                                                                                                                                                          |             |                                                                                                                                           |         |
|                                                                                                                                                                                        | <p>The local access point name. Setting this attribute to "*" means the resource is available at all local access points.</p>                                                                                                                                                                                                                                                                                                     |             |                                                                                                                                           |         |
|                                                                                                                                                                                        | TA_STATE:                                                                                                                                                                                                                                                                                                                                                                                                                         |             |                                                                                                                                           |         |
|                                                                                                                                                                                        | GET: "{VALid}"                                                                                                                                                                                                                                                                                                                                                                                                                    |             |                                                                                                                                           |         |
|                                                                                                                                                                                        | <p>A GET operation retrieves configuration information for the T_DM_EXPORT object. The following state indicates the meaning of a TA_STATE attribute value returned in response to a GET request. States not listed are not returned.</p>                                                                                                                                                                                         |             |                                                                                                                                           |         |
|                                                                                                                                                                                        | "VALid"                                                                                                                                                                                                                                                                                                                                                                                                                           |             | The object exists.                                                                                                                        |         |
|                                                                                                                                                                                        | SET: "{NEW   INVALid}"                                                                                                                                                                                                                                                                                                                                                                                                            |             |                                                                                                                                           |         |
|                                                                                                                                                                                        | <p>A SET operation updates configuration information for the selected T_DM_EXPORT object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.</p>                                                                                                                                                                                                                     |             |                                                                                                                                           |         |
|                                                                                                                                                                                        | "NEW"                                                                                                                                                                                                                                                                                                                                                                                                                             |             | A new object is created.                                                                                                                  |         |
|                                                                                                                                                                                        | <i>unset</i>                                                                                                                                                                                                                                                                                                                                                                                                                      |             | Modify an existing object. This combination is not allowed in the "INVALid" state. A successful return leaves the object state unchanged. |         |

---

|           |                        |
|-----------|------------------------|
| "INValid" | The object is deleted. |
|-----------|------------------------|

---

TA\_DMACLNAME: *string*[1..15]

The name of a T\_DM\_ACL object to use for security on this local service. When access is permitted from TOPEND remote access points, this optional attribute may be specified if TA\_DMRESOURCETYPE="SERVICE" or "QSPACE". This attribute is not permitted if TA\_DMRESOURCETYPE="QNAME".

TA\_DMCONV: "{Y|N}"

Specifies whether this local service is conversational or not. When access is permitted from TOPEND remote access points, this attribute must be set to "N" for entries of TA\_DMRESOURCETYPE="QSPACE" or "QNAME".

TA\_DMRESOURCETYPE: "{SERVICE|QSPACE|QNAME}"

Specifies whether this entry is for a "SERVICE", "QSPACE", or "QNAME". The default is "SERVICE".

TA\_DMREMOTENAME: *string*[1..30]

For entries of type "SERVICE" or "QSPACE", this attribute specifies the name exported through non-TOPEND remote access points.

### Attributes available from remote access points of TA\_DMTYPE=SNAX|OSITP|TOPEND

TA\_DMINBUFTYPE: *string*[0..513]

*type[:subtype]*—Input buffer type, optionally followed by subtype. If this attribute is present, it defines the buffer type [and subtype] accepted. This attribute should be defined for entries of TA\_DMRESOURCETYPE="SERVICE" when access is permitted from remote access points using OSITP with the UDT application context, or when using SNAX. When access is permitted from TOPEND remote access points, this optional attribute may be specified if TA\_DMRESOURCETYPE="SERVICE" or "QNAME". This attribute is not permitted if TA\_DMRESOURCETYPE="QSPACE".

For BEA TOPEND service and queue name entries, the valid values for *type* are: FML32, CARRAY, and X\_OCTET.

TA\_DMOUTBUFTYPE: *string*[0..513]

*type[:subtype]*—Output buffer type, optionally followed by subtype. If this attribute is present, it defines the buffer type [and subtype] output by the service. This attribute should be defined for entries of TA\_DMRESOURCETYPE="SERVICE" when access is permitted from remote

access points using OSITP with the UDT application context, or when using SNAX. When access is permitted from TOPEND remote access points, this optional attribute may be specified if TA\_DMRESOURCETYPE="SERVICE". This attribute is not permitted if TA\_DMRESOURCETYPE="QSPACE" or "QNAME".

For BEA TOP END service and queue name entries, the valid values for *type* are: FML32, CARRAY, and X\_OCTET.

## Attributes available from remote access points of TA\_DMTYPE=TOPEND

TA\_DMTE\_PRODUCT: *string*[1..32]

The BEA TOP END product name. When access is permitted from TOPEND remote access points, this attribute must be specified if TA\_DMRESOURCETYPE="SERVICE" and "QNAME". This attribute is not permitted if TA\_DMRESOURCETYPE="QSPACE".

TA\_DMTE\_FUNCTION: *string*[1..8]

The BEA TOP END function name. When access is permitted from TOPEND remote access points, this attribute must be specified if TA\_DMRESOURCETYPE="SERVICE" or "QNAME". This attribute is not permitted if TA\_DMRESOURCETYPE="QSPACE".

TA\_DMTE\_TARGET: *string*[1..8]

The BEA TOP END Message Sensitive Routing (MSR) target. This attribute is optional for entries of TA\_DMRESOURCETYPE="SERVICE", "QSPACE", or "QNAME" when access is permitted from TOPEND remote access points.

TA\_DMTE\_QUALIFIER: 0 <= *num* <= MAXLONG

The BEA TOP END function qualifier. This attribute is optional for entries of TA\_DMRESOURCETYPE="SERVICE" or "QNAME" when access is permitted from TOPEND remote access points. This attribute is not permitted if TA\_DMRESOURCETYPE="QSPACE".

TA\_DMTE\_RTQGROUP: *string*[1..32]

The BEA TOP END Recoverable Transaction Queuing (RTQ) queue group name. This attribute must be specified for TA\_DMRESOURCETYPE="QSPACE" when access is permitted from TOPEND remote access points. This attribute is not permitted if TA\_DMRESOURCETYPE="SERVICE" or "QNAME".

TA\_DMTE\_RTQNAME: *string*[1..8]

The BEA TOP END RTQ queue name. This attribute must be specified for TA\_DMRESOURCETYPE="QSPACE" and access is permitted from TOPEND

remote access points. This attribute is not permitted if  
`TA_DMRESOURCETYPE="SERVICE"` or `"QNAME"`.

**Limitations** On `SET` operations that add or update an instance of this class, and where a specific local domain access point is specified in the `TA_DMLACCESSPOINT` attribute, the access point must exist in the `T_DM_LOCAL` class. If it does not, a “not defined” error is returned for the `TA_DMLACCESSPOINT` attribute and the operation fails.

## T\_DM\_IMPORT Class Definition

**Overview** The `T_DM_IMPORT` class represents remote resources that are imported through one or more remote domain access points and made available to the local domain through one or more local domain access points.

### Attribute Table

**DM\_MIB(5): T\_DM\_IMPORT Class Definition Attribute Table**

| Attribute                                                                                 | Type   | Permissions | Values                           | Default            |
|-------------------------------------------------------------------------------------------|--------|-------------|----------------------------------|--------------------|
| <code>TA_DMRESOURCE(r)(k)(*)</code>                                                       | string | rw-r--r--   | <i>string</i> [1..15]            |                    |
| <code>TA_DMLACCESSPOINT(k)(*)</code>                                                      | string | rw-r--r--   | <i>string</i> [1..30]            | *<br>(meaning all) |
| <code>TA_DMRAccessPOINTLIST(k)(*)</code>                                                  | string | rw-r--r--   | <i>string</i> [1..92]            | *<br>(meaning all) |
| <code>TA_STATE(r)</code>                                                                  | string | rwxr-xr--   | GET: "VAL"<br>SET: "{NEW   INV}" | N/A<br>N/A         |
| <code>TA_DMAUTOTRAN</code>                                                                | string | rw-r--r--   | "{Y   N}"                        | "N"                |
| <code>TA_DMCONV</code>                                                                    | string | rw-r--r--   | "{Y   N}"                        | "N"                |
| <code>TA_DMLOAD</code>                                                                    | short  | rw-r--r--   | 1 <= num <= 32,767               | 50                 |
| <code>TA_DMPRIO</code>                                                                    | short  | rw-r--r--   | 1 <= num <= 100                  | 50                 |
| <code>TA_DMRESOURCETYPE</code>                                                            | string | rw-r--r--   | "{SERVICE   QSPACE   QNAME}"     | "SERVICE"          |
| <code>TA_DMREMOTENAME</code>                                                              | string | rw-r--r--   | <i>string</i> [1..30]            | N/A                |
| <code>TA_DMROUTINGNAME</code>                                                             | string | rw-r--r--   | <i>string</i> [1..15]            | N/A                |
| <code>TA_DMTRANTIME</code>                                                                | long   | rw-r--r--   | 0 <= num <= 32,767               | 30                 |
| <b>Attributes available from remote access points of TA_DMType=SNAX   OSITP   TOPEND:</b> |        |             |                                  |                    |
| <code>TA_DMInBUFTYPE</code>                                                               | string | rw-r--r--   | <i>string</i> [0..256]           | N/A                |
| <code>TA_DMOutBUFTYPE</code>                                                              | string | rw-r--r--   | <i>string</i> [0..256]           | N/A                |

## Section 5 - File Formats and Data Descriptions

**DM\_MIB(5): T\_DM\_IMPORT Class Definition Attribute Table**

| Attribute                                                                  | Type   | Permissions | Values                | Default  |
|----------------------------------------------------------------------------|--------|-------------|-----------------------|----------|
| <b>Attributes available from remote access points of TA_DMTYPE=TOPEND:</b> |        |             |                       |          |
| TA_DMTE_PRODUCT                                                            | string | rw-r--r--   | <i>string</i> [1..32] |          |
| TA_DMTE_FUNCTION                                                           | string | rw-r--r--   | <i>string</i> [1..8]  |          |
| TA_DMTE_TARGET                                                             | string | rw-r--r--   | <i>string</i> [1..8]  | Spaces   |
| TA_DMTE_QUALIFIER                                                          | long   | rw-r--r--   | 0 <= num <= MAXLONG   | 0 (Zero) |
| TA_DMTE_RTQGROUP                                                           | string | rw-r--r--   | <i>string</i> [1..32] |          |
| TA_DMTE_RTQNAME                                                            | string | rw-r--r--   | <i>string</i> [1..8]  |          |

(r) - required when a new object is created

(k) - a key field for object retrieval

(\*) - a required key field for all SET operations on the class

**Attribute Semantics** TA\_DMRESOURCENAME: *string*[1..15]  
 The remote resource name used for entries of resource type SERVICE (the service name), QSPACE (the queue space name), and QNAME (the queue name). This resource is imported from remote domains with the same name or with the alias defined in the TA\_DMREMOTENAME or TA\_DMTE\* attributes.

TA\_DMLACCESSPOINT: *string*[1..30]  
 The name of the local domain access point through which this imported resource should be made available. If this attribute is set to "\*", the resource is made available through all local domain access points.

TA\_DMRACCESSPOINTLIST: *string*[1..92]  
 Identifies the remote domain access point through which this resource should be imported. TA\_DMRACCESSPOINTLIST is a comma-separated failover domain list; it can contain up to three remote domain access points of up to 30 characters each. If this attribute is set to "\*", the resource can be imported from all remote access points.

TA\_STATE:

GET: "{VALid}"

A GET operation retrieves configuration information for the T\_DM\_IMPORT object. The following states indicate the meaning of a TA\_STATE attribute value returned in response to a GET request. States not listed are not returned.

|         |                    |
|---------|--------------------|
| "VALid" | The object exists. |
|---------|--------------------|

SET: "{NEW | INVALid}"

A SET operation updates the configuration information for the selected T\_DM\_IMPORT object. The following states indicate the meaning of TA\_STATE in a SET request. States not listed may not be set.

|              |                                                                                                                                           |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| "NEW"        | A new object is created. A state change is allowed in the state "INVALid" and results in the state "ACTive".                              |
| <i>unset</i> | Modify an existing object. This combination is not allowed in the "INVALid" state. A successful return leaves the object state unchanged. |
| "INVALid"    | The object is deleted. A state change is allowed in the state "ACTive" and results in the state "INVALid".                                |

TA\_DMAUTOTRAN: "{Y | N}"

When a request is received for a resource that is not already within a transaction, this attribute automatically starts a transaction for this resource. The default is "N".

TA\_DMCONV: "{Y | N}"

A boolean value ("Y" or "N") specifying whether the service is conversational. When access is permitted from TOPEND remote access points, this attribute must be set to "N" for entries of TA\_DMRESOURCETYPE="QSPACE" or "QNAME".

TA\_DMLoad: 1 <= num <= 32,767

The service load.

- TA\_DMPRIO:**  $1 \leq num \leq 100$   
The dequeuing priority. Service requests with a higher priority are serviced first.
- TA\_DMRESOURCETYPE:** "{ SERVICE | QSPACE | QNAME }"  
Specifies whether this entry is for a "SERVICE", "QSPACE", or "QNAME". The default is "SERVICE".
- TA\_DMREMOTENAME:** *string*[1..30]  
For entries of type "SERVICE" or "QSPACE", this attribute specifies the name imported through non-TOPEND remote access points.
- TA\_DMROUTINGNAME:** *string*[1..15]  
The name of a T\_DM\_ROUTING object to use for routing criteria for this "SERVICE" or "QSPACE".
- TA\_DMTRANTIME:**  $1 \leq num \leq 32,767$   
Transaction time value, in seconds, for transactions automatically started for this "SERVICE" or "QSPACE". Transactions are started automatically when a request not in transaction mode is received and the TA\_DMAUTOTRAN attribute is set to "Y".  
  
Limitation: Runtime updates to this attribute are not reflected in active requests.

### Attributes available from remote access points of TA\_DMTYPE=SNAX|OSITP|TOPEND

- TA\_DMINBUFTYPE:** *string*[0..256]  
*type[:subtype]*- Input buffer type, optionally followed by subtype. If this attribute is present, it defines the buffer type [and subtype] accepted. This attribute should be defined for entries of DMRESOURCETYPE="SERVICE" when access is permitted to remote access points using OSITP with the UDT application context, or when using SNAX. When access is permitted from TOPEND remote access points, this optional attribute may be specified if TA\_DMRESOURCETYPE="SERVICE" or "QNAME". This attribute is not permitted if TA\_DMRESOURCETYPE="QSPACE".  
  
For BEA TOP END service and queue name entries, the valid values for *type* are: FML32, CARRAY, and X\_OCTET.
- TA\_DMOUTBUFTYPE:** *string*[0..256]  
*type[:subtype]*- Output buffer type, optionally followed by subtype. If this attribute is present, it defines the buffer type [and subtype] output by the

service. This attribute should be defined for entries of `DMTYPE="SERVICE"` when access is permitted to remote access points using `OSITP` with the UDT application context, or when using `SNAX`. When access is permitted from `TOPEND` remote access points, this optional attribute may be specified if `TA_DMRESOURCETYPE="SERVICE"`. This attribute is not permitted if `TA_DMRESOURCETYPE="QSPACE"` or `"QNAME"`.

For `BEA TOP END` service and queue name entries, the valid values for *type* are: `FML32`, `CARRAY`, and `X_OCTET`.

## Attributes available from remote access points of `TA_DMTYPE=TOPEND`

`TA_DMTE_PRODUCT`: *string*[1..32]

The `BEA TOP END` product name. This attribute must be specified if `TA_DMRESOURCETYPE="SERVICE"` or `"QNAME"`. It is not permitted if `TA_DMRESOURCETYPE="QSPACE"`.

`TA_DMTE_FUNCTION`: *string*[1..8]

The `BEA TOP END` function name. This attribute must be specified if `TA_DMRESOURCETYPE="SERVICE"` or `"QNAME"`. It is not permitted if `TA_DMRESOURCETYPE="QSPACE"`.

`TA_DMTE_TARGET`: *string*[1..8]

The `BEA TOP END` Message Sensitive Routing (MSR) target. This attribute is optional attribute for entries of `TA_DMRESOURCETYPE="SERVICE"`, `"QSPACE"`, or `"QNAME"`.

`TA_DMTE_QUALIFIER`:  $0 \leq num \leq MAXLONG$

The `BEA TOP END` function qualifier. This attribute is optional for entries of `TA_DMRESOURCETYPE="SERVICE"` and `"QNAME"`. It is not permitted for entries of `TA_DMRESOURCETYPE="QSPACE"`.

`TA_DMTE_RTQGROUP`: *string*[1..32]

The `BEA TOP END` Recoverable Transaction Queuing (RTQ) queue group name. This attribute must be specified if `TA_DMRESOURCETYPE="QSPACE"`. It is not permitted if `TA_DMRESOURCETYPE="SERVICE"` or `"QNAME"`.

`TA_DMTE_RTQNAME`: *string*[1..8]

The `BEA TOP END` RTQ queue name. This attribute must be specified if `TA_DMRESOURCETYPE="QSPACE"`. It is not permitted if `TA_DMRESOURCETYPE="SERVICE"` or `"QNAME"`.

Limitations    None.

## T\_DM\_LOCAL Class Definition

**Overview** The T\_DM\_LOCAL class defines a local domain access point. A local domain access point is used to control access to local services exported to remote domains and to control access to remote services imported from remote domains.

### Attribute Table

**DM\_MIB(5): T\_DM\_LOCAL Class Definition Attribute Table**

| Attribute                                 | Type   | Permissions | Values                                                          | Default                                       |
|-------------------------------------------|--------|-------------|-----------------------------------------------------------------|-----------------------------------------------|
| <a href="#">TA_DMACCESSPOINT(r)(k)(*)</a> | string | rw-r--r--   | <i>string</i> [1..30]                                           | N/A                                           |
| <a href="#">TA_DMACCESSPOINTID(r)</a>     | string | rw-r--r--   | <i>string</i> [1..30]                                           | N/A                                           |
| <a href="#">TA_DMSRVGROUP(r)</a>          | string | rw-r--r--   | <i>string</i> [1..30]                                           | N/A                                           |
| <a href="#">TA_DMTYPE</a>                 | string | rw-r--r--   | "{TDOMAIN   OSITP   SNAX   TOPEND}"                             | "TDOMAIN"                                     |
| <a href="#">TA_STATE(r)</a>               | string | rw-r--r--   | GET: "VAL"<br>SET: "{NEW   INV}"                                | N/A<br>N/A                                    |
| <a href="#">TA_DMAUDITLOG</a>             | string | rw-r--r--   | <i>string</i> [1..78]                                           | N/A                                           |
| <a href="#">TA_DMBLOCKTIME</a>            | short  | rw-r--r--   | 0 <= num <= 32,767                                              | TA_BLOCKTIME<br>in T_DOMAIN <sup>Note 1</sup> |
| <a href="#">TA_DMMAXRAPTRAN</a>           | short  | rw-r--r--   | 0 <= num <= 32,767                                              | 16                                            |
| <a href="#">TA_DMMAXTRAN</a>              | short  | rw-r--r--   | 0 <= num <= 32,767                                              | TA_MAXGTT in<br>T_DOMAIN <sup>Note 2</sup>    |
| <a href="#">TA_DMSECURITY</a>             | string | rw-r--r--   | "{NONE   APP_PW   DM_PW   DM_USER_PW   CLEAR   SAFE   PRIVATE}" | "NONE"                                        |
| <a href="#">TA_DMTLOGDEV</a>              | string | rw-r--r--   | <i>string</i> [1..78]                                           | N/A                                           |
| <a href="#">TA_DMTLOGNAME</a>             | string | rw-r--r--   | <i>string</i> [1..30]                                           | "DMTLOG"                                      |
| <a href="#">TA_DMTLOGSIZE</a>             | long   | rw-r--r--   | 1 <= num <= 2048                                                | 100                                           |

## DM\_MIB(5): T\_DM\_LOCAL Class Definition Attribute Table

| Attribute                                                      | Type   | Permissions | Values                                     | Default     |
|----------------------------------------------------------------|--------|-------------|--------------------------------------------|-------------|
| <b>Attributes available when TA_DMTYPE=TDOMAIN   TOPEND:</b>   |        |             |                                            |             |
| TA_DMCONNECTION_POLICY                                         | string | rwxr--r--   | "{ON_DEMAND   ON_STARTUP   INCOMING_ONLY}" | "ON_DEMAND" |
| TA_DMRETRY_INTERVAL                                            | long   | rwxr--r--   | 0 <= num <= MAXLONG                        | 60          |
| TA_DMMAXRETRY                                                  | long   | rwxr--r--   | 0 <= num <= MAXLONG                        | 0           |
| TA_DMCONNPRINCIPALNAME                                         | string | rwxr--r--   | string[0..511]                             | " "         |
| TA_DMMACHINETYPE                                               | string | rw-r--r--   | string[0..15]                              | " "         |
| <b>Attributes available when TA_DMTYPE=OSITP   TOPEND:</b>     |        |             |                                            |             |
| TA_DMBLOB_SHM_SIZE                                             | long   | rw-r--r--   | 1 <= num <= MAXLONG                        | 1000000     |
| (r) - required when a new object is created                    |        |             |                                            |             |
| (k) - a key field for object retrieval                         |        |             |                                            |             |
| (*) - a required key field for all SET operations on the class |        |             |                                            |             |

Note 1 Current value of TA\_BLOCKTIME in the T\_DOMAIN class

Note 2 Current value of TA\_MAXGTT in the T\_DOMAIN class

|                     |                                                                                                                                                                                                 |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Attribute Semantics | TA_DMACCESSPOINT: <i>string</i> [1..30]<br>The name of this T_DM_LOCAL entry — an identifier unique within the scope of the T_DM_LOCAL and T_DM_REMOTE entry names in the domain configuration. |
|                     | TA_DMACCESSPOINTID: <i>string</i> [1..30]<br>The domain access point identifier. This identifier is unique across all local and remote domain access points.                                    |
|                     | TA_DMSRVGROUP: <i>string</i> [1..30]<br>The group name in which the administrative server and gateway process of the local domain reside.                                                       |
|                     | TA_DMTYPE: "{TDOMAIN   OSITP   SNAX   TOPEND}"<br>The type of domain: "TDOMAIN" for a BEA Tuxedo system domain, "OSITP" for an OSI domain, "SNAX" for an SNA domain, or "TOPEND" for a BEA TOP  |

END domain. The presence or absence of other attributes depends on the value of this attribute.

TA\_STATE:

GET: "{VALid}"

A GET operation retrieves configuration information for the T\_DM\_LOCAL object. The following state indicates the meaning of a TA\_STATE attribute value returned in response to a GET request. States not listed are not returned.

---

|         |                    |
|---------|--------------------|
| "VALid" | The object exists. |
|---------|--------------------|

---

SET: "{NEW | INVALid}"

A SET operation updates configuration information for the selected T\_DM\_LOCAL object. The following states indicate the meaning of a TA\_STATE set in a SET request. States not listed may not be set.

---

|       |                                                                                                                |
|-------|----------------------------------------------------------------------------------------------------------------|
| "NEW" | A new object is created. This state change is allowed in the state "INVALid" and results in the state "VALid". |
|-------|----------------------------------------------------------------------------------------------------------------|

---

|              |                                                                                                                                           |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>unset</i> | Modify an existing object. This combination is not allowed in the "INVALid" state. A successful return leaves the object state unchanged. |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------|

---

|           |                                                                                                              |
|-----------|--------------------------------------------------------------------------------------------------------------|
| "INVALid" | The object is deleted. This state change is allowed in the state "VALid" and results in the state "INVALid". |
|-----------|--------------------------------------------------------------------------------------------------------------|

---

TA\_DMAUDITLOG: *string*[1..78]

The name of the audit log file for this local domain.

TA\_DMBLOCKTIME: 0 <= *num* <= 32,767

Specifies the maximum wait time allowed for a blocking call. The value sets a multiplier of the SCANUNIT parameters specified in the T\_DOMAIN object. The value SCANUNIT \* TA\_BLOCKTIME must be greater than or equal to SCANUNIT and less than 32,768 seconds. If this attribute is not specified, the default is set to the value of the TA\_BLOCKTIME attribute specified in the T\_DOMAIN object. A timeout always implies a failure of the affected request.

Notice that the timeout specified for transactions in the `T_DOMAIN` is always used when the request is issued within a transaction.

`TA_DMMAXRAPTRAN`:  $0 \leq num \leq 32,767$

The maximum number of remote domain access points that can be involved in a single transaction.

`TA_DMMAXTRAN`:  $0 \leq num \leq 32,767$

The maximum number of simultaneous transactions allowed on this local domain access point. This number must be greater than or equal to `T_DOMAIN.TA_MAXGTT`.

`TA_DMSECURITY`: "{NONE | APP\_PW | DM\_PW | DM\_USER\_PW | CLEAR | SAFE | PRIVATE}"

The type of security enabled on this domain. This attribute must be set to one of the following.

"NONE"

No security is enabled.

"APP\_PW"

Valid only when `TA_DMTYPE="TDOMAIN"`. Application password-based security is enabled.

"DM\_PW"

Valid only when `TA_DMTYPE="TDOMAIN"`. Domain password-based security is enabled.

"DM\_USER\_PW"

Valid only when `TA_DMTYPE="SNAX"`. Translation of principal names is enabled.

"CLEAR"

Valid only when `TA_DMTYPE="TOPEND"`. BEA TOP END security is enabled between the local domain and the BEA TOP END system. Network messages are sent in plain text.

"SAFE"

Valid only when `TA_DMTYPE="TOPEND"`. BEA TOP END security is enabled between the local domain and the BEA TOP END system. Network messages are protected by a checksum.

"PRIVATE"

Valid only when `TA_DMTYPE="TOPEND"`. BEA TOP END security is enabled between the local domain and the BEA TOP END system. Network messages are encrypted.

`TA_DMTLOGDEV`: *string*[1..78]

The device (raw slice) or file containing the domain TLOG for this local domain access point. The TLOG is stored as a BEA Tuxedo System VTOC table on the device. For reliability, the use of a device (raw slice) is recommended.

`TA_DMTLOGNAME`: *string*[1..30]

The domain TLOG name for this local domain access point. If more than one TLOG exists on the same device, each TLOG must have a unique name.

`TA_DMTLOGSIZE`:  $1 \leq num \leq 2048$

The size in pages of the TLOG for this local domain access point. This size is constrained by the amount of space available on the device identified in `TA_DMTLOGDEV`.

### Attributes available when `TA_DMTYPE = TDOMAIN | TOPEND`

`TA_DMCONNECTION_POLICY`: "{ON\_DEMAND | ON\_STARTUP | INCOMING\_ONLY}"

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".

"ON\_DEMAND"

Means that a connection is attempted only when requested by either a client request to a remote service or an administrative "connect" command. The default setting for `TA_DMCONNECTION_POLICY` attribute is "ON\_DEMAND". The "ON\_DEMAND" policy provides the equivalent behavior to previous releases, in which the `TA_DMCONNECTION_POLICY` attribute was not explicitly available. Connection retry processing is not allowed with this policy.

"ON\_STARTUP"

Means that a domain gateway attempts to establish a connection with its remote domain access points at gateway server initialization time. Remote services (that is, services advertised by the domain gateway for this local access point) are advertised only if a connection is successfully established to that remote domain access point. Therefore, if there is no active connection to a remote domain access point, then the remote services are suspended. By default, this connection policy retries failed connections every 60 seconds; however, you can specify a different value for this interval using the `TA_DMMAXRETRY` and `TA_DMRETRY_INTERVAL` attributes.

"INCOMING\_ONLY"

Means that a domain gateway does not attempt an initial connection to remote domain access points upon starting, and remote services are initially suspended. The domain gateway is available for incoming connections from remote domain access points, and remote services are advertised when the domain gateway for this local domain access point receives an incoming connection. Connection retry processing is not allowed when the connection policy is "INCOMING\_ONLY".

TA\_DMRETRY\_INTERVAL: 0 <= num <= MAXLONG

The number of seconds between automatic attempts to establish a connection to remote domain access points. The minimum value is 0 and the maximum value is 2147483647. The default setting is 60. If TA\_DMMAXRETRY is set to 0, then setting TA\_DMRETRY\_INTERVAL is not allowed.

This attribute is valid only when the TA\_DMCONNECTION\_POLICY attribute is set to "ON\_STARTUP". For other connection policies, automatic retries are disabled.

TA\_DMMAXRETRY: 0 <= num <= MAXLONG

The number of times that a domain gateway tries to establish connections to remote domain access points. The minimum value is 0 and the maximum is MAXLONG. MAXLONG indicates that retry processing is repeated indefinitely, or until a connection is established. For a connection policy of "ON\_STARTUP", the default setting for TA\_DMMAXRETRY is MAXLONG. Setting this attribute to 0 turns off the auto retry mechanism. For other connection policies, auto retries are disabled.

The TA\_DMMAXRETRY attribute is valid only when the connection policy is "ON\_STARTUP".

TA\_DMCONNPRINCIPALNAME: string[0..511]

The connection principal name identifier, which is the principal name used for verifying the identity of this local domain access point when establishing a connection to a remote domain access point. This attribute only applies to domains of type TDOMAIN that are running BEA Tuxedo 7.1 or later software.

The TA\_DMCONNPRINCIPALNAME attribute may contain a maximum of 511 characters (excluding the terminating null character). If this attribute is not specified, the connection principal name defaults to the TA\_DMACCESSPOINTID string for this local domain access point.

For default authentication plug-ins, if a value is assigned to the `TA_DMCONNPRINCIPALNAME` attribute for this local domain access point, it must be the same as the value assigned to the `TA_DMACCESSPOINTID` attribute for this local domain access point. If these values do not match, the local domain gateway process will *not* boot, and the system will generate the following `userlog(3c)` message: `ERROR: Unable to acquire credentials.`

`TA_DMMACHINETYPE`: *string*[0..15]

Used for grouping domains so that encoding/decoding of messages between domains can be bypassed. If `TA_DMMACHINETYPE` is not specified, the default is to turn encoding/decoding on. If the value set for the `TA_DMMACHINETYPE` field is the same in both the `T_DM_LOCAL` and `T_DM_REMOTE` classes for a connection, data encoding/decoding is bypassed. The value set for `TA_DMMACHINETYPE` can be any string value up to 15 characters in length. It is used only for comparison.

This attribute is valid only when `TA_DMTYPE=TDOMAIN`.

### Attributes available when `TA_DMTYPE = OSITP | TOPEND`

`TA_DMBLOB_SHM_SIZE`:  $1 \leq num \leq MAXLONG$

Relevant only to local domain access point entries. Specifies the amount of shared memory allocated to storing binary large object log information specific to `OSITP` or `TOPEND`.

**Limitations** When the Domain gateway administration (`GWADM`) server supporting the local domain access point specified in the `TA_DMLACCESSPOINT` attribute is active, you cannot `SET` the `TA_STATE` to `INVALID` or update the following attributes: `TA_DMACCESSPOINTID`, `TA_DMMAXRAPTRAN`, `TA_DMMAXTRAN`, `TA_DMSRVGROUP`, `TA_DMTYPE`, `TA_DMTLOGDEV`, `TA_DMTLOGNAME`, `TA_DMTLOGSIZE`, `TA_DMMACHINETYPE`, or `TA_DMCODEPAGE`.

## T\_DM\_OSITP Class Definition

**Overview** The T\_DM\_OSITP class defines the OSI TP protocol related configuration information for a specific local or remote domain access point.

### Attribute Table

#### DM\_MIB(5): T\_DM\_OSITP Class Definition Attribute Table

| Attribute                 | Type   | Permissions | Values                                 | Default    |
|---------------------------|--------|-------------|----------------------------------------|------------|
| TA_DMACCESSPOINT(r)(k)(*) | string | rw-r--r--   | <i>string</i> [1..30]                  | N/A        |
| TA_STATE(r)               | string | rw-r--r--   | GET: "VAL"<br>SET: "{NEW   INV}"       | N/A<br>N/A |
| TA_DMAPT(r)               | string | rw-r--r--   | <i>string</i> [1..78]                  | N/A        |
| TA_DMAEQ(r)               | string | rw-r--r--   | <i>string</i> [1..78]                  | N/A        |
| TA_DMNWDEVICE             | string | rw-r--r--   | <i>string</i> [1..78]                  | N/A        |
| TA_DMACN                  | string | rw-r--r--   | "{XATMI   UDT}"                        | "XATMI"    |
| TA_DMAPID                 | short  | rw-r--r--   | 0 <= <i>num</i> <= 32767               | N/A        |
| TA_DMAEID                 | short  | rw-r--r--   | 0 <= <i>num</i> <= 32767               | N/A        |
| TA_DMURCH                 | string | rw-r--r--   | <i>string</i> [0..30]                  | N/A        |
| TA_DMMAXLISTENINGEP       | short  | rw-r--r--   | 1 <= <i>num</i> <= 32767               | 3          |
| TA_DMXATMIENCODING        | string | rw-r--r--   | "{CAE   PRELIMINARY  <br>OLTP_TM2200}" | "CAE"      |

(r) - required when a new object is created

(k) - a key field for object retrieval

(\*) - a required key field for all SET operations on the class

**Attribute Semantics** TA\_DMACCESSPOINT: *string*[1..30]  
The local or remote domain access point name for which this entry provides the protocol-specific configuration information. This field matches the domain access point name given in the T\_DM\_LOCAL or T\_DM\_REMOTE entry that defines the protocol independent configuration of the domain access point.

TA\_STATE:

GET: "{VALid}"

A GET operation retrieves configuration information for the T\_DM\_OSITP object. The following state indicates the meaning of a TA\_STATE attribute value returned in response to a GET request. States not listed are not returned.

|         |                    |
|---------|--------------------|
| "VALid" | The object exists. |
|---------|--------------------|

SET: "{NEW | INVALid}"

A SET operation updates configuration information for the selected T\_DM\_OSITP object. The following states indicate the meaning of TA\_STATE in a SET request. States not listed may not be set.

|       |                                                                                                                |
|-------|----------------------------------------------------------------------------------------------------------------|
| "NEW" | A new object is created. This state change is allowed in the state "INVALid" and results in the state "VALid". |
|-------|----------------------------------------------------------------------------------------------------------------|

|              |                                                                                                                                           |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>unset</i> | Modify an existing object. This combination is not allowed in the "INVALid" state. A successful return leaves the object state unchanged. |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------|

|           |                                                                                                              |
|-----------|--------------------------------------------------------------------------------------------------------------|
| "INVALid" | The object is deleted. This state change is allowed in the state "VALid" and results in the state "INVALid". |
|-----------|--------------------------------------------------------------------------------------------------------------|

TA\_DMAPT: *string*[1..78]

The application process title of the domain access point in object identifier form.

TA\_DMAEQ: *string*[1..78]

The application entity qualifier of the domain access point in integer form.

TA\_DMNWDEVICE: *string*[1..78]

Specifies the network device to be used.

**Note:** This attribute is relevant only when defining a local domain access point and is ignored for a remote domain access point.

TA\_DMACN: "{XATMI | UDT}"

The application context name to use with this domain access point. When establishing a dialogue to a remote domain access point, the application context name from the remote domain access point is used, if it is present. If it is absent, the application context name from the local domain access point is used. The value "XATMI" selects the use of the X/Open defined XATMI Application Service Element (ASE) and encoding. The value "UDT" selects the use of the ISO/IEC 10026-5 User Data Transfer encoding.

TA\_DMAPID: 0 <= num <= 32767

This optional attribute defines the application process invocation identifier to be used on this domain access point.

TA\_DMAEID: 0 <= num <= 32767

This optional attribute defines the application entity invocation identifier to be used on this domain access point.

TA\_DMURCH: string[0..30]

Specifies the user portion of the OSI TP recovery context handle. It may be required by an OSI TP provider in order to perform recovery of distributed transactions after a communications line or system failure.

**Note:** This attribute is relevant only when defining a local domain access point and is ignored for a remote domain access point.

TA\_DMMAXLISTENINGEP: 0 <= num <= 32767

Specifies the number of endpoints awaiting incoming OSI TP dialogues.

**Note:** This attribute is relevant only when defining a local domain access point and is ignored for a remote domain access point.

TA\_DMXXATMIENCODING: "{CAE | PRELIMINARY | OLTP\_TM2200}"

Specifies the version of the XATMI protocol used to communicate with a remote system. Valid values are:

"CAE"

"PRELIMINARY"

"OLTP\_TM2200"

**Note:** This attribute is relevant only for remote domain access points and is ignored for local domain access points.

Limitations    Deleting or updating an instance of this class is not permitted in the following scenarios:

- The instance of the class corresponds to a local domain access point and the Domain gateway administration (GWADM) server for the local access point is active.
- The instance of the class corresponds to a remote domain access point and any OSI TP Domain gateway administration (GWADM) server is active.

On SET operations that add or update an instance of this class, the specific domain access point specified in the TA\_DMACCESSPOINT attribute must exist in either the T\_DM\_LOCAL class or the T\_DM\_REMOTE class. If the access point does not exist, a “not defined” error is returned for the TA\_DMACCESSPOINT attribute and the operation fails.

## T\_DM\_PASSWORD Class Definition

**Overview** The T\_DM\_PASSWORD class represents configuration information for inter-domain authentication through access points of type TDOMAIN.

### Attribute Table

#### DM\_MIB(5): T\_DM\_PASSWORD Class Definition Attribute Table

| Attribute                  | Type   | Permissions | Values                                 | Default    |
|----------------------------|--------|-------------|----------------------------------------|------------|
| TA_DMLACCESSPOINT(r)(k)(*) | string | rw-r--r--   | string[1..30]                          | N/A        |
| TA_DMRACCESSPOINT(r)(k)(*) | string | rw-r--r--   | string[1..30]                          | N/A        |
| TA_DMLPWD(r)               | string | -w-----     | string[1..30]                          | N/A        |
| TA_DMRPWD(r)               | string | -w-----     | string[1..30]                          | N/A        |
| TA_STATE(r)                | string | rw-r--r--   | GET: "VAL"<br>SET: "{NEW   INV   REC}" | N/A<br>N/A |

(r) - required when a new object is created

(k) - a key field for object retrieval

(\*) - a required key field for all SET operations on the class

|                  |                                                                                                                                                                                                      |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Attribute</b> | TA_DMLACCESSPOINT: <i>string</i> [1..30]                                                                                                                                                             |
| <b>Semantics</b> | The name of the local domain access point to which the password applies.                                                                                                                             |
|                  | TA_DMRACCESSPOINT: <i>string</i> [1..30]                                                                                                                                                             |
|                  | The name of the remote domain access point to which the password applies.                                                                                                                            |
|                  | TA_DMLPWD: <i>string</i> [1..30]                                                                                                                                                                     |
|                  | The local password to be used to authenticate connections between the local domain access point identified by TA_DMLACCESSPOINT and the remote domain access point identified by TA_DMRACCESSPOINT.  |
|                  | TA_DMRPWD: <i>string</i> [1..30]                                                                                                                                                                     |
|                  | The remote password to be used to authenticate connections between the local domain access point identified by TA_DMLACCESSPOINT and the remote domain access point identified by TA_DMRACCESSPOINT. |

TA\_STATE:

GET: "{VALid}"

A GET operation retrieves configuration information for the selected T\_DM\_PASSWORD object. The following state indicates the meaning of a TA\_STATE attribute value returned in response to a GET request. States not listed are not returned.

---

|         |                    |
|---------|--------------------|
| "VALid" | The object exists. |
|---------|--------------------|

---

SET: "{NEW | INValid | RECrypt}"

A SET operation updates configuration information for the selected T\_DM\_PASSWORD object. The following states indicate the meaning of TA\_STATE in a SET request. States not listed may not be set.

---

|       |                                                                                                             |
|-------|-------------------------------------------------------------------------------------------------------------|
| "NEW" | A new object is created. A state change is allowed in the state "INValid" and results in the state "VALid". |
|-------|-------------------------------------------------------------------------------------------------------------|

---

|              |                                                                                    |
|--------------|------------------------------------------------------------------------------------|
| <i>unset</i> | Modify an existing object. This combination is not allowed in the state "INValid". |
|--------------|------------------------------------------------------------------------------------|

---

|           |                                                                                                           |
|-----------|-----------------------------------------------------------------------------------------------------------|
| "INValid" | The object is deleted. A state change is allowed in the state "VALid" and results in the state "INValid". |
|-----------|-----------------------------------------------------------------------------------------------------------|

---

|           |                                                                                                                                      |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------|
| "RECrypt" | Re-encrypt all passwords using a new encryption key. Applies to all password instances in the T_DM_PASSWORD and T_DM_TOPEND classes. |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------|

---

**Limitations** Passwords cannot be re-encrypted (SET TA\_STATE to "RECrypt") when any domain gateway administration server (GWADM) is running.

## T\_DM\_PRINCIPAL\_MAP Class Definition

**Overview** The T\_DM\_PRINCIPAL\_MAP class represents configuration information for mapping principal names to and from external principal names across access points of type SNAX.

### Attribute Table

#### DM\_MIB(5): T\_DM\_PRINCIPAL\_MAP Class Definition Attribute Table

| Attribute                  | Type   | Permissions | Values                               | Default    |
|----------------------------|--------|-------------|--------------------------------------|------------|
| TA_DMLACCESSPOINT(r)(k)(*) | string | rw-r--r--   | string[1..30]                        | N/A        |
| TA_DMRACCESSPOINT(r)(k)(*) | string | rw-r--r--   | string[1..30]                        | N/A        |
| TA_DMPRINNAME(r)(k)(*)     | string | rw-----     | string[1..30]                        | N/A        |
| TA_DMRPRINNAME(r)(k)(*)    | string | rw-----     | string[1..30]                        | N/A        |
| TA_DMDIRECTION(k)          | string | rw-r-----   | " { IN   OUT   BOTH } "              | "BOTH"     |
| TA_STATE(r)                | string | rw-r--r--   | GET: "VAL"<br>SET: " { NEW   INV } " | N/A<br>N/A |

(r) - required when a new object is created

(k) - a key field for object retrieval

(\*) - a required key field for all SET operations on the class

|                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Attribute Semantics</b> | <p>TA_DMLACCESSPOINT: <i>string</i>[1..30]<br/>The local domain access point to which the principal mapping applies.</p> <p>TA_DMRACCESSPOINT: <i>string</i>[1..30]<br/>The remote domain access point to which the principal mapping applies.</p> <p>TA_DMPRINNAME: <i>string</i>[1..30]<br/>The local principal name in the principal mapping.</p> <p>TA_DMRPRINNAME: <i>string</i>[1..30]<br/>The remote principal name in the principal mapping.</p> <p>TA_DMDIRECTION: " { IN   OUT   BOTH } "<br/>The direction to which the principal mapping applies.</p> |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

"IN"

Is INcoming to this BEA Tuxedo domain through the given remote domain access point and local domain access point.

"OUT"

Is OUTgoing from this BEA Tuxedo domain through the given local domain access point and remote domain access point.

"BOTH"

Applies to both INcoming and OUTgoing.

TA\_STATE:

GET: "{VALid}"

A GET operation retrieves configuration information for the selected T\_DM\_PRINCIPAL entry. The following state indicates the meaning of a TA\_STATE attribute value returned in response to a GET request. States not listed are not returned.

---

|         |                    |
|---------|--------------------|
| "VALid" | The object exists. |
|---------|--------------------|

---

SET: "{NEW | INValid}"

A SET operation updates configuration information for the selected T\_DM\_PRINCIPAL entry. The following states indicate the meaning of TA\_STATE in a SET request. States not listed may not be set.

---

|       |                                                                                                             |
|-------|-------------------------------------------------------------------------------------------------------------|
| "NEW" | A new object is created. A state change is allowed in the state "INValid" and results in the state "VALid". |
|-------|-------------------------------------------------------------------------------------------------------------|

---

|              |                                                                                    |
|--------------|------------------------------------------------------------------------------------|
| <i>unset</i> | Modify an existing object. This combination is not allowed in the state "INValid". |
|--------------|------------------------------------------------------------------------------------|

---

|           |                                                                                                           |
|-----------|-----------------------------------------------------------------------------------------------------------|
| "INValid" | The object is deleted. A state change is allowed in the state "VALid" and results in the state "INValid". |
|-----------|-----------------------------------------------------------------------------------------------------------|

---

**Limitations** In BEA Tuxedo Release 7.1, the T\_DM\_PRINCIPAL\_MAP class only applies to the SNAX domain gateway type.

## T\_DM\_REMOTE Class Definition

**Overview** The T\_DM\_REMOTE class represents remote domain access point configuration information. Local resources that may be exported through one or more local domain access points are made accessible to a remote domain through a remote domain access point. Similarly, remote resources are imported from a remote domain through a remote domain access point.

### Attribute Table

#### DM\_MIB(5): T\_DM\_REMOTE Class Definition Attribute Table

| Attribute                                                      | Type   | Permissions | Values                              | Default    |
|----------------------------------------------------------------|--------|-------------|-------------------------------------|------------|
| <a href="#">TA_DMACCESSPOINT(r)(k)(*)</a>                      | string | rw-r--r--   | <i>string</i> [1..30]               | N/A        |
| <a href="#">TA_DMACCESSPOINTID(r)</a>                          | string | rw-r--r--   | <i>string</i> [1..30]               | N/A        |
| <a href="#">TA_DMTYPE(k)</a>                                   | string | rw-r--r--   | "{TDOMAIN   OSITP   SNAX   TOPEND}" | "TDOMAIN"  |
| <a href="#">TA_STATE(r)</a>                                    | string | rw-r--r--   | GET: "VAL"<br>SET: "{NEW   INV}"    | N/A<br>N/A |
| <b>Attributes available when TA_DMTYPE=SNAX:</b>               |        |             |                                     |            |
| <a href="#">TA_DMCODEPAGE</a>                                  | string | rw-r--r--   | <i>string</i> [1..20]               | N/A        |
| <b>Attributes available when TA_DMTYPE=TDOMAIN:</b>            |        |             |                                     |            |
| <a href="#">TA_DMACLPOLICY</a>                                 | string | rwxr--r--   | "{LOCAL   GLOBAL}"                  | "LOCAL"    |
| <a href="#">TA_DMCONNPRINCIPALNAME</a>                         | string | rwxr--r--   | <i>string</i> [0..511]              | " "        |
| <a href="#">TA_DMLOCALPRINCIPALNAME</a>                        | string | rwxr--r--   | <i>string</i> [0..511]              | " "        |
| <a href="#">TA_DMMACHINETYPE</a>                               | string | rw-r--r--   | <i>string</i> [0..15]               | " "        |
| (r) - required when a new object is created                    |        |             |                                     |            |
| (k) - a key field for object retrieval                         |        |             |                                     |            |
| (*) - a required key field for all SET operations on the class |        |             |                                     |            |

## Section 5 - File Formats and Data Descriptions

---

**Attribute** TA\_DMACCESSPOINT: *string*[1..30]  
**Semantics** The name of this T\_DM\_REMOTE entry. TA\_DMACCESSPOINT is an identifier unique within the scope of T\_DM\_LOCAL and T\_DM\_REMOTE entry names in the domain configuration.

TA\_DMACCESSPOINTID: *string*[1..30]  
The access point identifier. This identifier is unique across all local and remote domain access points.

TA\_DMTYPE: "{TDOMAIN | OSITP | SNAX | TOPEND}"  
The type of domain: "TDOMAIN" for a BEA Tuxedo system domain, "OSITP" for an OSI domain, "SNAX" for an SNA domain, or "TOPEND" for a BEA TOP END domain. The presence or absence of other attributes depends on the value of this attribute.

TA\_STATE:

GET: "{VALid}"

A GET operation retrieves configuration information for the T\_DM\_REMOTE object. The following state indicates the meaning of a TA\_STATE attribute value returned in response to a GET request. States not listed are not returned.

---

|         |                    |
|---------|--------------------|
| "VALid" | The object exists. |
|---------|--------------------|

---

SET: "{NEW | INVALid}"

A SET operation updates configuration information for the selected T\_DM\_REMOTE object. The following states indicate the meaning of TA\_STATE in a SET request. States not listed may not be set.

---

|       |                          |
|-------|--------------------------|
| "NEW" | A new object is created. |
|-------|--------------------------|

---

|              |                                                                                                                                           |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>unset</i> | Modify an existing object. This combination is not allowed in the "INVALid" state. A successful return leaves the object state unchanged. |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------|

---

|           |                        |
|-----------|------------------------|
| "INVALid" | The object is deleted. |
|-----------|------------------------|

---

## Attributes available when TA\_DMTYPE=SNAX

TA\_DMCODEPAGE: *string*[1..20]

The name of the default translation tables to use in translating requests and replies sent through this access point.

## Attributes available when TA\_DMTYPE=TDOMAIN

TA\_DMACLPOLICY: { LOCAL | GLOBAL }

The access control list (ACL) policy for this remote domain access point. This attribute only applies to domains of type TDOMAIN that are running BEA Tuxedo 7.1 or later software.

LOCAL means that the local domain modifies the identity of service requests received from this remote domain to the principal name specified in the TA\_DMLOCALPRINCIPALNAME attribute for this remote domain access point. GLOBAL means that the local domain passes the service requests with no change. If not specified, the default is LOCAL.

TA\_DMCONNPRINCIPALNAME: *string*[0..511]

The connection principal name identifier, which is the principal name used for verifying the identity of this remote domain access point when establishing a connection to the local domain access point. This attribute only applies to domains of type TDOMAIN that are running BEA Tuxedo 7.1 or later software.

The TA\_DMCONNPRINCIPALNAME attribute may contain a maximum of 511 characters (excluding the terminating null character). If this attribute is not specified, the connection principal name defaults to the TA\_DMACCESSPOINTID string for this remote domain access point.

For default authentication plug-ins, if a value is assigned to the TA\_DMCONNPRINCIPALNAME attribute for this remote domain access point, it must be the same as the value assigned to the TA\_DMACCESSPOINTID attribute for this remote domain access point. If these values do not match, any attempt to set up a connection between the local domain gateway and the remote domain gateway will fail, and the system will generate the following `userlog(3c)` message: `ERROR: Unable to initialize administration key for domain domain_name.`

`TA_DMLOCALPRINCIPALNAME`: *string*[0..511]

The local principal name identifier, which is the identity assigned by the local domain to service requests received from this remote domain. This attribute only applies to domains of type `TDOMAIN` that are running BEA Tuxedo 7.1 or later software.

The `TA_DMLOCALPRINCIPALNAME` attribute is valid only if the `TA_DMACLPOLICY` attribute for this remote domain is set (or defaulted) to `LOCAL`. The `TA_DMLOCALPRINCIPALNAME` attribute may contain a maximum of 511 characters (excluding the terminating null character). If this attribute is not specified, the local principal name defaults to the `TA_DMACCESSPOINTID` string for this remote domain access point.

`TA_DMMACHINETYPE`: *string*[0..15]

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

**Limitations** When any Domain gateway administration server (`GWADM`) supporting a local domain access point of the same domain type as this request is active, you cannot `SET` the `TA_STATE` to `INVALID` or update the following attributes: `TA_DMACCESSPOINTID`, `TA_DMTYPE`, `TA_DMMACHINETYPE`, or `TA_DMCODEPAGE`.

You cannot delete an instance of the `T_DM_REMOTE` class if it is referenced by any instances of the following classes: `T_DM_ACL`, `T_DM_IMPORT`, `T_DM_OSITP`, `T_DM_ROUTING`, or `T_DM_TDOMAIN`.

## T\_DM\_RESOURCES Class Definition

**Overview** The T\_DM\_RESOURCES class represents Domains-specific configuration information.

**Attribute Table**

### DM\_MIB(5): T\_DM\_RESOURCES Class Definition Attribute Table

| Attribute       | Type   | Permissions | Values        | Default |
|-----------------|--------|-------------|---------------|---------|
| TA_DMVERSION(r) | string | rw-r--r--   | string[1..30] | N/A     |

(r) - required when a new object is created

(k) - a key field for object retrieval

(\*) - a required key field for all SET operations on the class

|                    |                                                           |
|--------------------|-----------------------------------------------------------|
| <b>Attribute</b>   | TA_DMVERSION: <i>string</i> [1..30]                       |
| <b>Semantics</b>   | A user-supplied identifier for the Domains configuration. |
| <b>Limitations</b> | None.                                                     |

## T\_DM\_ROUTING Class Definition

**Overview** The T\_DM\_ROUTING class represents routing criteria information for routing requests to a domain through a remote domain access point.

Attribute Table

**DM\_MIB(5): T\_DM\_ROUTING Class Definition Attribute Table**

| Attribute                 | Type   | Permissions | Values                                            | Default    |
|---------------------------|--------|-------------|---------------------------------------------------|------------|
| TA_DMROUTINGNAME(r)(k)(*) | string | rw-r--r--   | string[1..15]                                     | N/A        |
| TA_DMBUFTYPE(r)(k)(*)     | string | rw-r--r--   | string[1..256]                                    | N/A        |
| TA_DMFIELD(r)             | string | rw-r--r--   | string[1..30]                                     | N/A        |
| TA_DMFIELDTYPE            | string | rw-r--r--   | "{CHAR   SHORT   LONG   FLOAT   DOUBLE   STRING}" | N/A        |
| TA_DMANGES(r)             | string | rw-r--r--   | string[1..4096]                                   | N/A        |
| TA_STATE(r)               | string | rw-r--r--   | GET: "VAL"<br>SET: "{NEW   INV}"                  | N/A<br>N/A |

(r) - required when a new object is created

(k) - a key field for object retrieval

(\*) - a required key field for all SET operations on the class

**Attribute Semantics** TA\_DMROUTINGNAME: *string*[1..15]  
The name of the routing criteria table entry—an identifier unique within the scope of T\_DM\_ROUTING entries in the domain configuration.

TA\_DMBUFTYPE: *string*[1..256]  
"type1[:subtype1[,subtype2 ... ]];type2[:subtype3[,subtype4 ... ]]. . .]"

List of types and subtypes of data buffers for which this routing entry is valid. A maximum of 32 type/subtype combinations is allowed. The types are restricted to the following: FML, FML32, XML, VIEW, VIEW32, X\_C\_TYPE, or X\_COMMON. No subtype can be specified for type FML, FML32, or XML; subtypes are required for types VIEW, VIEW32, 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 cannot be specified for the same routing criterion name; more than one routing entry can have the same criterion 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\_DMFIELD: *string*[1..30]

The name of the field to which routing is applied.

For FML (and FML32) buffer types, TA\_DMFIELD contains an FML field name that must be defined in an FML field table. When routing is performed, the field name is retrieved using the FLDTBLDIR and FIELDTBLS (FLDTBLDIR32 and FIELDTBLS32 for FML32) environment variables.

For VIEW (and VIEW32) buffer types, TA\_DMFIELD contains a VIEW field name that must be defined in an FML VIEW table. When routing is performed, the field name is retrieved using the VIEWDIR and VIEWFILES (VIEWDIR32 and VIEWFILES32 for VIEW32) environment variables.

When routing a buffer to its correct remote domain access point, the appropriate table is used to get the data-dependent routing field value within a buffer.

For an XML buffer type, TA\_DMFIELD contains either a routing element type (or name) or a routing element attribute name.

The syntax of the TA\_DMFIELD parameter for an XML buffer type is as follows.

```
"root_element[/child_element][/child_element][/. .
.][/@attribute_name]"
```

The element is assumed to be an XML document or datagram element type. Indexing is not supported. Therefore, the BEA Tuxedo system recognizes only the first occurrence of a given element type when processing an XML buffer for data-dependent routing. This information is used to get the associated element content for data-dependent routing while sending a message. The content must be a string encoded in UTF-8.

The attribute is assumed to be an XML document or datagram attribute of the defined element. This information is used to get the associated attribute value for data-dependent routing while sending a message. The value must be a string encoded in UTF-8.

The combination of element name and attribute name may contain up to 30 characters.

The type of the routing field can be specified by the `TA_DMFIELDTYPE` attribute.

`TA_DMFIELDTYPE`: "{ CHAR | SHORT | LONG | FLOAT | DOUBLE | STRING }"

The type of the routing field specified in the `TA_DMFIELD` attribute. The type can be CHAR, SHORT, LONG, FLOAT, DOUBLE, or STRING; only one type is allowed. This attribute is required if `TA_DMBUFTYPE` is XML; it must be absent if `TA_DMBUFTYPE` is FML, VIEW, X\_C\_TYPE, or X\_COMMON.

`TA_DMRANGES`: *string*[1..4096]

The ranges and associated remote access points for the `TA_DMFIELD` routing field. The format of the string is a comma-separated, ordered list of range/group name pairs. A range/group name pair has the following format:

*"lower[-upper]:raccesspoint"*

*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 are ignored).

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 cannot 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 *raccesspoint* parameter indicates the remote domain access point to which the request is routed if the field matches the range. A *raccesspoint* of "\*" indicates that the request can go to any remote domain access point that imports the desired service.

TA\_STATE:

GET: "{VALid}"

A GET operation retrieves configuration information for the T\_DM\_ROUTING object. The following state indicates the meaning of a TA\_STATE attribute value returned in response to a GET request. States not listed are not returned.

---

|         |                    |
|---------|--------------------|
| "VALid" | The object exists. |
|---------|--------------------|

---

SET: "{NEW | INValid}"

A SET operation updates configuration information for the selected T\_DM\_ROUTING object. The following states indicate the meaning of TA\_STATE in a SET request. States not listed may not be set.

---

|       |                          |
|-------|--------------------------|
| "NEW" | A new object is created. |
|-------|--------------------------|

---

|              |                                                                                                                                         |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <i>unset</i> | Modify an existing object. This combination is not allowed in the "INValid" state. Successful return leaves the object state unchanged. |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------|

---

|           |                        |
|-----------|------------------------|
| "INValid" | The object is deleted. |
|-----------|------------------------|

---

**Limitations** You cannot delete an instance of the T\_DM\_ROUTING class if it is referenced by an instance of the T\_DM\_IMPORT class.

## T\_DM\_RPRINCIPAL Class Definition

**Overview** The T\_DM\_RPRINCIPAL class represents password configuration information for remote principal names.

### Attribute Table

DM\_MIB(5): T\_DM\_RPRINCIPAL Class Definition Attribute Table

| Attribute                  | Type   | Permissions | Values                           | Default    |
|----------------------------|--------|-------------|----------------------------------|------------|
| TA_DMRAccessPOINT(r)(k)(*) | string | rw-r--r--   | string[1..30]                    | N/A        |
| TA_DMPrINNAME(r)(k)(*)     | string | rw-----     | string[1..30]                    | N/A        |
| TA_DMPrINPASSWD(r)(*)      | string | -w-----     | string[0..30]                    | N/A        |
| TA_STATE(r)                | string | rw-r--r--   | GET: "VAL"<br>SET: "{NEW   INV}" | N/A<br>N/A |

(r) - required when a new object is created

(k) - a key field for object retrieval

(\*) - a required key field for all SET operations on the class

**Attribute** TA\_DMRAccessPOINT: *string*[1..30]

**Semantics** The remote domain access point to which the principal is applicable.

**Note:** The combination of TA\_DMRAccessPOINT and TA\_DMPrINNAME must be unique within the scope of TA\_DM\_RPRINCIPAL entries in the domain configuration.

TA\_DMPrINNAME: *string*[1..30]

The remote principal name.

**Note:** The combination of TA\_DMRAccessPOINT and TA\_DMPrINNAME must be unique within the scope of TA\_DM\_RPRINCIPAL entries in the domain configuration.

TA\_DMPrINPASSWD: *string*[0..30]

The remote password to be used for the principal name when communicating through the remote domain access point identified in TA\_DMRAccessPOINT.

TA\_STATE:

GET: "{VALid}"

A GET operation retrieves configuration information for the T\_DM\_RPRINCIPAL object. The following state indicates the meaning of a TA\_STATE attribute value returned in response to a GET request. States not listed are not returned.

---

|         |                    |
|---------|--------------------|
| "VALid" | The object exists. |
|---------|--------------------|

---

SET: "{NEW | INVALid}"

A SET operation updates configuration information for the selected T\_DM\_RPRINCIPAL object. The following states indicate the meaning of TA\_STATE in a SET request. States not listed may not be set.

---

|       |                                                                                                             |
|-------|-------------------------------------------------------------------------------------------------------------|
| "NEW" | A new object is created. A state change is allowed in the state "INVALid" and results in the state "VALid". |
|-------|-------------------------------------------------------------------------------------------------------------|

---

|              |                                                                                |
|--------------|--------------------------------------------------------------------------------|
| <i>unset</i> | Modify an existing object. This combination is not allowed in state "INVALid". |
|--------------|--------------------------------------------------------------------------------|

---

|           |                                                                                                           |
|-----------|-----------------------------------------------------------------------------------------------------------|
| "INVALid" | The object is deleted. A state change is allowed in the state "VALid" and results in the state "INVALid". |
|-----------|-----------------------------------------------------------------------------------------------------------|

---

**Limitations** In BEA Tuxedo Release 7.1, the T\_DM\_RPRINCIPAL class applies only to the SNAX domain gateway type.

## T\_DM\_SNACRM Class Definition

**Overview** The T\_DM\_SNACRM class defines the SNA-CRM-specific configuration for the named local domain access point.

Attribute Table

**DM\_MIB(5): T\_DM\_SNACRM Class Definition Attribute Table**

| Attribute               | Type   | Permissions | Values                           | Default    |
|-------------------------|--------|-------------|----------------------------------|------------|
| TA_DMSNACRM(k)(r)(*)    | string | rw-r--r--   | string[1..30]                    | N/A        |
| TA_DMLACCESSPOINT(k)(r) | string | rw-r--r--   | string[1..30]                    | N/A        |
| TA_STATE(r)             | string | rw-r--r--   | GET: "VAL"<br>SET: "{NEW   INV}" | N/A<br>N/A |
| TA_DMNWADDR(r)          | string | rw-r--r--   | string[1..78]                    | N/A        |
| TA_DMNWDEVICE           | string | rw-r--r--   | string[1..78]                    | N/A        |

(r) - required when a new object is created

(k) - a key field for object retrieval

(\*) - a required key field for all SET operations on the class

|                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Attribute Semantics</b> | <p>TA_DMSNACRM: <i>string</i>[1..30]<br/>The name of this T_DM_SNACRM entry. TA_DMSNACRM is an identifier unique within the scope of the SNA CRM entries within the domain configuration used to identify this SNA CRM entry.</p> <p>TA_DMLACCESSPOINT: <i>string</i>[1..30]<br/>The name of the local domain access point entry with which this SNA CRM is used.</p> <p>TA_STATE:<br/>GET: "{VALid}"<br/>A GET operation retrieves configuration information for the T_DM_SNACRM object. The following state indicates the meaning of a TA_STATE attribute value returned in response to a GET request. States not listed are not returned.</p> |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

---

|         |                    |
|---------|--------------------|
| "VALid" | The object exists. |
|---------|--------------------|

---

SET: "{NEW | INVALid}"

A SET operation updates configuration information for the selected T\_DM\_SNACRM object. The following states indicate the meaning of a TA\_STATE set in a SET request. States not listed may not be set.

---

|       |                                                                                                                |
|-------|----------------------------------------------------------------------------------------------------------------|
| "NEW" | A new object is created. This state change is allowed in the state "INVALid" and results in the state "VALid". |
|-------|----------------------------------------------------------------------------------------------------------------|

---

|              |                                                                                   |
|--------------|-----------------------------------------------------------------------------------|
| <i>unset</i> | Modify an existing entry. This combination is not allowed in the state "INVALid". |
|--------------|-----------------------------------------------------------------------------------|

---

|           |                                                                                                              |
|-----------|--------------------------------------------------------------------------------------------------------------|
| "INVALid" | The object is deleted. This state change is allowed in the state "VALid" and results in the state "INVALid". |
|-----------|--------------------------------------------------------------------------------------------------------------|

---

TA\_DMNWADDR: *string*[1..78]

Specifies the network address for communication between the domain gateway for the local domain access point and the SNA CRM.

TA\_DMNWDEVICE: *string*[1..78]

Specifies the network device to be used for communication between the domain gateway for the local domain access point and the SNA CRM.

**Limitations** Deleting or updating an instance of the T\_DM\_SNACRM class is not permitted if the Domain gateway administration (GWADM) server for the referenced local access point is active.

On SET operations that add or update an instance of this class, the local domain access point specified in the TA\_DMLACCESSPOINT must exist in the T\_DM\_LOCAL class. If the access point does not exist, a "not defined" error is returned for the TA\_DMLACCESSPOINT attribute and the operation fails.

## T\_DM\_SNALINK Class Definition

**Overview** The T\_DM\_SNALINK class represents SNAX-specific configuration information for a remote domain access point.

Attribute Table

**DM\_MIB(5): T\_DM\_SNALINK Class Definition Attribute Table**

| Attribute               | Type   | Permissions | Values                                                     | Default    |
|-------------------------|--------|-------------|------------------------------------------------------------|------------|
| TA_DMSNALINK(r)(k)(*)   | string | rw-r--r--   | string[1..30]                                              | N/A        |
| TA_DMSNASTACK(r)(k)     | string | rw-r--r--   | string[1..30]                                              | N/A        |
| TA_DMRAccessPOINT(r)(k) | string | rw-r--r--   | string[1..30]                                              | N/A        |
| TA_DMLSYSID(r)          | string | rw-r--r--   | string[1..4]                                               | N/A        |
| TA_DMRSYSID(r)          | string | rw-r--r--   | string[1..4]                                               | N/A        |
| TA_DMLUNAME(r)          | string | rw-r--r--   | string[1..8]                                               | N/A        |
| TA_DMMinWIN(r)          | short  | rw-r--r--   | 0 <= num <= 32767                                          | N/A        |
| TA_DMModENAME(r)        | string | rw-r--r--   | string[1..8]                                               | N/A        |
| TA_STATE(r)             | string | rw-r--r--   | GET: "VAL"<br>SET: "{NEW   INV}"                           | N/A<br>N/A |
| TA_DMSECTYPE            | string | rw-r--r--   | "{LOCAL   IDENTIFY  <br>VERIFY   PERSISTENT  <br>MIXIDPE}" | "LOCAL"    |
| TA_DMSTARTTYPE          | string | rw-r--r--   | "{AUTO   COLD}"                                            | "AUTO"     |
| TA_DMMAXSNASESS         | short  | rw-r--r--   | 0 <= num <= 32767                                          | 64         |
| TA_DMMAXSYNCLVL         | short  | r--r--r--   | 0 <= num <= 2                                              | 0          |

(r) - required when a new object is created

(k) - a key field for object retrieval

(\*) - a required key field for all SET operations on the class

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Attribute Semantics | <p>TA_DMSNALINK: <i>string</i>[1..30]<br/>The name of the T_DM_SNALINK entry. An identifier, unique within the scope of the SNA LINK entries within the domain configuration, used to identify this TA_DMSNALINK entry.</p> <p>TA_DMSNASTACK: <i>string</i>[1..30]<br/>The name of the SNAX stack entry to be used to reach this remote domain access point.</p> <p>TA_DMRAccessPOINT: <i>string</i>[1..30]<br/>Identifies the remote domain access point name for which this entry provides the SNAX configuration data.</p> <p>TA_DMLSYSID: <i>string</i>[1..4]<br/>The local SYSID to be used when establishing an SNA link to the remote logical unit (LU).</p> <p>TA_DMRSYSID: <i>string</i>[1..4]<br/>The remote SYSID to be used when establishing an SNA link to the remote LU.</p> <p>TA_DMLUNAME: <i>string</i>[1..8]<br/>Specifies the LU name associated with the remote domain access point.</p> <p>TA_DMMINWIN: 0 &lt;= num &lt;= 32767<br/>The minimum number of winner sessions to the remote LU.</p> <p>TA_DMMODENAME: <i>string</i>[1..8]<br/>Specifies the name associated with the session characteristics for sessions to the remote LU.</p> <p>TA_STATE:</p> <p>GET: "{VALid}"<br/>A GET operation retrieves configuration information for the T_DM_SNALINK object. The following state indicates the meaning of a TA_STATE attribute value returned in response to a GET request. States not listed are not returned.</p> |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

---

|         |                    |
|---------|--------------------|
| "VALid" | The object exists. |
|---------|--------------------|

---

SET: "{NEW | INValid}"

A SET operation updates configuration information for the selected T\_DM\_SNALINK object. The following states indicate the meaning of a TA\_STATE set in a SET request. States not listed may not be set.

|              |                                                                                |
|--------------|--------------------------------------------------------------------------------|
| "NEW"        | A new object is created.                                                       |
| <i>unset</i> | Modify an existing object. This combination is not allowed in state "INValid". |
| "INValid"    | The object is deleted.                                                         |

TA\_DMSECTYPE: "{LOCAL | IDENTIFY | VERIFY | PERSISTENT | MIXIDPE}"

Specifies the type of SNA security to be used on sessions to the remote LU. Valid values for this attribute are "LOCAL", "IDENTIFY", "VERIFY", "PERSISTENT", and "MIXIDPE".

TA\_DMSTARTTYPE: "{AUTO | COLD}"

Specifies the type of session start-up for the destination LU. Setting this attribute to "COLD" forces a COLDSTART with the LU. If set to "AUTO", the SNACRM in conjunction with the domain gateway choose whether to COLDSTART or WARMSTART the LU.

TA\_DMMAXSNAESS: 0 <= num <= 32767

Specifies maximum number of sessions to establish with the remote LU.

TA\_DMMAXSYNCLVL: 0 <= num <= 2

The maximum SYNC LEVEL that can be supported to this remote LU.

### Limitations

Deleting or updating an instance of the T\_DM\_SNALINK class that refers to a T\_DM\_SNASTACK class instance is not permitted under the following condition: the T\_DM\_SNASTACK class instance refers to a T\_DM\_SNACRM class instance that references a local domain access point for which the Domain gateway administration (GWADM) server is active.

On SET operations that add or update an instance of this class:

- The remote domain access point specified in the TA\_DMRACCESSPOINT attribute must exist in the T\_DM\_REMOTE class. If the access point does not exist, a "not defined" error is returned for the TA\_DMRACCESSPOINT attribute and the operation fails.

- The SNA stack reference name specified in the `TA_DMSNASTACK` attribute must exist in the `T_DM_SNASTACK` class. If the reference name does not exist, a “not defined” error is returned for the `TA_DMSNASTACK` attribute and the operation fails.

## T\_DM\_SNASTACK Class Definition

Overview The T\_DM\_SNASTACK class defines an SNA stack to be used by a specific SNA CRM.

Attribute Table

DM\_MIB(5): T\_DM\_SNASTACK Class Definition Attribute Table

| Attribute              | Type   | Permissions | Values                           | Default    |
|------------------------|--------|-------------|----------------------------------|------------|
| TA_DMSNASTACK(r)(k)(*) | string | rw-r--r--   | string[1..30]                    | N/A        |
| TA_DMSNACRM(r)(k)      | string | rw-r--r--   | string[1..30]                    | N/A        |
| TA_DMSTACKTYPE(r)      | string | rw-r--r--   | string[1..30]                    | N/A        |
| TA_DMLUNAME(r)         | string | rw-r--r--   | string[1..8]                     | N/A        |
| TA_DMTPTNAME(r)        | string | rw-r--r--   | string[1..8]                     | N/A        |
| TA_DMSTACKPARMS(r)     | string | rw-r--r--   | string[1..128]                   | N/A        |
| TA_STATE(r)            | string | rw-r--r--   | GET: "VAL"<br>SET: "{NEW   INV}" | N/A<br>N/A |

(r) - required when a new object is created

(k) - a key field for object retrieval

(\*) - a required key field for all SET operations on the class

|                     |                                                                                                                                                                                                |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Attribute Semantics | TA_DMSNASTACK: <i>string</i> [1..30]<br>The name of this T_DM_SNASTACK entry. TA_DMSNASTACK is an identifier unique within the scope of T_DM_SNASTACK entry names in the domain configuration. |
|                     | TA_DMSNACRM: <i>string</i> [1..30]<br>Identifies the T_DM_SNACRM entry of the SNA CRM in which this SNA protocol stack definition is used.                                                     |
|                     | TA_DMSTACKTYPE: <i>string</i> [1..30]<br>Identifies the protocol stack to be used.                                                                                                             |
|                     | TA_DMLUNAME: <i>string</i> [1..8]<br>Specifies the LU name to be used on sessions established using this stack definition.                                                                     |

TA\_DMTPNAME: *string*[1..8]

Specifies the TP name associated with the SNA stack. A value of "\*" means accept any TP name.

TA\_DMSTACKPARMS: *string*[1..128]

Provides protocol stack specific parameters.

TA\_STATE:

GET: "{VALid}"

A GET operation retrieves configuration information for the T\_DM\_SNASTACK object. The following state indicates the meaning of a TA\_STATE attribute value returned in response to a GET request. States not listed are not returned.

---

|         |                    |
|---------|--------------------|
| "VALid" | The object exists. |
|---------|--------------------|

---

SET: "{NEW | INVALid}"

A SET operation updates configuration information for the selected T\_DM\_SNASTACK object. The following states indicate the meaning of TA\_STATE in a SET request. States not listed may not be set.

---

|       |                                                                                                                |
|-------|----------------------------------------------------------------------------------------------------------------|
| "NEW" | A new object is created. This state change is allowed in the state "INVALid" and results in the state "VALid". |
|-------|----------------------------------------------------------------------------------------------------------------|

---

|              |                                                                                    |
|--------------|------------------------------------------------------------------------------------|
| <i>unset</i> | Modify an existing object. This combination is not allowed in the state "INVALid". |
|--------------|------------------------------------------------------------------------------------|

---

|           |                                                                                                              |
|-----------|--------------------------------------------------------------------------------------------------------------|
| "INVALid" | The object is deleted. This state change is allowed in the state "VALid" and results in the state "INVALid". |
|-----------|--------------------------------------------------------------------------------------------------------------|

---

**Limitations** Deleting or updating an instance of this class is not permitted if the instance of the class references a T\_DM\_SNACRM object which references a local domain access point for which the domain gateway administration (GWADM) server is active.

On SET operations that add or update an instance of this class, the SNA CRM name specified in the TA\_DMSNACRM attribute must exist in the T\_DM\_SNACRM class. If the name does not exist, a “not defined” error is returned for the TA\_DMSNACRM attribute and the operation fails.

## T\_DM\_TDOMAIN Class Definition

**Overview** The T\_DM\_TDOMAIN class defines the TDomain specific configuration for a local or remote domain access point.

### Attribute Table

#### DM\_MIB(5): T\_DM\_TDOMAIN Class Definition Attribute Table

| Attribute                 | Type   | Permissions | Values                                  | Default    |
|---------------------------|--------|-------------|-----------------------------------------|------------|
| TA_DMACCESSPOINT(r)(k)(*) | string | rw-r--r--   | <i>string</i> [1..30]                   | N/A        |
| TA_DMNWADDR(r)(k)(*)      | string | rw-r--r--   | <i>string</i> [1..78]                   | N/A        |
| TA_STATE(r)               | string | rw-r--r--   | GET: "VAL"<br>SET: "{NEW   INV}"        | N/A<br>N/A |
| TA_DMNWDEVICE             | string | rw-r--r--   | <i>string</i> [1..78]                   | N/A        |
| TA_DMCPLIMIT              | long   | rw-rw-r--   | 0 <= num <= MAXLONG                     | MAXLONG    |
| TA_DMFAILOVERSEQ          | short  | rw-rw-r--   | 0 <= num <= 32767                       | See below. |
| TA_DMMINENCRYPTBITS       | string | rw-----     | "{0   40   56   128}" <sup>Note 1</sup> | "0"        |
| TA_DMMAXENCRYPTBITS       | string | rw-----     | "{0   40   56   128}" <sup>Note 1</sup> | "128"      |

(r) - required when a new object is created

(k) - a key field for object retrieval

(\*) - a required key field for all SET operations on the class

<sup>Note 1</sup>The link-level encryption value of 40 bits is provided for backward compatibility.

**Attribute Semantics** TA\_DMACCESSPOINT: *string*[1..30]  
The local or remote domain access point name for which this entry provides the TDomain-specific configuration data.

When domain level failover is in use, more than one T\_DM\_TDOMAIN class entry can be defined with the same TA\_DMACCESSPOINT attribute value.

TA\_DMNWADDR: *string*[1..78]

Specifies the network address associated with the access point. For a local domain access point, this attribute supplies the address to be used for listening for incoming connections. For a remote domain access point, this attribute

supplies the destination address to be used when connecting to a remote domain access point. The value of this field must be unique across all T\_DM\_TDOMAIN entries.

TA\_STATE:

GET: "{VALid}"

A GET operation retrieves configuration information for the T\_DM\_TDOMAIN object. The following state indicates the meaning of a TA\_STATE attribute value returned in response to a GET request. States not listed are not returned.

---

|         |                    |
|---------|--------------------|
| "VALid" | The object exists. |
|---------|--------------------|

---

SET: "{NEW | INVALid}"

A SET operation updates configuration information for the selected T\_DM\_TDOMAIN object. The following states indicate the meaning of a TA\_STATE set in a SET request. States not listed may not be set.

---

|       |                                                                                                                |
|-------|----------------------------------------------------------------------------------------------------------------|
| "NEW" | A new object is created. This state change is allowed in the state "INVALid" and results in the state "VALid". |
|-------|----------------------------------------------------------------------------------------------------------------|

---

|              |                                                                                |
|--------------|--------------------------------------------------------------------------------|
| <i>unset</i> | Modify an existing object. This combination is not allowed in state "INVALid". |
|--------------|--------------------------------------------------------------------------------|

---

|           |                                                                                                              |
|-----------|--------------------------------------------------------------------------------------------------------------|
| "INVALid" | The object is deleted. This state change is allowed in the state "VALid" and results in the state "INVALid". |
|-----------|--------------------------------------------------------------------------------------------------------------|

---

TA\_DMNWDEVICE: *string*[1..78]

Specifies the network device to be used. For a local domain access point, this attribute specifies the device to be used for listening. For a remote domain access point, this attribute specifies the device to be used when connecting to the remote domain access point.

TA\_DMCMPLIMIT: 0 <= num <= MAXLONG

Relevant to remote domain access points only. Threshold message size at and above which compression occurs for traffic to this access point.

TA\_DMFAILOVERSEQ: 0 <= num <= 32767

Relevant to remote domain access points only. Specifies the position of this set of addressing in the failover sequence for this remote domain access point. If no failover sequence number is supplied, the first entry for this remote domain access point is allocated the number 10 greater than the highest failover sequence number known for the remote domain access point. Thus, the first entry gets 10, the second 20, and so on.

The domain gateway uses the T\_DM\_TDOMAIN addressing entries for a particular remote domain access point strictly in the order of their failover sequence numbers - lowest to highest.

**Note:** This attribute applies only to gateways running BEA Tuxedo Release 7.1 and is ignored by gateways running earlier releases of the BEA Tuxedo system.

TA\_DMMINENCRYPTBITS: "{0 | 40 | 56 | 128}"

Relevant to remote domain access points only. When establishing a connection to this access point, this attribute specifies the minimum level of encryption required. "0" means no encryption, while "40", "56", and "128" specify the encryption length (in bits). If this minimum level of encryption is not met, link establishment fails. The default value is "0".

The value of 40 bits is provided for backward compatibility.

**Note:** Modifications to this attribute do not affect established connections.

TA\_DMMAXENCRYPTBITS: "{0 | 40 | 56 | 128}"

Relevant to remote domain access points only. When establishing a network link to this access point, this attribute specifies the maximum level of encryption allowed. "0" means no encryption, while "40", "56", and "128" specify the encryption length (in bits). The default value is "128".

The value of 40 bits is provided for backward compatibility.

**Note:** Modifications to this attribute do not affect established connections.

**Limitations** Deleting an instance of this class or updating the TA\_DMNWDEVICE attribute of an instance of this class is not permitted in the following scenarios:

- If the instance of the class corresponds to a local domain access point and the Domain gateway administration (GWADM) server for the local access point is active.
- The instance of the class corresponds to a remote domain access point and any TDomain Domain gateway administration (GWADM) server is active.

## T\_DM\_TOPEND Class Definition

**Overview** The T\_DM\_TOPEND class defines the configuration for a local or remote domain access point specific to a BEA TOP END system.

Attribute Table

**DM\_MIB(5): T\_DM\_TOPEND Class Definition Attribute Table**

| Attribute                 | Type   | Permissions | Values                                 | Default    |
|---------------------------|--------|-------------|----------------------------------------|------------|
| TA_DMACCESSPOINT(k)(r)(*) | string | rw-r--r--   | string[1..30]                          | N/A        |
| TA_DMFAILOVERSEQ          | short  | rw-rw-r--   | 0 <= num <= 32767                      | See below. |
| TA_DMNWADDR(r)(k)(*)      | string | rw-r--r--   | string[1..78]                          | N/A        |
| TA_DMTE_TP_SYSTEM(r)      | string | rw-r--r--   | string[1..8]                           |            |
| TA_STATE(r)               | string | rw-r--r--   | GET: "VAL"<br>SET: "{NEW   INV   REC}" | N/A<br>N/A |
| TA_DMNWDEVICE             | string | rw-r--r--   | string[1..78]                          | N/A        |
| TA_DMTE_PWD               | string | rw-----     | string[1..12]                          |            |

(r) - required when a new object is created

(k) - a key field for object retrieval

(\*) - a required key field for all SET operations on the class

**Attribute Semantics** TA\_DMACCESSPOINT: string[1..30]  
Specifies the local or remote domain access point name for which this entry provides the BEA TOP END-specific configuration data.

TA\_DMFAILOVERSEQ: 0 <= num <= 32767

Relevant to remote domain access points only. Specifies the position of this set of addressing in the failover sequence for this remote domain access point. If no failover sequence number is supplied, the first entry for this remote domain access point is allocated the number 10 greater than the highest failover sequence number known for the remote domain access point. Thus, the first entry gets 10, the second 20, and so on.

The domain gateway uses the `T_DM_TOPEND` addressing entries for a particular remote domain access point strictly in the order of their failover sequence numbers - lowest to highest.

**Note:** This attribute applies only to gateways running BEA Tuxedo Release 7.1 and is ignored by gateways running earlier releases of the BEA Tuxedo system.

`TA_DMNWADDR`: *string*[1..78]

Specifies the network address associated with the access point. For a local domain access point, this attribute supplies the address to be used for listening for incoming connections. For a remote domain access point, this attribute supplies the destination address to be used when connecting to a remote domain access point. The value of this field must be unique across all `T_DM_TOPEND` entries.

`TA_DMTE_TP_SYSTEM`: *string*[1..8]

Specifies the name of the BEA TOP END system.

**Note:** All remote domain access points accessible through a local domain access point must have the same BEA TOP END system name.

`TA_STATE`:

GET: "{VALid}"

A GET operation retrieves configuration information for the `T_DM_TOPEND` object. The following state indicates the meaning of a `TA_STATE` attribute value returned in response to a GET request. States not listed are not returned.

---

|         |                    |
|---------|--------------------|
| "VALid" | The object exists. |
|---------|--------------------|

---

SET: "{NEW | INValid | RECrypt}"

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

---

|       |                                                                                                                |
|-------|----------------------------------------------------------------------------------------------------------------|
| "NEW" | A new object is created. This state change is allowed in the state "INValid" and results in the state "VALid". |
|-------|----------------------------------------------------------------------------------------------------------------|

---

## Section 5 - File Formats and Data Descriptions

---

|              |                                                                                                                                           |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>unset</i> | Modify an existing object. This combination is not allowed in the "INValid" state. A successful return leaves the object state unchanged. |
| "INValid"    | The object is deleted. This state change is allowed in the state "VALid" and results in the state "INValid".                              |
| "RECrypt"    | Re-encrypt all passwords using a new encryption key. Applies to all password instances in the T_DM_PASSWORD and T_DM_TOPEND classes.      |

TA\_DMNWDEVICE: *string*[1..78]

Specifies the network device associated with the local or remote domain access point.

TA\_DMTE\_PWD: *string*[1..12]

Specifies the password to be used when sending messages to the BEA TOP END system. Relevant only to local domain access point entries.

## T\_DM\_TRANSACTION Class Definition

**Overview** The T\_DM\_TRANSACTION class represents run-time information about transactions that span domains. This object can be used to find out what remote domain access points are involved in the transaction, the parent domain access point, the transaction state, and other information.

For GET operations, the attributes TA\_DMPTRANID, TA\_DMTXACCESSPOINT and TA\_DMTXNETTRANID may be supplied to select a particular transaction.

### Attribute Table

**DM\_MIB(5): T\_DM\_TRANSACTION Class Definition Attribute Table**

| Attribute                     | Type   | Permissions | Values                                                                                 | Default |
|-------------------------------|--------|-------------|----------------------------------------------------------------------------------------|---------|
| TA_DMLACCESSPOINT(k)(*)       | string | rw-r--r--   | string[1..30]                                                                          | N/A     |
| TA_DMPTRANID(k)               | string | rw-r--r--   | string[1..78]                                                                          | N/A     |
| TA_STATE(r)(k)                | string | rwxr-xr--   | GET: "{ABD   ABY   ACT   COM   DEC   DON   HAB   HCO   HEU   REA   UNK}"<br>SET: "INV" | N/A     |
| TA_DMTXACCESSPOINT(k)         | string | r--r--r--   | string[1..30]                                                                          | N/A     |
| TA_DMTXNETTRANID(k)           | string | r--r--r--   | string[1..78]                                                                          | N/A     |
| TA_DMBRANCHCOUNT              | long   | r--r--r--   | 0 <= num                                                                               | N/A     |
| TA_DMBRANCHINDEX              | long   | r--r--r--   | 0 <= num                                                                               | N/A     |
| <b>Per branch attributes:</b> |        |             |                                                                                        |         |
| TA_DMBRANCHNO                 | long   | r--r--r--   | 0 <= num                                                                               | N/A     |
| TA_DMRACCESSPOINT             | string | r--r--r--   | string[1..30]                                                                          | N/A     |
| TA_DMNETTRANID                | string | r--r--r--   | string[1..78]                                                                          | N/A     |
| TA_DMBRANCHSTATE              | string | r--r--r--   | GET: "{ABD   ABY   ACT   COM   DEC   DON   HAB   HCO   HHZ   HMI   REA   UNK}"         | N/A     |

**DM\_MIB(5): T\_DM\_TRANSACTION Class Definition Attribute Table**

| Attribute | Type | Permissions | Values | Default |
|-----------|------|-------------|--------|---------|
|-----------|------|-------------|--------|---------|

(r) - required when a new object is created  
(k) - a key field for object retrieval  
(\*) - a required key field for all SET operations on the class

**Attribute** TA\_DMLACCESSPOINT: *string*[1..30]  
**Semantics** Name of the local domain access point with which the transaction is associated. This is a required field for GET operations. For SET operations, TA\_DMLACCESSPOINT must be specified.

TA\_DMTPTANID: *string*[1..78]  
Transaction identifier 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\_STATE:

GET: "{ABorted | ABortonly | ACTive | COMcalled | DECided | DONE | HABort | HCOMmit | HEUristic | REAdy | UNKnown}"

A GET operation retrieves run-time information for the T\_DM\_TRANSACTION object. The following states indicate the meaning of a TA\_STATE attribute value returned in response to a GET request. States not listed are not returned.

|             |                                                               |
|-------------|---------------------------------------------------------------|
| "ABorted"   | The transaction is being rolled back.                         |
| "ABortonly" | The transaction has been identified for rollback.             |
| "ACTive"    | The transaction is active.                                    |
| "COMcalled" | The transaction has initiated the first phase of commitment.  |
| "DECided"   | The transaction has initiated the second phase of commitment. |
| "DONE"      | The transaction has completed the second phase of commitment. |
| "HABort"    | The transaction has been heuristically rolled back.           |

---

|             |                                                                                                                                                                                                  |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| "HCOmmit"   | The transaction has been heuristically committed.                                                                                                                                                |
| "HEUristic" | The transaction commitment or rollback has completed heuristically. The branch state may give further detail on which branch has completed heuristically.                                        |
| "REAdy"     | The transaction has completed the first phase of a two phase commit. All the participating groups and remote domains have completed the first phase of commitment and are ready to be committed. |
| "UNKknown"  | It was not possible to determine the state of the transaction.                                                                                                                                   |

---

SET: "{INValid}"

A SET operation updates run-time information for the selected T\_DM\_TRANSACTION object or objects. The following state indicates the meaning of a TA\_STATE set in a SET request. States not listed may not be set.

---

|           |                                                                                                                                                                                                                                                                                           |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| "INValid" | Forget the specified transaction object or objects. This state change is only valid in states "HCOmmit", "HABort", and "HEUristic". If a TA_DMTPTTRANID attribute value is not supplied, all heuristic transaction log records for the specified local domain access point are forgotten. |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

---

TA\_DMTXACCESSPOINT: *string*[1..30]

If the transaction originated from a remote domain, TA\_DMTXACCESSPOINT is the name of the remote domain access point through which it originated. If the transaction originated within this domain, TA\_DMTXACCESSPOINT is the name of the local domain access point.

TA\_DMTXNETTRANID: *string*[1..78]

If the transaction originated from a remote domain, TA\_DMTXNETTRANID is the external transaction identifier received from the remote domain access point through which it originated. If the transaction originated within this domain, TA\_DMTXNETTRANID contains the same value as the TA\_DMTPTTRANID attribute.

**Note:** This attribute is available only to gateways running BEA Tuxedo Release 7.1 and higher, and is set to the null string "" for gateways running earlier releases of the BEA Tuxedo system.

TA\_DMBRANCHCOUNT: 0 <= num

The number of branches to remote domain access points involved in the transaction. For a domain gateway that does not make branch information available, this value is zero.

TA\_DMBRANCHINDEX: 0 <= num

The index of the first branch-specific attribute values (TA\_DMBRANCHNO, TA\_DMRAccessPOINT, TA\_DMNETTRANID, and TA\_DMBRANCHSTATE) corresponding to this object.

### Per branch attributes

TA\_DMBRANCHNO: 0 <= num

The branch number of the participating branch (numbered from zero).

TA\_DMRAccessPOINT: string[1..30]

The name of the remote domain access point for this branch.

TA\_DMNETTRANID: string[1..78]

The external transaction identifier used with the remote domain access point for this branch. Some types of domain gateways do not return this information; in this scenario this attribute is set to the empty string. For example, TDomains uses the local transaction identifier in TA\_DMTPTTRANID for branches to remote domain access points and sets this value to the empty string.

TA\_DMBRANCHSTATE:

GET: "{ABD | ABY | ACT | COM | DEC | DON | HAB | HCO | HHZ | HMI | REA | UNK}"  
A GET operation will retrieve runtime information for the transaction branch (when it is available for a particular domain gateway type).

---

|             |                                                          |
|-------------|----------------------------------------------------------|
| "ABorted"   | The transaction branch is being rolled back              |
| "ABortonly" | The transaction branch has been identified for rollback. |
| "ACTive"    | The transaction branch is active.                        |

---

|                    |                                                                                                                                                                                                                                           |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| "COMcalled"        | The transaction branch has initiated the first phase of commitment.                                                                                                                                                                       |
| "DECided"          | The transaction branch has initiated the second phase of commitment.                                                                                                                                                                      |
| "DONE"             | The transaction branch has completed the second phase of commitment.                                                                                                                                                                      |
| "HABort"           | The transaction has been heuristically rolled back.                                                                                                                                                                                       |
| "HCOMmit"          | The transaction has been heuristically committed.                                                                                                                                                                                         |
| "Heuristic Hazard" | Communications for the transaction branch failed, and it has not been determined if rollback completed successfully.                                                                                                                      |
| "Heuristic Mixed"  | The commitment or rollback for the transaction branch has completed and the remote domain has reported that the state of some of the resources used for the commitment or rollback is not consistent with the outcome of the transaction. |
| "READY"            | The transaction has completed the first phase of a two-phase commit. All the participating groups and remote domains have completed the first phase of commitment and are ready to be committed.                                          |
| "UNKnown"          | The state of the transaction could not be determined.                                                                                                                                                                                     |

**Note:** This attribute is available only to gateways running BEA Tuxedo Release 7.1 and higher, and is set to "UNKnown" for gateways running earlier releases of the BEA Tuxedo system.

**Limitations** This object is never explicitly created by the administrator; it comes into existence when the application starts a multi-domain transaction. The only action an administrator can perform on this object is to set its state to "INValid", which has the effect of causing the transaction to forget heuristic transaction log records. No other attributes are writable. When a transaction state is set to "INValid", the state in the returned buffer is that of the transaction *before* the heuristic transaction log records are forgotten, not after.

On GET and SET operations, a specific local domain access point must be specified for the TA\_DMLACCESSPOINT attribute.

On GET and SET operations, the Domain gateway administration (GWADM) server for the local access point identified in the TA\_DMLACCESSPOINT attribute must be active. Otherwise, a “not defined” error is returned.

**Files**     \${TUXDIR}/include/tpadm.h  
           \${TUXDIR}/udataobj/tpadm

**See Also**   tpacall(3c), tpalloc(3c), tpcall(3c), tpdequeue(3c), tpenqueue(3c),  
              tpgetrply(3c), tprealloc(3c), **Introduction to FML Functions**, Fadd,  
              Fadd32(3fml), Fchg, Fchg32(3fml), Ffind, Ffind32(3fml), MIB(5),  
              TM\_MIB(5)

*Administering a BEA Tuxedo Application at Run Time*

*Setting Up a BEA Tuxedo Application*

*Programming a BEA Tuxedo Application Using C*

*Programming a BEA Tuxedo Application Using FML*

# EVENTS(5)

|                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name                           | EVENTS - list of system-generated events                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Description                    | <p>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.</p> <p>The BEA Tuxedo system periodically checks system capacities. If the system finds that a resource is exhausted or near capacity, it posts a system <code>WARN</code> or <code>ERROR</code> event. The system will continue to post these events until the condition subsides.</p> <p>This reference page first defines the generic event reporting fields, and then lists all system events detected in the current BEA Tuxedo release. System event names begin with a dot ( <code>.</code> ).</p> |
| Limitations                    | <p>Event reporting is currently limited to classes defined in <code>TM_MIB(5)</code>. Event reporting uses the MIB information base. See <code>MIB(5)</code> and <code>TM_MIB(5)</code> 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.</p> <p>It is possible that the system will not post an event related to a system capacity limit (for example, <code>.SysMachineFullMaxgtt</code>) if the condition only exists for a very short period of time.</p>                                                                                                                                                                                                                  |
| Generic Event Reporting Fields | <p><code>TA_OPERATION</code>: <i>string</i><br/>The literal string <code>EVT</code>, which identifies this buffer as an event report notification.</p> <p><code>TA_EVENT_NAME</code>: <i>string</i><br/>A string that uniquely identifies this event. All system-generated events begin with <code>.Sys</code>.</p> <p><code>TA_EVENT_SEVERITY</code>: <i>string</i><br/>The string <code>ERROR</code>, <code>WARN</code>, or <code>INFO</code>, to indicate the severity of this event.</p> <p><code>TA_EVENT_LMID</code>: <i>string</i><br/>A string identifying the machine where the event was detected.</p>                                                                                                                                                                                                          |

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

.SysLicenseInfo

INFO: .SysLicenseInfo: reached 100% of Tuxedo System  
Binary Licensed User Count, DBBL/BBL lockout canceled

.SysLicenseInfo: reached 90% of Tuxedo System  
Binary Licensed User Count

.SysLicenseInfo: reached 90% of Tuxedo System  
Binary Licensed User Count, DBBL/BBL lockout canceled

.SysLicenseInfo: reached below 90% of Tuxedo System  
Binary Licensed User Count, DBBL/BBL lockout canceled

---

**SysLicenseWarn**

WARN: .SysLicenseWarn: reached 100% of Tuxedo System  
Binary Licensed User Count

**SysLicenseError**

ERROR: .SysLicenseError: exceeded 110% of Tuxedo System  
Binary Licensed User Count, DBBL/BBL lockout occurs,  
no new clients can join the application

.SysLicenseError: exceeded 110% of Tuxedo System  
Binary Licensed User Count, %hour, %minutes,  
%seconds left before DBBL/BBL lockout occurs

**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 timeout

#### 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 - Management Information Base for Event Broker

**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                                          |
|---------------------------------|-----------------------------------------------------|
| <a href="#">T_EVENT_CLIENT</a>  | Subscriptions that trigger unsolicited notification |
| <a href="#">T_EVENT_COMMAND</a> | Subscriptions that trigger system commands          |
| <a href="#">T_EVENT_QUEUE</a>   | Subscriptions for queue-based notification          |
| <a href="#">T_EVENT_SERVICE</a> | Subscriptions for server-based notification         |
| <a href="#">T_EVENT_USERLOG</a> | Subscriptions for writing user log messages         |

Each object in these classes represents a single subscription request.

The pattern expression of `TA_EVENT_EXPR` in each class determines whether it is a `SYSTEM EVENT` request or an `USER EVENT` request. The determination on which one to query is made as follows:

- A basic GET request without `TA_EVENT_EXPR` or `TA_EVENT_SERVER` specified will always go to the `SYSTEM EVENT` request and will not return `USER EVENT` request.

- A GET request with `TA_EVENT_EXPR` specified but not `TA_EVENT_SERVER` will go to the `SYSTEM EVENT` request if the expressions starts with `"\."`. Otherwise, it will go to the `USER EVENT` request.
- A GET request with `TA_EVENT_SERVER` specified with a value of `"SYSTEM"` will go to the `SYSTEM EVENT` request. A value of `"USER"` will direct the request to the `USER EVENT`.

### 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    |
|----------------------------------------|--------|-------------|------------------------------|------------|
| <code>TA_EVENT_EXPR(r) (*)</code>      | string | R--R--R--   | <i>string</i> [1..255]       | N/A        |
| <code>TA_EVENT_FILTER(k)</code>        | string | R--R--R--   | <i>string</i> [1..255]       | none       |
| <code>TA_EVENT_FILTER_BINARY(k)</code> | carray | R--R--R--   | <i>carray</i> [1..64000]     | none       |
| <code>TA_STATE(r)</code>               | string | R-xR-xR-x   | GET: ACT<br>SET: {NEW   INV} | N/A<br>N/A |
| <code>TA_CLIENTID(r) (*)</code>        | string | R--R--R--   | <i>string</i> [1..78]        | N/A        |

(k) - a key field for object retrieval

(r) - the field is required when a new object is created

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

Check MIB(5) for an explanation of Permissions.

|                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Attribute Semantics</b> | <p><code>TA_EVENT_EXPR</code>: <i>string</i>[1..255]<br/>Event pattern expression. This expression, in <code>recomp()</code> format, controls which event names match this subscription.</p> <p><code>TA_EVENT_FILTER</code>: <i>string</i>[1..255]<br/>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.</p> <p><code>TA_EVENT_FILTER_BINARY</code>: <i>carray</i>[1..64000]<br/>Event filter expression, in binary (<i>carray</i>) format. Same as <code>TA_EVENT_FILTER</code>, but may contain arbitrary binary data. Only one of <code>TA_EVENT_FILTER</code> or <code>TA_EVENT_FILTER_BINARY</code> may be specified.</p> |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

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-----      | <i>string</i> [1..255]       | N/A        |
| TA_EVENT_FILTER(k)        | string | R-----      | <i>string</i> [1..255]       | none       |
| TA_EVENT_FILTER_BINARY(k) | carray | R-----      | <i>carray</i> [1..64000]     | none       |
| TA_STATE(r)               | string | R-x-----    | GET: ACT<br>SET: {NEW   INV} | N/A<br>N/A |
| TA_COMMAND(r) (*)         | string | R-----      | <i>string</i> [1..255]       | N/A        |

(k) - a key field for object retrieval

(r) - the field is required when a new object is created

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

Check MIB(5) for an explanation of Permissions.

|                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Attribute Semantics</b> | <p>TA_EVENT_EXPR: <i>string</i>[1..255]<br/>Event pattern expression. This expression, in <code>recomp()</code> format, controls which event names match this subscription.</p> <p>TA_EVENT_FILTER: <i>string</i>[1..255]<br/>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.</p> <p>TA_EVENT_FILTER_BINARY: <i>carray</i>[1..64000]<br/>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.</p> |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

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

(k) - a key field for object retrieval

(r) - the field is required when a new object is created

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

Check MIB(5) for an explanation of Permissions.

|           |                                                                                                                                                                                                      |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Attribute | TA_EVENT_EXPR: <i>string</i> [1..255]                                                                                                                                                                |
| Semantics | Event pattern expression. This expression, in <code>recomp()</code> format, controls which event names match this subscription.                                                                      |
|           | TA_EVENT_FILTER: <i>string</i> [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: <i>carray</i> [1..64000]                                                                                                                                                     |
|           | Event filter expression, in binary ( <i>carray</i> ) 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.  |
|           | <hr/>                                                                                                                                                                                                |
|           | NEW      Create T_EVENT_QUEUE object. Successful return leaves the object in the Active state.                                                                                                       |
|           | <hr/>                                                                                                                                                                                                |
|           | INValid    Delete T_EVENT_QUEUE object. Successful return leaves the object in the INValid state.                                                                                                    |
|           | <hr/>                                                                                                                                                                                                |
|           | TA_QSPACE: <i>string</i> [1..15]                                                                                                                                                                     |
|           | Enqueue a notification message to a reliable queue in this queue space when a matching event is detected.                                                                                            |
|           | TA_QNAME: <i>string</i> [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()`'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()`'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()`'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()`'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()`'s TPQCTL control structure.

TA\_QCTL\_PRIORITY: *short*

This value, if present, is passed in to `tpenqueue()`'s TPQCTL control structure.

TA\_QCTL\_MSGID: *string*[1..31]

This value, if present, is passed in to `tpenqueue()`'s TPQCTL structure.

TA\_QCTL\_CORRID: *string*[1..31]

This value, if present, is passed in to `tpenqueue()`'s TPQCTL control structure.

TA\_QCTL\_REPLYQUEUE: *string*[1..15]

This value, if present, is passed in to `tpenqueue()`'s TPQCTL control structure.

TA\_QCTL\_FAILUREQUEUE: *string*[1..15]

This value, if present, is passed in to `tpenqueue()`'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 `tpost()` call is transactional, include the `tpenqueue()` 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    |
|----------------------------------------|--------|-------------|------------------------------|------------|
| <code>TA_EVENT_EXPR(r) (*)</code>      | string | R--R--R--   | <i>string</i> [1..255]       | N/A        |
| <code>TA_EVENT_FILTER(k)</code>        | string | R--R--R--   | <i>string</i> [1..255]       | none       |
| <code>TA_EVENT_FILTER_BINARY(k)</code> | carray | R--R--R--   | <i>carray</i> [1..64000]     | none       |
| <code>TA_STATE(r)</code>               | string | R-xR-xR-x   | GET: ACT<br>SET: {NEW   INV} | N/A<br>N/A |
| <code>TA_SERVICENAME(r) (*)</code>     | string | R--R--R--   | <i>string</i> [1..15]        | N/A        |
| <code>TA_EVENT_PERSIST</code>          | short  | R-xR-xR-x   | <i>short</i>                 | 0          |
| <code>TA_EVENT_TRAN</code>             | short  | R-xR-xR-x   | <i>short</i>                 | 0          |

(k) - a key field for object retrieval

(r) - the field is required when a new object is created

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

Check MIB(5) for an explanation of permissions.

**Attribute Semantics** `TA_EVENT_EXPR`: *string*[1..255]  
Event pattern expression. This expression, in `recomp()` 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.

---

|                |                                                                                                 |
|----------------|-------------------------------------------------------------------------------------------------|
| <i>NEW</i>     | Create T_EVENT_SERVICE object. Successful return leaves the object in the <i>ACTive</i> state.  |
| <i>INValid</i> | Delete T_EVENT_SERVICE object. Successful return leaves the object in the <i>INValid</i> 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()` 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(3c)` 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 the BEA Tuxedo `userlog(3c)` function.

### Attribute Table

#### T\_EVENT\_USERLOG Class Definition Attribute Table

| Attribute                 | Type   | Permissions | Values                       | Default    |
|---------------------------|--------|-------------|------------------------------|------------|
| TA_EVENT_EXPR(r)          | string | R--R-----   | <i>string</i> [1..255]       | N/A        |
| TA_EVENT_FILTER(k)        | string | R--R-----   | <i>string</i> [1..255]       | none       |
| TA_EVENT_FILTER_BINARY(k) | carray | R--R-----   | <i>carray</i> [1..64000]     | none       |
| TA_STATE(r)               | string | R-xR-x---   | GET: ACT<br>SET: {NEW   INV} | N/A<br>N/A |
| TA_USERLOG(r)             | string | R--R-----   | <i>string</i> [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.

|                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Attribute Semantics</b> | <p>TA_EVENT_EXPR: <i>string</i>[1..255]<br/>Event pattern expression. This expression, in <code>recomp()</code> format, controls which event names match this subscription.</p> <p>TA_EVENT_FILTER: <i>string</i>[1..255]<br/>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.</p> <p>TA_EVENT_FILTER_BINARY: <i>carray</i>[1..64000]<br/>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.</p> |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

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(3c)` message when a matching event is detected.

Files `#{TUXDIR}/udataobj/evt_mib` `#{TUXDIR}/include/evt_mib.h`

See Also `EVENTS(5)`, `TM_MIB(5)`

# Error, Error32(5)

|             |                                                                                                                                                                              |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name        | Error, Error32 - FML error codes                                                                                                                                             |
| Synopsis    | <pre>#include "fml.h" #include "fml32.h"</pre>                                                                                                                               |
| Description | The numerical value represented by the symbolic name of an error condition is assigned to <code>Error</code> for errors that occur when executing many FML library routines. |

The name `Error` 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. `Error` need not be the identifier of an object; it might expand to a modifiable *lvalue* resulting from a function call. It is unspecified whether `Error` 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 `Error`, 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 `Error` 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 `Error` must include the `fml.h` header file.

`Error32` provides a similar capability for users of `fml32` routines. An application that checks the value of `Error32` 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 FTYPEERR 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.

Introduction to the C Language Application-Transaction Monitor Interface, `tperrordetail(3c)`, `tpstrerror(3c)`, `tpstrerrordetail(3c)`, Introduction to FML Functions, `F_error`, `F_error32(3fml)`

# field\_tables(5)

|                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name              | field_tables - FML mapping files for field names                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Description       | <p>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 vary 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.</p> <p>The numeric-identifier of a field is called its “field identifier” (fldid), and is typedef'd by <code>FLDID</code>. A field is named by relating an alphanumeric string (the name) to a <code>FLDID</code> in a field table.</p> <p>The original FML interface supports 16-bit field identifiers, field lengths, and buffer sizes. A newer 32-bit interface, <code>FML32</code>, 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 <code>FLDID32</code>).</p>                                                    |
| Field Identifiers | <p>FML functions allow field values to be typed. Currently the following types are supported: <code>char</code>, <code>string</code>, <code>short</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>carray</code> (character array), <code>ptr</code> (pointer to a buffer), <code>FML32</code> (embedded <code>FML32</code> buffer), and <code>VIEW32</code> (embedded <code>VIEW32</code> buffer). The <code>ptr</code>, <code>FML32</code>, and <code>VIEW32</code> types are supported only for the <code>FML32</code> interface. Constants for field types are defined in <code>fm1.h</code> (<code>fm132.h</code> for <code>FML32</code>). 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 <code>FLDID</code>. Thus, a <code>FLDID</code> 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.</p> |
| Field Mapping     | <p>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, <code>cpp</code>, by the usual <code>#define</code> macro. Run-time mapping is done by the function <code>Fldid()</code> (or <code>Fldid32()</code> for <code>FML32</code>), which maps its argument, a field name, to a field identifier by consulting the source field table files.</p>                                                                                                                                                                                                                                                                                                                       |
| Field Table Files | <p>Files containing field tables have the following format:</p> <ul style="list-style-type: none"> <li>■ blank lines and lines beginning with <code>#</code> are ignored.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

- lines beginning with \$ are ignored by the mapping functions but are passed through (without the \$) to header files generated by `mkfldhdr()` (the command name is `mkfldhdr32()` for FML32; see `mkfldhdr`, `mkfldhdr32(1)`). 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 the following: `char`, `string`, `short`, `long`, `float`, `double`, `carray`, `ptr`, `FML32`, or `VIEW32`.

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()` and `vuform()`, 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 `FLDTBLS` 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 carry 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: carry */
```

**See Also** `mkfldhdr`, `mkfldhdr32(1)`

*Programming a BEA Tuxedo Application Using FML*

## GWADM(5)

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name        | GWADM - Domains gateway administrative server                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Synopsis    | <pre>GWADM SRVGRP = "identifier" SRVID = "number" REPLYQ = "N" CLOPT = "-A -- [-a {on   off}] [-t {on   off}]"</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Description | <p>The gateway administrative server (GWADM) is a BEA Tuxedo system-supplied server that provides administrative functions for a Domains gateway group.</p> <p>GWADM should be defined in the <code>SERVERS</code> section of the <code>UBBCONFIG</code> file as a server running within a particular gateway group, that is, <code>SRVGRP</code> must be set to the corresponding <code>GRPNAME</code> tag specified in the <code>GROUPS</code> section. The <code>SRVID</code> parameter is also required and its value must consider the maximum number of gateways allowed within the gateway group.</p> <p>There should be only one instance of a GWADM per Domains gateway group, and it should <i>NOT</i> be part of the <code>MSSQ</code> defined for the gateways associated with the group. Also, GWADM should have the <code>REPLYQ</code> attribute set to <code>N</code>.</p> <p>The <code>CLOPT</code> option is a string of command line options that is passed to the GWADM when it is booted. This string has the following format:</p> <pre>CLOPT="-A -- gateway group run-time parameters&gt;"</pre> <p>The following run-time parameters are recognized for a gateway group:</p> <ul style="list-style-type: none"><li><code>-a {on   off}</code><br/>This option turns <code>off</code> or <code>on</code> the audit log feature for this local domain. The default is <code>off</code>. The <code>dmadmin</code> program can be used to change this setting while the gateway group is running (see <code>dmadmin(1)</code>).</li><li><code>-t {on   off}</code><br/>This option turns <code>off</code> or <code>on</code> the statistics gathering feature for the local domain. The default is <code>off</code>. The <code>dmadmin</code> program can be used to change this setting while the gateway group is running (see <code>dmadmin(1)</code>).</li></ul> <p>The GWADM server must be booted before the corresponding gateways.</p> |
| Portability | GWADM is a supported BEA Tuxedo system-supplied server on UNIX System platforms.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

**Interoperability** GWADM must be installed on BEA Tuxedo Release 4.2.1 or later; other machines in the same domain with a Release 4.2.2 gateway can be Release 4.1 or later.

**Examples** The following example illustrates the definition of the administrative server in the UBBCONFIG file. This example uses the GWTDOMAIN gateway process to provide connectivity with another BEA Tuxedo domain. To provide interoperability with a BEA TOP END system, use the GWTOPEND gateway process. For details on the GWTOPEND gateway process and an example using GWTOPEND, see GWTOPEND(5).

```
#
*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=N
RESTART=Y MIN=1 MAX=1
```

**See Also** dmadm(1), tmboot(1), DMADM(5), DMCONFIG(5), DMCONFIG for GWTOPEND(5), GWTOPEND(5), servopts(5), UBBCONFIG(5)

*Administering a BEA Tuxedo Application at Run Time*

*Setting Up a BEA Tuxedo Application*

*Using the BEA Tuxedo Domains Component*

## GWTDOMAIN(5)

**Name** GWTDOMAIN - TDomain 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 communicate 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 `SRVID` 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** `tmadmin(1)`, `tmboot(1)`, `DMADM(5)`, `DMCONFIG(5)`, `GWADM(5)`, `servopts(5)`,  
`UBBCONFIG(5)`

*Using the BEA Tuxedo Domains Component*

*Setting Up a BEA Tuxedo Application*

*Administering a BEA Tuxedo Application at Run Time*

## GWTOPEND(5)

Name GWTOPEND - TOP END Domain Gateway process

Synopsis *GWTOPEND* SRVGRP = "*identifier*" SRVID = "*number*" RQADDR = "*queue\_name*"  
REPLYQ = N RESTART = Y [MAXGEN = *value*] [GRACE = *value*]

Description GWTOPEND is the domain gateway process that provides communication between a BEA Tuxedo domain and a BEA TOP END system. GWTOPEND gateway processes communicate with the Network Interface (NI) component on one or more nodes of a single BEA TOP END system. Different GWTOPEND gateways (in different BEA Tuxedo groups) may be configured to access different BEA TOP END systems or to split the load. GWTOPEND supports request/reply, pseudo-conversations, queuing, and transactions.

Domain gateways are described in the `SERVERS` section of the `UBBCONFIG` file and the `BDMCONFIG` file. Domain gateways must be 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 specify 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 GWTOPEND process must be in the same group as the `GWADM(5)` process, with the `GWADM` listed first. Multiple GWTOPEND processes can be configured for a domain; each must be configured in a different BEA Tuxedo group.

If BEA TOP END security is configured for the gateway, the BEA TOP END Security Services product must be installed on the node and the `srvtab` file for the `TP_SYSTEM` name must be copied to the node in the location shown in the "Files" section. If long node names are to be supported, the `nodemap` file must be copied to the node in the location shown in the "Files" section.

Files `$TUXDIR/udataobj/nodemap`

`$APPPDIR/srvtab.system` (where *system* is the BEA TOP END system name)

`/usr/lib/libtp_krb.so` (installed on UNIX platforms on which BEA TOP END security is configured)

`%TOPENDDIR%\bin\krb.dll` (installed on Windows NT platforms on which BEA TOP END security is configured)

**Examples** The following example shows the definition of a domain 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
GWTOPEND SRVGRP="gwgrp" SRVID=1003 RQADDR="gwgrp" REPLYQ=N
RESTART=Y MAXGEN=5 GRACE=3600
```

See the “Examples” section of UBBCONFIG(5) and DMCONFIG for GWTOPEND(5) for additional information.

**See Also** tmadmin(1), tmbboot(1), DMADM(5), DMCONFIG for GWTOPEND(5), GWADM(5), servopts(5), UBBCONFIG(5)

*BEA TOP END Programmer's Reference Manual: ext\_srvtab(1T), nodemap(5T)*

*Administering a BEA Tuxedo Application at Run Time*

*Setting Up a BEA Tuxedo Application*

*Using the BEA Tuxedo Domains Component*

*Using the BEA Tuxedo TOP END Domain Gateway*

# GWTUX2TE, GWTE2TUX(5)

|             |                                                                                                                                                                                                                                                                                                                                                                            |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name        | GWTUX2TE, GWTE2TUX - BEA Tuxedo / BEA TOP END gateway servers                                                                                                                                                                                                                                                                                                              |
| Synopsis    | <pre>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]]"</pre> |
| Description | <p>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.</p>                                                                               |

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 BEA 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 BEA 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 *Setting Up a BEA Tuxedo Application* 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 <code>-u</code> option.<br><br>Set the password using the <code>-p</code> option.<br><br>Protect this file using operating system protection. |
| GWTE2TUX        | The BEA Tuxedo system is configured with<br><code>SECURITY=APP_PW,</code><br><code>USER_AUTH</code>   | No action is required.                                                                                                                                                    |
| GWTE2TUX        | The BEA Tuxedo system is configured with<br><code>SECURITY=ACL,</code><br><code>MANDATORY ACL.</code> | Set the user name using the <code>-u</code> option, and the group name, using the <code>-g</code> 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 Release 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`).

### 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"
```

### 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
```

### 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 the listing BEA TOP END Remote Configuration File) must match the port number of the BEA TOP END Network Agent.

- Software Requirements** The following software components are required:
- BEA Tuxedo Release 6.5
  - BEA TOP END 2.05
- Failures** A BEA Tuxedo client receives a `TPESVCFALL` 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 (`TPESVCFALL`), 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)`
- Setting Up a BEA Tuxedo Application*
- Administering a BEA Tuxedo Application at Run Time*
- BEA TOP END Remote Client/Server Services Guide*

# langinfo(5)

|             |                                                                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name        | <code>langinfo</code> - language information constants                                                                                                              |
| Synopsis    | <code>#include &lt;langinfo.h&gt;</code>                                                                                                                            |
| Description | This header file contains the constants used to identify items of <code>langinfo</code> data. The mode of <code>items</code> is given in <code>nl_types(5)</code> . |
|             | <code>DAY_1</code><br>Locale's equivalent of “sunday”                                                                                                               |
|             | <code>DAY_2</code><br>Locale's equivalent of “monday”                                                                                                               |
|             | <code>DAY_3</code><br>Locale's equivalent of “tuesday”                                                                                                              |
|             | <code>DAY_4</code><br>Locale's equivalent of “wednesday”                                                                                                            |
|             | <code>DAY_5</code><br>Locale's equivalent of “thursday”                                                                                                             |
|             | <code>DAY_6</code><br>Locale's equivalent of “friday”                                                                                                               |
|             | <code>DAY_7</code><br>Locale's equivalent of “saturday”                                                                                                             |
|             | <code>ABDAY_1</code><br>Locale's equivalent of “sun”                                                                                                                |
|             | <code>ABDAY_2</code><br>Locale's equivalent of “mon”                                                                                                                |
|             | <code>ABDAY_3</code><br>Locale's equivalent of “tue”                                                                                                                |
|             | <code>ABDAY_4</code><br>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"  |

## Section 5 - File Formats and Data Descriptions

---

|           |                              |
|-----------|------------------------------|
| 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(3c)`.

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(3c)`, `strftime(3c)`, `nl_types(5)`

# MIB(5)

Name MIB - 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(3c)`). 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()` or `tpsvrdone()` 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()` 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 *BEA Tuxedo FML Function Reference*. For example, one would use `Fchg32()` instead of using `Fchg()`.

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()`) 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.

## Section 5 - File Formats and Data Descriptions

---

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 the reply buffer and returned with the component MIB specific fields. The generic reply fields are described below.

**TA\_CLASS**

String valued FML32 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()` or `tpgetreply()`. 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()` 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()` and `Fchg32()`. 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()`.

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 rdbuf and contains cursor */
/*
 * GETNEXT 5 objects. Transfer TA_CURSOR from rdbuf.
 * 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(rdbuf, 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 rdbuf */
/* Use rdbuf as pre-image and update TA_MIBFIELD value
 * if matching
 */
Fcpy32(newrq, rdbuf);
Fconcat32(newrq, rdbuf); /* 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()`, `tpacall()` and `tpenqueue()`. 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()` or `tpacall()` to direct administrative requests within a transaction, the `TPNOTRAN` flag should be used or the user will get a failure (`TPETRAN`). When using `tpenqueue()` 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, rdbuf, 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()`, then a successful return from `tpcall()` indicates that the reply has been received. If the original request was generated using `tpacall()`, then the reply may

be received using `tpgetrply()`. If the original request was generated using `tpenqueue()` and a reply queue was specified in the queue control structure, then the reply may be received using `tpdequeue()`. 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", rdbuf, 0, rdbuf, rplen, flags);
/* Receive reply using call descriptor */
flags = TPNOCHANGE | TPSIGRSTRT;
rval = tpgetrply(cd, rdbuf, rplen, flags);
/* Receive reply using TPGETANY, may need to change buffer type */
flags = TPGETANY | TPSIGRSTRT;
rval = tpgetrply(rd, rdbuf, rplen, flags);
/* Dequeue reply, assumes qctl already setup */
flags = TPNOCHANGE | TPSIGRSTRT;
rval = tpdequeue("queue", "replyq", qctl, rdbuf, 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 (null) 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()` or `tpacall()` 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()` within a transaction. The enqueueing 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 be 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()`, which returns the FML32 define representing the data type; that is, `FLD_LONG`, `FLD_CHAR` and `FLD_STRING` (see `Fldtype`, `Fldtype32(3fml)`).

### 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 <code>ACTIVE</code> or <code>ACTIVE</code> equivalent. See the description of the <code>TA_STATE</code> attribute value for each class to determine which states qualify as <code>ACTIVE</code> 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 <code>ACTIVE</code> or <code>ACTIVE</code> equivalent. See the description of the <code>TA_STATE</code> attribute value for each class to determine which states qualify as <code>ACTIVE</code> equivalent.                                                              |

---

#### Position 2 - Inactive update permissions

---

|   |                                                                                                                                                                                                                                                                           |
|---|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| w | Attribute may be updated when the object is in an <code>INACTIVE</code> or <code>INACTIVE</code> equivalent state. See the description of the <code>TA_STATE</code> attribute value for each class to determine which states qualify as <code>INACTIVE</code> equivalent. |
| u | Attribute may be updated as described for the <code>w</code> permissions value. In addition, the combination of all attribute values identified with the <code>u</code> permissions character must be unique within the class.                                            |
| U | Attribute may be updated as described for the <code>w</code> 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.

---

|                           |                                                                                                                                         |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>LITSTRING</code>    | Literal string value.                                                                                                                   |
| <code>num</code>          | Numeric value.                                                                                                                          |
| <code>string[x..y]</code> | String value between <code>x</code> and <code>y</code> characters in length, not including the terminating <code>NULL</code> character. |
| <code>LMID</code>         | Shorthand for <code>string[1..30]</code> ( <i>no commas allowed</i> ). Represents a logical machine identifier.                         |
| <code>{x y z}</code>      | Select one of <code>x</code> , <code>y</code> or <code>z</code> .                                                                       |
| <code>{x y z}</code>      | Select zero or one of <code>x</code> , <code>y</code> or <code>z</code> .                                                               |
| <code>{x y z},*</code>    | Zero or more occurrences of <code>x</code> , <code>y</code> or <code>z</code> in a comma-separated list.                                |

---

|                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>low = num</i>                | Numeric value greater than or equal to <i>low</i> .                                                                                                                                                                                                                                                                                                                                                                                  |
| <i>low = num</i><br><i>high</i> | Numeric value greater than or equal to <i>low</i> and less than <i>high</i> .                                                                                                                                                                                                                                                                                                                                                        |
| 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--   | <i>string</i>        | N/A                  |
| TA_STATE(k)     | string | r--r--r--   | GET: VAL<br>SET: N/A | GET: N/A<br>SET: N/A |
| TA_GETSTATES    | string | r--r--r--   | <i>string</i>        | N/A                  |
| TA_INASTATES    | string | r--r--r--   | <i>string</i>        | N/A                  |
| TA_SETSTATES    | string | r--r--r--   | <i>string</i>        | 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)(*) | string | ru-r--r--   | <i>string</i>                | N/A                  |
| TA_ATTRIBUTE(r)(*) | long   | ru-r--r--   | 0 <= num                     | N/A                  |
| TA_STATE(k)        | string | rw-r--r--   | GET: VAL<br>SET: {NEW   INV} | GET: N/A<br>SET: N/A |
| TA_PERM(r)         | long   | rw-r--r--   | 0000 <= num <= 0777          | N/A                  |
| TA_FACTPERM        | long   | r--r--r--   | 0000 <= num <= 0777          | N/A                  |
| TA_MAXPERM         | long   | r--r--r--   | 0000 <= num <= 0777          | N/A                  |
| TA_ATTFLAGS        | long   | r--r--r--   | <i>long</i>                  | N/A                  |
| TA_DEFAULT         | string | r--r--r--   | <i>string</i>                | N/A                  |
| TA_VALIDATION      | string | r--r--r--   | <i>string</i>                | 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

|                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Attribute Semantics</b> | <p>TA_CLASSNAME: <i>string</i><br/>Class name. Only class names known to the system are accessible.</p> <p>TA_ATTRIBUTE: <i>long</i><br/>Attribute field identifier as defined in the system provided header file, for example, tpadm.h.</p> <p>TA_STATE:<br/><br/>GET: VALid<br/>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.</p> |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

---

`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.

---

|                  |                                                                                                                                                                                               |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>NEW</code> | Create <code>T_CLASSATT</code> object for application. State change allowed only when in the <code>INValid</code> state. Successful return leaves the object in the <code>VALid</code> state. |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

---

|                    |                                                                                                                                                |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>unset</code> | Modify <code>T_CLASSATT</code> object. Allowed only when in the <code>VALid</code> state. Successful return leaves the object state unchanged. |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------|

---

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>INValid</code> | Delete or reset <code>T_CLASSATT</code> object for application. State change allowed only when in the <code>VALid</code> state. Successful return leaves the object in either the <code>INValid</code> state or the <code>VALid</code> 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 <code>VALid</code> 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 <code>INValid</code> 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-max*

Numeric attribute value that must be between *min* and *max*, inclusive.

SIZE=*min-max*

String or array 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()`, `tpgetrply()` and `tpdequeue()`) 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()` or `tpgetrply()` 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.

[TAEYSTEM]

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 BEA 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** `tpacall(3c)`, `tpalloc(3c)`, `tpcall(3c)`, `tpdequeue(3c)`, `tpenqueue(3c)`, `tpgetreply(3c)`, `tprealloc(3c)`, **Introduction to FML Functions**, `Fadd`, `Fadd32(3fml)`, `Fchg`, `Fchg32(3fml)`, `Ffind`, `Ffind32(3fml)`, `AUTHSVR(5)`, `TM_MIB(5)`, `TMQFORWARD(5)`

*Setting Up a BEA Tuxedo Application*

*Administering a BEA Tuxedo Application at Run Time*

*Programming a BEA Tuxedo Application Using C*

*Programming a BEA Tuxedo Application Using FML*

---

# nl\_types(5)

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>        | nl_types - native language data types                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Synopsis</b>    | #include <nl_types.h>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Description</b> | The nl_types.h header file contains the following definitions:<br><br>nl_catd<br>used by the message catalog functions catopen(), catgets() and catclose() to identify a catalogue<br><br>nl_item<br>used by nl_langinfo() to identify items of langinfo() data. Values for objects of type nl_item are defined in langinfo.h.<br><br>NL_SETD<br>used by gencat() 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.<br><br>NL_MGSMAX<br>maximum number of messages per set<br><br>NL_SETMAX<br>maximum number of sets per catalogue.<br><br>NL_TEXTMAX<br>maximum size of a message.<br><br>DEF_NLSPATH<br>the default search path for locating catalogues. |
| <b>See Also</b>    | gencat(1), catgets(3c), catopen, catclose(3c), nl_langinfo(3c), langinfo(5)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

## servopts(5)

Name `servopts` - run-time options for 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][-t][ -- uargs][-v]`

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, 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.

The runtime association of service name with processing function within a server load module is called the dynamic service capability. The `tmadmin advertise` command can be used to change the list of services offered as the server continues to run.

Service names beginning with the '.' character are reserved for system servers. Application servers specifying such services will fail to boot.

-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 single-threaded servers; it cannot be used for multithreaded 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 messages per server.

--

The remaining arguments—*low\_water*, *terminate\_time*, *high\_water*, and *create\_time*—are used to control when servers

are spawned or deactivated. The following algorithm is used: If the load 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 message per server on the MSSQ or a workload of 50. *high\_water* defaults to an average of 2 messages per server, or a workload of 100. *create\_time* defaults to 50, and *terminate\_time* defaults to 60.

**Note:** For UNIX platforms only—The `alarm()` system call does not work as expected in servers running under server pool management. Because the code that terminates idle servers uses the `alarm()` call, user-written code intended to establish a customized signal handler fails to do so, despite the fact that calls to `Usignal()` do not result in errors.

`-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 (`TXTLCK`), data (`DATLOCK`), or the entire process (text and data - `PROCLCK`), 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`.
- `-t` specifies that the server in this BEA Tuxedo 7.1 or later application is allowed to interoperate with pre-Release 7.1 BEA Tuxedo software. The server may be a Workstation Listener (WSL) process (which when started with the `-t` option allows interoperability for all of its Workstation Handler—WSH—processes), a domain gateway (`GWTDOMAIN`) process, or a system or application server process.
- `--` 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()` function. `getopt()` should be used to parse them. Because all system arguments are processed prior to the call to `tpsvrinit()`, 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.
- `-v` prints out the service name/function name list to standard output, beginning with the following comment lines.
- ```
#
# List of services and corresponding handler functions built
# into the server
#
<servicename>:<functionname><NEWLINE>
<servicename>:<functionname><NEWLINE>
<servicename>:<functionname><NEWLINE>
. . . .
. . . .
```
- where the first three lines are comments and begin with a pound sign (#)

character. Each following line includes a service name and its corresponding function name built into the executable. The *servicename* field on any line can be an empty string if an “-s: *functionname*” is included on the *buildserver* command line. The *functionname* field is always present.

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)`, `tadmin(1)`, `tmboot(1)`, `txrpt(1)`, `tpsvrinit(3c)`, `UBBCONFIG(5)`

Setting Up a BEA Tuxedo Application

Administering a BEA Tuxedo Application at Run Time

`nice(2)`, `plock(2)`, `getopt(3)` in a UNIX system reference manual

TM_MIB(5)

- Name** TM_MIB - Management Information Base for core BEA Tuxedo system
- Synopsis** `#include <fm132.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()` 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_GROUP	Server groups
T_MACHINE	Machine specific attributes
T_MSG	Message queues
T_NETGROUP	Network groups
T_NETMAP	Machines to Netgroups

TM_MIB Classes

Class Name	Controls . . .
T_QUEUE	Server queue
T_ROUTING	Routing criteria
T_SERVER	Servers
T_SERVERCTXT	Server context
T_SERVICE	Services
T_SVCGRP	Service group
T_TLISTEN	BEA Tuxedo system 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 be 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	<p>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.</p> <p>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.</p> <p>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 and T_SERVERCTXT retrievals to restrict the retrieval to application servers only.</p> <p>TMIB_CONFIG A flag used to indicate that only configured groups and servers should be considered in satisfying the request.</p> <p>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.</p>
FML32 Field Tables	<p>The field table for the attributes described in this reference page is found in the file <code>udataobj/tpadm</code> relative to the root directory of the BEA Tuxedo system software installed on the system. The directory <code>\${TUXDIR}/udataobj</code> should be included by the application in the colon-separated list specified by the <code>FLDTBLDIR</code> environment variable, and the field table name <code>tpadm</code> should be included in the comma-separated list specified by the <code>FIELDTBLS</code> environment variable.</p>
Limitations	<p>Access to the header files and field tables for this MIB is being provided only on BEA Tuxedo Release 5.0 sites and later, both native and Workstation.</p> <p>Workstation access to this MIB is limited to runtime only access; the function <code>tpadmcall(3c)</code> is not supported on workstations.</p> <p>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.</p>

T_BRIDGE Class Definition

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

TM_MIB(5): 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 56 128} ⁵	N/A

(k) - GET key field

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

¹All attributes in Class T_BRIDGE are local attributes.

²The TA_LMID attribute must be fully specified for SET operations, that is, LMID1,LMID2.

³SET operation may only use TA_NETGROUP DEFAULTNET in BEA Tuxedo Release 6.4. GET operation may use any TA_NETGROUP defined for both LMID values.

⁴TA_SUSPTIME may be SET only if the TA_STATE is currently SUSPENDED or is being SET to SUSPENDED.

⁵The link-level encryption value of 40 bits is provided for backward compatibility.

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 BEA Tuxedo release (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.

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.

<i>unset</i>	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	<p>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.</p>
---------	---

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_RCVNUM: 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 | 56 | 128 }

The current encryption level for this link. The level is negotiated between machines when the link is established.

Note: The link-level encryption value of 40 bits is provided for backward compatibility.

Limitations None.

T_CLIENT Class Definition

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

TM_MIB(5): T_CLIENT Class Definition Attribute Table

Attribute ¹	Type	Permissions	Values	Default
TA_STATE(k)	string	R-XR-XR--	GET: "{ACT SUS DEA}" SET: "{ACT SUS DEA}"	N/A N/A
TA_CLIENTID(*)	string	R--R--R--	<i>string</i> [1..78]	N/A
TA_CLTNAME(k)	string	R--R--R--	<i>string</i> [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_CONTEXTID	long	R--R--R--	-2 <= num < 30,000	N/A
TA_SRVGRP(k)	string	R--R--R--	<i>string</i> [0..30]	N/A
TA_USRNAME(k)	string	R--R--R--	<i>string</i> [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--	<i>string</i> [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

Section 5 - File Formats and Data Descriptions

TM_MIB(5): T_CLIENT Class Definition Attribute Table

Attribute ¹	Type	Permissions	Values	Default
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_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 56 128} ²	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 THREAD 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

¹All attributes in Class T_CLIENT are local attributes.

²The link-level encryption value of 40 bits is provided for backward compatibility.

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.

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() or tpacall()) and from initiating further conversations (tpconnect()). 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.

<i>ACTIVE</i>	Activate a <i>SUSPENDED</i> <i>T_CLIENT</i> object. State change allowed only when in the <i>SUSPENDED</i> state. Successful return leaves the object in the <i>ACTIVE</i> state.
<i>unset</i>	Modify an existing <i>T_CLIENT</i> object. This combination is allowed only when in the <i>ACTIVE</i> or <i>SUSPENDED</i> state. Successful return leaves the object state unchanged.
<i>SUSPENDED</i>	Suspend the <i>T_CLIENT</i> object from making service requests (<i>tpcall()</i> or <i>tpacall()</i>), initiating conversations (<i>tpconnect()</i>), beginning transactions (<i>tpbegin()</i>), and enqueueing new requests (<i>tpenqueue()</i>). 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 <i>TPESYSTEM</i> error return and a system log message being generated indicating the situation. State change allowed only when in the <i>ACTIVE</i> state. Successful return leaves the object in the <i>SUSPENDED</i> state.

DEAD	<p>Abortively deactivate the <code>T_CLIENT</code> object. State change allowed only when in the <code>ACTIVE</code> or <code>SUSPENDED</code> state. The recommended method for deactivating clients is to first broadcast a warning message (<code>tpbroadcast</code>), then to suspend them (see <code>SET SUSPENDED</code> above), and finally to abortively deactivate them by setting the state to <code>DEAD</code>. Successful return leaves the object in the <code>DEAD</code> state.</p> <p>Limitation: Workstation handlers (<code>T_CLIENT:TA_WSH == Y</code>) may not be set to a state of <code>DEAD</code>.</p> <p>The system may not be able to <i>kill</i> 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.</p>
------	---

`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()` time via the `cltname` element of the `TPINIT` structure.

`TA_IDLETIME`: $0 \leq num$

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`: $1 \leq num$

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_CONTEXTID: $-2 \leq num < 30,000$

Identifier for this particular application association.

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()` time.

TA_USRNAME: *string*[0..30]

User name associated with client at `tpinit()` time via the `username` 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: $0 \leq num$

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: $0 \leq num$

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: $0 \leq num$

Number of conversations initiated by this client via `tpconnect()`.

- TA_NUMDEQUEUE:** 0 <= *num*
Number of dequeue operations initiated by this client via `tpdequeue()`.
- TA_NUMENQUEUE:** 0 <= *num*
Number of enqueue operations initiated by this client via `tpenqueue()`.
- TA_NUMPOST:** 0 <= *num*
Number of postings initiated by this client via `tppost()`.
- TA_NUMREQ:** 0 <= *num*
Number of requests made by this client via `tpcall()` or `tpacall()`.
- TA_NUMSUBSCRIBE:** 0 <= *num*
Number of subscriptions made by this client via `tpsubscribe()`.
- 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()` for details on this characteristic.
- TA_CURCONV:** 0 <= *num*
Number of conversations initiated by this client via `tpconnect()` that are still active.
- TA_CURENCRYPTBITS:** { 0 | 40 | 56 | 128 }
The current encryption level for this client. The level is negotiated when the link is established.
- Note:** The link-level encryption value of 40 bits is provided for backward compatibility.
- TA_CURREQ:** 0 <= *num*
Number of requests initiated by this client via `tpcall()` or `tpacall()` 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 are converted to one of the following formats.

- "0xhex-digits"
- "\\xhex-digits"

A string in either format must contain an even number of valid hex digits. Such a string is translated internally into a character array containing the hexadecimal representations of the string specified.

For TCP/IP addresses one of the following formats is used.

- "//hostname:port"
- "//#. #. #. #:port_number"

Each # (pound) sign represents a decimal number in the range of 0 to 255. The value of *port_number* is a decimal number in the range of 0 to 65535.

Note: Some port numbers may be reserved for the underlying transport protocols (such as TCP/IP) used by your system. Check the documentation for your transport protocols to find out which numbers, if any, are reserved on your system.

Non-Workstation clients 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 | THREAD | 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 Definition

Overview The T_CONN class represents run-time attributes of active conversations within an application.

Attribute Table

TM_MIB(5): 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" SET: N/A	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_CONSSNDCNT	long	R--R--R--	0 <= num	N/A
TA_CONSSRVID	long	R--R--R--	1 <= num < 30,001	N/A

(k) - GET key field

¹All 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.

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_CONNOSDCNT: 0 <= *num*

Number of `tpsend()` 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.

Section 5 - File Formats and Data Descriptions

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: $1 \leq num$

Process identifier for the subordinate in the conversation.

TA_CONSSSNCNT: $0 \leq num$

Number of `tpsend()` calls done by the subordinate.

TA_CONSSSRVID: $1 \leq num < 30,001$

Server identifier for the subordinate in the conversation.

Limitations None.

T_DEVICE Class Definition

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

TM_MIB(5): T_DEVICE Class Definition Attribute Table

Attribute ¹	Type	Permissions	Values	Default
TA_LMID(*)	string	ru-r--r--	<i>LMID</i>	local_lmld
TA_CFGDEVICE(r)(*)	string	ru-r--r--	<i>string</i> [2..64]	N/A
TA_DEVICE(*)	string	ru-r--r--	<i>string</i> [2..64]	TA_CFGDEVICE
TA_DEVOFFSET(*)	long	ru-r--r--	0 <= <i>num</i>	0
TA_DEVSIZE(r)	long	rw-r--r--	0 <= <i>num</i>	1000 ³
TA_DEVINDEX(*) ²	long	r--r--r--	0 <= <i>num</i>	N/A
TA_STATE(k)	string	rwxr--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

¹All 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	<p>TA_LMID: <i>LMID</i></p> <p>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 <code>T_MACHINE</code> entry is defined). It is required as a key field on <code>SET</code> operations when accessing a booted application. If specified when accessing the <code>T_DEVICE</code> class in an unconfigured application, this attribute is ignored.</p>
	<p>TA_CFGDEVICE: <i>string[2..64]</i></p> <p>Absolute pathname of the file or device where the BEA Tuxedo file system is stored or is to be stored.</p>
	<p>TA_DEVICE: <i>string[2..64]</i></p> <p>Absolute pathname of the device list entry.</p>
	<p>TA_DEVOFFSET: <i>0 <= num</i></p> <p>The offset, in blocks, at which space on this <code>TA_DEVICE</code> begins for use within the BEA Tuxedo System VTOC specified by <code>TA_CFGDEVICE</code>. Limitation: This attribute must be set to 0 for the first device list entry (<code>TA_DEVICE</code>) on the BEA Tuxedo filesystem (<code>TA_CFGDEVICE</code>).</p>
	<p>TA_DEVSZ: <i>0 <= num</i></p> <p>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 <code>NEW</code>.</p>
	<p>TA_DEVINDEX: <i>0 <= num</i></p> <p>Device index for <code>TA_DEVICE</code> within the device list addressed by <code>TA_CFGDEVICE</code>. This attribute value is used for identification purposes only in getting and setting attribute values relating to particular devices within a BEA Tuxedo filesystem.</p>
	<p>TA_STATE:</p> <p>GET: {<code>VALid</code>}</p> <p>A <code>GET</code> operation will retrieve run-time information for the selected <code>T_DEVICE</code> object(s). The following states indicate the meaning of a <code>TA_STATE</code> returned in response to a <code>GET</code> request.</p>

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 Definition

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

TM_MIB(5): T_DOMAIN Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_IPCKEY(r)	long	rw-r--r--	32,769 <= 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}" SET: "{NEW INV ACT INA FIN}"	N/A 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 < 32,768	N/A
TA_LICSERIAL	string	R--R--R--	string[0..78]	N/A
TA_MIBMASK	long	rwx-----	0 <= num <= 0777	0000

TM_MIB(5): T_DOMAIN Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_MAXACCESSERS	long	rwyr--r--	1 <= num < 32,768	50
TA_MAXCONV	long	rwyr--r--	0 <= num < 32,768	64
TA_MAXGTT	long	rwyr--r--	0 <= num < 32,768	100
TA_MAXBUFSTYPE	long	rw-r--r--	1 <= num < 32,768	32
TA_MAXBUFTYPE	long	rw-r--r--	1 <= num < 32,768	16
TA_MAXDRT	long	rw-r--r--	0 <= num < 32,768	0
TA_MAXGROUPS	long	rw-r--r--	100 <= num < 32,766	100
TA_MAXNETGROUPS	long	rw-r--r--	1 <= num <= 8,192	8
TA_MAXMACHINES	long	rw-r--r--	256 <= num < 8,191	256
TA_MAXQUEUES	long	rw-r--r--	1 <= num < 8,192	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 < 8,192	50
TA_MAXSERVICES	long	rw-r--r--	1 <= num < 32,766	100
TA_MAXACLGROUPS	long	rw-r--r--	1 <= num = < 16,384	16,384
TA_CMTRET	string	rwyr--r--	"{COMPLETE LOGGED}"	"COMPLETE"
TA_LDBAL	string	rwyr--r--	"{Y N}"	"Y"
TA_NOTIFY	string	rwyr--r--	"{DIPIN SIGNAL THREAD 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"

Section 5 - File Formats and Data Descriptions

TM_MIB(5): T_DOMAIN Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_MAXACCESSERS	long	rwyr--r--	1 <= num < 32,768	50
TA_MAXCONV	long	rwyr--r--	0 <= num < 32,768	64
TA_MAXGTT	long	rwyr--r--	0 <= num < 32,768	100
TA_MAXBUFSTYPE	long	rw-r--r--	1 <= num < 32,768	32
TA_MAXBUFTYPE	long	rw-r--r--	1 <= num < 32,768	16
TA_MAXDRT	long	rw-r--r--	0 <= num < 32,768	0
TA_MAXGROUPS	long	rw-r--r--	100 <= num < 32,766	100
TA_MAXNETGROUPS	long	rw-r--r--	1 <= num <= 8,192	8
TA_MAXMACHINES	long	rw-r--r--	256 <= num < 8,191	256
TA_MAXQUEUES	long	rw-r--r--	1 <= num < 8,192	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 < 8,192	50
TA_MAXSERVICES	long	rw-r--r--	1 <= num < 32,766	100
TA_MAXACLGROUPS	long	rw-r--r--	1 <= num = < 16,384	16,384
TA_CMTRET	string	rwyr--r--	"{COMPLETE LOGGED}"	"COMPLETE"
TA_LDBAL	string	rwyr--r--	"{Y N}"	"Y"
TA_NOTIFY	string	rwyr--r--	"{DIPIN SIGNAL THREAD 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"

TM_MIB(5): T_DOMAIN Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
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 < 32,768	300 ³
TA_BLOCKTIME	long	rwxr-xr--	0 <= num < 32,768	60 ³
TA_DBBLWAIT	long	rwxr-xr--	0 <= num < 32,768	20 ³
TA_SANITYSCAN	long	rwxr-xr--	0 <= num < 32,768	120 ³
TA_CURDRT	long	r--r--r--	0 <= num < 32,768	N/A
TA_CURGROUPS	long	r--r--r--	0 <= num < 32,768	N/A
TA_CURMACHINES	long	r--r--r--	0 <= num < 32,768	N/A
TA_CURQUEUES	long	r--r--r--	0 <= num < 32,768	N/A
TA_CURRFT	long	r--r--r--	0 <= num < 32,768	N/A
TA_CURRTDATA	long	r--r--r--	0 <= num < 32,768	N/A
TA_CURSERVERS	long	r--r--r--	0 <= num < 32,768	N/A
TA_CURSERVICES	long	r--r--r--	0 <= num < 32,768	N/A
TA_CURSTYPE	long	r--r--r--	0 <= num < 32,768	N/A
TA_CURTYPE	long	r--r--r--	0 <= num < 32,768	N/A

Section 5 - File Formats and Data Descriptions

TM_MIB(5): T_DOMAIN Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_HWDRT	long	r--r--r--	0 <= num < 32,768	N/A
TA_HWGROUPS	long	r--r--r--	0 <= num < 32,768	N/A
TA_HWMACHINES	long	r--r--r--	0 <= num < 32,768	N/A
TA_HWQUEUES	long	r--r--r--	0 <= num < 32,768	N/A
TA_HWRFT	long	r--r--r--	0 <= num < 32,768	N/A
TA_HWRDATA	long	r--r--r--	0 <= num < 32,768	N/A
TA_HWSERVERS	long	r--r--r--	0 <= num < 32,768	N/A
TA_HWSERVICES	long	r--r--r--	0 <= num < 32,768	N/A
TA_SEC_PRINCIPAL_NAME	string	rwxr--r--	string[0..511]	" "
TA_SEC_PRINCIPAL_LOCATION	string	rwxr--r--	string[0..511]	" "
TA_SEC_PRINCIPAL_PASSVAR	string	rwxr--r--	string[0..511]	" "
TA_SIGNATURE_AHEAD	long	rwxr--r--	1 <= num <= 2147483647	3600
TA_SIGNATURE_BEHIND	long	rwxr--r--	1 <= num <= 2147483647	604800
TA_SIGNATURE_REQUIRED	string	rwxr--r--	"{Y N}"	"N"
TA_ENCRYPTION_REQUIRED	string	rwxr--r--	"{Y N}"	"N"

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

¹UID and GID as known to the UNIX system

²num must be a multiple of 5

³Specify num so that num times TA_SCANUNIT is approximately "Default"

Attribute TA_IPCKEY: 32,769 <= num < 262,144

Semantics 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 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.

<i>ACTive</i>	<i>T_DOMAIN</i> 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_PPID 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.
<i>unset</i>	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* < 32,768

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: $0 \leq num \leq 0777$

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: $1 \leq num < 32,768$

Default maximum number of clients and servers that can be simultaneously connected to the bulletin board on any particular machine in this application. If not specified, the default maximum number is 50. The **T_DOMAIN** value for this attribute can be overridden in the **T_MACHINE** class on a per-machine basis.

System administration processes, such as the **BBL**, **restartsrv**, **cleanup**, **tmshutdown()**, and **tmadmin()**, need not be accounted for in this value, but the **DBBL**, all bridge processes, all system-supplied and application server processes, and all potential client processes at a particular site need to be counted. (Examples of system-supplied servers are **AUTHSVR**, **TMQUEUE**, **TMQFORWARD**, **TMUSREVT**, **TMSYSEVT**, **TMS**—see **T_GROUP** **TA_TMSNAME** attribute, **TMS_QM**, **GWTDOMAIN**, and **WSL**.) If the application is booting Workstation Listeners (WSLs) at a particular site, both the WSLs and the number of potential Workstation Handlers (WSHs) that may be booted need to be counted.

Note that for BEA Tuxedo pre-Release 7.1 (6.5 or earlier), both the **TA_MAXACCESSERS** and **TA_MAXSERVERS** attributes for an application play a part in the user license checking scheme. Specifically, a machine is not allowed to boot if the number of **TA_MAXACCESSERS** for that machine + the number of **TA_MAXACCESSERS** for the machine (or machines) already running in the application is greater than the number of **TA_MAXSERVERS** + user licenses for the application. Thus, the total number of **TA_MAXACCESSERS** for an application must be less than or equal to the number of **TA_MAXSERVERS** + user licenses for the application.

Note also that the user license checking scheme in BEA Tuxedo Release 7.1 or later considers only the following two factors when performing its checks: the number of user licenses for an application and the number of licenses currently in use for the application. When all user licenses are in use, no new clients are allowed to join the application.

Limitation: Changes to this attribute do not affect active or already configured T_MACHINE objects.

TA_MAXCONV: $0 \leq num < 32,768$

Maximum number of simultaneous conversations in which clients and servers on any particular machine in this application can be involved. If not specified, the default is 64 if any conversational servers are defined in the T_SERVER class, or 1 otherwise. The maximum number of simultaneous conversations per server is 64. The T_DOMAIN value for this attribute can be overridden in the T_MACHINE class on a per-machine basis.

Limitation: Changes to this attribute do not affect active or already configured T_MACHINE objects.

TA_MAXGTT: $0 \leq num < 32,768$

Maximum number of simultaneous global transactions in which any particular machine in this application can be involved. If not specified, the default is 100. The T_DOMAIN value for this attribute can be overridden in the T_MACHINE class on a per-machine basis.

Limitation: Changes to this attribute do not affect active or already configured T_MACHINE objects.

TA_MAXBUFSTYPE: $1 \leq num < 32,768$

Maximum number of buffer subtypes that can be accommodated in the bulletin board buffer subtype table.

TA_MAXBUFTYPE: $1 \leq num < 32,768$

Maximum number of buffer types that can be accommodated in the bulletin board buffer type table.

TA_MAXDRT: $0 \leq num < 32,768$

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: $100 \leq num < 32,766$

Maximum number of server groups that can be accommodated in the bulletin board server group table. Limitation: BEA Tuxedo 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: $1 \leq num < 8,192$

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: $256 \leq num < 8,191$

Maximum number of machines that can be accommodated in the bulletin board machine table. Limitation: BEA Tuxedo 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: $1 \leq num < 8,192$

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: $0 \leq num < 32,768$

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: $0 \leq num < 32,761$

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 < 8,192

Maximum number of servers to be accommodated in the bulletin board server table for this application. If not specified, the default is 50.

All instances of system-supplied and application servers available to an application need to be accounted for in the bulletin board server table, which is a global table, meaning that the same server table resides on each machine in the application. Examples of system-supplied servers are AUTHSVR, TMQUEUE, TMQFORWARD, TMUSREVT, TMSYSEVT, TMS (see T_GROUP TA_TMSNAME attribute), TMS_QM, GWTDOMAIN, and WSL.

Administration of each BEA Tuxedo system site adds approximately one system-supplied server. Additionally, the DBBL process and all BBL, bridge, and WSH processes must be accounted for in the TA_MAXSERVERS value.

TA_MAXSERVICES: 1 <= num < 32,766

Maximum number of services to be accommodated in the bulletin board service table. If not specified, the default is 100.

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 < 16,384

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 `tpsamt()` 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 | THREAD | 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()` 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.

The value `DIPIN` specifies that dip-in-based notification detection should be used. This means that the system will detect notification messages only 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 `THREAD` specifies that `THREAD` notification should be used. This means that the system dedicates a separate thread for the receipt of unsolicited messages and dispatches the unsolicited message handler in that thread. Only one unsolicited message handler executes at one time per BEA Tuxedo application association. This value is allowed only on platforms that offer support for multithreading. COBOL clients cannot use `THREAD` notification, and will default to `DIPIN` if `THREAD` is in effect.

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.

Limitations: Run-time modifications to this attribute do not affect active clients. All signaling of native client processes is done by administrative system processes and not by application processes. Therefore, only native clients running with the same UNIX system user identifier as the application administrator can be notified using the `SIGNAL` method. Workstation clients

may use the `SIGNAL` method, regardless of which user identifier they are running under.

Note: The `SIGNAL` notification method is not available for MS-DOS clients.

`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)` or `TPINITIALIZE(3cbl)`.

Limitations: (1) Updates to this attribute value in a running application affect only newly started clients and newly configured `T_SERVER` objects.

(2) Setting `TA_SYSTEM_ACCESS` to `PROTECTED` may not be effective for multithreaded servers because it is possible that while one thread is executing BEA Tuxedo code, which means it is attached to the Bulletin Board, another thread might be executing user code. The BEA Tuxedo system cannot prevent such situations.

`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`: $0 \leq \text{num} \leq 60$ (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`: $0 \leq \text{num} < 32,768$

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`: $0 \leq \text{num} < 32,768$

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: $0 \leq num < 32,768$

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: $0 \leq num < 32,768$

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: $0 \leq num < 32,768$

Current number of in use bulletin board routing table entries.

TA_CURGROUPS: $0 \leq num < 32,768$

Current number of in use bulletin board server group table entries.

TA_CURMACHINES: $0 \leq num < 32,768$

Current number of configured machines.

TA_CURQUEUES: $0 \leq num < 32,768$

Current number of in use bulletin board queue table entries.

TA_CURRFT: $0 \leq num < 32,768$

Current number of in use bulletin board routing criteria range table entries.

TA_CURRTDATA: $0 \leq num < 32,768$

Current size of routing table string pool.

TA_CURSERVERS: $0 \leq num < 32,768$

Current number of in use bulletin board server table entries.

TA_CURSERVICES: $0 \leq num < 32,768$

Current number of in use bulletin board service table entries.

TA_CURSTYPE: $0 \leq num < 32,768$

Current number of in use bulletin board subtype table entries.

TA_CURTYPE: $0 \leq num < 32,768$

Current number of in use bulletin board type table entries.

TA_HWDRT: $0 \leq num < 32,768$

High water number of in use bulletin board routing table entries.

TA_HWGROUPS: $0 \leq num < 32,768$

High water number of in use bulletin board server group table entries.

TA_HWMACHINES: $0 \leq num < 32,768$

High water number of configured machines.

TA_HWQUEUES: $0 \leq num < 32,768$

High water number of in use bulletin board queue table entries.

TA_HWRFT: $0 \leq num < 32,768$

High water number of in use bulletin board routing criteria range table entries.

TA_HWRDATA: $0 \leq num < 32,768$

High water size of routing table string pool.

TA_HWSERVERS: $0 \leq num < 32,768$

High water number of in use bulletin board server table entries.

TA_HWSERVICES: $0 \leq num < 32,768$

High water number of in use bulletin board service table entries.

TA_SEC_PRINCIPAL_NAME: *string*[0..511]

Security principal name identification string to be used for authentication purposes by an application running BEA Tuxedo 7.1 or later software. This attribute may contain a maximum of 511 characters (excluding the terminating null character). The principal name specified for this attribute becomes the identity of one or more system processes running in this domain.

TA_SEC_PRINCIPAL_NAME can be specified at any of the following four levels in the configuration hierarchy: T_DOMAIN class, T_MACHINE class, T_GROUP class, and T_SERVER class. A principal name at a particular configuration level can be overridden at a lower level. If TA_SEC_PRINCIPAL_NAME is not specified at any of these levels, the principal name for the application defaults to the TA_DOMAINID string for this domain.

Note that TA_SEC_PRINCIPAL_NAME is one of a trio of attributes, the other two being TA_SEC_PRINCIPAL_LOCATION and TA_SEC_PRINCIPAL_PASSVAR. The latter two attributes pertain to opening decryption keys during application booting for the system processes running in a BEA Tuxedo 7.1 or later application. When only

TA_SEC_PRINCIPAL_NAME is specified at a particular level, the system sets each of the other two attributes to a NULL (zero length) string.

TA_SEC_PRINCIPAL_LOCATION: *string*[0..511]

Location of the file or device where the decryption (private) key for the principal specified in TA_SEC_PRINCIPAL_NAME resides. This attribute may contain a maximum of 511 characters (excluding the terminating null character).

TA_SEC_PRINCIPAL_LOCATION can be specified at any of the following four levels in the configuration hierarchy: T_DOMAIN class, T_MACHINE class, T_GROUP class, and T_SERVER class. When specified at any of these levels, this attribute must be paired with the TA_SEC_PRINCIPAL_NAME attribute; otherwise, its value is ignored. (TA_SEC_PRINCIPAL_PASSVAR is optional; if not specified, the system sets it to a NULL—zero length—string.)

TA_SEC_PRINCIPAL_PASSVAR: *string*[0..511]

Variable in which the password for the principal specified in TA_SEC_PRINCIPAL_NAME is stored. This attribute may contain a maximum of 511 characters (excluding the terminating null character).

TA_SEC_PRINCIPAL_PASSVAR can be specified at any of the following four levels in the configuration hierarchy: T_DOMAIN class, T_MACHINE class, T_GROUP class, and T_SERVER class. When specified at any of these levels, this attribute must be paired with the TA_SEC_PRINCIPAL_NAME attribute; otherwise, its value is ignored. (TA_SEC_PRINCIPAL_LOCATION is optional; if not specified, the system sets it to a NULL—zero length—string.)

During initialization, the administrator must provide the password for each of the decryption keys configured with TA_SEC_PRINCIPAL_PASSVAR. The system automatically encrypts the password entered by the administrator and assigns each encrypted password to the associated password variable.

TA_SIGNATURE_AHEAD: 1 <= *num* <= 2147483647

Number of seconds into the future that a digital signature's timestamp is allowed to be, when compared to the local machine's clock. If not specified, the default is 3600 seconds (one hour). This attribute only applies to applications running BEA Tuxedo 7.1 or later software.

TA_SIGNATURE_BEHIND: 1 <= *num* <= 2147483647

Number of seconds into the past that a digital signature's timestamp is allowed to be, when compared to the local machine's clock. If not specified,

the default is 604800 seconds (one week). This attribute only applies to applications running BEA Tuxedo 7.1 or later software.

`TA_SIGNATURE_REQUIRED`: {Y | N}

If set to Y, every process running in this domain requires a digital signature on its input message buffer. If not specified, the default is N. This attribute only applies to applications running BEA Tuxedo 7.1 or later software.

`TA_SIGNATURE_REQUIRED` can be specified at any of the following four levels in the configuration hierarchy: `T_DOMAIN` class, `T_MACHINE` class, `T_GROUP` class, and `T_SERVICE` class. Setting `SIGNATURE_REQUIRED` to Y at a particular level means that signatures are required for all processes running at that level or below.

`TA_ENCRYPTION_REQUIRED`: {Y | N}

If set to Y, every process running in this domain requires an encrypted input message buffer. If not specified, the default is N. This attribute only applies to applications running BEA Tuxedo 7.1 or later software.

`TA_ENCRYPTION_REQUIRED` can be specified at any of the following four levels in the configuration hierarchy: `T_DOMAIN` class, `T_MACHINE` class, `T_GROUP` class, and `T_SERVICE` class. Setting `TA_ENCRYPTION_REQUIRED` to Y at a particular level means that encryption is required for all processes running at that level or below.

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()` 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 Definition

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

TM_MIB(5): T_GROUP Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_SRVGRP(r)(*)	string	rU-r--r--	<i>string</i> [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--r--	GET: "{ACT INA MIG}" SET: "{NEW INV ACT RAC INA MIG}"	N/A N/A
TA_CURLMID(k)	string	R--R--R--	LMID	N/A
TA_ENVFILE	string	rwyr--r--	<i>string</i> [0..78]	" "
TA_OPENINFO	string	rwyr--r--	<i>string</i> [0..256]	" "
TA_CLOSEINFO	string	rwyr--r--	<i>string</i> [0..256]	" "
TA_TMSCOUNT	long	rw-r--r--	0 or 2 <= num < 11	3
TA_TMSNAME(k)	string	rw-r--r--	<i>string</i> [0..78]	" "
TA_SEC_PRINCIPAL_NAME	string	rwxr--r--	<i>string</i> [0..511]	" "
TA_SEC_PRINCIPAL_LOCATION	string	rwxr--r--	<i>string</i> [0..511]	" "
TA_SEC_PRINCIPAL_PASSVAR	string	rwxr--r--	<i>string</i> [0..511]	" "
TA_SIGNATURE_REQUIRED	string	rwxr--r--	"{Y N}"	"N"
TA_ENCRYPTION_REQUIRED	string	rwxr--r--	"{Y N}"	"N"

(k) - GET key field

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

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

Section 5 - File Formats and Data Descriptions

Attribute TA_SRVGRP: *string*[1..30]
Semantics 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: $1 \leq num < 30,000$
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.

ACTIVE	T_GROUP object defined and active (TMS and/or application servers). Server groups with non null strings 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.
<i>unset</i>	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	<p>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).</p> <p>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.</p> <p>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.</p> <p>Successful return leaves the object in the ACTive state.</p>
ReACTivate	<p>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.</p> <p>The TMIB_NOTIFY TA_FLAG value should be used when reactivating a server group if status on individual servers is required.</p>
INActive	<p>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.</p> <p>The TMIB_NOTIFY TA_FLAG value should be used when deactivating a server group if status on individual servers is required.</p>

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_OPENINFO: *string*[0..256]

The resource manager instance-dependent information needed when opening the resource manager for this group. The value must be enclosed in double quotes and must be less than or equal to 256 characters in length.

If a non null string other than TMS is specified for the TA_TMSNAME attribute, the TA_OPENINFO attribute value provides the resource manager dependent information needed when initiating access to the resource manager. Otherwise, the TA_OPENINFO attribute value is ignored.

A null string value for the TA_OPENINFO attribute means that the resource manager for this group (if specified) does not require any application specific information to `open` access to the resource.

The format of the TA_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 the published name of the vendor's transaction (XA) interface followed immediately by a colon (:).

For BEA Tuxedo /Q databases, the format is:

On UNIX

```
OPENINFO = "TUXEDO/QM:qmconfig;qspace"
```

On Windows NT

```
OPENINFO = "TUXEDO/QM:qmconfig;qspace"
```

In AS/400 environment

```
OPENINFO = "TUXEDO/QM:qmconfig;qspace"
```

In OpenVMS environment

```
OPENINFO = "TUXEDO/QM,[a.b.c]qmconfig,qspace"
```

where TUXEDO/QM is the published name of the BEA Tuxedo /Q XA interface, *qmconfig* is replaced with the name of the QMCONFIG (see `qmadmin(1)`) on which the queue space resides, and *qspace* is replaced with the name of the queue space. For NT and AS/400, the separator after *qmconfig* must be a semicolon (;). For OpenVMS, the separator after TUXEDO/QM and after *qmconfig* must be a comma (,).

For other vendors' databases, the format of the TA_OPENINFO string is specific to the particular vendor providing the underlying resource manager.

Limitation: Run-time modifications to this attribute will not affect active servers in the group.

`TA_CLOSEINFO`: *string*[0..256]

The resource manager instance-dependent information needed when closing the resource manager for this group. The value must be enclosed in double quotes and must be less than or equal to 256 characters in length. Note that a `TA_CLOSEINFO` string is not used for BEA Tuxedo /Q databases.

If a non null string other than `TMS` is specified for the `TA_TMSNAME` attribute, the `TA_CLOSEINFO` attribute value provides the resource manager-dependent information needed when terminating access to the resource manager. Otherwise, the `TA_CLOSEINFO` attribute value is ignored.

A null string value for the `TA_CLOSEINFO` attribute means that the resource manager for this group (if specified) does not require any application specific information to `close` access to the resource.

The format of the `TA_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 the published name of the vendor's transaction (XA) interface followed immediately by a colon (:).

Limitation: Run-time modifications to this attribute will not affect active servers in the group.

`TA_TMSCOUNT`: 0 or $2 \leq num < 11$

If a non null string 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()`, and ended with `tpcommit()/tpabort()`).

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.

`TA_SEC_PRINCIPAL_NAME: string[0..511]`

Security principal name identification string to be used for authentication purposes by an application running BEA Tuxedo 7.1 or later software. This attribute may contain a maximum of 511 characters (excluding the terminating null character). The principal name specified for this attribute becomes the identity of one or more system processes running in this group.

`TA_SEC_PRINCIPAL_NAME` can be specified at any of the following four levels in the configuration hierarchy: `T_DOMAIN` class, `T_MACHINE` class, `T_GROUP` class, and `T_SERVER` class. A principal name at a particular configuration level can be overridden at a lower level. If

`TA_SEC_PRINCIPAL_NAME` is not specified at any of these levels, the principal name for the application defaults to the `TA_DOMAINID` string for this domain.

Note that `TA_SEC_PRINCIPAL_NAME` is one of a trio of attributes, the other two being `TA_SEC_PRINCIPAL_LOCATION` and `TA_SEC_PRINCIPAL_PASSVAR`. The latter two attributes pertain to opening decryption keys during application booting for the system processes running in a BEA Tuxedo 7.1 or later application. When only `TA_SEC_PRINCIPAL_NAME` is specified at a particular level, the system sets each of the other two attributes to a NULL (zero length) string.

`TA_SEC_PRINCIPAL_LOCATION: string[0..511]`

Location of the file or device where the decryption (private) key for the principal specified in `TA_SEC_PRINCIPAL_NAME` resides. This attribute may contain a maximum of 511 characters (excluding the terminating null character).

`TA_SEC_PRINCIPAL_LOCATION` can be specified at any of the following four levels in the configuration hierarchy: `T_DOMAIN` class, `T_MACHINE` class, `T_GROUP` class, and `T_SERVER` class. When specified at any of these levels, this attribute must be paired with the `TA_SEC_PRINCIPAL_NAME` attribute; otherwise, its value is ignored. (`TA_SEC_PRINCIPAL_PASSVAR` is optional; if not specified, the system sets it to a NULL—zero length—string.)

`TA_SEC_PRINCIPAL_PASSVAR: string[0..511]`

Variable in which the password for the principal specified in `TA_SEC_PRINCIPAL_NAME` is stored. This attribute may contain a maximum of 511 characters (excluding the terminating null character).

`TA_SEC_PRINCIPAL_PASSVAR` can be specified at any of the following four levels in the configuration hierarchy: `T_DOMAIN` class, `T_MACHINE` class, `T_GROUP` class, and `T_SERVER` class. When specified at any of these levels, this attribute must be paired with the `TA_SEC_PRINCIPAL_NAME` attribute; otherwise, its value is ignored. (`TA_SEC_PRINCIPAL_LOCATION` is optional; if not specified, the system sets it to a `NULL`—zero length—string.)

During initialization, the administrator must provide the password for each of the decryption keys configured with `TA_SEC_PRINCIPAL_PASSVAR`. The system automatically encrypts the password entered by the administrator and assigns each encrypted password to the associated password variable.

`TA_SIGNATURE_REQUIRED`: { Y | N }

If set to `Y`, every process running in this group requires a digital signature on its input message buffer. If not specified, the default is `N`. This attribute only applies to applications running BEA Tuxedo 7.1 or later software.

`TA_SIGNATURE_REQUIRED` can be specified at any of the following four levels in the configuration hierarchy: `T_DOMAIN` class, `T_MACHINE` class, `T_GROUP` class, and `T_SERVICE` class. Setting `SIGNATURE_REQUIRED` to `Y` at a particular level means that signatures are required for all processes running at that level or below.

`TA_ENCRYPTION_REQUIRED`: { Y | N }

If set to `Y`, every process running in this group requires an encrypted input message buffer. If not specified, the default is `N`. This attribute only applies to applications running BEA Tuxedo 7.1 or later software.

`TA_ENCRYPTION_REQUIRED` can be specified at any of the following four levels in the configuration hierarchy: `T_DOMAIN` class, `T_MACHINE` class, `T_GROUP` class, and `T_SERVICE` class. Setting `TA_ENCRYPTION_REQUIRED` to `Y` at a particular level means that encryption is required for all processes running at that level or below.

Limitations None.

T_MACHINE Class Definition

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

TM_MIB(5): T_MACHINE Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_LMID(r)(*) ¹	string	rU-r--r--	<i>string</i> [1..30]	N/A
TA_P MID(r)(*) ¹	string	rU-r--r--	<i>string</i> [1..30]	N/A
TA_TUXCONFIG(r)	string	rw-r--r--	<i>string</i> [2..64]	N/A
TA_TUXDIR(r)	string	rw-r--r--	<i>string</i> [2..78]	N/A
TA_APPDIR(r)	string	rw-r--r--	<i>string</i> [2..8]	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--	<i>string</i> [0..78]	" "
TA_PERM	long	rwyr--r--	0001 <= num <= 0777	(²)
TA_ULOGPFX	string	rwyr--r--	<i>string</i> [0..78]	(³)
TA_TYPE	string	rw-r--r--	<i>string</i> [0..15]	" "
TA_MAXACCESSERS	long	rw-r--r--	1 <= num < 32,768	(²)
TA_MAXCONV	long	rw-r--r--	0 <= num < 32,768	(²)
TA_MAXGTT	long	rw-r--r--	0 <= num < 32,768	(²)
TA_MAXWSCLIENTS	long	rw-r--r--	0 <= num < 32,768	0

Section 5 - File Formats and Data Descriptions

TM_MIB(5): T_MACHINE Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_MAXACLCACHE	long	rw-r--r--	10 <= num <= 32,000	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 < 2,049	100
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_FADDR	string	rw-r--r--	string[0..78]	" "
TA_FRANGE	long	rw-r--r--	1 <= num <= 65,535	1
TA_CMPLIMIT	string	rwyr-yr--	"remote[, local]"	MAXLONG, MAXLONG
TA_TMNETLOAD	long	rwyr-yr--	0 <= num < 32,768	0
TA_SPINCOUNT	long	rwyr-yr--	0 <= num	0
TA_ROLE	string	r--r--r--	"{MASTER 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_MINENCRYPTBITS	string	rwxrwx---	{0 40 56 128} ⁴	0
TA_MAXENCRYPTBITS	string	rwxrwx---	{0 40 56 128} ⁴	128
TA_MAXPENDINGBYTES	long	rw-r--r--	100000 <= num <= MAXLONG	2147483647

TM_MIB(5): T_MACHINE Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_SEC_PRINCIPAL_NAME	string	rwxr--r--	<i>string</i> [0..511]	""
TA_SEC_PRINCIPAL_LOCATION	string	rwxr--r--	<i>string</i> [0..511]	""
TA_SEC_PRINCIPAL_PASSVAR	string	rwxr--r--	<i>string</i> [0..511]	""
TA_SIGNATURE_REQUIRED	string	rwxr--r--	"{Y N}"	"N"
TA_ENCRYPTION_REQUIRED	string	rwxr--r--	"{Y N}"	"N"
T_MACHINE Class: LOCAL Attributes				
TA_CURACCESSERS	long	R--R--R--	0 <= <i>num</i> < 32,768	N/A
TA_CURCLIENTS	long	R--R--R--	0 <= <i>num</i> < 32,768	N/A
TA_CURCONV	long	R--R--R--	0 <= <i>num</i> < 32,768	N/A
TA_CURGTT	long	R--R--R--	0 <= <i>num</i> < 32,768	N/A
TA_CURRLOAD	long	R--R--R--	0 <= <i>num</i>	N/A
TA_CURWSCLIENTS	long	R--R--R--	0 <= <i>num</i> < 32,768	N/A
TA_HWACCESSERS	long	R--R--R--	0 <= <i>num</i> < 32,768	N/A
TA_HWCLIENTS	long	R--R--R--	0 <= <i>num</i> < 32,768	N/A
TA_HWCONV	long	R--R--R--	0 <= <i>num</i> < 32,768	N/A
TA_HWGTT	long	R--R--R--	0 <= <i>num</i> < 32,768	N/A
TA_HWWSCLIENTS	long	R--R--R--	0 <= <i>num</i> < 32,768	N/A
TA_NUMCONV	long	R-XR-XR--	0 <= <i>num</i>	N/A
TA_NUMDEQUEUE	long	R-XR-XR--	0 <= <i>num</i>	N/A
TA_NUMENQUEUE	long	R-XR-XR--	0 <= <i>num</i>	N/A
TA_NUMPOST	long	R-XR-XR--	0 <= <i>num</i>	N/A
TA_NUMREQ	long	R-XR-XR--	0 <= <i>num</i>	N/A
TA_NUMSUBSCRIBE	long	R-XR-XR--	0 <= <i>num</i>	N/A
TA_NUMTRAN	long	R-XR-XR--	0 <= <i>num</i>	N/A

Section 5 - File Formats and Data Descriptions

TM_MIB(5): T_MACHINE Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
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

(k) - GET key field

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

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

¹TA_LMID and TA_P MID 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.

²Default is same as value set for this attribute in Class T_DOMAIN.

³Default is TA_APPDIR for this machine followed by /ULOG.

⁴The link-level encryption value of 40 bits is provided for backward compatibility.

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_P MID: *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.

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.
<i>unset</i>	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	<p>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.</p> <p>The TMIB_NOTIFY TA_FLAG value should be used when activating a machine if status on individual servers is required.</p>

ReActivate	<p>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.</p> <p>The TMIB_NOTIFY TA_FLAG value should be used when reactivating a machine if status on individual servers is required.</p>
INActive	<p>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.</p> <p>The TMIB_NOTIFY TA_FLAG value should be used when deactivating a machine if status on individual servers is required.</p>
ForceINActive	<p>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.</p> <p>The TMIB_NOTIFY TA_FLAG value should be used when deactivating a machine if status on individual servers is required.</p>

<code>CLEaning</code>	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`: $0 \leq num$

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`: $0 \leq num$

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 (that is, 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()` 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_ULONGPFX: string[0..78]`

Absolute pathname prefix of the path for the `userlog()` file on this machine. The `userlog()` file name is formed by appending the string `.mmdyy` to the `TA_ULONGPFX` attribute value. `mmdyy` represents the month, day, and year that the messages were generated. All application and system `userlog()` 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 < 32,768`

Maximum number of clients and servers that can be simultaneously connected to the bulletin board on this machine. If not specified, the default is the `TA_MAXACCESSERS` value specified in the `T_DOMAIN` class.

System administration processes, such as the `BBL`, `restartsrv`, `cleanup`, `tmshutdown()`, and `tmadmin()`, need not be accounted for in

this value, but the DBBL, all bridge processes, all system-supplied and application server processes, and all potential client processes at this site need to be counted. (Examples of system-supplied servers are AUTHSVR, TMQUEUE, TMQFORWARD, TMUSREVT, TMSYSEVT, TMS—see T_GROUP TA_TMSNAME attribute, TMS_QM, GWTDOMAIN, and WSL.) If the application is booting Workstation Listeners (WSLs) on this site, both the WSLs and the number of potential Workstation Handlers (WSHs) that may be booted need to be counted.

Note that for BEA Tuxedo pre-Release 7.1 (6.5 or earlier), both the TA_MAXACCESSERS and TA_MAXSERVERS (see T_DOMAIN TA_MAXSERVERS) attributes for an application play a part in the user license checking scheme. Specifically, a machine is not allowed to boot if the number of TA_MAXACCESSERS for that machine + the number of TA_MAXACCESSERS for the machine (or machines) already running in the application is greater than the number of TA_MAXSERVERS + user licenses for the application. Thus, the total number of TA_MAXACCESSERS for an application must be less than or equal to the number of TA_MAXSERVERS + user licenses for the application.

Note also that the user license checking scheme in BEA Tuxedo Release 7.1 or later considers only the following two factors when performing its checks: the number of user licenses for an application and the number of licenses currently in use for the application. When all user licenses are in use, no new clients are allowed to join the application.

TA_MAXCONV: $0 \leq num < 32,768$

Maximum number of simultaneous conversations in which clients and servers on this machine can be involved. If not specified, the default is the TA_MAXCONV value specified in the T_DOMAIN class. The maximum number of simultaneous conversations per server is 64.

TA_MAXGTT: $0 \leq num < 32,768$

Maximum number of simultaneous global transactions in which this machine can be involved. If not specified, the default is the value specified in the T_DOMAIN class.

TA_MAXWSCLIENTS: $0 \leq num < 32,768$

Number of accessor entries on this machine to be reserved for Workstation clients (as opposed to native clients). If TA_MAXWSCLIENTS is not specified, the default is 0.

The number specified here takes a portion of the total accessor slots specified with TA_MAXACCESSERS, meaning that the accessor slots reserved for

`TA_MAXWSCLIENTS` are unavailable for use by other clients and servers on this machine. It is an error to set this number greater than `TA_MAXACCESSERS`.

The `TA_MAXWSCLIENTS` attribute is only used when the BEA Tuxedo System Workstation feature is used. The appropriate setting of this attribute helps to conserve interprocess communication (IPC) resources since Workstation client access to the system is multiplexed through a BEA Tuxedo system-supplied surrogate, the Workstation Handler (WSH).

`TA_MAXACLCACHE`: $10 \leq num \leq 32,000$

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`: $1 \leq num < 2,049$

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_FADDR: *string*[0..78]

Specifies the complete network address to which local processes such as `tmboot`, `BRIDGE`, `BSBRIDGE` and `tmloadcf` can bind before making an outbound connection. This address must be a TCP/IP address. This attribute, along with the TA_FRANGE attribute, determines the range of TCP/IP ports to which a process will attempt to bind before making an outbound connection. If this parameter is set to the NULL or empty string, then the operating system randomly chooses a local port with which to bind.

If *string* has the form "0xhex-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, one of the following formats is used:

- "//hostname:port"
- "//#.##.##:port"

TA_FRANGE: $1 \leq num \leq 65,535$

Specifies the range of TCP/IP ports to which local processes will attempt to bind before making an outbound connection. The TA_FADDR attribute specifies the base address of the range.

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 Release 4.2.2 or earlier site is activated, the configured value is not used.

TA_TMNETLOAD: $0 \leq num < 32,768$

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 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 Release 4.2.2 or earlier site is activated, the configured value is not used.

TA_SPINCOUNT: $0 \leq num$

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 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 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.

TA_MINENCRYPTBITS: {0 | 40 | 56 | 128}

specifies the minimum level of encryption required when establishing a network link to this machine. 0 means no encryption, while 40, 56, and 128 specify the encryption key length (in bits). If this minimum level of encryption cannot be met, link establishment will fail. The default is 0.

Note: The link-level encryption value of 40 bits is provided for backward compatibility.

Limitation: Modifications to this attribute will not effect established network links.

TA_MAXENCRYPTBITS: {0 | 40 | 56 | 128}

specifies the maximum level of encryption that can be negotiated when establishing a network link. 0 means no encryption, while 40, 56, and 128 specify the encryption length (in bits). The default is 128.

Note: The link-level encryption value of 40 bits is provided for backward compatibility.

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_SEC_PRINCIPAL_NAME: string[0..511]

Security principal name identification string to be used for authentication purposes by an application running BEA Tuxedo 7.1 or later software. This attribute may contain a maximum of 511 characters (excluding the

terminating null character). The principal name specified for this attribute becomes the identity of one or more system processes running on this machine.

TA_SEC_PRINCIPAL_NAME can be specified at any of the following four levels in the configuration hierarchy: T_DOMAIN class, T_MACHINE class, T_GROUP class, and T_SERVER class. A principal name at a particular configuration level can be overridden at a lower level. If TA_SEC_PRINCIPAL_NAME is not specified at any of these levels, the principal name for the application defaults to the TA_DOMAINID string for this domain.

Note that TA_SEC_PRINCIPAL_NAME is one of a trio of attributes, the other two being TA_SEC_PRINCIPAL_LOCATION and TA_SEC_PRINCIPAL_PASSVAR. The latter two attributes pertain to opening decryption keys during application booting for the system processes running in a BEA Tuxedo 7.1 or later application. When only TA_SEC_PRINCIPAL_NAME is specified at a particular level, the system sets each of the other two attributes to a NULL (zero length) string.

TA_SEC_PRINCIPAL_LOCATION: *string*[0..511]

Location of the file or device where the decryption (private) key for the principal specified in TA_SEC_PRINCIPAL_NAME resides. This attribute may contain a maximum of 511 characters (excluding the terminating null character).

TA_SEC_PRINCIPAL_LOCATION can be specified at any of the following four levels in the configuration hierarchy: T_DOMAIN class, T_MACHINE class, T_GROUP class, and T_SERVER class. When specified at any of these levels, this attribute must be paired with the TA_SEC_PRINCIPAL_NAME attribute; otherwise, its value is ignored. (TA_SEC_PRINCIPAL_PASSVAR is optional; if not specified, the system sets it to a NULL—zero length—string.)

TA_SEC_PRINCIPAL_PASSVAR: *string*[0..511]

Variable in which the password for the principal specified in TA_SEC_PRINCIPAL_NAME is stored. This attribute may contain a maximum of 511 characters (excluding the terminating null character).

TA_SEC_PRINCIPAL_PASSVAR can be specified at any of the following four levels in the configuration hierarchy: T_DOMAIN class, T_MACHINE class, T_GROUP class, and T_SERVER class. When specified at any of these levels, this attribute must be paired with the TA_SEC_PRINCIPAL_NAME attribute;

otherwise, its value is ignored. (TA_SEC_PRINCIPAL_LOCATION is optional; if not specified, the system sets it to a NULL—zero length—string.)

During initialization, the administrator must provide the password for each of the decryption keys configured with TA_SEC_PRINCIPAL_PASSVAR. The system automatically encrypts the password entered by the administrator and assigns each encrypted password to the associated password variable.

TA_SIGNATURE_REQUIRED: { Y | N }

If set to Y, every process running on this machine requires a digital signature on its input message buffer. If not specified, the default is N. This attribute only applies to applications running BEA Tuxedo 7.1 or later software.

TA_SIGNATURE_REQUIRED can be specified at any of the following four levels in the configuration hierarchy: T_DOMAIN class, T_MACHINE class, T_GROUP class, and T_SERVICE class. Setting SIGNATURE_REQUIRED to Y at a particular level means that signatures are required for all processes running at that level or below.

TA_ENCRYPTION_REQUIRED: { Y | N }

If set to Y, every process running on this machine requires an encrypted input message buffer. If not specified, the default is N. This attribute only applies to applications running BEA Tuxedo 7.1 or later software.

TA_ENCRYPTION_REQUIRED can be specified at any of the following four levels in the configuration hierarchy: T_DOMAIN class, T_MACHINE class, T_GROUP class, and T_SERVICE class. Setting TA_ENCRYPTION_REQUIRED to Y at a particular level means that encryption is required for all processes running at that level or below.

TA_CURACCESSERS: $0 \leq \text{num} < 32,768$

Number of clients and servers currently accessing the application either directly on this machine or through a workstation handler on this machine.

TA_CURCLIENTS: $0 \leq \text{num} < 32,768$

Number of clients, both native and workstation, currently logged in to this machine.

TA_CURCONV: $0 \leq \text{num} < 32,768$

Number of active conversations with participants on this machine.

TA_CURGTT: $0 \leq \text{num} < 32,768$

Number of in use transaction table entries on this machine.

- TA_CURRLOAD:** $0 \leq num$
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:** $0 \leq num < 32,768$
Number of Workstation clients currently logged in to this machine.
- TA_HWACCESSERS:** $0 \leq num < 32,768$
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:** $0 \leq num < 32,768$
High water number of clients, both native and workstation, logged in to this machine.
- TA_HWCONV:** $0 \leq num < 32,768$
High water number of active conversations with participants on this machine.
- TA_HWGTT:** $0 \leq num < 32,768$
High water number of in use transaction table entries on this machine.
- TA_HWWSCLIENTS:** $0 \leq num < 32,768$
High water number of Workstation clients currently logged in to this machine.
- TA_NUMCONV:** $0 \leq num$
Number of `tpconnect()` operations performed from this machine.
- TA_NUMDEQUEUE:** $0 \leq num$
Number of `tpdequeue()` operations performed from this machine.
- TA_NUMENQUEUE:** $0 \leq num$
Number of `tpenqueue()` operations performed from this machine.
- TA_NUMPOST:** $0 \leq num$
Number of `tppost()` operations performed from this machine.
- TA_NUMREQ:** $0 \leq num$
Number of `tpacall()` or `tpcall()` operations performed from this machine.
- TA_NUMSUBSCRIBE:** $0 \leq num$
Number of `tpsubscribe()` operations performed from this machine.

- TA_NUMTRAN:** 0 <= *num*
Number of transactions initiated (`tpbegin()`) from this machine.
- TA_NUMTRANABT:** 0 <= *num*
Number of transactions aborted (`tpabort()`) from this machine.
- TA_NUMTRANCMT:** 0 <= *num*
Number of transactions committed (`tpcommit()`) 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()` 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 Definition

Overview The T_MSG class represents run-time attributes of the BEA Tuxedo system managed UNIX System message queues.

Attribute Table

TM_MIB(5): 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_RUNTIME	long	R--R--R--	1 <= num	N/A
TA_MSG_STIME	long	R--R--R--	1 <= num	N/A

(k) - GET key field

¹All 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.

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 *msgid_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_RUNTIME: 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 Definition

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

TM_MIB(5): T_NETGROUP Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_NETGROUP(r)(*)	string	rU-----	<i>string</i> [1..30]	DEFAULTNET
TA_NETGRPNO(r)(*)	long	rU-----	1 <= <i>num</i> < 8192	N/A
TA_STATE(k)	string	rw-r--r--	GET: "VAL" SET: "{NEW INV}"	N/A N/A
TA_NETPRIO(*)	long	rwyrw----	1 <= <i>num</i> < 8,192	100

(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_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.

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.
<i>unset</i>	Modify an existing T_NETGROUP object. Only allowed in the VALid state. Successful return leaves the object state unchanged.
INValid	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: $1 \leq num < 8,192$

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 BEA Tuxedo 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 Definition

Overview The T_NETMAP class associates TA_LMID_s from the T_MACHINE class in the TM_MIB to a TA_NETGROUP object from the T_NETGROUP class. In other words, this class contains assignments of logical machines to network groups. A TA_LMID may be included in many 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 pair 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

TM_MIB(5): 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: "VAL" SET: "{NEW INV}"	N/A N/A
TA_NADDR	string	rw-r--r--	string[1..78]	" "
TA_FADDR	string	rw-r--r--	string[0..78]	" "
TA_FRANGE	long	rw-r--r--	1 <= num <= 65,535	1
TA_MINENCRYPTBITS	string	rw-rwx---	{0 40 56 128} ¹	0
TA_MAXENCRYPTBITS	string	rw-rwx---	{0 40 56 128} ¹	128

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

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

¹ The link-level encryption value of 40 bits is provided for backward compatibility.

Attribute TA_NETGROUP: *string*
Semantics 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.

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.
-----	--

<i>unset</i>	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 means by which one BRIDGE process is contacted by

others that are 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 “0x*hex-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 one of the following formats is used:

- “//*hostname:port*”
- “//#. #. #. #:*port*”

TA_FADDR: *string*[0..78]

Specifies the complete network address to which local processes such as tmbboot, BRIDGE, BSBRIDGE and tmlloadcf can bind before making an outbound connection. This address must be a TCP/IP address. This attribute, along with the TA_FRANGE attribute, determines the range of TCP/IP ports to which a process will attempt to bind before making an outbound connection. If this parameter is set to the NULL or empty string, then the operating system randomly chooses a local port with which to bind.

If *string* has the form “0x*hex-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, one of the following formats is used:

- “//*hostname:port*”
- “//#. #. #. #:*port*”

TA_FRANGE: 1<= *num* <= 65,535

Specifies the range of TCP/IP ports to which local processes will attempt to bind before making an outbound connection. The TA_FADDR attribute specifies the base address of the range.

TA_MINENCRYPTBITS: { 0 | 40 | 56 | 128 }

Specifies the minimum level of encryption required when a network link is being established. 0 means no encryption, while 40, 56, and 128 specify the encryption key length (in bits). If this minimum level of encryption cannot be met, link establishment fails. The default is 0.

Note: The link-level encryption value of 40 bits is provided for backward compatibility.

Limitation: Modifications to this attribute will not effect established network links.

TA_MAXENCRYPTBITS: { 0 | 40 | 56 | 128 }

Specifies the maximum level of encryption allowed when a link is being established. 0 means no encryption, while 40, 56, and 128 specify the encryption length (in bits). The default is 128.

Note: The link-level encryption value of 40 bits is provided for backward compatibility.

Limitation: Modifications to this attribute will not effect established network links.

When 128-bit encryption is licensed, TA_MAXENCRYPTBITS defaults to 128. When 56-bit encryption is licensed, the default is 56. When no encryption is licensed, the default is 0 bits. Note that when BRIDGE processes connect, they negotiate to the highest common TA_MAXENCRYPTBITS.

Limitations None.

T_QUEUE Class Definition

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

TM_MIB(5): T_QUEUE Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_RQADDR(*)	string	R--R--R--	<i>string</i> [1..30]	N/A
TA_SERVERNAME(k)	string	R--R--R--	<i>string</i> [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 <= <i>num</i>	N/A
TA_MAXGEN	long	R--R--R--	1 <= <i>num</i> < 256	N/A
TA_RCMD	string	R--R--R--	<i>string</i> [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--	<i>LMID</i>	N/A
TA_RQID	long	R--R--R--	1 <= <i>num</i>	N/A
TA_SERVERCNT	long	R--R--R--	1 <= <i>num</i> < 8,192	N/A
T_QUEUE Class:LOCAL Attributes				
TA_TOTNQUEUED	long	R-XR-XR--	0 <= <i>num</i>	N/A
TA_TOTWKQUEUED	long	R-XR-XR--	0 <= <i>num</i>	N/A
TA_SOURCE(k)	string	R--R--R--	<i>LMID</i>	N/A

TM_MIB(5): T_QUEUE Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
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.

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: $1 \leq num$

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: $1 \leq num < 8,192$

Number of active servers associated with this queue.

TA_TOTNQUEUED: $0 \leq num$

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: $0 \leq num$

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: $0 \leq num$

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 Definition

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

TM_MIB(5): T_ROUTING Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
<code>TA_ROUTINGNAME(r)(*)</code>	string	ru-r--r--	<i>string</i> [1..15]	N/A
<code>TA_BUFTYPE(r)(*)</code>	string	ru-r--r--	<i>string</i> [1..256]	N/A ¹
<code>TA_FIELD(r)(k)(*)</code>	string	ru-r--r--	<i>string</i> [1..30]	N/A ¹
<code>TA_FIELDTYPE</code>	string	ru-r--r--	[char short long float double string]	string
<code>TA_RANGES(r)</code>	carray	rw-r--r--	<i>carray</i> [1..2048]	N/A
<code>TA_STATE(k)</code>	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` only applies to BEA Tuxedo data-dependent routing criteria. The specified `u` (uniqueness) permission applies only in the relevant case. That is: the combination of `TA_ROUTINGNAME`, `TA_TYPE` and `TA_BUFTYPE` must be unique for `TA_TYPE=SERVICE`.

The `TA_TYPE` attribute determines the permissible attributes for the `TA_ROUTING` object. `TYPE=SERVICE` corresponds to BEA Tuxedo data-dependent routing criteria. The default is `SERVICE`. SET operations are assumed to be for data-dependent routing if no `TA_TYPE` is specified.

Attribute `TA_ROUTINGNAME: string[1..15]`
 Semantics 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 the following: FML, FML32, XML, VIEW, VIEW32, X_C_TYPE, and X_COMMON. No subtype can be specified for types FML, FML32, or XML; subtypes are required for types VIEW, VIEW32, 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 cannot 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]`

When `TA_TYPE=SERVICE`, the `TA_FIELD` field is assumed to be an FML or FML32 buffer, XML buffer, view field name that is identified in an FML field table (using the environment variables `FLDTBLDIR` and `FIELDTBLS`, or `FLDTBLDIR32` and `FIELDTBLS32`), or an FML view table (using the environment variables `VIEWDIR` and `VIEWFILES`, or `VIEWDIR32` and `VIEWFILES32`), respectively. This information is used to get the associated field value for data-dependent routing while sending a message.

For an XML buffer type, `TA_FIELD` contains either: a routing element type (or name) or a routing element attribute name.

The syntax of the `TA_FIELD` parameter for an XML buffer type is as follows.

`"root_element[/child_element][/child_element][/. . .][/@attribute_name]"`

The element is assumed to be an XML document or datagram element type. Indexing is not supported. Therefore, the BEA Tuxedo system recognizes only the first occurrence of a given element type when processing an XML buffer for data-dependent routing. This information is used to get the associated element content for data-dependent routing while sending a message. The content must be a string encoded in UTF-8.

The attribute is assumed to be an XML document or datagram attribute of the defined element. This information is used to get the associated attribute value for data-dependent routing while sending a message. The value must be a string encoded in UTF-8.

The combination of element name and attribute name may contain up to 30 characters.

The type of the routing field can be specified by the `TA_FIELDTYPE` attribute.

`TA_FIELDTYPE`: [char, short, long, float, double, string]

The type of the routing field specified in the `TA_FIELD` attribute. The type can be char, short, long, float, double, or string; only one type is allowed. This attribute is used only for routing XML buffers. The default type of the routing field is string.

`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 cannot 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.

Section 5 - File Formats and Data Descriptions

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.

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.
<i>unset</i>	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.

Limitations None.

T_SERVER Class Definition

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. Information returned will always include fields that are common among all contexts of a server. In addition, for those servers that are not defined to the system as multicontexted (that is, those for which the value of `TA_MAXDISPATCHTHREADS` is 1), this class includes information about the server's context. For those servers that are defined to the system as multicontexted, placeholder values are reported for per-context attributes. Per-context attributes can always be found as part of the `T_SERVERCTXT` class. The `T_SERVERCTXT` class is defined even for single-contexted servers.

The `TA_CLTMLID`, `TA_CLTPID`, `TA_CLTREPLY`, `TA_CMTRET`, `TA_CURCONV`, `TA_CURREQ`, `TA_CURRSERVICE`, `TA_LASTGRP`, `TA_SVCTIMEOUT`, `TA_TIMELEFT`, and `TA_TRANLEV` attributes are specific to each server dispatch context. All other attributes are common to all server dispatch contexts.

Attribute Table

TM_MIB(5): T_SERVER Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
<code>TA_SRVGRP(r)(*)</code>	string	<code>ru-r--r--</code>	<code>string[1..30]</code>	N/A
<code>TA_SRVID(r)(*)</code>	long	<code>ru-r--r--</code>	<code>1 <= num < 30,001</code>	N/A
<code>TA_SERVERNAME(k)(r)</code>	string	<code>rw-r--r--</code>	<code>string[1..78]</code>	N/A
<code>TA_GRPNO(k)</code>	long	<code>r--r--r--</code>	<code>1 <= num < 30,000</code>	N/A
<code>TA_STATE(k)</code>	string	<code>rwxr--xr--</code>	GET: "{ACT INA MIG CLE RES SUS PAR DEA}"	N/A
			SET: "{NEW INV ACT INA DEA}"	N/A
<code>TA_BASESRVID</code>	long	<code>r--r--r--</code>	<code>1 <= num < 30,001</code>	N/A
<code>TA_CLOPT</code>	string	<code>rwyr--r--</code>	<code>string[0..256]</code>	"-A"
<code>TA_ENVFILE</code>	string	<code>rwyr--r--</code>	<code>string[0..78]</code>	" "

Section 5 - File Formats and Data Descriptions

TM_MIB(5): T_SERVER Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
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_MINDISPATCHTHREADS	long	rwyr--r--	1 <= num < 1,000	1
TA_MAXDISPATCHTHREADS	long	rwyr--r--	0 <= num < 1,000	0
TA_THREADSTACKSIZE	long	rwyr--r--	0 <= num < 2,147,483,647	0
TA_CURDISPATCHTHREADS	long	R-XR-XR--	0 <= num	N/A
TA_HWDISPATCHTHREADS	long	R-XR-XR--	0 <= num	N/A
TA_NUMDISPATCHTHREADS	long	R-XR-XR--	0 <= num	N/A
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}"	(l)
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	(l)
TA_RQADDR(k)	string	rw-r--r--	string[0..30]	"GRPNO. SRVID"
TA_RQPERM	long	rw-r--r--	0001 <= num <= 0777	(l)
TA_LMID(k)	string	R--R--R--	LMID	N/A
TA_GENERATION	long	R--R--R--	1 <= num < 32,768	N/A
TA_PID(k)	long	R--R--R--	1 <= num	N/A

TM_MIB(5): T_SERVER Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
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
TA_SEC_PRINCIPAL_NAME	string	rwxr--r--	string[0..511]	" "
TA_SEC_PRINCIPAL_LOCATION	string	rwxr--r--	string[0..511]	" "
TA_SEC_PRINCIPAL_PASSVAR	string	rwxr--r--	string[0..511]	" "
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

Section 5 - File Formats and Data Descriptions

TM_MIB(5): T_SERVER Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_CURCONV	long	R--R--R--	0 <= num	N/A
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

¹Defaults to value set for this attribute in Class T_DOMAIN

Attribute TA_SRVGRP: *string*[1..30]
Semantics 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 | RESTARTING |
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.

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.
RESTARTING	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.
<i>unset</i>	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 REStating 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: $0 \leq \text{num}$

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: $1 \leq \text{num} < 256$

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: $1 \leq num < 1,001$

Maximum number of occurrences of the server to be booted. Initially, `tmboot()` 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: $1 \leq num < 1,001$

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_MINDISPATCHTHREADS: $1 \leq num < 1,000$

specifies the number of server dispatch threads started on initial server boot. This parameter is effective only if the server has been built with the `buildserver -t` command.

The separate dispatcher thread that is used when `TA_MAXDISPATCHTHREADS > 1` is not counted as part of the `TA_MINDISPATCHTHREADS` value. It is required that `TA_MINDISPATCHTHREADS <= TA_MAXDISPATCHTHREADS`. If `TA_MINDISPATCHTHREADS` is not specified, the default is 0.

Limitation: Run-time modifications to this attribute will not affect a running server.

TA_MAXDISPATCHTHREADS: $0 \leq num < 1,000$

specifies the maximum number of concurrently dispatched threads which each server process may spawn. This parameter is effective only if the server has been built with the `buildserver -t` command.

If `TA_MAXDISPATCHTHREADS > 1`, then a separate dispatcher thread is used and does not count against this limit. It is required that

TA_MINDISPATCHTHREADS <= TA_MAXDISPATCHTHREADS. If TA_MAXDISPATCHTHREADS is not specified, the default is 1.

Limitation: Run-time modifications to this attribute will not affect a running server.

TA_THREADSTACKSIZE = *number*

Size of the stack created for each dispatch thread in a multithreaded server. This option has an effect on the server only when a value greater than 1 is specified for TA_MAXDISPATCHTHREADS.

If this attribute is not specified or is set to 0, a default thread stack size is used. The default size used is the operating system default size *unless* that value is known to be insufficient for a multithreaded BEA Tuxedo application, in which case the BEA Tuxedo default size is used. Currently, the BEA Tuxedo default thread stack size is 1,024,000.

Note that if the thread stack size is exceeded, the server will core dump.

Limitation: Run-time modifications to this attribute will not affect a running server.

TA_CURDISPATCHTHREADS: 0 <= *num*

Current number of active service dispatch threads for this server.

TA_HWDISPATCHTHREADS: 0 <= *num*

Highest number of active service dispatch threads created for this server since its last restart. This number may differ from the number of service calls, since an administrator may specify parameters that control the caching of idle service threads.

TA_NUMDISPATCHTHREADS: 0 <= *num*

Total number of active service dispatch threads for this server since its last restart.

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: $1 \leq num < 10,000$

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()` in increasing order of sequence number and shutdown by `tmshutdown()` in decreasing order. Run-time modifications to this attribute affect only `tmboot()` and `tmshutdown()` and will affect the order in which running servers may be shutdown by a subsequent invocation of `tmshutdown()`.

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.

Limitations: (1) Run-time modifications to this attribute will not affect a running server. (2) Setting TA_SYSTEM_ACCESS to PROTECTED may not be effective for multithreaded servers because it is possible that while one thread

is executing BEA Tuxedo code, which means it is attached to the Bulletin Board, another thread might be executing user code. The BEA Tuxedo system cannot prevent such situations.

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 \leq num \leq 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 \leq num \leq 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 \leq num $<$ 32,768

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 \leq 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_SEC_PRINCIPAL_NAME: *string*[0..511]

Security principal name identification string to be used for authentication purposes by an application running BEA Tuxedo 7.1 or later software. This attribute may contain a maximum of 511 characters (excluding the terminating null character). The principal name specified for this attribute becomes the identity of the system processes running on this server.

`TA_SEC_PRINCIPAL_NAME` can be specified at any of the following four levels in the configuration hierarchy: `T_DOMAIN` class, `T_MACHINE` class, `T_GROUP` class, and `T_SERVER` class. A principal name at a particular configuration level can be overridden at a lower level. If

`TA_SEC_PRINCIPAL_NAME` is not specified at any of these levels, the

principal name for the application defaults to the `TA_DOMAINID` string for this domain.

Note that `TA_SEC_PRINCIPAL_NAME` is one of a trio of attributes, the other two being `TA_SEC_PRINCIPAL_LOCATION` and `TA_SEC_PRINCIPAL_PASSVAR`. The latter two attributes pertain to opening decryption keys during application booting for the system processes running in a BEA Tuxedo 7.1 or later application. When only `TA_SEC_PRINCIPAL_NAME` is specified at a particular level, the system sets each of the other two attributes to a `NULL` (zero length) string.

`TA_SEC_PRINCIPAL_LOCATION`: *string*[0..511]

Location of the file or device where the decryption (private) key for the principal specified in `TA_SEC_PRINCIPAL_NAME` resides. This attribute may contain a maximum of 511 characters (excluding the terminating null character).

`TA_SEC_PRINCIPAL_LOCATION` can be specified at any of the following four levels in the configuration hierarchy: `T_DOMAIN` class, `T_MACHINE` class, `T_GROUP` class, and `T_SERVER` class. When specified at any of these levels, this attribute must be paired with the `TA_SEC_PRINCIPAL_NAME` attribute; otherwise, its value is ignored. (`TA_SEC_PRINCIPAL_PASSVAR` is optional; if not specified, the system sets it to a `NULL`—zero length—string.)

`TA_SEC_PRINCIPAL_PASSVAR`: *string*[0..511]

Variable in which the password for the principal specified in `TA_SEC_PRINCIPAL_NAME` is stored. This attribute may contain a maximum of 511 characters (excluding the terminating null character).

`TA_SEC_PRINCIPAL_PASSVAR` can be specified at any of the following four levels in the configuration hierarchy: `T_DOMAIN` class, `T_MACHINE` class, `T_GROUP` class, and `T_SERVER` class. When specified at any of these levels, this attribute must be paired with the `TA_SEC_PRINCIPAL_NAME` attribute; otherwise, its value is ignored. (`TA_SEC_PRINCIPAL_LOCATION` is optional; if not specified, the system sets it to a `NULL`—zero length—string.)

During initialization, the administrator must provide the password for each of the decryption keys configured with `TA_SEC_PRINCIPAL_PASSVAR`. The system automatically encrypts the password entered by the administrator and assigns each encrypted password to the associated password variable.

`TA_NUMCONV`: $0 \leq num$

Number of conversations initiated by this server via `tpconnect()`.

TA_NUMDEQUEUE: 0 <= num

Number of dequeue operations initiated by this server via `tpdequeue()`.

TA_NUMENQUEUE: 0 <= num

Number of enqueue operations initiated by this server via `tpenqueue()`.

TA_NUMPOST: 0 <= num

Number of postings initiated by this server via `tppost()`.

TA_NUMREQ: 0 <= num

Number of requests made by this server via `tpcall()` or `tpacall()`.

TA_NUMSUBSCRIBE: 0 <= num

Number of subscriptions made by this server via `tpsubscribe()`.

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_CLTMLID: *LMID*

Logical machine for the initiating client or server.

This field element is also contained in the `T_SERVERCTXT` class, both for single-context servers and for multicontext servers.

The initiating client or server is the process that made the service request on which the server is currently working. The value in this field has meaning

only for single-context servers. In multicontext servers, a null string is returned as a placeholder.

TA_CLTPID: 1 <= *num*

UNIX System process identifier for the initiating client or server.

This field element is also contained in the T_SERVERCTXT class, both for single-context servers and for multicontext servers.

The value in this field has meaning only for single-context servers; in multicontexted servers 0 is returned as a placeholder.

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").

This field element is also contained in the T_SERVERCTXT class, both for single-context servers and for multi-context servers.

The value in this field has meaning only for single-context servers; in multicontexted servers a null string is returned as a placeholder.

TA_CMTRET: {COMPLETE | LOGGED}

Setting of the TP_COMMIT_CONTROL characteristic for this server.

This field element is also contained in the T_SERVERCTXT class, both for single-context servers and for multi-context servers.

See the description of the ATMI function call `tpscmt()` for details on this characteristic. The value in this field has meaning only for single-context servers; in multicontext servers a null string is returned as a placeholder.

TA_CURCONV: 0 <= *num*

Number of conversations initiated by this server via `tpconnect()` that are still active. For multicontext servers, this field represents the total for all server contexts. Values for individual server contexts can be found in the T_SERVERCTXT class.

TA_CURREQ: 0 <= *num*

Number of requests initiated by this server via `tpcall()` or `tpacall()` that are still active. For multicontext servers, this field represents the total for all

server contexts. Values for individual server contexts can be found in the T_SERVERCTXT class.

TA_CURRSERVICE: *string*[0..15]

Service name that the server is currently working on, if any.

This field element is also contained in the T_SERVERCTXT class, both for single-context servers and for multicontext servers.

The value in this field has meaning only for single-context servers; in multicontext servers 0 is returned as a placeholder.

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.

This field element is also contained in the T_SERVERCTXT class, both for single-context servers and for multicontext servers.

The value in this field has meaning only for single-context servers; in multicontexted servers 0 is returned as a placeholder.

TA_SVCTIMEOUT: 0 <= *num*

Time left, in seconds, for this server to process the current service request, if any.

This field element is also contained in the T_SERVERCTXT class, both for single-context servers and for multicontext servers.

A value of 0 for an active service indicates that no timeout processing is being done. See T_SERVICE:TA_SVCTIMEOUT for more information. The value in this field has meaning only for single-context servers; in a multicontext server 0 is returned as a placeholder.

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 field element is also contained in the T_SERVERCTXT class, both for single-context servers and for multicontext servers.

This timeout may be a transactional timeout or a blocking timeout.

The value in this field has meaning only for single-context servers; in a multicontext server 0 is returned as a placeholder.

TA_TRANLEV: 0 <= *num*

Current transaction level for this server.

This field element is also contained in the T_SERVERCTXT class, both for single-context servers and for multicontext servers.

0 indicates that the server is not currently involved in a transaction. The value in this field has meaning only for single-context servers; in multicontext servers 0 is returned as a placeholder.

Limitations None.

T_SERVERCTXT Class Definition

Overview The `T_SERVERCTXT` class represents configuration and runtime attributes of individual server dispatch contexts within an application. This class is defined for both single-context and multi-context servers. For single-context servers, the values in this class are repeated as part of the `T_SERVER` class. The attributes in the `T_SERVERCTXT` class are read-only.

These attribute values provide runtime tracking of statistics and resources associated with each server dispatch context.

Attribute Table

TM_MIB(5): T_SERVERCTXT Class Definition Attribute Table

Attribute ¹	Type	Permissions	Values	Default
TA_SRVGRP(k)	string	r--r--r--	<i>string</i> [1..30]	N/A
TA_SRVID(k)	long	r--r--r--	1 <= <i>num</i> < 30,001	N/A
TA_CONTEXTID(k)	long	r--r--r--	-2 <= <i>num</i> < 30,000	N/A
TA_CLTLMID	string	r--r--r--	<i>LMID</i>	N/A
TA_CLTPID	long	r--r--r--	1 <= <i>num</i>	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 <= <i>num</i>	N/A
TA_CURREQ	long	r--r--r--	0 <= <i>num</i>	N/A
TA_CURRSERVICE	string	r--r--r--	<i>string</i> [0..15]	N/A
TA_LASTGRP	long	r--r--r--	1 <= <i>num</i> <30,000	N/A
TA_SVCTIMEOUT	long	r--r--r--	0 <= <i>num</i>	N/A
TA_TIMELEFT	long	r--r--r--	0 <= <i>num</i>	N/A
TA_TRANLEV	long	r--r--r--	0 <= <i>num</i>	N/A

(k) - GET key field

¹All attributes in Class `T_SERVERCTXT` are local attributes.

Section 5 - File Formats and Data Descriptions

Attribute	TA_SRVGRP: <i>string</i> [1..30]
Semantics	Logical name of the server group. Server group names cannot contain an asterisk (*), comma, or colon.
	TA_SRVID: 1 <= <i>num</i> < 30,001
	Unique (within the server group) server identification number.
	TA_CONTEXTID: 0 <= <i>num</i> < 30000
	Identifier of this particular server context.
	TA_CLTMLID: <i>LMID</i>
	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 <= <i>num</i>
	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 ATMI function <code>tpscmt(3c)</code> for details on this characteristic.
	TA_CURCONV: 0 <= <i>num</i>
	Number of conversations initiated by this server via <code>tpconnect()</code> that are still active.
	TA_CURREQ: 0 <= <i>num</i>
	Number of requests initiated by this server via <code>tpcall()</code> or <code>tpacall()</code> that are still active.
	TA_CURRSERVICE: <i>string</i> [0..15]
	Service name that the server is currently working on, if any.
	TA_LASTGRP: 1 <= <i>num</i> < 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 Definition

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()`, 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

TM_MIB(5): T_SERVICE Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_SERVICENAME(r)(*)	string	ru-r--r--	<i>string</i> [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 < 32,768	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--	<i>string</i> [1..256]	"ALL"
TA_ROUTINGNAME	string	rwxr--r--	<i>string</i> [0..15]	" "
TA_SIGNATURE_REQUIRED	string	rwxr--r--	"{Y N}"	"N"
TA_ENCRYPTION_REQUIRED	string	rwxr--r--	"{Y N}"	"N"

(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 TA_SERVICENAME: *string*[1..15]
Semantics 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.

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.
<i>unset</i>	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 < 32,768

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 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 the BEA Tuxedo system are FML and FML32 (for FML buffers), XML (for XML buffers), VIEW, VIEW32, 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, VIEW32, 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 `tuatypes(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 `tuextypes(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.

`TA_SIGNATURE_REQUIRED`: {Y | N}

If set to `Y`, every instance of this service requires a digital signature on its input message buffer. If not specified, the default is `N`. This attribute only applies to applications running BEA Tuxedo 7.1 or later software.

`TA_SIGNATURE_REQUIRED` can be specified at any of the following four levels in the configuration hierarchy: `T_DOMAIN` class, `T_MACHINE` class, `T_GROUP` class, and `T_SERVICE` class. Setting `SIGNATURE_REQUIRED` to `Y` at a particular level means that signatures are required for all processes running at that level or below.

`TA_ENCRYPTION_REQUIRED`: {Y | N}

If set to `Y`, every instance of this service requires an encrypted input message buffer. If not specified, the default is `N`. This attribute only applies to applications running BEA Tuxedo 7.1 or later software.

`TA_ENCRYPTION_REQUIRED` can be specified at any of the following four levels in the configuration hierarchy: `T_DOMAIN` class, `T_MACHINE` class, `T_GROUP` class, and `T_SERVICE` class. Setting `TA_ENCRYPTION_REQUIRED` to `Y` at a particular level means that encryption is required for all processes running at that level or below.

Limitations None.

T_SVCGRP Class Definition

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()`, 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.

Section 5 - File Formats and Data Descriptions

Attribute Table

TM_MIB(5): T_SVCGRP Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_SERVICENAME(r)(*)	string	ru-r--r--	<i>string</i> [1..15]	N/A
TA_SRVGRP(r)(*)	string	ru-r--r--	<i>string</i> [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 < 32,768	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--	<i>LMID</i>	N/A
TA_RQADDR(*)	string	R--R--R--	<i>string</i> [1..30]	N/A
TA_SRVID(*)	long	R--R--R--	1 <= num < 30,001	N/A
TA_SVCRNAM	string	R-XR-XR--	<i>string</i> [1..15]	(?)
TA_BUFTYPE	string	r--r--r--	<i>string</i> [1..256]	N/A
TA_ROUTINGNAME	string	r--r--r--	<i>string</i> [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 < 32,768	N/A

TM_MIB(5): T_SVCGRP Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
(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				

¹SET 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.

²If 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.

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	<p>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.</p> <p>Limitation: State change not permitted for service names (TA_SERVICENAME) beginning with the reserved string “.”.</p>
INActive	<p>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.</p> <p>Limitation: State change not permitted for service names (TA_SERVICENAME) beginning with the reserved string “_”.</p>
SUSPended	<p>Suspend the T_SVCGRP object. State change allowed only when in the ACTIVE state. Successful return leaves the object in the SUSPended state.</p> <p>Limitation: State change not permitted for service names (TA_SERVICENAME) beginning with the reserved string “_”.</p>

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 < 32,768

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: $0 \leq num$

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 Release 4.2.2 sites or earlier.

TA_TRANTIME: $0 \leq num$

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: $1 \leq num < 30,001$

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()` 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* < 32,768

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 Definition

Overview The T_TLISTEN class represents run-time attributes of the BEA Tuxedo system listener processes for a distributed application.

Attribute Table

TM_MIB(5): 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}" SET: N/A	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.

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()` interface.

T_TLOG Class Definition

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, and so on.

Attribute Table

TM_MIB(5): 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	(²)
TA_TLOGDATA	string	r-xr-xr--	string[1..256]	(²)

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

¹All attributes in Class T_TLOG are local attributes

²One 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.

<i>ACTive</i>	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.
<i>INACTive</i>	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 <code>tlisten(1)</code> process running; otherwise, the site is unreachable and a object will not be returned.
<i>WARmstart</i>	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 <i>ACTive</i> 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.

<i>unset</i>	Modify T_TLOG object. Allowed only when in the <i>ACTive</i> 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.
<i>WARmstart</i>	Initiate warmstart for the T_TLOG object. State change allowed only when in the <i>ACTive</i> state. Successful return leaves the object in the <i>WARmstart</i> 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: $0 \leq num$

Index of the first object specific attribute values (TA_GRPNO and TA_TLOGDATA) corresponding to this object.

TA_GRPNO: $1 \leq num < 30,000$

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 Definition

Overview The T_TRANSACTION class represents run-time attributes of active transactions within the application.

Attribute Table

TM_MIB(5): 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)

(k) - GET key field

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

¹All attributes in Class T_TRANSACTION are local attributes.

²One or more TA_GRPNO 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()` 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()` 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. 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.

<code>PREPrepare</code>	Indicates that the transaction group contains servers that have called <code>xa_end (TMSUSPEND)</code> during the course of transactional work and that commit processing is beginning. This state will exist until either all servers that called <code>xa_end (TMSUSPEND)</code> have caused a call to <code>xa_end (TMSUCCESS)</code> , at which point the group state will become <code>READY</code> , or until one of the target servers does a rollback of the transaction at which point the group state will become either <code>PostABOrT</code> or <code>ABOrteD</code> .
<code>PostABOrT</code>	Indicates that a server called <code>xa_end (TPFAIL)</code> and that the TMS has not yet called <code>xa_rollback()</code> . (that is, that other servers that had called <code>xa_end (TMSUSPEND)</code> are being notified by the TMS in order to clean up their associated CORBA objects.
<code>PostCOMmit</code>	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.

<code>unset</code>	Modify an existing <code>T_TRANSACTION</code> object. This combination is allowed only when in the <code>READY</code> state and only for the purpose of updating an individual group's state. Successful return leaves the object state unchanged.
<code>ABOrteD</code>	Abort the <code>T_TRANSACTION</code> object for the application. State change allowed only when in the <code>ACTive</code> , <code>ABOrtonly</code> , or <code>COMcalled</code> states. Successful return leaves the object in the <code>ABOrteD</code> 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_GRP_COUNT: 1 <= num

Number of groups identified as participants in the transaction by the information returned from the retrieval site.

TA_GRP_INDEX: 1 <= num

Index of the first group specific attribute values (TA_GRPNO) corresponding to this object.

TA_GRPNO: 1 <= num < 30,000

Group number of the participating group.

Limitations None.

T_ULONG Class Definition

Overview The T_ULONG class represents run-time attributes of `userlog()` files within an application.

Attribute Table

TM_MIB(5): T_ULONG Class Definition Attribute Table

Attribute ¹	Type	Permissions	Values	Default
TA_LMID(k)	string	R--R--R--	<i>LMID</i>	(²)
TA_PMID(x)	string	R--R--R--	<i>string</i> [1..30]	(²)
TA_MMDDYY(k)	long	R--R--R--	<i>mmdyy</i>	Current date
TA_STATE	string	R--R--R--	GET: "ACT" SET: N/A	N/A N/A
TA_ULONGTIME(k)	long	R--R--R--	<i>hhmmss</i>	000000
TA_ENDTIME(k)	long	K--K--K--	<i>hhmmss</i>	235959
TA_ULONGLINE(k)	long	R--R--R--	1 <= <i>num</i>	1
TA_ULONGMSG(x)	string	R--R--R--	<i>string</i> [1..256]	N/A
TA_TPTRANID(k)	string	R--R--R--	<i>string</i> [1..78]	N/A
TA_XID(k)	string	R--R--R--	<i>string</i> [1..78]	N/A
TA_PID(k)	long	R--R--R--	1 <= <i>num</i>	N/A
TA_THREADID	integer	r--r--r--	0 <= <i>num</i>	NA
TA_CONTEXTID(k)	long	r--r--r--	-2 <= <i>num</i> < 30,000	N/A
TA_SEVERITY(x)	string	R--R--R--	<i>string</i> [1..30]	N/A
TA_ULONGCAT(x)	string	R--R--R--	<i>string</i> [1..30]	N/A
TA_ULONGMSGNUM(k)	long	R--R--R--	1 <= <i>num</i>	N/A
TA_ULONGPROCNM(x)	string	R--R--R--	<i>string</i> [1..30]	N/A

TM_MIB(5): T_ULOG Class Definition Attribute Table

Attribute ¹	Type	Permissions	Values	Default
(k) - GET key field				
(x) - Regular expression GET key field				

¹All attributes in Class T_ULOG are local attributes.

²TA_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_P MID 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_P MID as a regular expression key field.

**Attribute
Semantics**

TA_LMID: *LMID*

Retrieval machine logical machine identifier.

TA_P MID: *string*[1..30]

Physical machine identifier.

TA_MMDDYY: *mmdyy*

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.

ACTIVE	The object returned reflects an existing userlog file on the indicated logical machine.
--------	---

SET:

SET operations are not permitted on this class.

TA_ULONGTIME: *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 multiplied 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_ULONGLINE: $1 \leq num$

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_ULONGMSG: *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()`. 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()`. 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: $1 \leq num$

Process identifier of the client or server that generated the userlog message.

TA_THREADID: $0 \leq num$

Identifier for the thread that wrote this userlog message.

TA_CONTEXTID: $-2 \leq num < 30,000$

Identifier for this particular application association.

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(), tpgetrply(), and tpdequeue()) 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() and tpcall() 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 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 BEA Tuxedo 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()` and `tpcall()`. 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>
```

Libraries

```
`${TUXDIR}/lib/libtmib.a, `${TUXDIR}/lib/libqgm.a,
`${TUXDIR}/lib/libtmib.so.<rel>, `${TUXDIR}/lib/libqgm.so.<rel>,
`${TUXDIR}/lib/libtmib.lib
```

The libraries must be linked manually when using `buildclient`. The user must use:

```
-L$ (TUXDIR) /lib -ltmib -lqgm
```

Initial Configuration

The following code creates and populates an FML32 buffer that is then passed to `tpadmcalls()` for processing. This example also shows interpretation of `tpadmcalls()` 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_P MID, 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 tpadmcalls() */
```

```
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_P MID, 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);

tpadmcalls(...); /* 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);

tpadmcalls(...); /* 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);

tpadmcalls(...); /* 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()` 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);

tpadmcalls(...); /* 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() 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().

```
/* 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);
```

Section 5 - File Formats and Data Descriptions

```
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 tpacall(3c), tpalloc(3c), tpcall(3c), tpdequeue(3c), tpenqueue(3c),
tpgetrply(3c), tprealloc(3c), **Introduction to FML Functions**, Fadd,
Fadd32(3fml), Fchg, Fchg32(3fml), Ffind, Ffind32(3fml), MIB(5),
WS_MIB(5)

Setting Up a BEA Tuxedo Application

Administering a BEA Tuxedo Application at Run Time

Programming a BEA Tuxedo Application Using C

Programming a BEA Tuxedo Application Using FML

TMQFORWARD(5)

Name	TMQFORWARD - Message Forwarding Server
Synopsis	<pre>TMQFORWARD SRVGRP="<i>identifier</i>" SRVID="<i>number</i>" REPLYQ=N CLOPT=" [-A] [<i>servopts options</i>] -- -q <i>queuename[,queuename...]</i> [-t <i>trantime</i>] [-i <i>idletime</i>] [-e] [-d] [-n] [-f <i>delay</i>] "</pre>
Description	<p>The message forwarding server is a BEA Tuxedo system-supplied server that forwards messages that have been stored using <code>tpenqueue()</code> for later processing. The application administrator enables automated message processing for the application servers by specifying this server as an application server in the <code>SERVERS</code> section.</p>

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()` 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. That is, the original request message is deleted from the queue—not put back on the queue—if the service fails *and* a reply message (non-zero in length) is received from the server.

The reply message is enqueued to the failure queue, if one is associated with the message and the queue exists. If the message is to be deleted at the same time as the retry limit configured for the queue is reached, the original request message is put into the error queue.

-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 cause 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 is enqueued, if 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.

An application may be able to specify the quality of service for a reply to a message when the original message is enqueued. If a reply quality of service is not specified, then the default delivery policy specified for the reply queue is used. Note that the default delivery policy is determined when the reply to a message is enqueued. That is,

if the default delivery policy of the reply queue is modified between the time that the original message is enqueued and the reply to the message is enqueued, the policy used is the one in effect when the reply is finally enqueued.

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 dequeuing that applied when it was first put on the queue are (in effect) suspended for *delay* seconds; this opens up the possibility, for example, that a message of a lower priority may be dequeued ahead of the restored message on a queue ordered by priority.

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. If the message is to be deleted at the same time as the retry limit for the queue is reached, the original request message is put into the error queue.

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. See the description in *Using the BEA Tuxedo /Q Component*.

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 (space or tab) 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 all supported server platforms.

Interoperability `TMQFORWARD` may be run in an interoperating application, but it must run on a BEA Tuxedo Release 4.2 or later node.

Examples

```
*GROUPS # For NT, :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 buildserver(1), tpdequeue(3c), tpenqueue(3c), servopts(5), TMQUEUE(5),
UBBCONFIG(5)

Setting Up a BEA Tuxedo Application

Programming a BEA Tuxedo Application Using C

TMQUEUE(5)

Name TMQUEUE - Message Queue Manager

Synopsis TMQUEUE
 SRVGRP="*identifier*"
 SRVID="*number*" CLOPT=" [-A][servopts options] -- [-t *timeout*]"

Description The message queue manager is a BEA Tuxedo system-supplied server that enqueues and dequeues messages on behalf of programs calling `tpenqueue()` and `tpdequeue()`, respectively. The application administrator enables message enqueueing 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 timeout`

is used to indicate the timeout to be used for queuing operations when not in transaction mode (for example, `tpenqueue()` or `tpdequeue()` 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 operation 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 enqueueing 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)`. See the description in *Using the BEA Tuxedo /Q Component*.

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 qspace:TMQUEUE -r TUXEDO/QM \  
-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 qspace, qspace :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 (space or tab) 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 all supported server platforms.

Interoperability `TMQUEUE` may be run in an interoperating application, but it must run on a BEA Tuxedo Release 4.2 or later node.

Examples

```
*GROUPS
# For NT, :myqueue becomes ;myqueue
TMQUEUEGRP1 GRPNO=1 TMSNAME=TMS_QM
  OPENINFO="TUXEDO/QM:/dev/device1:myqueue"
# For NT, :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"
```

Section 5 - File Formats and Data Descriptions

```
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 `buildserver(1)`, `tpdequeue(3c)`, `tpenqueue(3c)`, `servopts(5)`, `TMQFORWARD(5)`, `UBBCONFIG(5)`

Setting Up a BEA Tuxedo Application

Administering a BEA Tuxedo Application at Run Time

Programming a BEA Tuxedo Application Using C

TMSYSEVT(5)

Name	TMSYSEVT - system event reporting process
Synopsis	<pre>TMSYSEVT SRVGRP="<i>identifier</i>" SRVID="<i>number</i>" [CLOPT="[-A] [<i>servopts options</i>] [-- [-S] [-p <i>poll-seconds</i>] [-f <i>control-file</i>]]"]</pre>
Description	<p>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.</p> <p>Filtering and notification rules are stored in <i>control-file</i>, which defaults to <code>\${APPDIR}/tmsysevt.dat</code>. Control file syntax is defined in <code>EVENT_MIB(5)</code>; specifically, the attributes of the classes in <code>EVENT_MIB</code> can be set to activate subscriptions under the full range of notification rules.</p> <p>It is possible to boot one or more secondary TMSYSEVT processes for increased availability. Additional servers must be booted with the <code>-S</code> command line option, which indicates a “secondary” server.</p> <p>When the <code>EVENT_MIB(5)</code> 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 <code>-p</code> option, and is 30 seconds by default.</p>
Interoperability	TMSYSEVT must run on a BEA Tuxedo Release 6.0 or later machine.
Notices	<p>To migrate the primary TMSYSEVT server to another machine, the system administrator must provide a current copy of <i>control-file</i>. Each secondary TMSYSEVT server automatically maintains a recent copy.</p> <p>TMSYSEVT needs access to the system's FML32 field table definitions for system events. <code>FLDTBLDIR32</code> should include <code>\$TUXDIR/udataobj</code>, and <code>FIELDTBLS32</code> should include <code>evt_mib</code>. These environment variables may be set in the machine's or server's environment file.</p>
Example	<pre>*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"</pre>
See Also	<code>tsubscribe(3c)</code> , <code>EVENTS(5)</code> , <code>EVENT_MIB(5)</code> , <code>TMUSREVT(5)</code>

tmtrace(5)

Name `tmtrace` - 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 `undyed` 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 implicit 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, for example, 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 `:undy`.

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.

`undy`
 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:

hhmss.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="*:u!og: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 *ENVFILES* 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 *ENVFILES* 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(3c)`

TMUSREVT(5)

Name	TMUSREVT - user event reporting process
Synopsis	<pre>TMUSREVT SRVGRP="<i>identifier</i>" SRVID="<i>number</i>" [CLOPT="[-A] [<i>servopts options</i>] -- [-S] [-p <i>poll-seconds</i>] [-f <i>control-file</i>]]"</pre>
Description	<p>TMUSREVT is a BEA Tuxedo system provided server that processes event report message buffers from <code>tppost(3c)</code>, and acts as an Event Broker to filter and distribute them.</p> <p>Filtering and notification rules are stored in <i>control-file</i>, which defaults to <code>/\${APPDIR}/tmusrevt.dat</code>. Control file syntax is defined in <code>EVENT_MIB(5)</code>; specifically, the attributes of the classes in <code>EVENT_MIB</code> can be set to activate subscriptions under the full range of notification rules.</p> <p>It is possible to boot one or more secondary TMUSREVT processes for increased availability. Additional servers must be booted with the <code>-S</code> command line option, which indicates a “secondary” server.</p> <p>When the <code>EVENT_MIB(5)</code> 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 <code>-p</code> option, and is 30 seconds by default.</p>
Interoperability	TMUSREVT must run on a BEA Tuxedo Release 6.0 or later machine.
Notices	<p>To migrate the primary TMUSREVT server to another machine, the system administrator must provide a current copy of <i>control-file</i>. Each secondary TMUSREVT server automatically maintains a recent copy.</p> <p>If <code>tppost()</code> will be called in transaction mode, all TMUSREVT server groups must have transactional capability (a TMS process).</p> <p>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.</p>

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(3c)`, `tpsubscribe(3c)`, `EVENTS(5)`, `EVENT_MIB(5)`, `TMSYSEVT(5)`

tperrno(5)

Name	tperrno - 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 <code>tperrno</code> 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.

TPE DIAGNOSTIC

The enqueueing of a message on the specified queue failed. The reason for failure can be determined by the diagnostic returned via *ctl*.

TPE EVENT

An event occurred; the event type is returned in *revent*.

TPE GOTSIG

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

TPE HAZARD

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

TPE HEURISTIC

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

TPE INVAL

An invalid argument was detected.

TPE ITYPE

The type and sub-type of the input buffer is not one of the types and sub-types that the service accepts.

TPE LIMIT

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

TPE MATCH

svcname is already advertised for the server but with a function other than *func*.

TPE MIB

The administrative request failed. *outbuf* is updated and returned to the caller with FML32 fields indicating the cause of the error, as described in *MIB(5)* and *TM_MIB(5)*.

TPE NOENT

Cannot send to *svc* because it does not exist or is not the correct type of service.

TPE OS

An operating system error has occurred.

TPEOTYPE

The type and sub-type of the reply are not known to the caller.

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.

TPERELEASE

When the `TPACK` is set and the target is a client from a prior release of the BEA Tuxedo system that does not support the acknowledgment protocol.

TPERMERR

A resource manager failed to open or close correctly.

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.

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 `tpurcode` - 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()`. `tpurcode` may be used by the application to return additional information to the process that calls an application service. For details, see `tpreturn()`.

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 tuxenv - 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 and View Files
- File System and TLOG Variables
- Workstation Variables
- BEA Tuxedo /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 `nlpaths(5)`.

`PAGER`

specifies the paging command used for paging output in `qadmin(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 a directory in which temporary files may be written. (Temporary files may also be written to a location specific to an operating system, as specified with the `tmpnam()` function, which is called by the BEA Tuxedo MIB and code used for transferring message files. When a call is made to `tmpnam()`, then the `TMPDIR` variable is ignored by the BEA Tuxedo system.)

`TZ`

on systems where the ANSI C `mktime` functions does not exist, `TZ` must be set to use the BEA Tuxedo `gp_mktime(3c)` 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 *Programming a BEA Tuxedo Application Using C*, *Setting Up a BEA Tuxedo Application*, and *Administering a BEA Tuxedo Application at Run Time*.

**Variables for
Field Table Files
and View Files**

The following environment variables are used by `FML` and `VIEWS`:

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.

More information about these variables can be found in *Setting Up a BEA Tuxedo Application*, *Administering a BEA Tuxedo Application at Run Time*, *Programming a BEA Tuxedo Application Using C*, and the *Programming a BEA Tuxedo Application Using FML*.

Section 5 - File Formats and Data Descriptions

File System and TLOG Variables	<p>The following variables are used by the BEA Tuxedo system file system and the transaction log.</p> <p>FSCONFIG the pathname for the Universal Device List.</p> <p>FSMAXCOMMIT sets the maximum size of the commit buffer.</p> <p>FSMAXUPDATE sets the size of the update list and the maximum number of updates.</p> <p>FMSGREGP sets the message repetition interval.</p> <p>FSOFFSET specifies an offset into the Universal Device List.</p>
Workstation Variables	<p>The following variables are used on Workstation client machines:</p> <p>WSALLOWPRE71 determines whether the workstation machine running BEA Tuxedo 7.1 or later software is allowed to interoperate with pre-Release 7.1 BEA Tuxedo applications. Setting the variable to Y (WSALLOWPRE71=Y) allows interoperability.</p> <p>WSBUFFERS the number of packets per application.</p> <p>WSDEVICE the network device to be used for network access. For Workstation clients in BEA Tuxedo Release 6.4 and higher, this variable is no longer required.</p> <p>WSENVFILE pathname of a file containing Workstation client environment variables.</p> <p>WSFADDR the network address used by the Workstation client when connecting to other machines. This variable, along with the WSFRANGE variable, determine the range of TCP/IP ports to which a process will attempt to bind before making an outbound connection.</p>

WSFRANGE

the range of TCP/IP ports to which a native process will attempt to bind before making an outbound connection. The **WSFADDR** variable specifies the base address of the range.

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 variables can be found in *Using the BEA Tuxedo Workstation Component*.

**BEA Tuxedo /Q
Variables**

The following environment variable is used by BEA Tuxedo /Q:

QMCONFIG

sets the device where queue space is available to BEA Tuxedo /Q.

There is more information on this in *Using the BEA Tuxedo /Q Component*.

**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.

Note: On a Windows NT system, the **ALTCC** and **ALTCFLAGS** environment variables are not applicable and setting them will produce unexpected results. You must compile your application first using a COBOL compiler and then pass the resulting object file to the `buildclient(1)` or `buildserver(1)` command.

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 *Programming a BEA Tuxedo Application Using COBOL*.

Additional 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 somewhat less compression than 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 deferral of signals, set this variable to "Y".

See Also `buildclient(1)`, `buildserver(1)`, `viewc`, `viewc32(1)`
`cc(1)`, `environ(5)` in a UNIX system reference manual

tuxtypes(5)

Name tuxtypes - buffer type switch; descriptions of buffer types provided by the 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 tmltype_sw_t tm_typesw[] = {
{
    "CARRAY",      /* type */
    "*",          /* subtype */
    0              /* dfltsize */},
},
{
    "STRING",     /* type */
    "*",          /* subtype */
    512,          /* dfltsize */
    NULL,         /* initbuf */
    NULL,         /* reinitbuf */
    NULL,         /* uninitbuf */
    _strpresend, /* presend */
    NULL,         /* postsend */
    NULL,         /* postrecv */
    _strencdec,   /* encdec */
    NULL,         /* route */
    _sfilter,     /* filter */
    _sformat,     /* format */},
{
    "FML",        /* type */
    "*",          /* subtype */
    1024,         /* dfltsize */
    _finit,       /* initbuf */
    _freinit,     /* reinitbuf */
    _funinit,     /* uninitbuf */

```

Section 5 - File Formats and Data Descriptions

```

        _fpresend,          /* presend */
        _fpostsend,        /* postsend */
        _fpostrecv,        /* postrecv */
        _fencdec,          /* encdec */
        _froute,           /* route */
        _ffilter,          /* filter */
        _fformat,          /* format */},
    NULL,                   /* presend2 */
},
{
    "VIEW",                 /* type */
    "*",                    /* subtype */
    1024,                   /* dfltsize */
    _vinit,                 /* initbuf */
    _vreinit,              /* reinitbuf */
    NULL,                   /* uninitbuf */
    _vpresend,             /* presend */
    NULL,                   /* postsend */
    NULL,                   /* postrecv */
    _vencdec,              /* encdec */
    _vrout,                 /* route */
    _vfilter,              /* filter */
    _vformat,              /* format */
},
{
    /* XATMI - identical to CARRAY */
    "X_OCTET",             /* type */
    "*",                    /* subtype */
    0,                      /* dfltsize */
},
{
    /* XATMI - identical to VIEW */
    {'X','_','C','_','T','Y','P','E'}, /* type */
    "*",                    /* subtype */
    1024,                   /* dfltsize */
    _vinit,                 /* initbuf */
    _vreinit,              /* reinitbuf */
    NULL,                   /* uninitbuf */
    _vpresend,             /* presend */
    NULL,                   /* postsend */
    NULL,                   /* postrecv */
    _vencdec,              /* encdec */
    _vrout,                 /* route */
    _vfilter,              /* filter */
    _vformat,              /* format */
},
{
    /* XATMI - identical to VIEW */
    {'X','_','C','O','M','M','O','N'}, /* type */
    "*",                    /* subtype */
}

```

```

1024,          /* dfltsize */
_vinit,       /* initbuf */
_vreinit,     /* reinitbuf */
NULL,         /* uninitbuf */
_vpresend,    /* presend */
NULL,         /* postsend */
NULL,         /* postrecv */
_vencdec,     /* encdec */
_vroute,     /* route */
_vfilter,     /* filter */
_vformat,     /* format */
},
{
"FML32",      /* type */
"***",       /* subtype */
1024,        /* dfltsize */
_finit32,    /* initbuf */
_freinit32,  /* reinitbuf */
_funinit32,  /* uninitbuf */
_fpresend32, /* presend */
_fpostsend32, /* postsend */
_fpostrecv32, /* postrecv */
_fencdec32,  /* encdec */
_froute32,   /* route */
_ffilter32,  /* filter */
_ffformat32, /* format */
_fpresend232 /* present2 */
},
{
"VIEW32",    /* type */
"***",       /* subtype */
1024,        /* dfltsize */
_vinit32,    /* initbuf */
_vreinit32,  /* reinitbuf */
NULL,        /* uninitbuf */
_vpresend32, /* presend */
NULL,        /* postsend */
NULL,        /* postrecv */
_vencdec32,  /* encdec */
_vroute32,   /* route */
_vfilter32,  /* filter */
_vformat32,  /* format */
},
{
"XML",       /* type */
"***",       /* subtype */
0,           /* dfltsize */
NULL,        /* initbuf */
NULL,        /* reinitbuf */

```

Section 5 - File Formats and Data Descriptions

```

        NULL,          /* uninitbuf */
        NULL,          /* present */
        NULL,          /* postsend */
        NULL,          /* postrecv */
        NULL,          /* encdec */
        _xroute,       /* route */
        NULL,          /* filter */
        NULL,          /* format */
    },
    {
    "
    }
};

struct tmtypesw_t _TM_FAR *
_TMDLLENTY
_tmtypeswaddr(void)
{
    return(tm_typesw);
}

```

Description The following table lists the ten buffer types provided by the BEA Tuxedo system.

CARRAY	character array (possibly containing NULL characters) that is neither encoded nor decoded during transmission
STRING	NULL-terminated character array
FML	FML fielded buffer
XML	buffer for XML documents
VIEW	C structure or FML view
X_OCTET	equivalent to CARRAY; provided for XATMI compatibility
X_C_TYPE	equivalent to VIEW; provided for XATMI compatibility
X_ COMMON	equivalent to VIEW; provided for XATMI compatibility
FML32	FML32 fielded buffer, using 32-bit identifiers and offsets
VIEW32	C structure or 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 programmer who wants to supply a custom 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`.

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` may need to be rebuilt using special commands such as `buildwsh(1)`.

See `buffer(3c)` 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 three routing functions provided by the system, `_froute()`, `_vroute()`, and `_xroute()`, are used for data-dependent routing of `FML` buffers, `VIEW` buffers, and `XML` buffers, respectively. See `UBBCONFIG(5)` for instructions on defining the routing criteria to be used by these three 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(3c)`, `typesw(5)`, `UBBCONFIG(5)`

typesw(5)

Name typesw - buffer type switch structure; parameters and routines needed for each buffer type

Synopsis Buffer Type Structure

```

/*
 * The following definitions are in $TUXDIR/include/tmtypes.h
 */
#define TMTYPELEN ED_TYPELEN
#define TMSTYPELEN ED_STYPELEN

struct tmtime_sw_t {
    char type[TMTYPELEN];      /* type of buffer */
    char subtype[TMSTYPELEN]; /* sub-type of buffer */
    long dfltsize;            /* default size of buffer */
    /* buffer initialization function pointer */
    int (_TMDLLENTY *initbuf) _((char _TM_FAR *, long));
    /* buffer re-initialization function pointer */
    int (_TMDLLENTY *reinitbuf) _((char _TM_FAR *, long));
    /* buffer un-initialization function pointer */
    int (_TMDLLENTY *uninitbuf) _((char _TM_FAR *, long));
    /* pre-send buffer manipulation func pointer */
    long (_TMDLLENTY *presend) _((char _TM_FAR *, long, long));
    /* post-send buffer manipulation func pointer */
    void (_TMDLLENTY *postsend) _((char _TM_FAR *, long, long));
    /* post-receive buffer manipulation func pointer */
    long (_TMDLLENTY *postrecv) _((char _TM_FAR *, long, long));
    /* encode/decode function pointer */
    long (_TMDLLENTY *encdec) _((int, char _TM_FAR *, long, char _TM_FAR *, long));
    /* routing function pointer */
    int (_TMDLLENTY *route) _((char _TM_FAR *, char _TM_FAR *, char _TM_FAR *,
        long, char _TM_FAR *));
    /* buffer filtering function pointer */
    int (_TMDLLENTY *filter) _((char _TM_FAR *, long, char _TM_FAR *, long));
    /* buffer formatting function pointer */
    int (_TMDLLENTY *format) _((char _TM_FAR *, long, char _TM_FAR *,
        char _TM_FAR *, long));
    /* process buffer before sending, possibly generating copy */
    long (_TMDLLENTY *presend2) _((char _TM_FAR *, long, char _TM_FAR *, long,
        long _TM_FAR *));
    /* this space reserved for future expansion */

```

```
void (_TMDLLENTY *reserved[9]) _((void));
};

/*
 * application types switch pointer
 * always use this pointer when accessing the table
 */
extern struct tmttype_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 programmer who wants to supply a customized 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 that must be supplied when adding a new type are specified in `buffer(3c)`.

Files `$TUXDIR/tuxedo/include/tmtypes.h` - the type switch definition
`$TUXDIR/lib/tmtypesw.c` - the type switch instantiation

See Also `buffer(3c)`, `tuxtypes(5)`

UBBCONFIG(5)

Name	UBBCONFIG - text version of a BEA Tuxedo configuration file
Description	<p>When a BEA Tuxedo application is booted, the <code>tmboot</code> command refers to a binary configuration file called <code>TUXCONFIG</code> to get the information necessary for starting application servers and initializing the bulletin boards in an orderly sequence. This binary file cannot be created directly; it must be created from a text file called <code>UBBCONFIG</code>. To configure an application, an administrator creates a <code>UBBCONFIG</code> file (with a text editor) and loads the file into a binary version (<code>TUXCONFIG</code>) by running the <code>tmloadcf(1)</code> command. During the life of the application, the <code>TUXCONFIG</code> file is used by various BEA Tuxedo administrative tools. <code>tmadmin(1)</code> uses the configuration file (or a copy of it) in its monitoring activity. <code>tmshutdown(1)</code> references the configuration file for information needed to shut the application down.</p>
Definitions	<p>A server is a process that accepts requests and sends replies for clients and other servers. A client originates requests and gets replies.</p> <p>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 <code>tpbegin()</code>, and ended with <code>tpcommit()</code> or <code>tpabort()</code>.</p> <p>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 <code>GROUPS</code> section. The client group is not associated with a resource manager.</p> <p>A remote domain is defined to be an environment for which the bulletin board for this BEA Tuxedo system configuration is not available. Remote domains are not specified in the <code>UBBCONFIG</code> file, but rather through host-specific environment variables that are specified in host-specific reference pages.</p>

Configuration File Format

A UBBCONFIG file is made up of nine possible 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`, and `ROUTING`. The `RESOURCES` and `MACHINES` sections must be the first two sections and must be included in that order. The `GROUPS` section must precede the `SERVERS`, `SERVICES`, and `ROUTING` sections. The `NETGROUPS` section must precede the `NETWORK` section.

Parameters (except in the `RESOURCES` section) are generally specified by: `KEYWORD = value`; white space (space or tab character) is allowed on either side of the equal sign (=). This format sets `KEYWORD` to `value`. Valid keywords are described within each section.

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* (a string value already known to the BEA Tuxedo system such as `APP_PW` for the `SECURITY` parameter), standard C rules are typically used. A standard C *identifier* starts with an alphabetic character or underscore and contains only alphanumeric characters or underscores. The maximum allowable length of an identifier is 30 (not including the terminating null).

Note: There is no need to enclose an identifier in double quotes.

A value that is neither an integer number nor an identifier must be enclosed in double quotes. This value is a user-defined *string*. The maximum allowable length of a user-defined string is 78 characters (not including the terminating null). Exceptions to this rule are as follows:

- The CLOPT, BUFTYPE, OPENINFO, and CLOSEINFO parameters, which can be 256 characters in length
- The SEC_PRINCIPAL_NAME, SEC_PRINCIPAL_LOCATION, and SEC_PRINCIPAL_PASSVAR parameters, which can be 511 characters in length (not including the terminating null)
- The RANGES parameter, which can be 2048 characters in length (except in Domains, 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.

"#" (pound sign) introduces a comment. A newline ends a comment.

An identifier or a numeric constant must always be followed by white space (space or tab character), a newline character, 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 cannot be continued.

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 file 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_value

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 as 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 clients and servers that can be simultaneously connected to the bulletin board on any particular machine in this application. This value must be greater than 0 and less than 32,768. If not specified, the default maximum number is 50. The *RESOURCES* value for this parameter can be overridden in the *MACHINES* section on a per-machine basis.

System administration processes, such as the *BBL*, *restartsrv*, *cleanup**srv*, *tmshutdown()*, and *tmadmin()*, need not be accounted for in this value, but the *DBBL*, all bridge processes, all system-supplied and application server processes, and all potential client processes at a particular site need to be counted. (Examples of system-supplied servers are *AUTHSVR*, *TMQUEUE*, *TMQFORWARD*, *TMUSREVT*, *TMSYSEVT*, *TMS*—see *TMSNAME* parameter in *GROUPS* section, *TMS_QM*, *GWTDOMAIN*, and *WSL*.) If the application is booting Workstation Listeners (*WSLs*) at a particular site, both the *WSLs* and the number of potential Workstation Handlers (*WSHs*) that may be booted need to be counted.

Note that for BEA Tuxedo pre-Release 7.1 (6.5 or earlier), both the `MAXACCESSERS` and `MAXSERVERS` parameters for an application play a part in the user license checking scheme. Specifically, a machine is not allowed to boot if the number of `MAXACCESSERS` for that machine + the number of `MAXACCESSERS` for the machine (or machines) already running in the application is greater than the number of `MAXSERVERS` + user licenses for the application. Thus, the total number of `MAXACCESSERS` for an application must be less than or equal to the number of `MAXSERVERS` + user licenses for the application.

Note also that the user license checking scheme in BEA Tuxedo Release 7.1 or later considers only the following two factors when performing its checks: the number of user licenses for an application and the number of licenses currently in use for the application. When all user licenses are in use, no new clients are allowed to join the application.

`MAXSERVERS` *numeric_value*

specifies the maximum number of servers to be accommodated in the bulletin board server table for this application. This value must be greater than 0 and less than 8192. If not specified, the default is 50.

All instances of system-supplied and application servers available to an application need to be accounted for in the bulletin board server table, which is a global table, meaning that the same server table resides on each machine in the application. Examples of system-supplied servers are `AUTHSVR`, `TMQUEUE`, `TMQFORWARD`, `TMUSREVT`, `TMSYSEVT`, `TMS` (see `TMSNAME` parameter in `GROUPS` section), `TMS_QM`, `GWTDOMAIN`, and `WSL`.

Administration of each BEA Tuxedo system site adds approximately one system-supplied server. Additionally, the `DBBL` process and all `BBL`, `bridge`, and `WSH` processes must be accounted for in the `MAXSERVERS` value.

`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 be 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 `TUXCONFIG` file. 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 be 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 be 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 16,384. If not specified, the default is 16,384.

`MODEL` { `SHM` | `MP` }

specifies the configuration type. This parameter is required and only one of the two settings can be specified. `SHM` (for shared memory) 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.

`CMTRET` { `COMPLETE` | `LOGGED` }

specifies the initial setting of the `TP_COMMIT_CONTROL` characteristic for all client and server processes in a BEA Tuxedo system 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, `tpsamt`, for details on the setting of this characteristic.

OPTIONS {[LAN | MIGRATE], *}

specifies options that are used. If two options are given, they are separated by commas. 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 {FASTPATH | PROTECTED}[, NO_OVERRIDE]

specifies the default mode used by BEA Tuxedo system libraries within application processes to gain access to internal tables of the BEA Tuxedo system. FASTPATH specifies that the internal tables are accessible by BEA Tuxedo system libraries via unprotected 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 using flags available for use with `tpinit(3c)` or `TPINITIALIZE(3cbl)`. If SYSTEM_ACCESS is not specified, the default mode is FASTPATH.

Limitation: Setting SYSTEM_ACCESS to PROTECTED may not be effective for multithreaded servers because it is possible that while one thread is executing BEA Tuxedo code, which means it is attached to the Bulletin Board, another thread might be executing user code. The BEA Tuxedo system cannot prevent such situations.

SECURITY {NONE | APP_PW | USER_AUTH | ACL | MANDATORY_ACL}

specifies the type of application security to be enforced. If not specified, 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 default service name, if not specified, is `..AUTHSVC`.

Note that the system-supplied authentication server, `AUTHSVR`, advertises the authentication service as `AUTHSVC` when `SECURITY` is set to `USER_AUTH`, and as `..AUTHSVC` when `SECURITY` is set to `ACL` or `MANDATORY_ACL`. `AUTHSVC` and `..AUTHSVC` point to the same authentication service.

Note also that string values `AUTHSVC` and `..AUTHSVC` are *identifiers*, meaning that there is no need to surround `AUTHSVC` or `..AUTHSVC` with double quotes.

`MAXGTT` *numeric_value*

specifies the maximum number of simultaneous global transactions in which any particular machine in this application 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-machine basis.

`MAXCONV` *numeric_value*

specifies the maximum number of simultaneous conversations in which clients and servers on any particular machine in this application can be involved. It must be greater than 0 and less than 32,768. If not specified, the default is 64 if any conversational servers are defined in the `SERVERS` section, or 1 otherwise. The maximum number of simultaneous conversations per server is 64. The `RESOURCES` value for this parameter can be overridden in the `MACHINES` section on a per-machine basis.

`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()` 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 default 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 | THREAD | 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()` 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()`).

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 native client processes is done by administrative system processes and not by application processes. Therefore, only native clients running with the same UNIX system user identifier as the application administrator can be notified using the `SIGNAL` method. Workstation clients may use the `SIGNAL` method, regardless of which user identifier they are running under.

Note: The `SIGNAL` notification method is not available for MS-DOS clients, and is not available for multithreaded or multicontexted clients.

The value `THREAD` specifies that `THREAD` notification detection should be used. This means that the system dedicates a separate thread for the receipt of unsolicited messages and dispatches the unsolicited message handler in that thread. Only one unsolicited message handler executes at one time per BEA Tuxedo application association. This value is allowed only on platforms that offer support for multi-threading. COBOL clients cannot use `THREAD` notification. Clients that are written in COBOL or that run on a platform on which threads are not supported will have their notification method changed to `DIPIN` if they accept the `UBBCONFIG` default notification method and the `UBBCONFIG` default notification method is `THREAD`. In contrast, if such a client specifies thread notification explicitly in the parameters to `tpinit()` or `TPINITIALIZE()`, then the call to this function will return an error.

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.

`SEC_PRINCIPAL_NAME` *string_value* [0..511]

specifies the security principal name identification string to be used for authentication purposes by an application running BEA Tuxedo 7.1 or later software. This parameter may contain a maximum of 511 characters (excluding the terminating null character). The principal name specified for this parameter becomes the identity of one or more system processes running in this application.

`SEC_PRINCIPAL_NAME` can be specified at any of the following four levels in the configuration hierarchy: `RESOURCES` section, `MACHINES` section, `GROUPS` section, and `SERVERS` section. A principal name at a particular configuration level can be overridden at a lower level. If `SEC_PRINCIPAL_NAME` is not specified at any of these levels, the principal name for the application defaults to the `DOMAINID` string specified in the `RESOURCES` section for this application.

Note that `SEC_PRINCIPAL_NAME` is one of a trio of parameters, the other two being `SEC_PRINCIPAL_LOCATION` and `SEC_PRINCIPAL_PASSVAR`. The latter two parameters pertain to opening decryption keys during application booting for the system processes running in a BEA Tuxedo 7.1 or later application. When only `SEC_PRINCIPAL_NAME` is specified at a particular level, the system sets each of the other two parameters to a `NULL` (zero length) string.

`SEC_PRINCIPAL_LOCATION` *string_value* [0..511]

specifies the location of the file or device where the decryption (private) key for the principal specified in `SEC_PRINCIPAL_NAME` resides. This parameter may contain a maximum of 511 characters (excluding the terminating null character).

`SEC_PRINCIPAL_LOCATION` can be specified at any of the following four levels in the configuration hierarchy: `RESOURCES` section, `MACHINES` section, `GROUPS` section, and `SERVERS` section. When specified at any of these levels, this parameter must be paired with the `SEC_PRINCIPAL_NAME` parameter; otherwise, its value is ignored. (`SEC_PRINCIPAL_PASSVAR` is optional; if not specified, the system sets it to a `NULL`—zero length—string.)

`SEC_PRINCIPAL_PASSVAR` *string_value* [0..511]

specifies the variable in which the password for the principal specified in `SEC_PRINCIPAL_NAME` is stored. This parameter may contain a maximum of 511 characters (excluding the terminating null character).

SEC_PRINCIPAL_PASSVAR can be specified at any of the following four levels in the configuration hierarchy: RESOURCES section, MACHINES section, GROUPS section, and SERVERS section. When specified at any of these levels, this parameter must be paired with the SEC_PRINCIPAL_NAME parameter; otherwise, its value is ignored. (SEC_PRINCIPAL_LOCATION is optional; if not specified, the system sets it to a NULL—zero length—string.)

During initialization, the administrator must provide the password for each of the decryption keys configured with SEC_PRINCIPAL_PASSVAR. (tmloadcf(1) prompts for the password.) The system automatically encrypts the password entered by the administrator and assigns each encrypted password to the associated password variable.

SIGNATURE_AHEAD *numeric_value* (1 <= num <= 2147483647)

specifies the number of seconds into the future that a digital signature's timestamp is allowed to be, when compared to the local machine's clock. If not specified, the default is 3600 seconds (one hour). This parameter only applies to applications running BEA Tuxedo 7.1 or later software.

SIGNATURE_BEHIND *numeric_value* (1 <= num <= 2147483647)

specifies the number of seconds into the past that a digital signature's timestamp is allowed to be, when compared to the local machine's clock. If not specified, the default is 604800 seconds (one week). This parameter only applies to applications running BEA Tuxedo 7.1 or later software.

SIGNATURE_REQUIRED {Y | N}

specifies whether or not every process running in this application requires a digital signature on its input message buffer. If not specified, the default is N. This parameter only applies to applications running BEA Tuxedo 7.1 or later software.

SIGNATURE_REQUIRED can be specified at any of the following four levels in the configuration hierarchy: RESOURCES section, MACHINES section, GROUPS section, and SERVICES section. Setting SIGNATURE_REQUIRED to Y at a particular level means that signatures are required for all processes running at that level or below.

ENCRYPTION_REQUIRED {Y | N}

specifies whether or not every process running in this application requires an encrypted input message buffer. If not specified, the default is N. This parameter only applies to applications running BEA Tuxedo 7.1 or later software.

ENCRYPTION_REQUIRED can be specified at any of the following four levels in the configuration hierarchy: RESOURCES section, MACHINES section, GROUPS section, and SERVICES section. Setting ENCRYPTION_REQUIRED to Y at a particular level means that encryption is required for all processes running at that level or below.

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. On a Windows NT system, the value can be set using the Computer Name value in the Network Control Panel and must be specified in upper case. 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 *KEYWORDS* 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 path name 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.

Note: The path name specified for this parameter must match exactly (including case) the path name specified for the `TUXCONFIG` environment variable. Otherwise, `tmloadcf(1)` cannot be run successfully.

`TUXDIR = string_value`

This is the absolute path name of the directory where the BEA Tuxedo system software is found on this machine. This parameter must be specified for each machine and the path name 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 *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 path name of the application directory and is the current directory for all application and administrative servers booted on this machine. The absolute path name can optionally be followed by a colon-separated list of other path names. 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 clients and servers that can be simultaneously connected to the bulletin board on this machine. This value must be greater than 0 and less than 32,768. If not specified, the default is the `MAXACCESSERS` value specified in the `RESOURCES` section.

System administration processes, such as the `BBL`, `restartsrv`, `cleanupsrv`, `tmshutdown()`, and `tmadmin()`, need not be accounted for in this value, but the `DBBL`, all bridge processes, all system-supplied and application server processes, and all potential client processes at this site need to be counted. (Examples of system-supplied servers are `AUTHSVR`, `TMQUEUE`, `TMQFORWARD`, `TMUSREVT`, `TMSYSEVT`, `TMS`—see `TMSNAME` parameter in `GROUPS` section, `TMS_QM`, `GWTDOMAIN`, and `WSL`.) If the application is booting Workstation Listeners (WSLs) at this site, both the WSLs and the number of potential Workstation Handlers (WSHs) that may be booted need to be counted.

Note that for BEA Tuxedo pre-Release 7.1 (6.5 or earlier), both the `MAXACCESSERS` and `MAXSERVERS` (see `MAXSERVERS` in `RESOURCES` section) parameters for an application play a part in the user license checking scheme. Specifically, a machine is not allowed to boot if the number of `MAXACCESSERS` for that machine + the number of `MAXACCESSERS` for the machine (or machines) already running in the application is greater than the number of `MAXSERVERS` + user licenses for the application. Thus, the total number of `MAXACCESSERS` for an application must be less than or equal to the number of `MAXSERVERS` + user licenses for the application.

Note also that the user license checking scheme in BEA Tuxedo Release 7.1 or later considers only the following two factors when performing its checks: the number of user licenses for an application and the number of licenses currently in use for the application. When all user licenses are in use, no new clients are allowed to join the application.

`MAXWSCLIENTS = number`

specifies the number of accesser entries on this machine to be reserved for Workstation clients (as opposed to native clients). If specified, the value must be greater than or equal to 0 and less than 32,768. If not specified, the default is 0.

The number specified here takes a portion of the total accesser slots specified with `MAXACCESSERS`, meaning that the accesser slots reserved for

MAXWSCLIENTS are unavailable for use by other clients and servers on this machine. It is an error to set this number greater than MAXACCESSERS.

The MAXWSCLIENTS parameter is only used when the BEA Tuxedo System Workstation feature is used. The appropriate setting of this parameter helps to conserve interprocess communication (IPC) resources since Workstation client access to the system is multiplexed through a BEA Tuxedo system-supplied surrogate, the Workstation Handler (WSH).

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 32,000. The default is 100.

MAXCONV = *number*

specifies the maximum number of simultaneous conversations in which clients and servers on this machine can be involved. It must be greater than 0 and less than 32,768. If not specified, the default is the MAXCONV 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 this 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 path name 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. “mmddy” (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` file 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 `tmbboot(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. `tmbboot` and `tlisten` process the machine `ENVFILE` before starting the server, allowing

for the environment to indicate necessary path names 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 path names 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 path name. `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`.

`SEC_PRINCIPAL_NAME = string_value [0..511]`

specifies the security principal name identification string to be used for authentication purposes by an application running BEA Tuxedo 7.1 or later software. This parameter may contain a maximum of 511 characters (excluding the terminating null character). The principal name specified for this parameter becomes the identity of one or more system processes running on this machine.

`SEC_PRINCIPAL_NAME` can be specified at any of the following four levels in the configuration hierarchy: `RESOURCES` section, `MACHINES` section, `GROUPS` section, and `SERVERS` section. A principal name at a particular configuration level can be overridden at a lower level. If `SEC_PRINCIPAL_NAME` is not specified at any of these levels, the principal name for the application defaults to the `DOMAINID` string specified in the `RESOURCES` section for this application.

Note that `SEC_PRINCIPAL_NAME` is one of a trio of parameters, the other two being `SEC_PRINCIPAL_LOCATION` and `SEC_PRINCIPAL_PASSVAR`. The latter two parameters pertain to opening decryption keys during application

booting for the system processes running in a BEA Tuxedo 7.1 or later application. When only `SEC_PRINCIPAL_NAME` is specified at a particular level, the system sets each of the other two parameters to a NULL (zero length) string.

`SEC_PRINCIPAL_LOCATION = string_value [0..511]`

specifies the location of the file or device where the decryption (private) key for the principal specified in `SEC_PRINCIPAL_NAME` resides. This parameter may contain a maximum of 511 characters (excluding the terminating null character).

`SEC_PRINCIPAL_LOCATION` can be specified at any of the following four levels in the configuration hierarchy: `RESOURCES` section, `MACHINES` section, `GROUPS` section, and `SERVERS` section. When specified at any of these levels, this parameter must be paired with the `SEC_PRINCIPAL_NAME` parameter; otherwise, its value is ignored. (`SEC_PRINCIPAL_PASSVAR` is optional; if not specified, the system sets it to a NULL—zero length—string.)

`SEC_PRINCIPAL_PASSVAR = string_value [0..511]`

specifies the variable in which the password for the principal specified in `SEC_PRINCIPAL_NAME` is stored. This parameter may contain a maximum of 511 characters (excluding the terminating null character).

`SEC_PRINCIPAL_PASSVAR` can be specified at any of the following four levels in the configuration hierarchy: `RESOURCES` section, `MACHINES` section, `GROUPS` section, and `SERVERS` section. When specified at any of these levels, this parameter must be paired with the `SEC_PRINCIPAL_NAME` parameter; otherwise, its value is ignored. (`SEC_PRINCIPAL_LOCATION` is optional; if not specified, the system sets it to a NULL—zero length—string.)

During initialization, the administrator must provide the password for each of the decryption keys configured with `SEC_PRINCIPAL_PASSVAR`. (`tmloadcf(1)` prompts for the password.) The system automatically encrypts the password entered by the administrator and assigns each encrypted password to the associated password variable.

`SIGNATURE_REQUIRED = { Y | N }`

specifies whether or not every process running on this machine requires a digital signature on its input message buffer. If not specified, the default is `N`. This parameter only applies to applications running BEA Tuxedo 7.1 or later software.

`SIGNATURE_REQUIRED` can be specified at any of the following four levels in the configuration hierarchy: `RESOURCES` section, `MACHINES` section, `GROUPS` section, and `SERVICES` section. Setting `SIGNATURE_REQUIRED` to `Y` at a particular level means that signatures are required for all processes running at that level or below.

`ENCRYPTION_REQUIRED` = {`Y` | `N`}

specifies whether or not every process running on this machine requires an encrypted input message buffer. If not specified, the default is `N`. This parameter only applies to applications running BEA Tuxedo 7.1 or later software.

`ENCRYPTION_REQUIRED` can be specified at any of the following four levels in the configuration hierarchy: `RESOURCES` section, `MACHINES` section, `GROUPS` section, and `SERVICES` section. Setting `ENCRYPTION_REQUIRED` to `Y` at a particular level means that encryption is required for all processes running at that level or below.

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`, `wtmconfig(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()`, and ended with `tpcommit()/tpabort()`). 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.

`SEC_PRINCIPAL_NAME` = *string_value* [0..511]

specifies the security principal name identification string to be used for authentication purposes by an application running BEA Tuxedo 7.1 or later software. This parameter may contain a maximum of 511 characters (excluding the terminating null character). The principal name specified for this parameter becomes the identity of one or more system processes running in this group.

`SEC_PRINCIPAL_NAME` can be specified at any of the following four levels in the configuration hierarchy: `RESOURCES` section, `MACHINES` section, `GROUPS` section, and `SERVERS` section. A principal name at a particular configuration level can be overridden at a lower level. If `SEC_PRINCIPAL_NAME` is not specified at any of these levels, the principal name for the application defaults to the `DOMAINID` string specified in the `RESOURCES` section for this application.

Note that `SEC_PRINCIPAL_NAME` is one of a trio of parameters, the other two being `SEC_PRINCIPAL_LOCATION` and `SEC_PRINCIPAL_PASSVAR`. The latter two parameters pertain to opening decryption keys during application booting for the system processes running in a BEA Tuxedo 7.1 or later application. When only `SEC_PRINCIPAL_NAME` is specified at a particular level, the system sets each of the other two parameters to a NULL (zero length) string.

`SEC_PRINCIPAL_LOCATION = string_value [0..511]`

specifies the location of the file or device where the decryption (private) key for the principal specified in `SEC_PRINCIPAL_NAME` resides. This parameter may contain a maximum of 511 characters (excluding the terminating null character).

`SEC_PRINCIPAL_LOCATION` can be specified at any of the following four levels in the configuration hierarchy: `RESOURCES` section, `MACHINES` section, `GROUPS` section, and `SERVERS` section. When specified at any of these levels, this parameter must be paired with the `SEC_PRINCIPAL_NAME` parameter; otherwise, its value is ignored. (`SEC_PRINCIPAL_PASSVAR` is optional; if not specified, the system sets it to a NULL—zero length—string.)

`SEC_PRINCIPAL_PASSVAR = string_value [0..511]`

specifies the variable in which the password for the principal specified in `SEC_PRINCIPAL_NAME` is stored. This parameter may contain a maximum of 511 characters (excluding the terminating null character).

`SEC_PRINCIPAL_PASSVAR` can be specified at any of the following four levels in the configuration hierarchy: `RESOURCES` section, `MACHINES` section, `GROUPS` section, and `SERVERS` section. When specified at any of these levels, this parameter must be paired with the `SEC_PRINCIPAL_NAME` parameter; otherwise, its value is ignored. (`SEC_PRINCIPAL_LOCATION` is optional; if not specified, the system sets it to a NULL—zero length—string.)

During initialization, the administrator must provide the password for each of the decryption keys configured with `SEC_PRINCIPAL_PASSVAR`. (`tmloadcf(1)` prompts for the password.) The system automatically encrypts the password entered by the administrator and assigns each encrypted password to the associated password variable.

`SIGNATURE_REQUIRED = { Y | N }`

specifies whether or not every process running in this group requires a digital signature on its input message buffer. If not specified, the default is `N`. This

parameter only applies to applications running BEA Tuxedo 7.1 or later software.

SIGNATURE_REQUIRED can be specified at any of the following four levels in the configuration hierarchy: RESOURCES section, MACHINES section, GROUPS section, and SERVICES section. Setting SIGNATURE_REQUIRED to Y at a particular level means that signatures are required for all processes running at that level or below.

ENCRYPTION_REQUIRED = { Y | N }

specifies whether or not every process running in this group requires an encrypted input message buffer. If not specified, the default is N. This parameter only applies to applications running BEA Tuxedo 7.1 or later software.

ENCRYPTION_REQUIRED can be specified at any of the following four levels in the configuration hierarchy: RESOURCES section, MACHINES section, GROUPS section, and SERVICES section. Setting ENCRYPTION_REQUIRED to Y at a particular level means that encryption is required for all processes running at that level or below.

OPENINFO = *string_value*

specifies the resource manager dependent information needed when opening the resource manager for this group. 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 the TMSNAME parameter for this group is *not* set or is set to TMS. If the TMSNAME parameter is set to a value other than TMS but the OPENINFO string is set to the null string (" ") or is not specified, a resource manager exists for the group but does not require any information for executing an `open` operation.

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 the published name of the vendor's transaction (XA) interface followed immediately by a colon (:).

For BEA Tuxedo /Q databases, the format is:

On UNIX

```
OPENINFO = "TUXEDO/QM:qmconfig:qspace"
```

On Windows NT

```
OPENINFO = "TUXEDO/QM:qmconfig:qspace"
```

In AS/400 environment

```
OPENINFO = "TUXEDO/QM:qmconfig;qspace"
```

In OpenVMS environment

```
OPENINFO = "TUXEDO/QM,[a.b.c]qmconfig,qspace"
```

where TUXEDO/QM is the published name of the BEA Tuxedo/Q XA interface, *qmconfig* is replaced with the name of the QMCONFIG (see `qmadmin(1)`) on which the queue space resides, and *qspace* is replaced with the name of the queue space. For NT and AS/400, the separator after *qmconfig* must be a semicolon (;). For OpenVMS, the separator after TUXEDO/QM and after *qmconfig* must be a comma (,).

For other vendors' databases, the format of the OPENINFO string is specific to the particular vendor providing the underlying resource manager. As an example, the following OPENINFO string demonstrates the type of information needed when opening the Oracle resource manager.

```
OPENINFO="Oracle_XA:  
Oracle_XA+Acc=P/Scott/*****+SesTm=30+LogDit=/tmp"
```

Oracle_XA is the published name of the Oracle XA interface. The series of five asterisks (*) in the OPENINFO string pertains to the encrypting of a password, which is described in the paragraphs that follow.

Passwords passed to a resource manager in the OPENINFO string can be stored in either clear text or encrypted form. To encrypt a password, first enter a series of five or more continuous asterisks in the OPENINFO string at the place where you want the password to go. Then load the UBBCONFIG file by running `tmloadcf(1)`. When `tmloadcf()` encounters the string of asterisks, it prompts you to create a password. For example:

```
tmloadcf -y /usr5/apps/bankapp/myubconfig  
Password for OPENINFO (SRVGRP=BANKB3):  
password
```

`tmloadcf(1)` stores the password in the TUXCONFIG file in encrypted form. If you then regenerate the UBBCONFIG file from the TUXCONFIG file using `tmunloadcf()`, the password is printed in the regenerated UBBCONFIG file in encrypted form with @@ as delimiters. For example:

```
OPENINFO="Oracle_XA:  
Oracle_XA+Acc=P/Scott/@A0986F7733D4@@+SesTm=30+LogDit=/tmp"
```

When `tmloadcf()` encounters an encrypted password in a `UBBCONFIG` file generated by `tmunloadcf()`, it does not prompt the user to create a password.

`CLOSEINFO = string_value`

specifies the resource manager dependent information needed when closing the resource manager for this group. The value must be enclosed in double quotes and must be less than or equal to 256 characters in length. Note that a `CLOSEINFO` string is not used for BEA Tuxedo /Q databases.

This value is ignored if the `TMSNAME` parameter for this group is *not* set or is set to `TMS`. If the `TMSNAME` parameter is set to a value other than `TMS` but the `CLOSEINFO` string is set to the null string (" ") or is not specified, a resource manager exists for the group but does not require any information for executing a `close` operation.

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 the published name of the vendor's transaction (XA) interface followed immediately by a colon (:).

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 fallback 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 *Setting Up a BEA Tuxedo Application*.) 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: Parallel data circuits are prioritized by network group number (NETGRPNO) within priority group number.

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, which 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.

Note: Some port numbers may be reserved for the underlying transport protocols (such as TCP/IP) used by your system. Check the documentation for your transport protocols to find out which numbers, if any, are reserved on your system.

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 path name 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 reboot the master site after failure.

`FADDR = string_value`

specifies the network address used by the local machine when connecting to other machines. This parameter, along with the `FRANGE` parameter, determines the range of TCP/IP ports to which a process will attempt to bind before making an outbound connection. This address must be a TCP/IP address. The port portion of the TCP/IP address represents the base address from which a range of TCP/IP ports can be bound by the process. The `FRANGE` parameter specifies the size of the range. For example, if this address is `//mymachine.bea.com:30000` and `FRANGE` is 200, then all native processes attempting to make outbound connections from this *LMID* will bind a port on `mymachine.bea.com` between 30000 and 30200. If not set, this parameter defaults to the empty string, which implies the operating system chooses a local port randomly.

`FRANGE = number`

specifies the range of TCP/IP ports to which a native process will attempt to bind before making an outbound connection. The `FADDR` parameter specifies the base address of the range. For example, if the `FADDR` parameter is set to `//mymachine.bea.com:30000` and `FRANGE` is set to 200, then all native

processes attempting to make outbound connections from this LMID will bind a port on `mymachine.bea.com` between 30000 and 30200. The valid range is 1-65535. The default is 1.

`MINENCRYPTBITS = { 0 | 40 | 56 | 128 }`

specifies the minimum level of encryption required when a network link to this machine is being established. 0 means no encryption, while 40, 56, and 128 specify the encryption key length (in bits). If this minimum level of encryption cannot be met, link establishment fails. The default is 0.

Note: The link-level encryption value of 40 bits is provided for backward compatibility.

`MAXENCRYPTBITS = { 0 | 40 | 56 | 128 }`

specifies the maximum level of encryption allowed when a network link is being established. 0 means no encryption, while 40, 56, and 128 specify the encryption length (in bits). The default is 128.

Note: The link-level encryption value of 40 bits is provided for backward compatibility.

`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.

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 path name 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 `MIN` 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 path names 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()` or `tpcall()`) 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` and same executable name 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 path name or relative to *APPDIR* (do not 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.

Limitation: Setting `SYSTEM_ACCESS` to `PROTECTED` may not be effective for multithreaded servers because it is possible that while one thread is executing BEA Tuxedo code, which means it is attached to the Bulletin Board, another thread might be executing user code. The BEA Tuxedo system cannot prevent such situations.

`MAXDISPATCHTHREADS = number`

specifies the maximum number of concurrently dispatched threads which each server process may spawn. This parameter is effective only if the server has been built with the `buildserver -t` command.

If `MAXDISPATCHTHREADS > 1`, then a separate dispatcher thread is used and does not count against this limit. It is required that `MINDISPATCHTHREADS <= MAXDISPATCHTHREADS`. If this parameter is not specified, the default is 1.

`MINDISPATCHTHREADS = number`

specifies the number of server dispatch threads started on initial server boot. This parameter is effective only if the server has been built with the `buildserver -t` command.

The separate dispatcher thread that is used when `MAXDISPATCHTHREADS > 1` is not counted as part of the `MINDISPATCHTHREADS` value. It is required that `MINDISPATCHTHREADS <= MAXDISPATCHTHREADS`. The default for this parameter is 0.

`THREADSTACKSIZE = number`

If this parameter is not specified or if the value specified is 0, the operating system default is used. This option has an effect on the server only when a value greater than 1 is specified for `MAXDISPATCHTHREADS`.

`SEC_PRINCIPAL_NAME = string_value [0..511]`

specifies the security principal name identification string to be used for authentication purposes by an application running BEA Tuxedo 7.1 or later software. This parameter may contain a maximum of 511 characters (excluding the terminating null character). The principal name specified for this parameter becomes the identity of one or more system processes running on this server.

`SEC_PRINCIPAL_NAME` can be specified at any of the following four levels in the configuration hierarchy: `RESOURCES` section, `MACHINES` section, `GROUPS` section, and `SERVERS` section. A principal name at a particular configuration level can be overridden at a lower level. If `SEC_PRINCIPAL_NAME` is not specified at any of these levels, the principal name for the application defaults

to the DOMAINID string specified in the RESOURCES section for this application.

Note that SEC_PRINCIPAL_NAME is one of a trio of parameters, the other two being SEC_PRINCIPAL_LOCATION and SEC_PRINCIPAL_PASSVAR. The latter two parameters pertain to opening decryption keys during application booting for the system processes running in a BEA Tuxedo 7.1 or later application. When only SEC_PRINCIPAL_NAME is specified at a particular level, the system sets each of the other two parameters to a NULL (zero length) string.

SEC_PRINCIPAL_LOCATION = *string_value* [0..511]

specifies the location of the file or device where the decryption (private) key for the principal specified in SEC_PRINCIPAL_NAME resides. This parameter may contain a maximum of 511 characters (excluding the terminating null character).

SEC_PRINCIPAL_LOCATION can be specified at any of the following four levels in the configuration hierarchy: RESOURCES section, MACHINES section, GROUPS section, and SERVERS section. When specified at any of these levels, this parameter must be paired with the SEC_PRINCIPAL_NAME parameter; otherwise, its value is ignored. (SEC_PRINCIPAL_PASSVAR is optional; if not specified, the system sets it to a NULL—zero length—string.)

SEC_PRINCIPAL_PASSVAR = *string_value* [0..511]

specifies the variable in which the password for the principal specified in SEC_PRINCIPAL_NAME is stored. This parameter may contain a maximum of 511 characters (excluding the terminating null character).

SEC_PRINCIPAL_PASSVAR can be specified at any of the following four levels in the configuration hierarchy: RESOURCES section, MACHINES section, GROUPS section, and SERVERS section. When specified at any of these levels, this parameter must be paired with the SEC_PRINCIPAL_NAME parameter; otherwise, its value is ignored. (SEC_PRINCIPAL_LOCATION is optional; if not specified, the system sets it to a NULL—zero length—string.)

During initialization, the administrator must provide the password for each of the decryption keys configured with SEC_PRINCIPAL_PASSVAR. (tmloadcf(1) prompts for the password.) The system automatically encrypts the password entered by the administrator and assigns each encrypted password to the associated password variable.

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 dequeuing 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.

A lower priority message does not remain forever enqueued because every tenth message is retrieved on a FIFO basis. Response time should not be a concern of the lower priority interface or service.

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. The BEA Tuxedo system provides the following types of data buffers: FML and FML32 (for FML buffers), XML (for XML buffers), VIEW, VIEW32, 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, VIEW32, 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 this parameter is 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. Note that this signal affects all threads in the server. The default for this parameter is 0.

SIGNATURE_REQUIRED = {Y | N}

specifies whether or not every instance of this service requires a digital signature on its input message buffer. If not specified, the default is N. This parameter only applies to applications running BEA Tuxedo 7.1 or later software.

`SIGNATURE_REQUIRED` can be specified at any of the following four levels in the configuration hierarchy: `RESOURCES` section, `MACHINES` section, `GROUPS` section, and `SERVICES` section. Setting `SIGNATURE_REQUIRED` to `Y` at a particular level means that signatures are required for all processes running at that level or below.

`ENCRYPTION_REQUIRED` = { `Y` | `N` }

specifies whether or not every instance of this service requires an encrypted input message buffer. If not specified, the default is `N`. This parameter only applies to applications running BEA Tuxedo 7.1 or later software.

`ENCRYPTION_REQUIRED` can be specified at any of the following four levels in the configuration hierarchy: `RESOURCES` section, `MACHINES` section, `GROUPS` section, and `SERVICES` section. Setting `ENCRYPTION_REQUIRED` to `Y` at a particular level means that encryption is required for all processes running at that level or below.

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.

ROUTING Section

This section provides information for data-dependent routing of service requests using FML buffers, XML buffers, and views. The routing criteria specified here are used only if the default routing functions `_froute`, `_xroute`, 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 or FML32 buffer, XML element or element attribute, view field name that is identified in an FML field table (using two environment variables—`FLDTBLDIR` and `FIELDTBLS`, or `FLDTBLDIR32` and `FIELDTBLS32`), or an FML view table (using two environment variables—`VIEWDIR` and `VIEWFILES`, or `VIEWDIR32` and `VIEWFILES32`), 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 FML or an FML32 buffer is used for routing, the value of that field must be a number less than or equal to 8191.

To route XML documents on the basis of element content or element attribute, you must define the value of the `FIELD` parameter with the following syntax.

```
FIELD="root_element[/child_element][[/child_element][/. . .][/@attribute_name]"
```

The value of `FIELD` specifies the name of the routing element or an element attribute. This element is assumed to be an element type (or name) or an element attribute name of an XML document or datagram. This information is used to identify the element content or element attribute value for data-dependent routing while sending a document or datagram. The element name and attribute name combined may contain no more than 30 characters. Because indexing is not supported, the BEA Tuxedo system recognizes only the first occurrence of a given element type when processing an XML buffer for data-dependent routing.

XML strictly defines the set of characters that may be used in an attribute name. An attribute name must be a string consisting of a single letter, underscore, or colon followed by one or more name characters. Both element names and attribute names are case-sensitive.

You can find more information about XML on the World Wide Web Consortium Web site at <http://www.w3c.org/XML>.

`FIELDTYPE = type`

indicates the type of routing field specified in the `FIELD` parameter. This parameter is used only for routing XML buffers. The value `type` can be set to one of the following: `CHAR`, `SHORT`, `LONG`, `FLOAT`, `DOUBLE`, or `STRING`. The default type of the routing field is `STRING`.

`RANGES = string_value`

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, `carry`, and character field types must be placed inside a pair of single quotes and cannot 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. The meta-character "*" (wild-card) indicates that the request goes to the default group if the field value does not match the range or if there is match but no viable server in the group associated with the range entry, then the service request is forwarded to the default group specified on the wildcard "*" range entry.

Within a range/group pair, the range is separated from the group name by a ":".

An XML element content and attribute value must be encoded in UTF-8 and can be used for routing if it can be converted to the data type specified by the `FIELDTYPE` parameter.

When used for routing, the element content cannot contain character references, entity references, or CDATA sections.

An XML attribute value (encoded in UTF-8) can be used for routing if the element to which the attribute belongs is defined.

`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 must be one of the following: `FML`, `FML32`, `XML`, `VIEW`, `VIEW32`, `X_C_TYPE`, or `X_COMMON`. No subtype can be specified for types `FML`, `FML32`, or `XML`. Subtypes are required for type `VIEW`, `VIEW32`, `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 cannot 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.

An example of a routing entry based on the XML element CODE is:

```
PRODUCT FIELD="ORDER/CODE" RANGES="'AAA' - 'FFF':DBG1,  
'GGG-ZZZ':DBG2" BUFTYPE="XML"
```

Here, CODE is a child element of the root element ORDER.

A routing entry based on the attribute ORDERNO might look like the following example.

```
ORDER FIELD="ORDER/HEADER/@ORDERNO" FIELDTYPE=long  
RANGES="0-9999:DBG1,10000-MAX:DBG3" BUFTYPE="XML"
```

Here, ORDERNO is the attribute of the XML child element HEADER of the root element ORDER.

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.  
*RESOURCE  
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  
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, :bankdb: becomes ;bankdb;
BANKB1      LMID=SITE1 GRPNO=1
        OPENINFO="TUXEDO/SQL:/usr2/apps/bank/bankd11:bankdb:readwrite"
# For NT, :bankdb: becomes ;bankdb;
BANKB2      LMID=SITE2 GRPNO=2
        OPENINFO="TUXEDO/SQL:/usr4/apps/bank/bankd12: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

```

Section 5 - File Formats and Data Descriptions

```
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 multiple releases of the BEA Tuxedo system are interoperating with each other.

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 on.

See Also `buildserver(1)`, `tmadmin(1)`, `tmboot(1)`, `tmloadcf(1)`, `tmshutdown(1)`, `tmunloadcf(1)`, `buffer(3c)`, `tpinit(3c)`, `servopts(5)`

Setting Up a BEA Tuxedo Application

Administering a BEA Tuxedo Application at Run Time

Programming a BEA Tuxedo Application Using C

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()` command, the viewfile forms the basis for a binary file (filename `view_filename.V`) and a header file (`view_filename.h`) (see `viewc`, `viewc32(1)`).

The binary `.v` files are used two ways in the BEA Tuxedo system:

- for programs that use `Fvftos()` and `Fvstof()`, 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` and `VIEW32`, the `.v` file is searched for a structure of the name provided in the `subtype` argument of `tpalloc()`

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(3c)`, 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  ffname  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 cannot 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, 'z' should be specified, and the view compiler will compute the size.

null

is the user-specified null value or 'z' 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.

Section 5 - File Formats and Data Descriptions

```
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`, `viewc32(1)`, `tpalloc(3c)`, `Fvftos`, `Fvftos32(3fml)`, `Fvstof`, `Fvstof32(3fml)`

Programming a BEA Tuxedo Application Using FML

WS_MIB(5)

- Name** WS_MIB - Management Information Base for Workstation
- 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.

Section 5 - File Formats and Data Descriptions

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 be 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 <code>udataobj/tpadm</code> relative to the root directory of the BEA Tuxedo system software installed on the system. The directory <code>\${TUXDIR}/udataobj</code> should be included by the application in the colon separated list specified by the <code>FLDTBLDIR</code> environment variable and the field table name <code>tpadm</code> should be included in the comma separated list specified by the <code>FIELDTBLS</code> 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 TM_MIB(5)) by the common key field TA_WSHCLIENTID.

Attribute Table

WS_MIB(5): T_WSH Class Definition Attribute Table

Attribute ¹	Type	Permissions	Values	Default
TA_CLIENTID(*)	string	R--R--R--	<i>string</i> [1..78]	N/A
TA_WSHCLIENTID(*)	string	R--R--R--	<i>string</i> [1..78]	N/A
TA_SRVGRP(*)	string	R--R--R--	<i>string</i> [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--	<i>LMID</i>	N/A
TA_PID(*)	long	R--R--R--	1 <= num	N/A
TA_NADDR	string	R--R--R--	<i>string</i> [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

Section 5 - File Formats and Data Descriptions

WS_MIB(5): T_WSH 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

¹All 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: $1 \leq num < 30,001$

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 targeted 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: $1 \leq num < 32,767$

High water number of clients accessing application via this WSH.

TA_MULTIPLEX: $1 \leq num < 32,767$

Maximum number of clients that may access the application via this WSH.

TA_CURCLIENTS: $1 \leq num < 32,767$

Current number of clients accessing application via this WSH.

TA_TIMELEFT: $0 \leq num$

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.

The attributes of WSH servers are meaningful only in a run-time environment. Therefore they cannot be changed, in an unbooted environment, by using the `tpadmcall(3c)` function.

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

WS_MIB(5): T_WSL Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
<code>TA_SRVGRP(r)(*)</code>	string	<code>ru-r--r--</code>	<i>string</i> [1..30]	N/A
<code>TA_SRVID(r)(*)</code>	long	<code>ru-r--r--</code>	$1 \leq num < 30,001$	N/A
<code>TA_GRPNO(k)</code>	long	<code>r--r--r--</code>	$1 \leq num < 30,001$	N/A
<code>TA_STATE(k)</code>	string	<code>rwxr-xr--</code>	See <code>T_SERVER</code> Class in <code>TM_MIB(5)</code>	
<code>TA_LMID(k)</code>	string	<code>R--R--R--</code>	<i>LMID</i>	N/A
<code>TA_PID(k)</code>	long	<code>R--R--R--</code>	$1 \leq num$	N/A
<code>TA_DEVICE</code>	string	<code>rw-r--r--</code>	<i>string</i> [0..78]	N/A
<code>TA_NADDR(r)</code>	string	<code>rw-r--r--</code>	<i>string</i> [1..78]	N/A
<code>TA_EXT_NADDR</code>	string	<code>rw-r--r--</code>	<i>string</i> [0..78]	""
<code>TA_WSHNAME</code>	string	<code>rw-r--r--</code>	<i>string</i> [1..78]	"WSH"
<code>TA_MINHANDLERS</code>	long	<code>rwxr-xr--</code>	$0 \leq num < 256$	0
<code>TA_MAXHANDLERS</code>	long	<code>rw-r--r--</code>	$0 \leq num < 32,767$	See note ¹ .
<code>TA_MULTIPLEX</code>	long	<code>rw-r--r--</code>	$1 \leq num < 32,767$	10
<code>TA_MINENCRYPTBITS</code>	string	<code>rwxrwx---</code>	{0 40 56 128} See note ² .	0
<code>TA_MAXENCRYPTBITS</code>	string	<code>rwxrwx---</code>	{0 40 56 128} See note ² .	128

WS_MIB(5): T_WSL Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
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
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_KEEPLIVE	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

Section 5 - File Formats and Data Descriptions

WS_MIB(5): T_WSL Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
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--	<i>threshold</i>	MAXLONG
TA_CLOPT	string	rwxr--r--	<i>string</i> [0..128]	"-A"
TA_ENVFILE	string	rwxr--r--	<i>string</i> [0..78]	" "
TA_GRACE	long	rwxr--r--	0 <= num	0
TA_KEEPAKIVE	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--	<i>string</i> [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

WS_MIB(5): T_WSL Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
-----------	------	-------------	--------	---------

(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.

² The link-level encryption value of 40 bits is provided for backward compatibility.

Attribute Semantics	TA_SRVGRP: <i>string</i> [1..30] Logical name of the server group. Server group names cannot contain an asterisk (*), comma, or colon.
	TA_SRVID: 1 <= <i>num</i> < 30,001 Unique (within the server group) server identification number.
	TA_GRPNO: 1 <= <i>num</i> < 30,001 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: <i>LMID</i> Current logical machine on which the server is running.
	TA_PID: 1 = <i>num</i> 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: <i>string</i> [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. The value of *string* may also be represented in either of the following 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 string `#. #. #. #` is the dotted decimal format in which each `#` represents a decimal number in the range 0 to 255. *Port_number* is a decimal number in the range 0 to 65535.

Note: Some port numbers may be reserved for the underlying transport protocols (such as TCP/IP) used by your system. Check the documentation for your transport protocols to find out which numbers, if any, are reserved on your system.

`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 `0x0002MMMMddddddd` and WSH network address is `0x00021111ffffff` then the well known network address will be `0x00021111ddddddd`. 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()`. See the Customization section and the `buildwsh(1)` reference page for more details.

TA_MINHANDLERS: $0 \leq num < 256$

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: $0 \leq num < 32,767$

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 minimum number of handlers.

TA_MULTIPLEX: $1 \leq num < 32,767$

Maximum number of clients that are to be supported by any one handler process concurrently.

TA_MINENCRYPTBITS: { 0 | 40 | 56 | 128 }

Specifies the minimum level of encryption required when connecting to the BEA Tuxedo system. 0 means no encryption, while 40, 56, and 128 specify the encryption key length (in bits). If this minimum level of encryption cannot be met, link establishment fails. The default is 0.

Note: The link-level encryption value of 40 bits is provided for backward compatibility.

TA_MAXENCRYPTBITS: { 0 | 40 | 56 | 128 }

Specifies the maximum level of encryption that can be negotiated when connecting to the BEA Tuxedo system. 0 means no encryption, while 40, 56, and 128 specify the encryption length (in bits). The default is 128.

Note: The link-level encryption value of 40 bits is provided for backward compatibility.

TA_MINWSPORT: $0 \leq num < 65,535$

The lower end of the range of available port numbers that may be allocated to WSH processes by this listener.

TA_MAXWSPORT: $0 \leq num < 65,535$

The upper end of the range of available port numbers that may be allocated to WSH processes by this listener.

TA_MAXIDLETIME: $0 \leq num < 35,204,650$

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: $1 \leq num < 32,767$

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 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 (that is, 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, that is, 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_KEEPLIVE: "{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 timeout.

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: $1 \leq num < 10,000$

Specifies when this server should be booted (`tmbboot(1)`) or shut down (`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 `tmbboot()` in increasing order of sequence number and shutdown by `tmshutdown()` in decreasing order. Run-time modifications to this attribute affect only `tmbboot()` and `tmshutdown()` and will affect the order in which running servers may be shutdown by a subsequent invocation of `tmshutdown()`.

TA_CURHANDLERS: $0 \leq num$

Number of currently active handlers associated with this WSL.

TA_HWHANDLERS: $0 \leq num$

Maximum number of currently active handlers associated with this WSL at any one time.

TA_WSPROTO: $0 \leq num$

The BEA Tuxedo 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()`, `tpgetrply()` and `tpdequeue()`) 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()` and `tpgetrply()` 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 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 = tmalloc("FML32",
NULL, 1000);
obuf = tmalloc("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\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 */
}
/* 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 = tmalloc("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 = tmalloc("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 = tmalloc("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), **Introduction to FML Functions**, Fadd, Fadd32(3fml), Fchg, Fchg32(3fml), Ffind, Ffind32(3fml), MIB(5), TM_MIB(5)

Setting Up a BEA Tuxedo Application

Administering a BEA Tuxedo Application at Run Time

Programming a BEA Tuxedo Application Using C

Programming a BEA Tuxedo Application Using FML

WSL(5)

Name	WSL - Workstation Listener server
Synopsis	<pre> WSL SRVGRP="<i>identifier</i>" SRVID="<i>number</i>" CLOPT="[-A] [<i>servopts options</i>] -- -n <i>netaddr</i> [-d <i>device</i>] [-w <i>WSHname</i>] [-t <i>timeout-factor</i>] [-T <i>Client-timeout</i>] [-m <i>minh</i>] [-M <i>maxh</i>] [-x <i>mpx-factor</i>] [-p <i>minwshport</i>] [-P <i>maxwshport</i>] [-I <i>init-timeout</i>] [-c <i>compression-threshold</i>] [-k <i>compression-threshold</i>] [-K {<i>client handler both none</i>}] [-z <i>bits</i>] [-Z <i>bits</i>] [-H <i>external-netaddr</i>][-N <i>network-timeout</i>]" </pre>
Description	<p>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 <code>SERVERS</code> section. The associated command line options are used to parameterize the processing of the workstation listener and workstation handlers.</p> <p>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.</p> <p>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.</p> <p>The following WSL-specific command line options are available and may be listed after the double-dash (--) in the <code>CLOPT</code> parameter.</p>

`-n netaddr`

Specifies the complete network address to be used by the WSL process as its listening address. This is the only required parameter.

The listening address for a WSL is the means by which it is contacted by Workstation client processes participating in the application. If *netaddr* (which may contain from 1 to 78 characters) 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. The address may also be represented in either of the following forms:

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

The string *#. #. #. #* is the dotted decimal format in which each *#* represents a decimal number in the range 0 to 255. The value of *port_number* is a decimal number in the range 0 to 65535.

Note: Some port numbers may be reserved for the underlying transport protocols (such as TCP/IP) used by your system. Check the documentation for your transport protocols to find out which numbers, if any, are reserved on your system.

`[-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()`. 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 BEA Tuxedo 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*.

Note: Some port numbers may be reserved for the underlying transport protocols (such as TCP/IP) used by your system. Check the documentation for your transport protocols to find out which numbers, if any, are reserved on your system.

`[-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 Release 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 | 56 | 128]]`

This option specifies the minimum level of encryption required when a network link is being established between a Workstation client and the Workstation Handler. 0 means no encryption, while 40, 56, and 128 specify the length (in bits) of the encryption key. If this minimum level of encryption cannot be met, link establishment fails. The default is 0. This option is available only if BEA Tuxedo Security (either International, or US and Canada) is installed.

Note: The link-level encryption value of 40 bits is provided for backward compatibility.

`[-z [0 | 40 | 56 | 128]]`

This option specifies the maximum level of encryption allowed when a network link is being established between a Workstation client and the Workstation Handler. 0 means no encryption, while 40, 56, and 128 specify the length (in bits) of the encryption key. The default is 128. This option is available only if BEA Tuxedo Security (either International, or US and Canada) is installed.

Note: The link-level encryption value of 40 bits is provided for backward compatibility.

`[-H external-netaddr]`

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 `0x0002MMMMddddddd` and WSH network address is `0x00021111ffffff` 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.

`[-N network-timeout]`

The network-timeout option will establish a wait period, in seconds, for any Tuxedo operation by the Workstation client that receives data from the network. If the period is exceeded, the operation will fail and the client will be disconnected from the application. A value of 0 (zero) indicates no time-out; this is the default. Note: setting this value too low may cause an unacceptably high number of disconnects.

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 BEA Tuxedo Release 4.2 or later node.

Examples

```
*SERVERS
WSL SRVGRP="WSLGRP" SRVID=1000 RESTART=Y GRACE=0
  CLOPT="-A -- -n 0x0002fffffaaaaaaaaa -d /dev/tcp"
WSL SRVGRP="WSLGRP" SRVID=1001 RESTART=Y GRACE=0
  CLOPT="-A -- -n 0x0002aaaaaffffffff -d /dev/tcp -H 0x0002MMMMddddddd"
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)

Setting Up a BEA Tuxedo Application

Administering a BEA Tuxedo Application at Run Time

Programming a BEA Tuxedo Application Using C