



BEA Tuxedo®

File Formats, Data Descriptions, MIBs, and System Processes Reference

Version 10.0
Document Released: September 28, 2007

Contents

Section 5 - File Formats, Data Descriptions, MIBs, and System Processes Reference

Introduction to Tables and Files	4
ACL_MIB(5).	5
T_ACLGROUP Class Definition	6
T_ACLPERM Class Definition	8
T_ACLPRINCIPAL Class Definition.	10
ACL_MIB(5) Additional Information	13
APPQ_MIB(5)	16
T_APPQ Class Definition.	18
T_APPQMSG Class Definition	25
T_APPQSPACE Class Definition.	31
T_APPQTRANS Class Definition	43
APPQ_MIB(5) Additional Information	45
AUTHSVR(5).	52
SECURITY USER_AUTH.	52
SECURITY ACL or MANDATORY_ACL.	54
AUTHSVR Additional Information	55
compilation(5).	57
DMADM(5)	61
DMCONFIG(5).	63
DM_LOCAL Section	67

DM_REMOTE Section	72
DM_EXPORT Section	77
DM_IMPORT Section	78
DM_RESOURCES	81
DM_ROUTING Section	81
DM_ACCESS_CONTROL Section	84
DM_TDOMAIN Section	85
DMCONFIG(5) Additional Information	93
DM_MIB(5)	97
T_DM_ACL Class Definition	101
T_DM_CONNECTION Class Definition	103
T_DM_EXPORT Class Definition	106
T_DM_IMPORT Class Definition	109
T_DM_LOCAL Class Definition	113
T_DM_OSITP Class Definition	121
T_DM_OSITPX Class Definition	124
T_DM_PASSWORD Class Definition	129
T_DM_PRINCIPAL_MAP Class Definition	131
T_DM_REMOTE Class Definition	133
T_DM_RESOURCES Class Definition	138
T_DM_ROUTING Class Definition	139
T_DM_RPRINCIPAL Class Definition	143
T_DM_SNACRM Class Definition	145
T_DM_SNALINK Class Definition	147
T_DM_SNASTACK Class Definition	150
T_DM_TDOMAIN Class Definition	152
T_DM_TRANSACTION Class Definition	161
DM_MIB(5) Additional Information	166

EVENTS(5)	167
EVENT_MIB(5).....	173
T_EVENT_CLIENT Class Definition	174
T_EVENT_COMMAND Class Definition	176
T_EVENT_QUEUE Class Definition	178
T_EVENT_SERVICE Class Definition	181
T_EVENT_USERLOG Class Definition	183
EVENT_MIB(5) Additional Information	185
factory_finder.ini(5)	186
Error, Ferror32(5)	190
field_tables(5).....	192
GAUTHSVR(5)	196
SECURITY USER_AUTH	196
SECURITY ACL or MANDATORY_ACL.....	197
GAUTHSVR Additional Information	197
GWADM(5)	198
GWTDOMAIN(5)	200
ISL(5)	201
KAUTHSVR(5)	210
langinfo(5)	213
LAUTHSVR(5)	217
SECURITY USER_AUTH	218
SECURITY ACL or MANDATORY_ACL.....	218
LAUTHSVR Additional Information	218
METAREPOS(5)	219
MIB(5)	234
Usage	242
T_CLASS Class Definition	253

T_CLASSATT Class Definition	255
MIB(5) Additional Information	259
nl_types(5)	263
servopts(5)	264
TM_MIB(5)	270
T_BRIDGE Class Definition	273
T_CLIENT Class Definition	277
T_CONN Class Definition	285
T_DEVICE Class Definition	288
T_DOMAIN Class Definition	290
T_FACTORY MIB	309
T_GROUP Class Definition	310
T_IFQUEUE Class	319
T_INTERFACE Class	323
T_MACHINE Class Definition	330
T_MSG Class Definition	350
T_NETGROUP Class Definition	352
T_NETMAP Class Definition	354
T_QUEUE Class Definition	357
T_ROUTING Class Definition	362
T_SERVER Class Definition	366
T_SERVERCTXT Class Definition	384
T_SERVICE Class Definition	387
T_SVCGRP Class Definition	392
T_TLISTEN Class Definition	399
T_TLOG Class Definition	400
T_TRANSACTION Class Definition	403
T_ULONG Class Definition	407

TM_MIB(5) Additional Information	411
TMFFNAME(5)	421
TMIFRSVR(5)	424
TMMETADATA(5)	425
TMQFORWARD(5)	427
TMQUEUE(5)	432
TMSYSEVT(5)	436
tmtrace(5)	438
TMUSREVT(5)	444
tperrno(5)	446
tpurcode(5)	450
tuxenv(5)	452
tuxtypes(5)	463
typesw(5)	471
UBBCONFIG(5)	473
RESOURCES Section	476
MACHINES Section	488
GROUPS Section	497
NETGROUPS Section	502
NETWORK Section	503
SERVERS Section	505
SERVICES Section	511
INTERFACES Section	514
ROUTING Section	516
UBBCONFIG(5) Additional Information	519
viewfile(5)	524
WS_MIB(5)	531
T_WSH Class Definition	532

T_WSL Class Definition	537
WS_MIB(5) Additional Information	545
WSL(5).	551

Section 5 - File Formats, Data Descriptions, MIBs, and System Processes Reference

Table 1 BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes

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
DM_MIB(5)	Management Information Base for Domains
EVENTS(5)	List of system-generated events
EVENT_MIB(5)	Management Information Base for EventBroker
factory_finder.ini(5)	FactoryFinder Domains configuration file
Ferror, Ferror32(5)	FML error codes
field_tables(5)	FML mapping files for field names

Table 1 BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes (Continued)

Name	Description
GAUTHSVR(5)	General LDAP-based authentication server
GWADM(5)	Domains gateway administrative server
GWTDOMAIN(5)	TDomain gateway process
ISL(5)	Enables access to BEA Tuxedo objects by remote BEA Tuxedo clients using IIOP.
KAUTHSVR(5)	Kerberos-based Tuxedo authorization server
langinfo(5)	Language information constants
LAUTHSVR(5)	WebLogic Server embedded LDAP-based authentication server
METAREPOS(5)	Tuxedo service metadata repository buffer format
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
TMFFNAME(5)	Server that runs the FactoryFinder and NameManager services
TMIFRSVR(5)	The Interface Repository server
TMMETADATA(5)	Tuxedo service metadata repository server
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

Table 1 BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes (Continued)

Name	Description
<code>tuxenv(5)</code>	List of environment variables in the BEA Tuxedo system
<code>tuxtypes(5)</code>	Buffer type switch; descriptions of buffer types provided by the BEA Tuxedo system
<code>typesw(5)</code>	Buffer type switch structure; parameters and routines needed for each buffer type
<code>UBBCONFIG(5)</code>	Text version of a BEA Tuxedo configuration file
<code>viewfile(5)</code>	Source file for view descriptions
<code>WS_MIB(5)</code>	Management Information Base for Workstation
<code>WSL(5)</code>	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 <fm132.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. For additional information pertaining to all `ACL_MIB(5)` class definitions, see [“ACL_MIB\(5\) Additional Information” on page 13](#).

`ACL_MIB(5)` consists of the following classes.

Table 2 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 `udataobj/tpadm` relative to the root directory of the BEA Tuxedo system software installed on the system. The directory `#{TUXDIR}/udataobj` should be included by the application in the colon-separated list specified by the `FLDTBLDIR` environment variable and the field table name `tpadm()` should be included in the comma-separated list specified by the `FIELDTBLS` environment variable.

Limitations

Access to the header files and field tables for this MIB is 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

Table 3 ACL_MIB(5): T_ACLGROUP Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_GROUPNAME(<i>r</i>)(<i>*</i>)	string	rU-----	<i>string</i> [1..30]	N/A
TA_GROUPID(<i>k</i>)	long	rw-----	0 <= <i>num</i> < 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.
<i>unset</i>	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

Table 4 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

Table 5 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.

ACL_MIB(5) Additional Information

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, 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 ATMI Application Using C

Programming a BEA Tuxedo ATMI Application Using FML

APPQ_MIB(5)

Name

APPQ_MIB—Management Information Base for /Q

Synopsis

```
#include <fm132.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. For additional information pertaining to all APPQ_MIB(5) class definitions, see “[APPQ_MIB\(5\) Additional Information](#)” on page 45.

APPQ_MIB(5) consists of the following classes.

Table 6 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_FORCEDDELETE

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 user log.

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 (semicolon-separated list on Windows and colon-separated list otherwise) specified by the `FLDTBLDIR` environment variable and the field table name `tpadm` should be included in the comma-separated list specified by the `FLDTBLS` 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, 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()`, all `T_APPQ` objects within the specified queue space will be retrieved.

Attribute Table

Table 7 APPQ_MIB(5): T_APPQ Class Definition Attribute Table

Attribute ^a	Type	Permissions	Values	Default
TA_APPQNAME(k)(r)(*)	string	ru-r--r--	<i>string</i> [1..15]	N/A
TA_APPQSPACENAME(k)(r)(*)	string	ru-r--r--	<i>string</i> [1..15]	N/A
TA_QMCONFIG(k)(r)(*)	string	ru-r--r--	<i>string</i> [1..78]	N/A
TA_LMID(k)(r)(*) ^b	string	ru-r--r--	<i>string</i> [1..30]	N/A
TA_STATE ^c	string	rw-r--r--	GET: "VAL" SET: "{NEW INV}"	N/A N/A
TA_APPQORDER ^d	string	rw-r--r--	{PRIO TIME LIFO FIFO EXPIR}	FIFO
TA_DEFEXPIRATIONTIME	string	rw-r--r--	{+seconds NONE}	N/A
TA_DEFDELIVERYPOLICY	string	rw-r--r--	{PERSIST NONPERSIST}	PERSIST
TA_CMD	string	rw-r--r--	<i>shell-command</i> <i>-string</i> [0..127] ^e	" "
TA_CMDHW	string	rw-r--r--	0 <= num [bBm%]	100%
TA_CMDLW	string	rw-r--r--	0 <= num [bBm%]	0%
TA_CMDNONPERSIST	string	rw-r--r--	<i>shell-command-string</i> [0..127] ^e	" "
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

Table 7 APPQ_MIB(5): T_APPQ Class Definition Attribute Table (Continued)

Attribute ^a	Type	Permissions	Values	Default
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 ^f
(r)—required field for object creation
(*)—required SET key field

^a All attributes of class T_APPQ are local attributes.

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

^c All 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.

^d TA_APPQORDER cannot be modified after the application queue is created.

^e Maximum string length for this attribute is 78 bytes for BEA Tuxedo 8.0 or earlier.

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

Attribute Semantics

TA_APPQNAME: *string*[1..15]

Name of the application queue.

TA_APPQSPACENAME: *string*[1..15]

Name of the application queue space containing the application queue.

TA_QMCONFIG: *string*[1..78]

Absolute pathname of the file or device where the application queue space is located.

TA_LMID: *string*[1..30] (no comma)

Identifier of the logical machine where the application queue space is located.

TA_STATE:

GET: {VALid}

A GET operation retrieves information about the selected application queues. The following list describes the meaning of the TA_STATE attribute returned in response to a GET request.

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

SET: {NEW | INValid}

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

NEW	Create a new queue in the specified queue space. The queue is left in state VALid following successful creation.
-----	--

INValid	Delete the specified queue. The queue must be in state VALid to be deleted. If the queue space has processes attached to it (that is, it is in the ACTive state), the queue will not be deleted unless the TA_FLAGS attribute includes the QMIB_FORCEDELETE flag. In addition, if the queue has messages in it, it will not be deleted unless QMIB_FORCEPURGE is specified. Successful return leaves the object in the INValid state.
---------	---

unset	Modify an application queue. Successful return leaves the state unchanged.
-------	--

TA_APPQORDER:

The order in which messages in the queue are to be processed. Legal values are PRIO, 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..127]

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.

For BEA Tuxedo 8.0 or earlier, the maximum string length for the TA_CMD attribute is 78 bytes.

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 key letters and the key letters 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, 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..127]

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.

For BEA Tuxedo 8.0 or earlier, the maximum string length for the `TA_CMDNONPERSIST` attribute is 78 bytes.

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 key letters. The key letters 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, 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_APPQSPACE`, `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()`, all T_APPQMSG objects in all queues of the specified queue space will be retrieved.

Attribute Table

Table 8 APPQ_MIB(5): T_APPQMSG Class Definition Attribute Table

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

Table 8 APPQ_MIB(5): T_APPQMSG Class Definition Attribute Table (Continued)

Attribute ^a	Type	Permissions	Values	Default
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: *string*[1..32]

A unique identifier for the queue message, which can be used to select the message for GET or SET operations. No significance should be placed on this value beyond using it for equality comparisons.

TA_APPQNAME: *string*[1..15]

Name of the application queue in which the message is stored.

TA_APPQSPACENAME: *string*[1..15]

Name of the application queue space containing the message.

TA_QMCONFIG: *string*[1..78]

Absolute pathname of the file or device where the application queue space is located.

TA_LMID: *string*[1..30] (no comma)

Identifier of the logical machine where the application queue space is located.

TA_STATE:

GET: {VALid}

A GET operation retrieves information about the selected messages. The following list describes the meaning of the TA_STATE attribute returned in response to a GET request.

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.

INVALid	The message is deleted from its queue space. The message must be in state VALid before attempting this operation. Successful return leaves the object in the INVALid state.
---------	---

<i>unset</i>	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 <= num <= 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

Table 9 APPQ_MIB(5): T_APPQSPACE Class Definition Attribute Table

Attribute ^a	Type	Permissions	Values	Default
TA_APPQSPACENAME(k)(r)(*)	string	ru-r--r--	<i>string</i> [1..15]	N/A
TA_QMCONFIG(k)(r)(*)	string	ru-r--r--	<i>string</i> [1..78]	N/A
TA_LMID(k)(r)(*) ^b	string	ru-r--r--	<i>string</i> [1..30]	N/A
TA_STATE(k) ^c	string	rw-rw-r--	GET: "{ INA INI OPE ACT }" SET: "{ NEW OPE CLE INV }"	N/A N/A
TA_BLOCKING	long	rw-r--r--	0 <= <i>num</i>	16
TA_ERRORQNAME	string	rw-r--r--	<i>string</i> [0..15]	" "
TA_FORCEINIT	string	rw-r--r--	{ Y N }	N
TA_IPCKEY(r)	long	rw-r--r--	32769 <= <i>num</i> <= 262143	N/A
TA_MAXMSG(r)	long	rw-r--r--	0 <= <i>num</i>	N/A
TA_MAXPAGES(r)	long	rw-r--r--	0 <= <i>num</i>	N/A
TA_MAXPROC(r)	long	rw-r--r--	0 <= <i>num</i>	N/A
TA_MAXQUEUES(r) ^d	long	rw-r--r--	0 <= <i>num</i>	N/A
TA_MAXTRANS(r)	long	rw-r--r--	0 <= <i>num</i>	N/A
TA_MAXACTIONS	long	rw-r--r--	0 <= <i>num</i>	0
TA_MAXHANDLES	long	rw-r--r--	0 <= <i>num</i>	0
TA_MAXOWNERS	long	rw-r--r--	0 <= <i>num</i>	0
TA_MAXTMPQUEUES	long	rw-r--r--	0 <= <i>num</i>	0
TA_MAXCURSORS	long	rw-r--r--	0 <= <i>num</i>	0
TA_MEMNONPERSIST	string	rw-r--r--	0 <= <i>num</i> [bB]	0
TA_MEMFILTERS	long	rw-r--r--	0 <= <i>num</i>	0
TA_MEMOVERFLOW	long	rw-r--r--	0 <= <i>num</i>	0
TA_MEMSYSTEMRESERVED	long	r--r--r--	0 <= <i>num</i>	N/A
TA_MEMTOTALALLOCATED	long	r--r--r--	0 <= <i>num</i>	N/A

Table 9 APPQ_MIB(5): T_APPQSPACE Class Definition Attribute Table (Continued)

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

Name of the application queue space.

TA_QMCONFIG: **string[1..78]**

Absolute pathname of the file or device where the application queue space is located.

TA_LMID: *string*[1..30] (no comma)

Identifier of the logical machine where the application queue space is located.

TA_STATE:

GET: {INActive | INItializing | OPEN | ACTive}

A GET operation retrieves information about the selected application queue space. The following list describes the meaning of the TA_STATE attribute returned in response to a GET request.

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 <code>qadmin(1)</code> , 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.

<code>INValid</code>	Delete the queue space. Unless the <code>QMIB_FORCEDELETE</code> flag is passed, an error is reported if the state is <code>ACTive</code> or if messages exist on any queues in the queue space. Successful return leaves the object in the <code>INValid</code> state. When successful, all non-persistent messages are permanently lost.
<code>unset</code>	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 Tuxedo infrastructure 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 Tuxedo infrastructure 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 Tuxedo infrastructure 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 Tuxedo infrastructure 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 Tuxedo infrastructure. 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 Tuxedo infrastructure for message selection in dequeuing and cursor operations. Filters may be specified using various grammars but are compiled into an BEA Tuxedo infrastructure 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 an request using `tpcall()`, 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

Table 10 APPQ_MIB(5): T_APPQTRANS Class Definition Attribute Table

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

Attribute Semantics

TA_XID: string[1..78]

Transaction identifier as returned by `tx_info()` and mapped to a string representation. The data in this field should not be interpreted directly by the user except for equality comparison.

TA_APPQSPACENAME: string[1..15]

Name of the application queue space associated with the transaction.

TA_QMCONFIG: string[1..78]

Absolute pathname of the file or device where the application queue space is located.

TA_LMID: string[1..30] (no comma)

Identifier of the logical machine where the application queue space is located.

TA_STATE:

GET: {ACTIVE | ABORTONLY | ABORTED | COMCALLED | READY | DECIDED | HABORD | HCOMMIT}

A GET operation retrieves run-time information about the selected transactions. The following list describes the meaning of the TA_STATE attribute returned in response to a GET request. All states are ACTIVE equivalent for purposes of permissions checking.

ACTIVE	The transaction is active.
ABORTONLY	The transaction has been identified for rollback.
ABORTED	The transaction has been identified for rollback and rollback has been initiated.
COMCALLED	The initiator of the transaction has called <code>tpcommit()</code> and the first phase of two-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 two-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.

APPQ_MIB(5) Additional Information

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, 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, 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, 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", rqbuf, 0, rpbuf, rplen, flags);
```

See [MIB\(5\)](#) for additional information.

Field Tables

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

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

Header Files

The following header files are needed.

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

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

/* Make the request; see above */

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

/* Build the request to create the queue space */
Fchg32(rqbuf, TA_OPERATION, 0, "SET", 0);
Fchg32(rqbuf, TA_CLASS, 0, "T_APPQSPACE", 0);
Fchg32(rqbuf, TA_STATE, 0, "NEW", 0);
Fchg32(rqbuf, TA_APPQSPACENAME, 0, "QSPACE1", 0);
Fchg32(rqbuf, TA_QMCONFIG, 0, "/dev/q/dsk001", 0);
Fchg32(rqbuf, TA_LMID, 0, "SITE1", 0);
Fchg32(rqbuf, TA_ERRORQNAME, 0, "errque", 0);
ipckey = 123456;
Fchg32(rqbuf, TA_IPCKEY, 0, (char *)ipckey, 0);
maxmsg = 100;
Fchg32(rqbuf, TA_MAXMSG, 0, (char *)maxmsg, 0);
maxpages = 200;
Fchg32(rqbuf, TA_MAXPAGES, 0, (char *)maxpages, 0);
maxproc = 50;
Fchg32(rqbuf, TA_MAXPROC, 0, (char *)maxproc, 0);
maxqueues = 10;

```

```

Fchg32(rqbuf, TA_MAXQUEUES, 0, (char *)maxqueues, 0);
maxtrans = 100;
Fchg32(rqbuf, TA_MAXTRANS, 0, (char *)maxtrans, 0);

/* Make the request; see above */

```

Add a Queue to an Application Queue Space

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

```

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

/* Make the request; see above */

```

List Application Queue Spaces Known to the Application

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

Listing 1 List Application Queue Spaces Known to the Application

```

/* Build the request to retrieve all TMS_QM groups */
Fchg32(rqbuf, TA_OPERATION, 0, "GET", 0);
Fchg32(rqbuf, TA_CLASS, 0, "T_GROUP", 0);
Fchg32(rqbuf, TA_TMSNAME, 0, "TMS_QM", 0);
fldid1 = TA_OPENINFO;
fldid2 = TA_LMID;
Fchg32(rqbuf, TA_FILTER, 0, (char *)fldid1, 0);
Fchg32(rqbuf, TA_FILTER, 0, (char *)fldid2, 1);

/* Make the request, assuming we are joined to the application */

```

```

rval = tpcall(".TMIB", rqbuf, 0, rdbuf, rplen, flags);

/* For each TMS_QM group, build the request to retrieve its queue space */
rval = Fget32(*rdbuf, 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 Windows <resource-mgr>:<qmconfig>;<appqspacename> */
    rval = Fget32(rdbuf, 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] = '\e0'; /* NULL-terminate qmconfig */
    appqspacename++; /* bump past the NULL */

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

    /* Get the LMID (assume no migration for this group) */
    rval = Fget32(rdbuf, 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\(3c\)](#), [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 ATMI Application Using C

APPQ_MIB(5)

Programming a BEA Tuxedo ATMI 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, 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.

For additional information pertaining to AUTHSVR, see [“AUTHSVR Additional Information” on page 55](#).

SECURITY USER_AUTH

If SECURITY is set to USER_AUTH, 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 pathname 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 filename 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 username 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 username may be a wildcard. 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`, 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 username and client name. The entry must match exactly on the username. 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 wildcard. 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.

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

AUTHSVR Additional Information

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

[tpaddusr\(1\)](#), [tpusradd\(1\)](#), [UBBCONFIG\(5\)](#)

Setting Up a BEA Tuxedo Application

Administering a BEA Tuxedo Application at Run Time

Programming a BEA Tuxedo ATMI Application Using C

compilation(5)

Name

`compilation`—Instructions for compilation of BEA Tuxedo ATMI 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 *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

In terms of header file sequence, UNIX header files should always be included before any BEA Tuxedo system header files. Commonly used UNIX header files are `stdio.h` and `ctype.h`.

Environment Variables

The following environment variables should be set and exported:

TUXDIR

Specifies the topmost directory in which the BEA Tuxedo system software resides.

PATH

Should include `$TUXDIR/bin`.

ULOGPFIX

Prefix of the filename of the central event log; by default, the value of ULOGPFIX is ULOG.

If ...	Then you must first set and export the following environment variables
	...
You want to run <ul style="list-style-type: none"><code>buildclient(1)</code><code>buildserver(1)</code>	<ul style="list-style-type: none">TUXDIR—always required for servers; also required for native clientsCC—if you want to use a non-default compilerCFLAGS—if you want to specify flags to be passed to the compiler
A default or validation routine references FML fields	<ul style="list-style-type: none">FIELDTBLS—a comma-separated list of field table filesFLDTBLDIR—a colon-separated list of directories to search for the FIELDTBLS
You want to execute a server	TUXCONFIG—full pathname of the binary configuration file (default is the current directory)
<ul style="list-style-type: none">Security is turned on in your applicationYou are going to supply input indirectly (that is, from a source other than standard input) for any of the following system-supplied clients: <code>tmadmin(1)</code>, <code>tmconfig</code> or <code>wtmconfig</code> (see <code>tmconfig</code>, <code>wtmconfig(1)</code>), or <code>ud</code> or <code>wud</code> (see <code>ud</code>, <code>wud(1)</code>)	<ul style="list-style-type: none">APP_PW—application passwordUSR_PW—user password
You want to execute a Workstation client	<ul style="list-style-type: none">WSENVFILE—file containing environment variable settingsWSDEVICE—network device to use for connectionWSTYPE—workstation machine type

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

On this platform . . .	Set the following environment variable . . .
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

In terms of 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.

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 viewfiles; the default is the current directory.

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

Description

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

DMADM is described in the SERVERS section of the UBBCONFIG file as a server running within a group, 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”).

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

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

Portability

DMADM is supported as a BEA Tuxedo system-supplied server on all supported server 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

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.

```
#
*GROUPS
DMADMGRP LMID=mach1 GRPNO=1
gwgrp    LMID=mach1 GRPNO=2
```

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

See Also

[dmadmin\(1\)](#), [tmboot\(1\)](#), [DMCONFIG\(5\)](#), [GWADM\(5\)](#), [servopts\(5\)](#), [UBBCONFIG\(5\)](#)

Setting Up a BEA Tuxedo Application

Administering a BEA Tuxedo Application at Run Time

Using the BEA Tuxedo TOP END Domain Gateway with ATMI Applications

DMCONFIG(5)

Name

DMCONFIG—Text version of a Domains configuration file

Description

A Domains configuration is a set of two or more *domains* (business applications) that can communicate and share services with the help of the BEA Tuxedo Domains component. How multiple domains are connected and which services they make accessible to each other are defined in a Domains configuration file for each BEA Tuxedo domain participating in the Domains configuration. The text version of a Domains configuration file is known as the DMLoadcf file, although the configuration file may have any name as long as the content of the file conforms to the format described on this reference page.

The DMLoadcf file is parsed and loaded into a binary version, called BDMCONFIG, by the `dmloadcf(1)` utility. As with DMLoadcf, the BDMCONFIG file may be given any name; the actual name is the device or system filename specified in the BDMCONFIG environment variable. One BDMCONFIG file is required for each Tuxedo domain participating in a Domains configuration.

The DMLoadcf and BDMCONFIG files are analogous to the UBBCONFIG and TUXCONFIG files used to define a BEA Tuxedo domain. For a description of the UBBCONFIG and TUXCONFIG files, see [UBBCONFIG\(5\)](#).

For additional information pertaining to the DMLoadcf file, including examples, see “[DMCONFIG\(5\) Additional Information](#)” on page 93. For a detailed description of the BEA Tuxedo Domains component for both ATMI and CORBA environments, see [Using the BEA Tuxedo Domains Component](#).

Definitions

A BEA Tuxedo domain is defined as the environment described in a single TUXCONFIG file. In BEA Tuxedo terminology, a *domain* is the same as an *application*—a business application.

There is one *Domains administrative server* (DMADM) process running in each BEA Tuxedo domain involved in a Domains configuration. The DMADM is the administrative server for all domain gateway groups running in a particular BEA Tuxedo domain.

A *domain gateway group* consists of a BEA Tuxedo system *gateway administrative server* (GWADM) process and a BEA Tuxedo system domain gateway process.

A BEA Tuxedo system *domain gateway* process provides communication services with a specific type of transaction processing (TP) domain; for example, the GWTDOMAIN process enables BEA

Tuxedo applications to communicate with other BEA Tuxedo applications. A domain gateway relays requests to another domain and receives replies.

A *local domain access point* is a user-specified logical name representing a set of services of the BEA Tuxedo domain that is made available to other domains (remote domains). A local domain access point maps to a domain gateway group; both terms are used as synonyms.

A *remote domain access point* is a user-specified logical name representing a set services of a remote domain that is made available to the local domain. The remote domain may be another BEA Tuxedo application or an application running on another TP system.

A *remote service* is a service provided by a remote domain that is made available to the local domain through a remote domain access point and a local domain access point.

A *local service* is a service of the local domain that is made available to remote domains through a local domain access point.

Configuration File Purpose

You use a `DMCONFIG` file to:

- Define the local domain access points through which application clients on a remote domain can access services on the local domain
- Define the local services available through each local domain access point
- Define the remote domain access points through which application clients on the local domain can access services on a remote domain
- Define the remote services available through each remote domain access point
- Map local domain access points and remote domain access points to specific domain gateway groups and network addresses

Configuration File Format

The `DMCONFIG` file is made up of the following specification sections:

- `DM_LOCAL` (also known as `DM_LOCAL_DOMAINS`)
- `DM_REMOTE` (also known as `DM_REMOTE_DOMAINS`)
- `DM_EXPORT` (also known as `DM_LOCAL_SERVICES`)
- `DM_IMPORT` (also known as `DM_REMOTE_SERVICES`)
- `DM_RESOURCES`

- [DM_ROUTING](#)
- [DM_ACCESS_CONTROL](#)
- [DM_TDOMAIN](#) (section for domain gateways of type TDOMAIN)
- `DM_dom`, where *dom* may be any of the following sections for other domain gateway types: SNACRM, SNASTACKS, SNALINKS, OSITP, OSITPX.

Lines in a DMCNFIG file beginning with an asterisk (*) indicate the beginning of a specification section. Each such line contains the name of the section immediately following the *. The asterisk is required when specifying a section name. The `DM_LOCAL` section must precede the `DM_REMOTE` section.

This reference page describes how to configure a domain gateway of type TDOMAIN (the TDomain gateway), which is implemented by the `GWTDOMAIN` gateway process. For information about how to configure a `SNAX`, `OSITP`, or `OSITPX` domain gateway, see [BEA eLink Documentation at http://e-docs.bea.com/mlink/mainfram/mainfram.htm](http://e-docs.bea.com/mlink/mainfram/mainfram.htm).

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 component 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 bytes (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

For BEA Tuxedo release 7.1 or later, the Domains MIB uses improved class and attribute terminology to describe the interaction between local and remote domains. The improved terminology has been applied to the `DMCONFIG(5)` reference page, section names, parameter names, and error messages, and to the `DM_MIB(5)` reference page, classes, and error messages.

For backwards compatibility, aliases are provided between the `DMCONFIG` terminology used prior to BEA Tuxedo 7.1 and the improved Domains MIB terminology. For BEA Tuxedo release 7.1 or later, both versions of `DMCONFIG` terminology are accepted. The following table shows the mapping of the previous and improved terminology for the `DMCONFIG` file.

Previous Terminology		Improved Terminology	
Section Name	Parameter Name	Section Name	Parameter Name
DM_LOCAL_DOMAINS		DM_LOCAL	
DM_REMOTE_DOMAINS		DM_REMOTE	
	DOMAINID		ACCESSPOINTID
	MAXRDOM		MAXACCESSPOINT
	MAXRDTRAN		MAXRAPTRAN
DM_LOCAL_SERVICES		DM_EXPORT	
DM_REMOTE_SERVICES		DM_IMPORT	
	LDOM		LACCESSPOINT
	RDOM		RACCESSPOINT

For BEA Tuxedo release 7.1 or later, the `dmunloadcf` command generates by default a DMCONFIG file that uses the improved domains terminology. Use the `-c` option to print a DMCONFIG file that uses the previous domains terminology. For example:

```
prompt> dmunloadcf -c > dmconfig_prev
```

DM_LOCAL Section

This section, also known as the `DM_LOCAL_DOMAINS` section, defines one or more local domain access point identifiers and their associated gateway groups. The section must have a local domain access point entry for each active gateway group defined in the `UBBCONFIG` file. Each entry specifies the parameters required for the domain gateway process running in that group.

Entries within the `DM_LOCAL` section have the following form:

```
LocalAccessPoint required_parameters [optional_parameters]
```

where *LocalAccessPoint* is the local domain access point identifier (logical name) that you choose to represent a particular gateway group defined in the `UBBCONFIG` file.

LocalAccessPoint must be unique across the local and remote domains involved in a Domains configuration. As you will see in the description of the `DM_EXPORT` section, you use the local domain access point to associate local services with the gateway group. The local services available through the local domain access point will be available to clients in one or more remote domains.

Required parameters for the DM_LOCAL section

`GWGRP = identifier`

Specifies the name of the domain gateway group (the name provided in the `GROUPS` section of the `TUXCONFIG` file) representing this local domain access point. There is a one-to-one relationship between a local domain access point and a domain gateway group.

`TYPE = identifier`

Specifies the type of domain gateway associated with this local domain access point. `TYPE` can be set to one of the following values: `TDOMAIN`, `SNAX`, `OSITP`, or `OSITPX`.

The `TDOMAIN` value indicates that this local domain access point is associated with a `GWTDOMAIN` gateway instance and therefore can communicate with another BEA Tuxedo application.

The `SNAX` value indicates that this local domain access point is associated with a `GWSNAX` gateway instance and therefore can communicate with another TP domain via the SNA protocol.

The `OSITP` or `OSITPX` value indicates that this local domain access point is associated with a `GWOSITP` gateway instance and therefore can communicate with another TP domain via the OSI TP protocol. The `OSITP` value indicates the use of the OSI TP 1.3 protocol, and the `OSITPX` value indicates the use of the OSI TP 4.0 or later protocol. The `OSITPX` value is supported only by BEA Tuxedo 8.0 or later software.

Domain types must be defined in the `DMTYPE` file: `%TUXDIR%\udataobj\DMTYPE` for Windows or `$TUXDIR/udataobj/DMTYPE` for UNIX.

`ACCESSPOINTID` (also known as `DOMAINID`) = *string*[1..30]

Used to identify the domain gateway group associated with this local domain access point for purposes of security when setting up connections to remote domains. `ACCESSPOINTID` must be unique across all local and remote domain access points.

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”). `ACCESSPOINTID` must be 30 bytes or fewer in length. If the value is a string, it must be 30 characters or fewer (counting the trailing `NULL`).

Optional parameters for the `DM_LOCAL` section

The following optional parameters for the `DM_LOCAL` section describe resources and limits used in the operation of domain gateways:

`AUDITLOG` = *string*[1..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

Specifies the name of the audit log file for this local domain access point. The audit log feature is activated from the `dmadmin(1)` command and records all the operations for this local domain access point. 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` parameter of the `MACHINES` section of the `TUXCONFIG` file.

`BLOCKTIME` = *numeric*

Specifies the maximum wait time allowed for a blocking call for this local domain access point. The value is a multiplier of the `SCANUNIT` parameters specified in the `RESOURCES` section of 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 `RESOURCES` section of the `TUXCONFIG` file. A blocking timeout condition implies that the affected service request has failed.

Be aware that *interdomain* transactions generate blocking timeout conditions when transaction duration exceeds `BLOCKTIME`. That is, for an interdomain transaction, if the

`BLOCKTIME` value is less than (a) the `TRANTIME` timeout value specified in the `SERVICES` section of the `TUXCONFIG` file or (b) the timeout value passed in a `tpbegin()` call to start the transaction, the timeout for the transaction is reduced to the `BLOCKTIME` value. In contrast, for *intradomain* transactions (that is, transactions handled within a single BEA Tuxedo domain), the `BLOCKTIME` value specified in the `RESOURCES` section of the `TUXCONFIG` file has *no* effect on the timeout of an intradomain transaction.

`CONNECTION_POLICY = {ON_DEMAND | ON_STARTUP | INCOMING_ONLY}`

Specifies the conditions under which the domain gateway associated with this local domain access point tries to establish connections to remote domains. Supported values are `ON_DEMAND`, `ON_STARTUP`, or `INCOMING_ONLY`. This parameter applies only to domain gateways of type `TDOMAIN`.

A connection policy of `ON_DEMAND` means that a domain gateway attempts to establish a connection with a remote domain only when requested by either a client request to a remote service or a `dadmin(1)` `connect` command. The default for `CONNECTION_POLICY` is `ON_DEMAND`. 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. If `CONNECTION_POLICY` is set to `ON_STARTUP`, the remote services for a particular remote domain (that is, services advertised by the domain gateway) are advertised only if a connection is successfully established to the remote domain. Thus, if there is no active connection to the remote domain, the remote services are suspended. By default, this connection policy retries failed connections every 60 seconds, but you can specify a different value for this interval using the `RETRY_INTERVAL` parameter. Also, see the `MAXRETRY` parameter.

A connection policy of `INCOMING_ONLY` means that a domain gateway does not attempt an initial connection upon startup and that remote services are initially suspended. The domain gateway is available for incoming connections from remote domains, and remote services are advertised when the domain gateway receives an incoming connection or an administrative connection (using the `dadmin(1)` `connect` command) is made. Connection retry processing is not allowed when the connection policy is `INCOMING_ONLY`.

Note: For domain gateways of type `TDOMAIN` running BEA Tuxedo 8.1 or later software, `CONNECTION_POLICY` can be specified on a per remote domain basis in the `DM_TDOMAIN` section.

`MAXRETRY = {numeric | MAXLONG}`

Specifies the number of times that the domain gateway associated with this local domain access point tries to establish connections to remote domains. This parameter applies only to domain gateways of type `TDOMAIN` and is valid only when the `CONNECTION_POLICY` parameter for this local domain access point is set to `ON_STARTUP`. For other connection policies, automatic retries are disabled.

The minimum value for `MAXRETRY` is 0, and the maximum value is `MAXLONG` (2147483647). `MAXLONG`, the default, indicates that retry processing will be repeated indefinitely, or until a connection is established. Setting `MAXRETRY=0` turns off the automatic retry mechanism.

`RETRY_INTERVAL = numeric`

Specifies the number of seconds that the domain gateway associated with this local domain access point waits between automatic attempts to establish a connection to remote domains. This parameter applies only to domain gateways of type `TDOMAIN` and is valid only when the `CONNECTION_POLICY` parameter for this local domain access point is set to `ON_STARTUP`. For other connection policies, automatic retries are disabled.

The minimum value for `RETRY_INTERVAL` is 0, and the maximum value is 2147483647. The default is 60. If `MAXRETRY` is set to 0, setting `RETRY_INTERVAL` is not allowed.

`CONNECTION_PRINCIPAL_NAME = string[0..511]`

Specifies the connection principal name identifier, which is the principal name for verifying the identity of the domain gateway associated with this local domain access point when establishing a connection to a remote domain. This parameter applies only to domain gateways of type `TDOMAIN` 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 `ACCESSPOINTID` string for this local domain access point.

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

DMTLOGDEV = *string*[1..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

Specifies the BEA Tuxedo filesystem that contains the Domains transaction log (TLOG) for this local domain access point. The TLOG is stored as a BEA Tuxedo system VTOC table on the device. If this parameter is not specified, the domain gateway group associated with this local domain access point is not allowed to process requests in transaction mode. Multiple local domain access points for the same machine can share the same BEA Tuxedo filesystem, but each local domain access point must have its own log (a table in the DMTLOGDEV) named as specified by the DMTLOGNAME parameter.

DMTLOGNAME = *string*[1..30]

Specifies the name of the TLOG for this local domain access point. This name must be unique when the same BEA Tuxedo filesystem (as specified in DMTLOGDEV) is used for several local domain access points. If this parameter is 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 TLOG for this local domain access point. It must be greater than 0 and less than the amount of available space on the BEA Tuxedo filesystem. If this parameter is not specified, the default is 100 pages.

MAXRAPTRAN (also known as MAXRDTRAN) = *numeric*

Specifies the maximum number of domains that can be involved in a transaction for this local domain access point. It must be greater than 0 and less than 32,768. If this parameter is not specified, the default is 16.

MAXTRAN = *numeric*

Specifies the maximum number of simultaneous global transactions allowed for this local domain access point. It must be greater than or equal to 0 and less than or equal to the MAXGTT parameter specified in the RESOURCES section of the TUXCONFIG file. If MAXTRAN is not specified, the default is the value of MAXGTT.

MTYPE = *string*[1..15]

Used for grouping domains so that encoding/decoding of messages can be bypassed between the machine associated with this local domain access point and the machines associated with the remote domain access points. This parameter applies only to domain gateways of type TDOMAIN.

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 and the DM_REMOTE section of a DMCNFIG 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.

SECURITY = {NONE | APP_PW | DM_PW}

Specifies the type of application security to be enforced for this local domain access point. The SECURITY parameter currently has three valid values for domain gateways of type TDOMAIN: NONE, APP_PW, or DM_PW. The value NONE (the default) indicates that no security is used. 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 is defined in the TUXCONFIG file. The value DM_PW indicates that Domains password security is to be enforced when a connection is established from a remote domain; Domains passwords are defined through the `dmadmin(1)` command.

The SECURITY parameter does not apply to domain gateways of type OSITP. For gateways of type OSITPX, the values NONE or DM_PW can be used. For gateways of type SNAX, the values NONE or DM_USER_PW can be used.

Non-TDomain parameters for the DM_LOCAL section

The following DM_LOCAL section parameters do not apply to domain gateways of type TDOMAIN but are included here for completeness:

- BLOB_SHM_SIZE = *numeric* — applicable to domain gateways of type SNAX
- MAXACCESSPOINT (also known as MAXRDOM) = *numeric* — applicable to domain gateways of type OSITP
- MAXDATALEN = *numeric* — applicable to domain gateways of type OSITP

For detailed descriptions of SNAX and OSITP parameters, see [BEA eLink Documentation at http://e-docs.bea.com/mlink/mainfram/mainfram.htm](http://e-docs.bea.com/mlink/mainfram/mainfram.htm).

DM_REMOTE Section

This section, also known as the DM_REMOTE_DOMAINS section, defines one or more remote domain access point identifiers and their characteristics.

Entries within the DM_REMOTE section have the following form:

```
RemoteAccessPoint required_parameters [optional_parameters]
```

where *RemoteAccessPoint* is a remote domain access point identifier (logical name) that you choose to identify each remote domain known to the local BEA Tuxedo application.

RemoteAccessPoint must be unique across the local and remote domains involved in a Domains configuration. As you will see in the description of the DM_IMPORT section, you use a remote domain access point to associate remote services with a particular remote domain. The

remote services available through the remote domain access point will be available to clients in the local domain through a remote domain access point and a local domain access point.

Required parameters for the DM_REMOTE section

`TYPE = identifier`

Specifies the type of local domain gateway needed to communicate with the remote domain associated with this remote domain access point. `TYPE` can be set to one of the following values: `TDOMAIN`, `SNAX`, `OSITP`, or `OSITPX`.

The `TDOMAIN` value indicates that a local instance of the `GWTDOMAIN` process will communicate with a remote BEA Tuxedo application.

The `SNAX` value indicates that a local instance of the `GWSNAX` process will communicate with a remote TP domain via the SNA protocol.

The `OSITP` value indicates that a local instance of the `GWOSITP` process will communicate with a remote TP domain via the OSI TP 1.3 protocol.

The `OSITPX` value indicates that a local instance of the `GWOSITP` process will communicate with a remote TP domain via the OSI TP 4.0 or later protocol. The `OSITPX` value is supported only by BEA Tuxedo 8.0 or later software.

`ACCESSPOINTID (also known as DOMAINID) = string[1..30]`

Used to identify the remote domain associated with this remote domain access point for purposes of security when setting up a connection to the remote domain. For a local domain gateway of type `TDOMAIN`, this value may also be used by the `TDomain` gateway (local instance of the `GWTDOMAIN` process) as the user ID for incoming requests from this remote domain access point connection. `ACCESSPOINTID` must be unique across local and remote domain access points.

`ACCESSPOINTID` must be 30 bytes or fewer in length. If the value is a string, it must be 30 characters or fewer (counting the trailing `NULL`). The value of `string` can be a sequence of characters or a sequence of hexadecimal digits preceded by `0x`.

Optional parameters for the DM_REMOTE section

The following optional parameters for the `DM_REMOTE` section describe resources and limits used in the operation of the local domain gateways:

`ACL_POLICY = {LOCAL | GLOBAL}`

Specifies the access control list (ACL) policy for this remote domain access point. This parameter applies only to domain gateways of type `TDOMAIN` running BEA Tuxedo 7.1 or

later software and domain gateways of type OSITPX running BEA Tuxedo 8.0 or later software.

LOCAL means that the local domain *replaces* the credential (identity) of any service request received from the remote domain *with* the principal name specified in the LOCAL_PRINCIPAL_NAME parameter for this remote domain access point. GLOBAL means that the local domain does not replace the credential received with a remote service request; if no credential is received with a remote service request, the local domain forwards the service request to the local service *as is* (which usually fails). If this parameter is not specified, the default is LOCAL.

Note that the ACL_POLICY parameter controls whether or not the local domain replaces the credential of a service request received from a remote domain with the principal name specified in the LOCAL_PRINCIPAL_NAME parameter. The CREDENTIAL_POLICY parameter is related to this parameter and controls whether or not the local domain removes the credential from a local service request before sending the request to a remote domain.

LOCAL_PRINCIPAL_NAME = *string*[0..511]

The local principal name identifier (credential) assigned by the local domain to service requests received from the remote domain when the ACL_POLICY parameter for this remote domain access point is set (or defaulted) to LOCAL. This parameter applies only to domain gateways of type TDOMAIN running BEA Tuxedo 7.1 or later software and domain gateways of type OSITPX running BEA Tuxedo 8.0 or later software.

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 ACCESSPOINTID string for this remote domain access point.

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 access point when establishing a connection to the local domain. This parameter applies only to domain gateways of type TDOMAIN 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 ACCESSPOINTID string for this remote domain access point.

For default authentication plug-ins, if a value is assigned to the CONNECTION_PRINCIPAL_NAME parameter for this remote domain access point, it must be the same as the value assigned to the ACCESSPOINTID parameter for this remote

domain access point. If these values do not match, any attempt to set up a connection between the local TDomain gateway and the remote TDomain gateway will fail, and the system will generate the following `userlog(3c)` message: `ERROR: Unable to initialize administration key for domain domain_name.`

`CREDENTIAL_POLICY = {LOCAL | GLOBAL}`

Specifies the credential policy for this remote domain access point. This parameter applies only to domain gateways of type `TDOMAIN` running BEA Tuxedo 8.0 or later software.

`LOCAL` means that the local domain removes the credential (identity) from a local service request destined for this remote domain access point. `GLOBAL` means that the local domain does not remove the credential from a local service request destined for this remote domain access point. If this parameter is not specified, the default is `LOCAL`.

Note that the `CREDENTIAL_POLICY` parameter controls whether or not the local domain removes the credential from a local service request before sending the request to a remote domain. The `ACL_POLICY` parameter is related to this parameter and controls whether or not the local domain replaces the credential of a service request received from a remote domain with the principal name specified in the `LOCAL_PRINCIPAL_NAME` parameter.

`MTYPE = string[1..15]`

Used for grouping domains so that encoding/decoding of messages can be bypassed between the machine associated with this remote domain access point and the machine associated with the local domain access point. This parameter applies only to domain gateways of type `TDOMAIN`.

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` and the `DM_REMOTE` section of a `DMCONFIG` 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.

`PRIORITY_TYPE = {LOCAL_RELATIVE | LOCAL_ABSOLUTE | GLOBAL}`

`INPRIORITY = numeric`

Together, the `PRIORITY_TYPE` and `INPRIORITY` parameters specify the message priority handling for this remote domain access point. These parameters are supported by BEA Tuxedo 8.0 or later software.

For the `PRIORITY_TYPE` parameter, the `LOCAL_RELATIVE` and `LOCAL_ABSOLUTE` values are valid for all remote domain types; the `GLOBAL` value is valid only for remote domains of type `TDOMAIN`. If not set, the `PRIORITY_TYPE` parameter defaults to `LOCAL_RELATIVE`.

`PRIORITY_TYPE=LOCAL_RELATIVE` means that the priority associated with a request from this remote domain access point (for example, via the `tpsprio` call) is not used by

the local domain. Instead, the priority of incoming requests from this remote domain access point is set relative to the `INPRIORITY` value; this value may be greater than or equal to -99 (lowest priority) and less than or equal to 99 (highest priority), with 0 being the default. The setting of `INPRIORITY` increments or decrements a service's default priority as follows: up to a maximum of 100 or down to a minimum of 1, depending on its sign, where 100 is the highest priority. For requests to the remote domain access point, the priority associated with a request will accompany the request to the remote domain access point.

`PRIORITY_TYPE=LOCAL_ABSOLUTE` means that the priority associated with a request from this remote domain access point is not used by the local domain. Instead, the priority of incoming requests from this remote domain access point is set relative to the `INPRIORITY` value; this value may be greater than or equal to 1 (lowest priority) and less than or equal to 100 (highest priority), with 50 being the default. The setting of `INPRIORITY` increments or decrements a service's default priority as follows: up to a maximum of 100 or down to a minimum of 1, depending on its sign, where 100 is the highest priority. For requests to the remote domain access point, the priority associated with a request will accompany the request to the remote domain access point.

`PRIORITY_TYPE=GLOBAL` means that the priority associated with a request from this remote domain access point is adjusted by the local domain. The priority of incoming requests from this remote domain access point is adjusted relative to the `INPRIORITY` value; this value may be greater than or equal to -99 (lowest priority) and less than or equal to 99 (highest priority), with 0 being the default. If `INPRIORITY` is set, the priority accompanying the incoming request is added to the `INPRIORITY` value to create an absolute priority setting for the incoming request. If `INPRIORITY` is not set or is set to 0, the priority accompanying the incoming request is used *as is* by the local domain. For requests to the remote domain access point, the priority associated with a request will accompany the request to the remote domain access point.

Non-TDomain parameters for the `DM_REMOTE` section

The following `DM_REMOTE` section parameter does not apply to domain gateways of type `TDOMAIN` but is included here for completeness:

`CODEPAGE = string` — applicable to domain gateways of type `SNAX` and `OSITPX`

For detailed descriptions of `SNAX` and `OSITPX` parameters, see [BEA eLink Documentation at `http://e-docs.bea.com/mlink/mainfram/mainfram.htm`](http://e-docs.bea.com/mlink/mainfram/mainfram.htm).

DM_EXPORT Section

This section, also known as the `DM_LOCAL_SERVICES` section, provides information on the services exported by each local domain access point. If this section is absent, or is present but empty, all local domain access points defined in the `DM_LOCAL` section accept remote requests to all services advertised by the local BEA Tuxedo application. If this section is specified, it should be used to restrict the set of local services that can be requested from a remote domain.

A *local service* is a service made available to one or more remote domains through a local domain access point.

Entries within the `DM_EXPORT` section have the following form:

```
service [optional_parameters]
```

where *service* is the identifier name of a particular local service; it must be 15 characters or fewer in length. This name is advertised by one or more servers running within the local BEA Tuxedo application.

A local service made available to one or more remote domains inherits many of its properties from 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 for the DM_EXPORT section

`LACCESSPOINT` (**also known as** `LDOM`) = *identifier*

Specifies the name of the local domain access point exporting this service. If this parameter is not specified, all local domain access points defined in the `DM_LOCAL` section accept remote requests to this local service.

`ACL` = *identifier*

Specifies the name of the access control list (ACL) to be used by the local domain access point to restrict requests made to this local service by remote domains. The name of the ACL is defined in the `DM_ACCESS_CONTROL` section.

`CONV` = {Y | N}

Specifies whether (Y) or not (N) this local service is a conversational service. The default is N.

`RNAME` = *string*[1..30]

Specifies an alternative identity, or “alias,” for the name of this local service to the remote domains. This name will be used by the remote domains to request this service. If this parameter is not specified, the actual name of this local service name—the *service* identifier—is the name used by the remote domains to request this service.

Non-TDomain parameters for the DM_EXPORT section

The following DM_EXPORT section parameters do not apply to domain gateways of type TDOMAIN but are included here for completeness.

- INBUFTYPE = *string* — applicable to domain gateways of type SNAX, OSITP, and OSITPX
- OUTBUFTYPE = *string* — applicable to domain gateways of type SNAX, OSITP, and OSITPX
- COUPLING = {TIGHT | LOOSE} — applicable to domain gateways of type OSITPX
- INRECTYPE = *string* — applicable to domain gateways of type OSITPX
- OUTRECTYPE = *string* — applicable to domain gateways of type OSITPX

For detailed descriptions of SNAX, OSITP, and OSITPX parameters, see [BEA eLink Documentation](http://e-docs.bea.com/mlink/mainfram/mainfram.htm) at <http://e-docs.bea.com/mlink/mainfram/mainfram.htm>.

DM_IMPORT Section

This section, also known as the DM_REMOTE_SERVICES section, provides information on services imported and available to the local domain through remote domain access points defined in the DM_REMOTE section. If the DM_IMPORT section is absent, or is present but empty, no remote services are available to the local domain.

A *remote service* is a service made available to the local domain through a remote domain access point and a local domain access point.

Entries within the DM_IMPORT section have the following form:

```
service [optional_parameters]
```

where *service* is the identifier name advertised by the local BEA Tuxedo application for a particular remote service; it must be 15 characters or fewer in length. A remote service may be imported from one or more remote domains.

A remote BEA Tuxedo service made available to the local domain inherits many of its properties from the SERVICES section of the remote TUXCONFIG file, or their defaults. Some of the properties that may be inherited are LOAD, PRIO, AUTOTRAN, ROUTING, BUFTYPE, and TRANTIME.

Optional parameters for the DM_IMPORT section

RACCESSPOINT (**also known as** RDOM) =

identifier1[, *identifier2*][, *identifier3*][, *identifier4*]...[, *identifier 10*]

Specifies the remote domain access point through which this service is imported. If a remote domain access point is specified for this service *and* a local domain access point is specified (using the LACCESSPOINT parameter) for this service, only the named local domain access point is allowed to send local requests to this remote service through the named remote domain access point.

If a remote domain access point is specified for this service but no local domain access point is specified, any local domain access point defined in the DM_LOCAL section having the same gateway type (TDOMAIN, ...) as the remote domain access point is allowed to send local requests to this remote service through the named remote domain access point.

If no remote domain access point is specified for this service and no local domain access point is specified, any local domain access point defined in the DM_LOCAL section may send requests to this service through any remote domain access point defined in the DM_REMOTE section.

If you want to configure alternate remote domain access points with the *identifier2*, *identifier3*, *identifier4* arguments, you must specify ON_STARTUP as the value of the CONNECTION_POLICY parameter in the DM_LOCAL section. (CONNECTION_POLICY may also be specified in the DM_TDOMAIN section for a BEA Tuxedo 8.1 or later application.) If *identifier2* is configured, it is used for failover: When the remote domain associated with *identifier1* is unavailable, the remote domain associated with *identifier2* is used. Similarly, if *identifier3* *ND* *identifier4* are configured, they are used for failover: When the remote domains associated with *identifier1*, *identifier2* and *identifier3* are unavailable, the remote domain associated with *identifier4* is used.

LACCESSPOINT (**also known as** LDOM) = *identifier*

Specifies the name of a local domain access point that is allowed to send requests to this remote service. The gateway group associated with this local domain access point advertises the name—the *service* identifier—of the remote service in the BEA Tuxedo system bulletin board.

BLOCKTIME *numeric_value*

Specifies the nontransactional client blocking time value, in seconds, per service indicating the minimum amount of time a blocking API call will delay before timing out for a particular service. The blocktime value is controlled by the local domain.

This parameter lets the client know that (after a specified time in seconds), no reply has been received by the server while the service request is still processing.

numeric_value can be between 0 and 32,767 inclusive. If not specified, the default is 0 which indicates that the system-wide BLOCKTIME value specified in the UBBCONFIG RESOURCES section is used for the service.

CONV = {Y | N}

Specifies whether (Y) or not (N) this remote service is a conversational service. The default is N.

LOAD = *numeric*

Specifies the service load for this remote service. The value must be greater than or equal to 1 and less than or equal to 32767. The default is 50. Interface loads are used for load balancing purposes, that is, queues with higher enqueued workloads are less likely to be chosen for a new request.

RNAME = *string*[1..30]

Specifies an alternative identity, or “alias,” for the name of this remote service to the local domain. This name will be used by the local domain to request this service. If this parameter is not specified, the actual name of this remote service name—the *service* identifier—is the name used by the local domain to request this service.

ROUTING = *identifier*

Specifies the name of the routing criteria table used for data-dependent routing for this remote service. When more than one remote domain access point offers the same service, a local domain access point can perform data-dependent routing if this optional parameter is specified. If this parameter is not specified, data-dependent routing is not used for this service.

The *identifier* is a *ROUTING_CRITERIA_NAME* defined in the DM_ROUTING section. The value of *identifier* must be 15 characters or less in length. If multiple entries for the same service name are included with different remote domain access points (specified using the RACCESSPOINT parameter), the value of the ROUTING parameter should be the same for all of these entries.

Non-TDomain parameters for the DM_IMPORT section

The following DM_IMPORT section parameters do not apply to domain gateways of type TDOMAIN but are included here for completeness:

- INBUFTYPE = *string* — applicable to domain gateways of type SNAX, OSITP, and OSITPX

- `OUTBUFTYPE = string` — applicable to domain gateways of type SNAX, OSITP, and OSITPX
- `AUTOPREPARE = {Y | N}` — applicable to domain gateways of type OSITPX
- `INRECTYPE = string` — applicable to domain gateways of type OSITPX
- `OUTRECTYPE = string` — applicable to domain gateways of type OSITPX
- `TPSUT_TYPE = {INTEGER | PRINTABLESTRING}` — applicable to domain gateways of type OSITPX
- `REM_TPSUT = string` — applicable to domain gateways of type OSITPX

For detailed descriptions of SNAX, OSITP, and OSITPX parameters, see [BEA eLink Documentation](http://e-docs.bea.com/mlink/mainfram/mainfram.htm) at <http://e-docs.bea.com/mlink/mainfram/mainfram.htm>.

DM_RESOURCES

This optional section is used for defining global Domains configuration information, specifically a user-supplied configuration version string. This field is not checked by the software.

The only parameter for the `DM_RESOURCES` section is:

`VERSION = string`

where *string* is a field in which users can enter a version number for the current `DMCONFIG` file.

DM_ROUTING Section

This section provides information for data-dependent routing of local service requests using FML, FML32, VIEW, VIEW32, X_C_TYPE, X_COMMON, or XML typed buffers to one of several remote domains offering the same service.

Entries within the `DM_ROUTING` section have the following form:

`ROUTING_CRITERIA_NAME required_parameters`

where `ROUTING_CRITERIA_NAME` is the *identifier* name assigned to the `ROUTING` parameter for the particular service entry in the `DM_IMPORT` section. `ROUTING_CRITERIA_NAME` must be 15 characters or less in length.

Required parameters for the DM_ROUTING section

`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.

An XML element content encoded in UTF-8 can be used for routing. When used for routing, the element content cannot contain character references, entity references, or CDATA sections. An XML element attribute encoded in UTF-8 can also be used for routing if the element to which the attribute belongs is defined.

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`.

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

`RANGES = "string[1..4096]"`

Specifies the ranges and associated remote domain access point names for the routing field. `string` must be enclosed in double quotes. The format of `string` is a comma-separated ordered list of pairs, where each pair consists of a range and a remote domain access point separated by a colon (:); for example,

```
RANGES = "MIN-1000:b01,1001-3000:b02,*:b03".
```

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, "`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 `*` (wildcard) in the position of a range indicates any values not covered by the other ranges previously seen in the entry; only one wildcard 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(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.

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

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

A list of types and subtypes of data buffers for which this routing entry is valid. The types are restricted to FML, FML32, VIEW, VIEW32, X_C_TYPE, X_COMMON, or XML. 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). 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 wildcard range has not been specified, an error is returned to the application process that requested the execution of the remote service.

DM_ACCESS_CONTROL Section

This section specifies one or more access control list (ACL) names and associates one or more remote domain access points with each specified ACL name. You can use the `ACL` parameter in the `DM_EXPORT` section by setting `ACL=ACL_NAME` to restrict access to a local service exported through a particular local domain access point to just those remote domain access points associated with the `ACL_NAME`.

Entries within the `DM_ACCESS_CONTROL` section have the following form:

ACL_NAME required_parameters

where `ACL_NAME` is an identifier value used to specify an access control list; it may contain no more than 15 characters.

The only required parameter for the `DM_ACCESS_CONTROL` section is:

ACLIST = identifier[, identifier]

where an `ACLIST` is composed of one or more remote domain access point names separated by commas. The wildcard character (*) can be used to specify that all remote domain access points defined in the `DM_REMOTE` section can access a particular local service exported through a particular local domain access point.

DM_TDOMAIN Section

This section defines the network-specific information for TDomain gateways. The `DM_TDOMAIN` section should have an entry per local domain access point if requests from remote domains to local services are accepted through that local domain access point, and at least one entry per remote domain access point if requests from the local domain to remote services are accepted through that access point.

The `DM_TDOMAIN` section is used to configure the following network properties for an access point entry:

- For a local domain access point entry, specify the network address to be used for listening for incoming connections.
- For a remote domain access point entry, specify the destination network address to be used when connecting to the remote domain associated with that access point.
- For a local or remote domain access point entry, specify the conditions under which the TDomain gateway tries to establish connections. This optional configuration is available only to BEA Tuxedo 8.1 or later applications.
- For a local or remote domain access point entry, specify whether the TDomain gateway sends keepalive messages on the connection to the remote domain. This optional configuration is available only to BEA Tuxedo 8.1 or later applications.

Entries within the `DM_TDOMAIN` section have the following form:

```
AccessPoint required_parameters [optional_parameters]
```

where *AccessPoint* is an identifier value used to identify either a local domain access point or a remote domain access point. The *AccessPoint* identifier must match a previously defined local domain access point in the `DM_LOCAL` section or a previously defined remote domain access point in the `DM_REMOTE` section.

Required parameters for the DM_TDOMAIN section

`NWADDR = string[1..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)`

Specifies the network address associated with this local or remote domain access point. For a local domain access point, this parameter supplies the address to be used for listening for incoming connections from other BEA Tuxedo applications. For a remote domain access point, this parameter supplies the destination address to be used when connecting to the BEA Tuxedo application associated with the remote domain access point. The value of this parameter must be unique across all `DM_TDOMAIN` entries.

If *string* has the form "0x*hex-digits*" or "\\x*hex-digits*", it must contain an even number of valid hexadecimal 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:

```
//hostname: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.

Optional parameters for the DM_TDOMAIN section

NWDEVICE = *string*[1..78]

Specifies the network device to be used when binding to the network address of this local or remote domain access point. 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.

The NWDEVICE parameter is not required. In earlier releases, if the networking functionality is TLI-based, the network device name must be an absolute pathname.

CMPLIMIT = *numeric*

Specifies the compression threshold to be used when sending data to this remote domain access point. This parameter is relevant only to remote domain access points. Its minimum value is 0, and its maximum value is 2147483647. The default is 2147483647. Application buffers larger than the CMPLIMIT value are compressed.

MINENCRYPTBITS = {0 | 40 | 56 | 128 | 256}

Specifies the minimum level of encryption required when establishing a network link to the remote domain associated with this remote domain access point. This parameter is relevant only to remote domain access points.

A value of 0 means no encryption, while a value of 40, 56, 128, or 256 specifies the encryption key length (in bits). The default is 0. If the minimum level of encryption cannot be met, link establishment fails.

Note: The link-level encryption value of 40 bits is provided for backward compatibility. 256-bit encryption is currently possible only when using SSL.

MAXENCRYPTBITS = {0 | 40 | 56 | 128 | 256}

Specifies the maximum level of encryption allowed when establishing a network link to the remote domain associated with this remote domain access point. This parameter is relevant only to remote domain access points.

A value of 0 means no encryption, while a value of 40, 56, 128 or 256 specifies the encryption key length (in bits). The default is 128.

Note: The link-level encryption value of 40 bits is provided for backward compatibility. 256-bit encryption is currently possible only when using SSL.

NWPROTOCOL = {LLE | SSL | SSL_ONE_WAY}

Specifies SSL, LLE, or one-way SSL encryption. The default value is LLE. The SSL option requires the domains at both end of the connection to authenticate each other; the SSL_ONE_WAY option does not.

If SSL_ONE_WAY is set, the domain that *accepts* an SSL connection needs to authenticate itself to the domain that *initiates* the connection using an SSL certificate. The initiating domain *does not* need to authenticate itself to the other domain. This value is mainly intended for use with a CONNECTION_POLICY to INCOMING_ONLY, and should only be set when the domain that accepts incoming connections *does not* need to authenticate connecting domains.

Note: If NWPROTOCOL is not set or is set to LLE and SSL_RENEGOTIATION is set to a non-zero value, `dmloadcf` prints a warning message.

SSL_RENEGOTIATION *numeric*

Specifies the renegotiation interval (in seconds) for SSL information. It must be greater than or equal to 0 and less than or equal to 2,147,483,647. The default value is 0 (which indicates that no renegotiation takes place).

Note: If NWPROTOCOL is not set or set to LLE and SSL_RENEGOTIATION is set to a non-zero value, `dmloadcf` prints a warning message.

CONNECTION_POLICY = {LOCAL | ON_DEMAND | ON_STARTUP | INCOMING_ONLY}

Specifies the conditions under which the TDomain gateway associated with this local or remote domain access point tries to establish connections. Supported values are LOCAL, ON_DEMAND, ON_STARTUP, or INCOMING_ONLY. LOCAL is relevant only to remote domain access points.

The CONNECTION_POLICY parameter is available in the DM_TDOMAIN section when running BEA Tuxedo 8.1 or later software. Its value in the DM_TDOMAIN section for a

particular local or remote domain access point takes precedence over its global value in the `DM_LOCAL` section. The ability to override the global connection policy enables you to configure connection policy on a per TDomain session basis.

Specifying no connection policy for a *local domain access point* defaults to the global connection policy specified in the `DM_LOCAL` section. If you choose to specify a global connection policy in the `DM_TDOMAIN` section, do not specify a global connection policy in the `DM_LOCAL` section.

A connection policy of `LOCAL` means that a remote domain access point accepts the global connection policy defined in the `DM_LOCAL` section. `LOCAL` is the default connection policy for remote domain access points. Excluding `LOCAL`, the connection policy value for a remote domain access point takes precedence over the connection policy value for a local domain access point.

A connection policy of `ON_DEMAND` means that the TDomain gateway attempts a connection only when requested by either a client request to a remote service or a `dmadmin(1) connect` command. Connection retry processing is not allowed when the connection policy is `ON_DEMAND`.

A connection policy of `ON_STARTUP` means that the TDomain gateway attempts to establish a connection at gateway server initialization time. For `ON_STARTUP`, the remote services for a particular remote domain (that is, services advertised by the TDomain gateway) are advertised only if a connection is successfully established to the remote domain. Thus, if there is no active connection to the remote domain, the remote services are suspended. By default, this connection policy retries failed connections every 60 seconds, but you can specify a different value for this interval using the `RETRY_INTERVAL` parameter in the `DM_TDOMAIN` section. Also, see the `MAXRETRY` parameter in this section.

A connection policy of `INCOMING_ONLY` means that the TDomain gateway does not attempt an initial connection upon startup and that remote services are initially suspended. The TDomain gateway is available for incoming connections from a remote domain, and remote services are advertised when the gateway receives an incoming connection or an administrative connection (using the `dmadmin(1) connect` command) is made. Connection retry processing is not allowed when the connection policy is `INCOMING_ONLY`.

`FAILOVERSEQ = -1 <= num <= 32767`

Specifies the failover sequence and establishes the primary record for a TDomain session between remote and local access points in Tuxedo release 9.0 and later. The TDomain session record with the lowest `FAILOVERSEQ` number is the primary record for that session. If not specified, `FAILOVERSEQ` defaults to -1.

There is only one primary record for a TDomain session, all remaining records for the same TDomain session are called secondary/backup records. With the exceptions of `NWADDR`, `NWDEVICE`, and `FAILOVERSEQ`, the primary record is the source for all TDomain session configuration parameters and attributes. All other parameters and attributes listed in secondary/backup records are ignored.

Based on the `CONNECTION_POLICY` attribute you select, the local domain will try to connect to a TDomain session's primary record. If the primary record fails to connect, it will then try to connect to the next sequential secondary/backup record. If all secondary record connections fail, it will retry the primary record information at a later time as determined by `RETRY_INTERVAL` until `MAXRETRY` is exhausted.

`LACCESSPOINT` (also known as `LDM`) = *“string”*[1..30]

Specifies the name of a local domain access point listed in the `DM_LOCAL` section of the `DMCONFIG` file in Tuxedo release 9.0 and later. The `LACCESSPOINT` parameter is used exclusively to define TDomain session gateways and can contain only one local domain access point as its value.

If not specified, `LACCESSPOINT` defaults to *“*”* and the TDomain session will connect to all local domain access points listed in the `DM_LOCAL` section. You can substitute `LDM` for the `LACCESSPOINT` parameter.

Note: `LACCESSPOINT` can also use regular expression values to define multiple local domain access points. When the `DMCONFIG` file is compiled using `dmloadcf`, the regular expression values are expanded to their full local domain names in the `BDMCONFIG` file. `LACCESSPOINT` can only use regular expressions in the `DMCONFIG` file.

[`MAXRETRY` = {*numeric* | `MAXLONG`}

Specifies the number of times that the TDomain gateway associated with this local or remote domain access point tries to establish a connection. This parameter is available in the `DM_TDOMAIN` section when running BEA Tuxedo 8.1 or later software, and is valid when the `CONNECTION_POLICY` parameter for this access point is set to `ON_STARTUP`. For other connection policies, automatic retries are disabled.

The minimum value for `MAXRETRY` is 0, and the maximum value is `MAXLONG` (2147483647). `MAXLONG`, the default, indicates that retry processing will be repeated indefinitely, or until a connection is established.

`RETRY_INTERVAL` = *numeric*

Specifies the number of seconds that the TDomain gateway associated with this local or remote domain access point waits between automatic attempts to establish a connection. This parameter is available in the `DM_TDOMAIN` section when running BEA Tuxedo 8.1 or

later software, and is valid when the `CONNECTION_POLICY` parameter for this access point is set to `ON_STARTUP`. For other connection policies, automatic retries are disabled.

The minimum value for `RETRY_INTERVAL` is 0, and the maximum value is 2147483647. The default is 60. If `MAXRETRY` is set to 0, setting `RETRY_INTERVAL` is not allowed.

`TCPKEEPALIVE = {LOCAL | NO | YES}`

Enables TCP-level keepalive for this local or remote domain access point. Supported values are `LOCAL`, `N` (no), or `Y` (yes). `LOCAL` is relevant only to remote domain access points.

The `TCPKEEPALIVE` parameter applies only to domain gateways of type `TDOMAIN` running BEA Tuxedo 8.1 or later software. Its value for a remote domain access point takes precedence over its value for a local domain access point. The ability to override the local domain access point value enables you to configure TCP-level keepalive on a per remote domain basis.

A value of `LOCAL` means that a remote domain access point accepts the TCP-level keepalive value defined for the local domain access point. `LOCAL` is the default TCP-level keepalive value for remote domain access points.

A value of `NO` means that TCP-level keepalive is disabled for this access point. `N` is the default TCP-level keepalive value for local domain access points.

A value of `YES` means that TCP-level keepalive is enabled for this access point. When TCP-level keepalive is enabled for a connection, the keepalive interval used for the connection is the system-wide value configured for the operating system's TCP keepalive timer. This interval is the maximum time that the TDomain gateway will wait without receiving any traffic on the connection. If the maximum time is exceeded, the gateway sends a TCP-level keepalive request message. If the connection is still open and the remote TDomain gateway is still alive, the remote gateway responds by sending an acknowledgement. If the local TDomain gateway does not receive an acknowledgement within a fixed period of time of sending the request message, it assumes that the connection is broken and releases any resources associated with the connection.

Not only does TCP-level keepalive keep BEA Tuxedo interdomain connections open during periods of inactivity, but it also enable TDomain gateways to quickly detect connection failures.

Note: The `TCPKEEPALIVE` and `DMKEEPALIVE` parameters are *not* mutually exclusive, meaning that you can configure an interdomain connection using both parameters.

`DMKEEPALIVE = numeric`

Controls application-level keepalive for this local or remote domain access point. This value must be greater than or equal to -1 and less than or equal to 2147483647. The value -1 is relevant only to remote domain access points.

The `DMKEEPALIVE` parameter applies only to domain gateways of type `TDOMAIN` running BEA Tuxedo 8.1 or later software. Its value for a remote domain access point takes precedence over its value for a local domain access point. The ability to override the local domain access point value enables you to configure application-level keepalive on a per remote domain basis.

A value of -1 means that a remote domain access point accepts the application-level keepalive value defined for the local domain access point. -1 is the default application-level keepalive value for remote domain access points.

A value of 0 means that application-level keepalive is disabled for this access point. 0 is the default application-level keepalive value for local domain access points.

A value greater than or equal to 1 and less than or equal to 2147483647, in milliseconds, currently rounded up to the nearest second by the Domains software, means that application-level keepalive is enabled for this access point. The time that you specify is the maximum time that the TDomain gateway will wait without receiving any traffic on the connection. If the maximum time is exceeded, the gateway sends an application-level keepalive request message. If the connection is still open and the remote TDomain gateway is still alive, the remote gateway responds by sending an acknowledgement. If the local TDomain gateway does not receive an acknowledgement within a configurable period of time (see the `DMKEEPALIVEWAIT` parameter) of sending the request message, it assumes that the connection is broken and releases any resources associated with the connection.

Not only does application-level keepalive keep BEA Tuxedo interdomain connections open during periods of inactivity, but it also enable TDomain gateways to quickly detect connection failures.

Note: The `DMKEEPALIVE` and `TCPKEEPALIVE` parameters are *not* mutually exclusive, meaning that you can configure an interdomain connection using both parameters.

`DMKEEPALIVEWAIT = numeric`

Specifies the maximum time for this local or remote domain access point that the TDomain gateway will wait without receiving an acknowledgement to a sent keepalive message. This value must be greater than or equal to 0 and less than or equal to 2147483647, in milliseconds, currently rounded up to the nearest second by the Domains

software. The default is 0. This parameter applies only to domain gateways of type TDOMAIN running BEA Tuxedo 8.1 or later software.

If DMKEEPALIVE is 0 (keepalive disabled) for this access point, setting DMKEEPALIVEWAIT has no effect.

If DMKEEPALIVE is enabled for this access point and DMKEEPALIVEWAIT is set to a value greater than DMKEEPALIVE, the local TDomain gateway will send more than one application-level keepalive message before the DMKEEPALIVEWAIT timer expires. This combination of settings is allowed.

If DMKEEPALIVE is enabled for this access point and DMKEEPALIVEWAIT is set to 0, receiving an acknowledgement to a sent keepalive message is unimportant: any such acknowledgement is ignored by the TDomain gateway. The gateway continues to send keepalive messages every time the DMKEEPALIVE timer times out. *Use this combination of settings to keep an idle connection open through a firewall.*

Multiple entries for the same access point in the DM_TDOMAIN section

If this DM_TDOMAIN entry is a local domain access point (as specified in the DM_LOCAL section), its NWADDR is a network address to be used to listen for incoming connections. Entries associated with a local domain access point can be specified more than once in the DM_TDOMAIN section, to allow for migration of the services associated with a local access point to another machine in the BEA Tuxedo domain.

Entries associated with a remote domain access point (as specified in the DM_REMOTE section) can also be specified more than once in the DM_TDOMAIN section. If FAILOVERSEQ is not specified, the first entry is considered to be the primary address, which means its NWADDR is the first network address tried when a connection is being attempted to the remote domain access point. The second entry is considered to be the secondary address, which means its NWADDR is the second network address tried when a connection cannot be established using the primary address.

Note: If the FAILOVERSEQ parameter is used, it determines the primary and secondary addresses for TDomain session connection policies.

If this DM_TDOMAIN entry is another occurrence of a remote domain access point, the entry points to a secondary remote gateway that must reside in a different BEA Tuxedo domain than the BEA Tuxedo domain in which the primary remote gateway resides. The secondary and primary remote gateways must have the same ACCESSPOINTID defined in the DM_LOCAL section of their associated DMCONFIG files; 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 remote gateway is available.

Note: For multiple entries of a local or remote domain access point in the DM_TDOMAIN section, only the multiple instances of the NWADDR parameter are read by the Domains software. For multiple instances of any other parameter, only the first instance of the parameter is read by the Domains software; all other instances are ignored.

DMCONFIG(5) Additional Information

Files

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

Example 1

The following Domains configuration file defines a five-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 Domains Configuration File for the Central Bank
#
#
*DM_LOCAL
#
DEFAULT: SECURITY = NONE

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

c02  GWGRP = bankg2
      TYPE = OSITP
      ACCESSPOINTID = "BA.CENTRAL02"
      DMTLOGDEV = "/usr/apps/bank/DMTLOG"
      DMTLOGNAME = "DMTLG_C02"

#
*DM_REMOTE
#
b01  TYPE = TDOMAIN
      ACCESSPOINTID = "BA.BANK01"

b02  TYPE = TDOMAIN
```

```

ACCESSPOINTID = "BA.BANK02"

b03 TYPE = TDOMAIN
ACCESSPOINTID = "BA.BANK03"

b04 TYPE = OSITP
ACCESSPOINTID = "BA.BANK04"

*DM_TDOMAIN
#
# 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
#
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_EXPORT
#
open_act ACL = branch
close_act ACL = branch
credit
debit
balance
loan LACCESSPOINT = c02 ACL = loans

*DM_IMPORT
#
tlr_add LACCESSPOINT = c01 ROUTING = ACCOUNT
tlr_bal LACCESSPOINT = c01 ROUTING = ACCOUNT
tlr_add RACCESSPOINT = b04 LACCESSPOINT = c02 RNAME = "TPSU002"
tlr_bal RACCESSPOINT = b04 LACCESSPOINT = c02 RNAME = "TPSU003"
tlr_bal RACCESSPOINT = b02,b03" LACCESSPOINT = c02

*DM_ROUTING
#
ACCOUNT FIELD = branchid BUFTYPE = "VIEW:account"

```

```

RANGES = "MIN-1000:b01,1001-3000:b02,*:b03"

*DM_ACCESS_CONTROL
#
branch ACLIST = "b01,b02,b03"
loans  ACLIST = b04

```

Example 2

This example shows the BEA Tuxedo Domains configuration file for one of the Bank Branches (BANK01).

```

#
#BEA Tuxedo Domains Configuration file for a Bank Branch
#
#
*DM_LOCAL
#
b01  GWGRP = auth
      TYPE = TDOMAIN
      ACCESSPOINTID = "BA.BANK01"
      DMTLOGDEV = "/usr/apps/bank/DMTLOG"

*DM_REMOTE
#
c01  TYPE = TDOMAIN
      ACCESSPOINTID = "BA.CENTRAL01"

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

*DM_IMPORT
#

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

*DM_ACCESS_CONTROL
#
central  ACLIST = c01

```

Network Addresses

Suppose the local machine on which a 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 `155.2.193.12` 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\(5\)](#), [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 ATMI Application Using C

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

For BEA Tuxedo release 7.1 or later, the Domains MIB uses improved class and attribute terminology to describe the interaction between local and remote 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 domain access points. These `DMCONFIG` section name changes are shown in the following table.

This <code>DMCONFIG</code> section name. . .	Has changed to. . .
<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.

This parameter name. . .	Has changed to. . .
<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

The improved Domains terminology introduced in BEA Tuxedo release 7.1 has been applied to the `DM_MIB` reference page, classes, and error messages, and to the `DMCONFIG` reference page, section names, parameter names, and error messages.

For backwards compatibility, aliases are provided between the `DMCONFIG` terminology used prior to BEA Tuxedo 7.1 and the improved Domains MIB terminology. For BEA Tuxedo release 7.1 or later, `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 component, which is described in [Using the BEA Tuxedo Domains Component](#).

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. For additional information pertaining to all `DM_MIB(5)` class definitions, see [“DM_MIB\(5\) Additional Information” on page 166](#).

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

Table 11 DM_MIB Classes

Class Name	Attributes
T_DM_ACL	Domain access control list
T_DM_CONNECTION	Connection status between two domains
T_DM_EXPORT	Exported resource
T_DM_IMPORT	Imported resource
T_DM_LOCAL	Local access point
T_DM_OSITP	OSI TP 1.3 specific configuration for an access point
T_DM_OSITPX	OSI TP 4.0 or later specific configuration for an access point
T_DM_PASSWORD	Domain password entry
T_DM_PRINCIPAL_MAP	Principal mapping entry
T_DM_REMOTE	Remote access point
T_DM_RESOURCES	Global Domains configuration information

Table 11 DM_MIB Classes

Class Name	Attributes
T_DM_ROUTING	Access point routing criteria
T_DM_RPRINCIPAL	Remote principal entry
T_DM_SNACRM	SNA-CRM-specific configuration for a local access point
T_DM_SNALINK	SNAX-specific configuration for a remote domain access point
T_DM_SNASTACK	SNA stack to be used by a specific SNA CRM
T_DM_TDOMAIN	TDomain-specific configuration for an access point
T_DM_TRANSACTION	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, 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, 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

Table 12 DM_MIB(5): T_DM_ACL Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
<code>TA_DMACLNAME</code> (r) (k) (*)	string	<code>rw-r--r--</code>	<i>string</i> [1..15]	N/A
<code>TA_DMRAccessPOINTLIST</code> (*)	string	<code>rw-r--r--</code>	<i>string</i> [0..1550]	" "

Table 12 DM_MIB(5): T_DM_ACL Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_STATE(r)	string	rw-r--r--	GET: "VAL" SET: "{NEW INV}"	N/A 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 Domains 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 domain 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 domain 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 domain 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

Table 13 DM_MIB(5): T_DM_CONNECTION Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_DMLACCESSPOINT(k)(*)	string	rw-r--r--	<i>string</i> [1..30]	N/A
TA_DMRACTESSPOINT(k)	string	rw-r--r--	<i>string</i> [1..30]	N/A
TA_DMTYPE	string	r--r--r--	"{TDOMAIN}"	N/A
TA_STATE(k)(*)	string	rwxr-xr--	GET: "{ACT SUS INI INA UNK}" SET: "{ACT INA}"	N/A N/A

Attributes available when TA_DMTYPE=TDOMAIN:

Table 13 DM_MIB(5): T_DM_CONNECTION Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_DMCURENCRYPTBITS	string	r-----	"{0 40 56 128}" ^{Note 1}	"0"

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

^{Note 1}The link-level encryption value of 40 bits is provided for backward compatibility.

Attribute Semantics

TA_DMLACCESSPOINT: *string*[1..30]

The name of the local domain access point identifying the connection between the domains.

On GET and SET operations, a specific local domain access point must be specified for this attribute.

TA_DMRACCESSPOINT: *string*[1..30]

The name of the remote domain access point identifying the connection between the domains.

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.

TA_DMSTYPE: "{TDOMAIN}"

The type of domain, which can be "TDOMAIN".

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

Table 14 DM_MIB(5): T_DM_EXPORT Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_DMRESOURCENAME(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" SET: "{NEW INV}"	N/A N/A
TA_DMACLNAME	string	rw-r--r--	string[1..15]	N/A
TA_DMCONV	string	rw-r--r--	"{Y N}"	"N"
TA_DMREMOTENAME	string	rw-r--r--	string[1..30]	N/A
Attributes available from remote domain access points of TA_DMTYPE=SNAX OSITP OSITPX:				
TA_DMINBUFTYPE	string	rw-r--r--	string[0..513]	N/A
TA_DMOUTBUFTYPE	string	rw-r--r--	string[0..513]	N/A
Attributes available from remote domain access points of TA_DMTYPE=OSITPX:				
TA_DMCOUPLING(r)	string	rw-r--r--	"{TIGHT LOOSE}"	"LOOSE"
TA_DMINRECTYPE(r)	string	rw-r--r--	string[0..78]	" "
TA_DMOUTRECTYPE(r)	string	rw-r--r--	string[0..78]	" "

(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 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 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 local access point name through which this local resource is available. Setting this attribute to "*" means the resource is available at all local access points.

TA_STATE:

GET: "{VALid}"

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.

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

SET: "{NEW | INVALid}"

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.

"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 for this local resource. This attribute is not permitted if `TA_DMRESOURCETYPE="QNAME"`.

TA_DMCONV: "{Y | N}"

Specifies whether this local resource is conversational.

TA_DMREMOTENAME: *string*[1..30]

Specifies the name of this local resource exported through the remote domain access points. If this attribute is not specified, the name of the local resource defaults to the name specified in TA_DMRESOURCENAME.

Attributes available from remote domain access points of TA_DMTYPE=SNAXIOSITPIOSITPX

TA_DMINBUFTYPE: *string*[0..513]

type[:subtype]—Specifies the input buffer type, optionally followed by the subtype, for this local resource. 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 using SNAX, or when access is permitted from remote domain access points using OSITP or OSITPX with the UDT application context.

TA_DMOUTBUFTYPE: *string*[0..513]

type[:subtype]— Specifies the output buffer type, optionally followed by subtype, for this local resource. 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 using SNAX, or when access is permitted from remote domain access points using OSITP or OSITPX with the UDT application context.

Attributes available from remote domain access points of TA_DMTYPE=OSITPX

TA_DMCOUPLING: *string*"{TIGHT | LOOSE}"

Specifies whether the transaction coupling is to be tight or loose when requests for this local service come through the same remote domain access point. The default is "LOOSE". Setting TA_DMCOUPLING="LOOSE" means that database updates made by the first request to this local service cannot be seen by the second request to the local service even though both requests are involved in the same global transaction. Setting TA_DMCOUPLING="TIGHT" means that multiple calls to the same local service through the same remote domain access point are tightly coupled: database updates made by the first request can be seen by the second request.

TA_DMCOUPLING="TIGHT" applies only when duplicate service requests come through the same remote domain access point. When the service requests are through different remote domain access points, the requests are always loosely coupled.

TA_DMINRECTYPE: *string*[1..78]

type[:subtype]—Specifies the type, optionally followed by subtype, and in some case the format of the reply buffer that a particular client requires for this local service. This attribute can be omitted if the local service sends a buffer that is identical in type and

structure to the buffer that the remote client expects. If you do not specify `TA_DMINRECTYPE`, the type of buffer is unchanged.

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

type[:subtype]—Specifies the type, optionally followed by subtype, of the buffer sent by the remote client for this local service. This attribute is used to enforce stronger type checking.

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

Table 15 DM_MIB(5): T_DM_IMPORT Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
<code>TA_DMRESOURCENAME(r)(k)(*)</code>	string	rw-r--r--	<i>string</i> [1..15]	
<code>TA_DMRACCESSPOINTLIST(k)(*)</code>	string	rw-r--r--	<i>string</i> [1..92]	* (meaning all)
<code>TA_DMLACCESSPOINT(k)(*)</code>	string	rw-r--r--	<i>string</i> [1..30]	* (meaning all)
<code>TA_STATE(r)</code>	string	rwxr-xr--	GET: "VAL" SET: "{NEW INV}"	N/A N/A
<code>TA_DMBLOCKTIME</code>	long	rwyr--r--	0 <= num <= 32,767	0
<code>TA_DMCONV</code>	string	rw-r--r--	"{Y N}"	"N"

Table 15 DM_MIB(5): T_DM_IMPORT Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
TA_DMLOAD	short	rw-r--r--	1 <= num <= 32,767	50
TA_DMREMOTENAME	string	rw-r--r--	string[1..30]	N/A
TA_DMRESOURCETYPE	string	rw-r--r--	"{SERVICE QSPACE QNAME}"	"SERVICE"
TA_DMROUTINGNAME	string	rw-r--r--	string[1..15]	N/A
Attributes available from remote domain access points of TA_DMTYPE=SNAX OSITP OSITPX:				
TA_DMINBUFTYPE	string	rw-r--r--	string[0..256]	N/A
TA_DMOUTBUFTYPE	string	rw-r--r--	string[0..256]	N/A
Attributes available from remote domain access points of TA_DMTYPE=OSITPX:				
TA_DMAUTOPREPARE(r)	string	rw-r--r--	"{Y N}"	"N"
TA_DMINRECTYPE(r)	string	rw-r--r--	string[0..78]	" "
TA_DMOUTRECTYPE(r)	string	rw-r--r--	string[0..78]	" "
TA_DMTPSUTTYPE(r)	string	rw-r--r--	"{INTEGER PRINTABLESTRING}"	" "
TA_DMREMTSUT(r)	string	rw-r--r--	string[0..64]	" "

(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_DMRACCESSPOINTLIST: *string*[1..92]

Identifies the remote domain access point through which this remote resource should be imported. **TA_DMRACCESSPOINTLIST** is a comma-separated failover domain list; it can contain up to ten remote domain access points of up to 30 characters each. If this attribute is set to "*", the resource can be imported from all remote domain access points.

TA_DMLACCESSPOINT: *string*[1..30]

The name of the local domain access point through which this remote resource should be made available. If this attribute is set to "*", the resource is made available through all local domain 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_DMBLOCKTIME: 0 <= num <= 32,767

Blocktime limit, in seconds, indicating the minimum amount of time a blocking API call will delay before timing out for a particular service. This attribute lets the client know that

(after a specified time in seconds), no reply has been received by the server while the service request is still processing.

If not specified, the default is 0 which indicates that the system-wide BLOCKTIME value specified in the UBBCONFIG RESOURCES section is used for the service.

TA_DMCONV: "{Y | N}"

A boolean value ("Y" or "N") specifying whether this remote resource is conversational.

TA_DMLOAD: 1 <= num <= 32,767

The service load for this remote resource. Interface 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_DMREMOTENAME: string[1..30]

Specifies the name of this remote resource imported through the remote domain access points. If this attribute is not specified, the name of the remote resource defaults to the name specified in TA_DMRESOURCENAME.

TA_DMROUTINGNAME: string[1..15]

The name of a T_DM_ROUTING object to use for routing criteria for this remote resource ("SERVICE" or "QSPACE").

Attributes available from remote domain access points of TA_DMTYPE=SNAXIOSITPIOSITPX

TA_DMINBUFTYPE: string[0..256]

type[:subtype]—Specifies the input buffer type, optionally followed by subtype, for this remote resource. If this attribute is present, it defines the buffer type [and subtype] accepted. This attribute should be defined for entries of DMRESOURCECTYPE="SERVICE" when using SNAX, or when access is permitted to remote domain access points using OSITP or OSITPX with the UDT application context.

TA_DMOUTBUFTYPE: string[0..256]

type[:subtype]—Specifies the output buffer type, optionally followed by subtype, for this remote resource. 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 using SNAX, or when access is permitted to remote domain access points using OSITP or OSITPX with the UDT application context.

Attributes available from remote domain access points of TA_DMTYPE=OSITPX

TA_DMAUTOPREPARE: *string*"{Y | N}"

Allows a single `tpcall()` involved in a global transaction to this remote service to automatically prepare the call. This optimization reduces the two-phase commit process to a single step. The remote OSITP domain must support this feature. The default is "N".

TA_DMINRECTYPE: *string*[1..78]

type[:subtype]—Specifies the type, optionally followed by subtype, and in some case the format of the request buffer that this remote service requires. This attribute can be omitted if the local client sends a buffer that is identical in type and structure to the buffer that this remote service expects. If you do not specify `TA_DMINRECTYPE`, the type of buffer is unchanged.

TA_DMOUTRECTYPE: *string*[1..78]

type[:subtype]—Specifies the type, optionally followed by subtype, of the buffer sent by this remote service. This attribute is used to enforce stronger type checking.

TA_DMTPSUTTYPE: *string*"{INTEGER | PRINTABLESTRING}"

Specifies the type of encoding to be performed on the `TA_DMREMPSTUT` value for this remote service. "INTEGER" and "PRINTABLESTRING" are ASN.1 types. The default is "PRINTABLESTRING".

TA_DMREMPSTUT: *string*[1..64]

Identifies the TP service user title for the remote system providing this remote service. Some users of OSI TP implementations require this attribute. It is not required for OS 2200 OLTP-TM2200, OpenTI, A Series Open/OLTP, or BEA eLink OSI TP. If the `TA_DMTPSUTTYPE` value is "PRINTABLESTRING", the maximum length is 60 characters, which must comply with the ASN.1 type of PRINTABLESTRING. If the `TA_DMTPSUTTYPE` value is "INTEGER", the maximum length must fit into a LONG. The value must be defined prior to defining the remote TPSUT.

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

Table 16 DM_MIB(5): T_DM_LOCAL Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_DMACCESSPOINT(r)(k)(*)	string	rw-r--r--	string[1..30]	N/A
TA_DMACCESSPOINTID(r)	string	rw-r--r--	string[1..30]	N/A
TA_DMSRVGROUP(r)	string	rw-r--r--	string[1..30]	N/A
TA_DMTYPE	string	rw-r--r--	"{TDOMAIN SNAX OSITP OSITPX}"	"TDOMAIN"
TA_STATE(r)	string	rw-r--r--	GET: "VAL" SET: "{NEW INV}"	N/A N/A
TA_DMAUDITLOG	string	rw-r--r--	string[1..256] ^{Note 3}	N/A
TA_DMBLOCKTIME	short	rw-r--r--	0 <= num <= 32,767	TA_BLOCKTIME in T_DOMAIN ^{Note 1}
TA_DMTLOGDEV	string	rw-r--r--	string[1..256] ^{Note 3}	N/A
TA_DMTLOGNAME	string	rw-r--r--	string[1..30]	"DMTLOG"
TA_DMTLOGSIZE	long	rw-r--r--	1 <= num <= 2048	100
TA_DMMAXRAPTRAN	short	rw-r--r--	0 <= num <= 32,767	16
TA_DMMAXTRAN	short	rw-r--r--	0 <= num <= 32,767	TA_MAXGTT in T_DOMAIN ^{Note 2}
TA_DMSECURITY	string	rw-r--r--	"{NONE APP_PW DM_PW DM_USER_PW CLEAR SAFE PRIVATE}"	"NONE"

Table 16 DM_MIB(5): T_DM_LOCAL Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
Attributes available when TA_DMTYPE=TDOMAIN:				
TA_DMCONNECTION_POLICY	string	rwxr--r--	" {ON_DEMAND ON_STARTUP INCOMING_ONLY} "	"ON_DEMAND"
TA_DMMAXRETRY	long	rwxr--r--	0 <= num <= MAXLONG	0
TA_DMRETRY_INTERVAL	long	rwxr--r--	0 <= num <= MAXLONG	60
Attributes available when TA_DMTYPE=TDOMAIN:				
TA_DMCONNPRINCIPALNAME	string	rwxr--r--	string[0..511]	" "
TA_DMMACHINETYPE	string	rw-r--r--	string[0..15]	" "
Attributes available when TA_DMTYPE=SNAX:				
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.

Note 3 Maximum string length for this attribute is 78 bytes for BEA Tuxedo 8.0 or earlier.

Attribute Semantics

TA_DMACCESSPOINT: *string*[1..30]

The name of this T_DM_LOCAL entry—a user-specified local domain access point identifier (logical name) unique within the scope of the T_DM_LOCAL and T_DM_REMOTE access point names in this Domains configuration.

TA_DMACCESSPOINTID: *string*[1..30]

The identifier of the domain gateway group associated with this local domain access point for purposes of security when setting up connections to remote domains. This identifier is unique across all local and remote domain access points.

TA_DMSRVGROUP: *string*[1..30]

The group name of the domain gateway group (the name provided in the `GROUPS` section of the `TUXCONFIG` file) representing this local domain access point. There is a one-to-one relationship between a local domain access point and a gateway server group.

TA_DMTYPE: "{TDOMAIN | SNAX | OSITP | OSITPX}"

The type of domain for this local domain access point: "TDOMAIN" for a BEA Tuxedo domain, "SNAX" for an SNA domain, "OSITP" for an OSI TP 1.3 domain, or "OSITPX" for an OSI TP 4.0 or later domain. The presence or absence of other attributes depends on the value of this attribute.

Setting `TA_DMTYPE="OSITPX"` is supported only by BEA Tuxedo 8.0 or later software.

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..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

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

TA_DMBLOCKTIME: 0 <= num <= **32,767**

Specifies the maximum wait time allowed for a blocking call for this local domain access point. The value is 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 for the T_DOMAIN object. A blocking timeout condition implies that the affected service request has failed.

Be aware that *interdomain* transactions generate blocking timeout conditions when transaction duration exceeds the value of the TA_DMBLOCKTIME attribute. That is, for an interdomain transaction, if the value of the TA_DMBLOCKTIME attribute is less than (a) the value of the TA_TRANTIME attribute specified for the T_SERVICE object or (b) the timeout value passed in the tpbegin() call to start the transaction, the timeout for the transaction is reduced to the TA_DMBLOCKTIME value. In contrast, for *intradomain* transactions (that is, transactions handled within a single BEA Tuxedo domain), the value of the TA_BLOCKTIME attribute specified for the T_DOMAIN object has *no* effect on the timeout value of an intradomain transaction.

TA_DMTLOGDEV: string[1..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

The device (raw slice) or file containing the Domains transaction log (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.

If this attribute is not specified, the domain gateway group associated with this local domain access point is not allowed to process requests in transaction mode. Multiple local domain access points for the same machine can share the same BEA Tuxedo filesystem, but each local domain access point must have its own log (a table in the TA_DMTLOGDEV) named as specified by the TA_DMTLOGNAME keyword.

TA_DMTLOGNAME: string[1..30]

The 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 <= num <= **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.

TA_DMMAXRAPTRAN: 0 <= num <= **32,767**

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

TA_DM_MAXTRAN: 0 <= num <= 32,767

The maximum number of simultaneous transactions allowed for this local domain access point. This number must be greater than or equal to the T_DOMAIN:TA_MAXGTT attribute value.

TA_DM_SECURITY: "{NONE | APP_PW | DM_PW | DM_USER_PW}"

The type of security enabled for the domain gateway associated with this local domain access point. This attribute must be set to one of the following values:

"NONE"

No security is enabled.

"APP_PW"

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

"DM_PW"

Valid only when TA_DM_TYPE="TDOMAIN" or "OSITPX". Domain password-based security is enabled.

"DM_USER_PW"

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

Attributes available when TA_DM_TYPE=TDOMAIN

TA_DM_CONNECTION_POLICY: "{ON_DEMAND | ON_STARTUP | INCOMING_ONLY}"

Specifies the conditions under which the domain gateway associated with this local domain access points tries to establish connections to remote domains. 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 a `dmadmin(1) connect` command. The default setting for TA_DM_CONNECTION_POLICY attribute is "ON_DEMAND". The "ON_DEMAND" policy provides the equivalent behavior to previous releases, in which the TA_DM_CONNECTION_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 domains at gateway server initialization time. Remote services for a particular remote domain (that is, services advertised by the domain gateway) are advertised only if a connection is successfully established to the remote domain. Therefore, if there is no active connection to a remote domain, 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_DMRETRY_INTERVAL` attribute. Also, see the `TA_DMMAXRETRY` attribute.

`"INCOMING_ONLY"`

Means that a domain gateway does not attempt an initial connection to remote domains upon startup and that remote services are initially suspended. The domain gateway is available for incoming connections from remote domains, and remote services are advertised when the domain gateway receives an incoming connection or an administrative connection (using the `dmadmin(1) connect` command) is made. Connection retry processing is not allowed when the connection policy is `"INCOMING_ONLY"`.

`TA_DMMAXRETRY: 0 <= num <= MAXLONG`

The number of times that the domain gateway associated with this local domain access point tries to establish connections to remote domains. The minimum value is 0 and the maximum is `MAXLONG` (2147483647). `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 automatic retry mechanism. For other connection policies, automatic retries are disabled.

The `TA_DMMAXRETRY` attribute is valid only when the connection policy is `"ON_STARTUP"`.

`TA_DMRETRY_INTERVAL: 0 <= num <= MAXLONG`

The number of seconds that the domain gateway associated with this local domain access point waits between automatic attempts to establish a connection to remote domains. The minimum value is 0 and the maximum value is `MAXLONG` (2147483647). The default is 60. If `TA_DMMAXRETRY` is set to 0, 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.

Attributes available when `TA_DMTYPE=TDOMAIN`

`TA_DMCONNPRINCIPALNAME: string[0..511]`

The connection principal name identifier, which is the principal name used for verifying the identity of the domain gateway associated with this local domain access point when establishing a connection to a remote domain. This attribute applies only to domain gateways of type `TDOMAIN` 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 can be bypassed between the machine associated with this local domain access point and the machines associated with the remote domain access points. This attribute applies only to domain gateways of type TDOMAIN.

If TA_DMMACHINETYPE is not specified, the default is to turn encoding/decoding on. If the value set for the TA_DMMACHINETYPE attribute 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.

Attributes available when TA_DMTYPE=SNAX

TA_DMBLOB_SHM_SIZE: **1** <= *num* <= MAXLONG

Specifies the shared memory allocated to storing binary large object log information specific to this SNAX local domain access point. This attribute applies only to local domain access points and domain gateways of type SNAX.

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_DMSRVGROUP, TA_DMTYPE, TA_DMTLOGDEV, TA_DMTLOGNAME, TA_DMTLOGSIZE, TA_DMMAXRAPTRAN, TA_DMMAXTRAN, or TA_DMMACHINETYPE.

T_DM_OSITP Class Definition

Overview

The T_DM_OSITP class defines the OSI TP 1.3 protocol related configuration information for a specific local or remote domain access point.

Attribute Table

Table 17 DM_MIB(5): T_DM_OSITP Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_DMACCESSPOINT(r)(k)(*)	string	rw-r--r--	string[1..30]	N/A
TA_STATE(r)	string	rw-r--r--	GET: "VAL" SET: "{NEW INV}"	N/A N/A
TA_DMAPT(r)	string	rw-r--r--	string[1..78]	N/A
TA_DMAEQ(r)	string	rw-r--r--	string[1..78]	N/A
TA_DMNWDEVICE	string	rw-r--r--	string[1..78]	N/A
TA_DMACN	string	rw-r--r--	"{XATMI UDT}"	"XATMI"
TA_DMAPID	short	rw-r--r--	0 <= num <= 32767	N/A
TA_DMAEID	short	rw-r--r--	0 <= num <= 32767	N/A
TA_DMURCH	string	rw-r--r--	string[0..30]	N/A
TA_DMMAXLISTENINGEP	short	rw-r--r--	1 <= num <= 32767	3
TA_DMXXATMIENCODING	string	rw-r--r--	"{CAE PRELIMINARY 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 this local or remote domain access point in object identifier form.

TA_DMAEQ: *string*[1..78]

The application entity qualifier of this local or remote domain access point in integer form.

TA_DMNWDEVICE: *string*[**1..78**]

Specifies the network device to be used for this local domain access point. This attribute is relevant only when defining a local domain access point; it is ignored for a remote domain access point.

TA_DMACN: "{XATMI | UDT}"

The application context name to use with this local or remote 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 local or remote domain access point.

TA_DMAEID: 0 <= num <= 32767

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

TA_DMURCH: *string*[**0..30**]

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

This attribute is relevant only when defining a local domain access point; it is ignored for a remote domain access point.

TA_DMMAXLISTENINGEP: 0 <= num <= 32767

Specifies the number of endpoints awaiting incoming OSI TP dialogues for this local domain access point. This attribute is relevant only when defining a local domain access point; it 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. This attribute is valid only when describing a remote domain access point. Valid values are:

"CAE" (default)

"PRELIMINARY" (used with Unisys MCP OLTP systems)

"OLTP_TM2200" (used with Unisys TM 2200 systems)

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 group associated with the local domain access point is active.
- The instance of the class corresponds to a remote domain access point and the domain gateway group associated with the remote domain access point is active.

On SET operations that add or update an instance of this class, the specific local or remote domain access point specified in the TA_DMACCESSPOINT attribute must exist in the T_DM_LOCAL class or the T_DM_REMOTE class. If the domain access point does not exist, a "not defined" error is returned for the TA_DMACCESSPOINT attribute, and the operation fails.

T_DM_OSITPX Class Definition

Overview

The T_DM_OSITPX class defines the OSI TP 4.0 or later protocol related configuration information for a specific local or remote domain access point. The T_DM_OSITPX class is supported only by BEA Tuxedo 8.0 or later software.

Attribute Table

Table 18 DM_MIB(5): T_DM_OSITPX Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_DMACCESSPOINT(r)(k)(*)	string	rw-r--r--	string[1..30]	N/A
TA_STATE(r)	string	rw-r--r--	GET: "VAL" SET: "{NEW INV}"	N/A N/A
TA_DMAET(r)	string	rw-r--r--	string[1..78]	N/A
TA_DMNWADDR(r)	string	rw-r--r--	string[1..631]	N/A

Table 18 DM_MIB(5): T_DM_OSITPX Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
TA_DMTSEL	string	rw-r--r--	<i>string</i> [1..66]	N/A
TA_DMDNSRESOLUTION	string	rw-r--r--	"{STARTUP RUNTIME}"	"STARTUP"
TA_DMPSEL	short	rw-r--r--	<i>string</i> [1..10]	" "
TA_DMSSEL	short	rw-r--r--	<i>string</i> [1..34]	" "
TA_DMTAILORPATH	short	rw-r--r--	<i>string</i> [1..78]	" "
TA_DMCHATMIENCODING	string	rw-r--r--	"{CAE PRELIMINARY OLTP_TM2200 NATIVE_A_SERIES}"	"CAE"
TA_DMEXTENSIONS	short	rw-r--r--	<i>string</i> [1..78]	" "
TA_DMOPTIONS	short	rw-r--r--	"{SECURITY_SUPPORTED}"	" "

(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_OSITPX 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_OSITPX object. The following states indicate the meaning of TA_STATE in a SET request. States not listed may not be set.

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

TA_DMAET: *string*[1..78]

The application entity title of this local or remote domain access point. This address must be unique among all hosts communicating in the OSI TP network; it matches the local AE Title on the remote (OLTP) node.

The value of this attribute consists of the application process title as an object identifier form followed by the application entity qualifier as an integer, using the following form: "{*object identifier*},{*integer qualifier*}". The braces are part of the syntax and must be included within the quotes.

TA_DMNWADDR: *string*[1..631]

The semicolon-separated list of network addresses to use for this local or remote domain access point. A network address may be either an IP address, if using TCP/IP networks, or a DNS name. The network address takes one of the following forms:

```
"#.#.#.#:port_number" IP Address  
"//hostname:port_number" DNS Name  
"//hostname:port_number;//hostname:port_number; ..."
```

If the port_number component is absent, the default port 102 is used.

For a local domain access point, the value of this attribute contains a semicolon-separated list of up to eight addresses on which to listen for connection requests. For a remote domain access point, the value of this attribute contains the preferred address for the

destination domain followed by up to seven alternative addresses (in preference order) to be tried if the first is unavailable.

TA_DMTSEL: *string*[1..66]

The Transport Service Access Point address to be used for this local or remote domain access point. The value may be one to 32 ASCII non-control characters (those represented by the hexadecimal numbers 20 to 7E), one to 32 hexadecimal octets preceded by 0x, or "NONE"—the NULL string.

TA_DMDNRESOLUTION: "{STARTUP | RUNTIME}"

Specifies when the DNS name for the network address defined by the TA_DMNWADDR attribute should be resolved for the domain gateway (GWOSITP) associated with this local domain access point. If this attribute is set (or defaulted) to "STARTUP", the resolution of hostname to an actual IP address takes place at gateway startup. If this attribute is set to "RUNTIME", the resolution of hostname to an actual IP address takes place at gateway run time.

This attribute is relevant only when defining a local domain access point; it is ignored for a remote domain access point. On GET calls for remote domain access point instances, this attribute is set to the NULL string.

TA_DMPSEL: *string*[1..10]

The Presentation Service Access Point address to be used for this local or remote domain access point. Values may be one to four ASCII non-control characters (those represented by the hexadecimal numbers 20 to 7E), one to four hexadecimal octets preceded by 0x, or "NONE" (default).

TA_DMSSEL: *string*[1..34]

The Session Service Access Point address to be used for this local or remote domain access point. Values may be one to 16 ASCII non-control characters (those represented by the hexadecimal numbers 20 to 7E), one to 16 hexadecimal octets preceded by 0x, or "NONE" (default).

TA_DMTAILORPATH: *string*[1..78]

Indicates the full pathname of the optional OSI TP tailor file used for tuning the OSI TP stack for this local domain access point. Double quotes are required. If no value is supplied or the value is set to the NULL string, the OSI TP stack will run using defaults for tuning parameters.

This attribute is relevant only when defining a local domain access point; it is ignored for a remote domain access point.

TA_DMXTMIENCODING: "{CAE | PRELIMINARY | OLTP_TM2200 | NATIVE_A_SERIES}"

Specifies the version of the XATMI protocol used to communicate with a remote system. This attribute is valid only when describing a remote domain access point. Valid values are:

"CAE" (default)

"PRELIMINARY" (used with Unisys MCP OLTP systems)

"OLTP_TM2200" (used with Unisys TM 2200 systems)

"NATIVE_A_SERIES" (used with Unisys MCP OLTP systems that support this encoding type)

TA_DMEXTENSIONS: *string*[1..78]

Controls operations for the remote domain associated with this remote domain access point. Valid values are separated by a semicolon (;) and include "ONLINE=N/Y" (Y is the default) and "RdomAssocRetry=nn", where *nn* is the number of seconds to retry connecting to the online remote domain. This attribute defaults to the `RdomAssocRetry` tailor parameter if present, or 60 seconds if `RdomAssocRetry` is not present and *nn* is not specified.

TA_DMOPTIONS: "{SECURITY_SUPPORTED}"

Indicates optional parameters for this remote domain access point. The "SECURITY_SUPPORTED" value indicates that the remote domain associated with this remote domain access point supports the OSITP security extension. This attribute provides backward compatibility; it is valid only when describing a remote domain access point.

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 group associated with the local domain access point is active.
- The instance of the class corresponds to a remote domain access point and the domain gateway group associated with the remote domain access point is active.

On SET operations that add or update an instance of this class, the specific local or remote domain access point specified in the `TA_DMACCESSPOINT` attribute must exist in the `T_DM_LOCAL` class or the `T_DM_REMOTE` class. If the domain 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 interdomain authentication through access points of type `TDOMAIN`.

Attribute Table

Table 19 DM_MIB(5): T_DM_PASSWORD Class Definition Attribute Table

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

The name of the local domain access point to which the password applies.

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

The name of the remote domain access point to which the password applies.

`TA_DMLPWD`: *string*[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: *string*[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 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

Table 20 DM_MIB(5): T_DM_PRINCIPAL_MAP Class Definition Attribute Table

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

The local domain access point to which the principal mapping applies.

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

The remote domain access point to which the principal mapping applies.

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

The local principal name in the principal mapping.

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

The remote principal name in the principal mapping.

TA_DMDIRECTION: "{IN | OUT | BOTH}"

The direction to which the principal mapping applies.

"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".
unset	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 or later, the `T_DM_PRINCIPAL_MAP` class applies only 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

Table 21 DM_MIB(5): T_DM_REMOTE Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
<code>TA_DMACCESSPOINT(r)(k)(*)</code>	string	rw-r--r--	<i>string</i> [1..30]	N/A
<code>TA_DMACCESSPOINTID(r)</code>	string	rw-r--r--	<i>string</i> [1..30]	N/A
<code>TA_DMTYPE(k)</code>	string	rw-r--r--	"{TDOMAIN SNAX OSITP OSITPX}"	"TDOMAIN"
<code>TA_STATE(r)</code>	string	rw-r--r--	GET: "VAL" SET: "{NEW INV}"	N/A N/A
<code>TA_DMPRIORITY_TYPE</code>	string	rw-r--r--	"{LOCAL_RELATIVE LOCAL_ABSOLUTE GLOBAL}"	"LOCAL_RELATIVE"
<code>TA_DMINPRIORITY</code>	string	rw-r--r--	-99 <= <i>num</i> <= 100	0 or 50
Attributes available when TA_DMTYPE=TDOMAIN OSITPX:				
<code>TA_DMACLPOLICY</code>	string	rw-r--r--	"{LOCAL GLOBAL}"	"LOCAL"
<code>TA_DMLOCALPRINCIPALNAME</code>	string	rw-r--r--	<i>string</i> [0..511]	" "
Attributes available when TA_DMTYPE=TDOMAIN:				
<code>TA_DMCONNPRINCIPALNAME</code>	string	rw-r--r--	<i>string</i> [0..511]	" "

Table 21 DM_MIB(5): T_DM_REMOTE Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
TA_DM_CREDENTIALPOLICY	string	rwxr--r--	"{LOCAL GLOBAL}"	"LOCAL"
TA_DM_MACHINE_TYPE	string	rw-r--r--	string[0..15]	" "
Attributes available when TA_DMTYPE=SNAX OSITPX:				
TA_DM_CODEPAGE	string	rw-r--r--	string[1..20]	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_DM_ACCESSPOINT: *string*[1..30]

The name of this T_DM_REMOTE entry—a user-specified remote domain access point identifier (logical name) unique within the scope of the T_DM_LOCAL and T_DM_REMOTE access point names in this Domains configuration.

TA_DM_ACCESSPOINTID: *string*[1..30]

The identifier for the remote domain associated with this remote domain access point for purposes of security when setting up a connection to the remote domain. This identifier is unique across all local and remote domain access points.

TA_DM_TYPE: "{TDOMAIN | SNAX | OSITP | OSITPX}"

The type of domain for this remote domain access point: "TDOMAIN" for a BEA Tuxedo domain, "SNAX" for an SNA domain, "OSITP" for an OSI TP 1.3 domain, or "OSITPX" for an OSI TP 4.0 or later domain. The presence or absence of other attributes depends on the value of this attribute.

Setting TA_DMTYPE="OSITPX" is supported only by BEA Tuxedo 8.0 or later software.

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.
-----------	------------------------

TA_DMPRIORITY_TYPE = "{LOCAL_RELATIVE | LOCAL_ABSOLUTE | GLOBAL}"

TA_DMINPRIORITY = **-99** <= num <= **100**

Together, the TA_DMPRIORITY_TYPE and TA_DMINPRIORITY attributes specify the message priority handling for this remote domain access point. These attributes are supported by BEA Tuxedo 8.0 or later software.

For the TA_DMPRIORITY_TYPE attribute, the "LOCAL_RELATIVE" and "LOCAL_ABSOLUTE" values are valid for all remote domain types; the "GLOBAL" value is valid only for remote domains of type TDOMAIN. If not set, the TA_DMPRIORITY_TYPE attribute defaults to "LOCAL_RELATIVE".

TA_DMPRIORITY_TYPE="LOCAL_RELATIVE" means that the priority associated with a request from this remote domain access point (for example, via the *tpsrio* call) is not used by the local domain. Instead, the priority of incoming requests from this remote domain access point is set relative to the TA_DMINPRIORITY value; this value may be greater than or equal to -99 (lowest priority) and less than or equal to 99 (highest priority), with 0 being the default. The setting of TA_DMINPRIORITY increments or decrements a service's default priority as follows: up to a maximum of 100 or down to a minimum of 1, depending on its sign, where 100 is the highest priority. For requests to the remote domain access point, the priority associated with a request will accompany the request to the remote domain access point.

TA_DMPRIORITY_TYPE="LOCAL_ABSOLUTE" means that the priority associated with a request from this remote domain access point is not used by the local domain. Instead, the priority of incoming requests from this remote domain access point is set relative to the TA_DMINPRIORITY value; this value may be greater than or equal to 1 (lowest priority) and less than or equal to 100 (highest priority), with 50 being the default. The setting of TA_DMINPRIORITY increments or decrements a service's default priority as follows: up to a maximum of 100 or down to a minimum of 1, depending on its sign, where 100 is the highest priority. For requests to the remote domain access point, the priority associated with a request will accompany the request to the remote domain access point.

TA_DMPRIORITY_TYPE="GLOBAL" means that the priority associated with a request from this remote domain access point is adjusted by the local domain. The priority of incoming requests from this remote domain access point is adjusted relative to the TA_DMINPRIORITY value; this value may be greater than or equal to -99 (lowest priority) and less than or equal to 99 (highest priority), with 0 being the default. If TA_DMINPRIORITY is set, the priority accompanying the incoming request is added to the TA_DMINPRIORITY value to create an absolute priority setting for the incoming request. If TA_DMINPRIORITY is not set or is set to 0, the priority accompanying the incoming request is used *as is* by the local domain. For requests to the remote domain access point, the priority associated with a request will accompany the request to the remote domain access point.

Attributes available when TA_DMTYPE=TDOMAINIOSITPX

TA_DMACLPOLICY: {LOCAL | GLOBAL}

The access control list (ACL) policy for this remote domain access point. This attribute applies only to domain gateways of type TDOMAIN running BEA Tuxedo 7.1 or later software and domain gateways of type OSITPX running BEA Tuxedo 8.0 or later software.

LOCAL means that the local domain *replaces* the credential (identity) of any service request received from the remote domain *with* the principal name specified in the TA_DMLOCALPRINCIPALNAME attribute for this remote domain access point. GLOBAL means that the local domain does not replace the credential received with a remote service request; if no credential is received with a remote service request, the local domain forwards the service request to the local service *as is* (which usually fails). If this attribute is not specified, the default is LOCAL.

Note that the TA_DMACLPOLICY attribute controls whether or not the local domain replaces the credential of a service request received from a remote domain with the principal name specified in the TA_DMLOCALPRINCIPALNAME attribute. The TA_DMCREENTIALPOLICY attribute is related to this attribute and controls whether or not the local domain removes the credential from a local service request before sending the request to a remote domain.

TA_DMLOCALPRINCIPALNAME: string[0..511]

The local principal name identifier (credential) assigned by the local domain to service requests received from the remote domain when the `TA_DMACLPOLICY` attribute for this remote domain access point is set (or defaulted) to `LOCAL`. This attribute applies only to domain gateways of type `TDOMAIN` running BEA Tuxedo 7.1 or later software and domain gateways of type `OSITPX` running BEA Tuxedo 8.0 or later software.

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.

Attributes available when `TA_DMTYPE=TDOMAIN`

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 applies only to domain gateways of type `TDOMAIN` 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_DMCREENTIALPOLICY: {LOCAL | GLOBAL}

The credential policy for this remote domain access point. This attribute applies only to domain gateways of type `TDOMAIN` running BEA Tuxedo 8.0 or later software.

`LOCAL` means that the local domain removes the credential (identity) from a local service request destined for this remote domain access point. `GLOBAL` means that the local domain does not remove the credential from a local service request destined for this remote domain access point. If this attribute is not specified, the default is `LOCAL`.

Note that the `TA_DM_CREDENTIALPOLICY` attribute controls whether or not the local domain removes the credential from a local service request before sending the request to a remote domain. The `TA_DM_ACLPOLICY` attribute controls whether or not the local domain replaces the credential of a service request received from a remote domain with the principal name specified in the `TA_DM_LOCALPRINCIPALNAME` attribute.

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

Used for grouping domains so that encoding/decoding of messages can be bypassed between the machine associated with this remote domain access point and the machine associated with the local domain access point. If `TA_DMMACHINETYPE` is not specified, the default is to turn encoding/decoding on. If the value set for the `TA_DMMACHINETYPE` attribute 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.

Attributes available when `TA_DM_TYPE=SNAXIOSITPX`

`TA_DM_CODEPAGE`: *string*[1..20]

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

Limitations

When any gateway administrative 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_DM_ACCESSPOINTID`, `TA_DM_TYPE`, `TA_DMMACHINETYPE`, or `TA_DM_CODEPAGE`.

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_OSITPX`, `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

Table 22 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				

Attribute Semantics

TA_DMVERSION: *string*[1..30]

A user-supplied identifier for the Domains configuration.

Limitations

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

Table 23 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

Table 23 DM_MIB(5): T_DM_ROUTING Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_DMRRANGES(r)	string	rw-r--r--	string[1..4096]	N/A
TA_STATE(r)	string	rw-r--r--	GET: "VAL" SET: "{NEW INV}"	N/A 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 Domains 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 attribute 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_DMANGES: *string*[1..4096]

The ranges and associated remote domain 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 `"*"` (wildcard) in the position of a range indicates any values not covered by the other ranges previously seen in the entry. Only one wildcard 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, 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 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.

<code>"VALid"</code>	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.

<code>"NEW"</code>	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

Table 24 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_DMRPRINNAME(r)(k)(*)	string	rw-----	string[1..30]	N/A
TA_DMRPRINPASSWD(r)(*)	string	-w-----	string[0..8]	N/A
TA_STATE(r)	string	rw-r--r--	GET: "VAL" SET: "{NEW INV}"	N/A 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_DMRAccessPOINT: *string[1..30]*

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

Note: The combination of TA_DMRACCESSPOINT and TA_DMRPRINNAME must be unique within the scope of TA_DM_RPRINCIPAL entries in the Domains configuration.

TA_DMRPRINNAME: *string*[1..30]

The remote principal name.

Note: The combination of TA_DMRACCESSPOINT and TA_DMRPRINNAME must be unique within the scope of TA_DM_RPRINCIPAL entries in the Domains configuration.

TA_DMRPRINPASSWD: *string*[0..8]

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 or later, 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

Table 25 DM_MIB(5): T_DM_SNACRM Class Definition Attribute Table

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

Attribute Semantics

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

The name of this `T_DM_SNACRM` entry. `TA_DMSNACRM` is an identifier unique within the scope of the SNA CRM entries within the Domains configuration used to identify this SNA CRM entry.

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

The name of the local domain access point entry with which this SNA CRM is used.

`TA_STATE`:

GET: "{VALid}"

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.

"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

Table 26 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" SET: "{NEW INV}"	N/A N/A
TA_DMSECTYPE	string	rw-r--r--	"{LOCAL IDENTIFY VERIFY PERSISTENT 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

TA_DMSNALINK: *string*[1..30]

The name of the T_DM_SNALINK entry. An identifier, unique within the scope of the SNA LINK entries within the Domains configuration, used to identify this TA_DMSNALINK entry.

TA_DMSNASTACK: *string*[1..30]

The name of the SNAX stack entry to be used to reach this remote domain access point.

TA_DMRACCESSPOINT: *string*[1..30]

Identifies the remote domain access point name for which this entry provides the SNAX configuration data.

TA_DMLSYSID: *string*[1..4]

The local SYSID to be used when establishing an SNA link to the remote logical unit (LU).

TA_DMRSYSID: *string*[1..4]

The remote SYSID to be used when establishing an SNA link to the remote LU.

TA_DMLUNAME: *string*[1..8]

Specifies the LU name associated with the remote domain access point.

TA_DMMINWIN: 0 <= num <= 32767

The minimum number of winner sessions to the remote LU.

TA_DMMODENAME: *string*[1..8]

Specifies the name associated with the session characteristics for sessions to the remote LU.

TA_STATE:

GET: "{VALid}"

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.

"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_DMMAXSNASESS: 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

Table 27 DM_MIB(5): T_DM_SNASTACK Class Definition Attribute Table

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

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 Domains configuration.

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

Identifies the `T_DM_SNACRM` entry of the SNA CRM in which this SNA protocol stack definition is used.

TA_DMSTACKTYPE: *string*[1..30]

Identifies the protocol stack to be used.

TA_DMLUNAME: *string*[1..8]

Specifies the LU name to be used on sessions established using this stack definition.

TA_DMTTPNAME: *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

Table 28 DM_MIB(5): T_DM_TDOMAIN Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_DMACCESSPOINT(r)(k)(*)	string	rw-r--r--	string[1..30]	N/A
TA_DMNWADDR(r)(k)(*)	string	rw-r--r--	string[1..256] ^{Note 1}	N/A
TA_STATE(r)	string	rw-r--r--	GET: "VAL" SET: "{NEW INV}"	N/A N/A
TA_DMNWDEVICE	string	rw-r--r--	string[1..78]	N/A
TA_DMCPLIMIT	long	rw-rw-r--	0 <= num <= MAXLONG	MAXLONG
TA_DMMINENCRYPTBITS	string	rw-----	"{0 40 56 128 256}" ^{Note 2}	"0"
TA_DMMAXENCRYPTBITS	string	rw-----	"{0 40 56 128 256}" ^{Note 2}	"128"
TA_DMCONNECTION_POLICY	string	rwxr--r--	"{LOCAL ON_DEMAND ON_STARTUP INCOMING_ONLY}"	"LOCAL" ^{Note 3} (Also see ^{Note 5})
TA_DMMAXRETRY	long	rwxr--r--	0 <= num <= MAXLONG	0
TA_DMRETRY_INTERVAL	long	rwxr--r--	0 <= num <= MAXLONG	60
TA_DMNW_PROTOCOL	string	rw-r--r--	"{SSL LLE}"	"LLE"
TA_DMSSL_RENEGOTIATION	long	rwxr--r--	0 <= num <= 2147483647	0

Table 28 DM_MIB(5): T_DM_TDOMAIN Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
TA_DMTCKEEPALIVE	string	rwxr--r--	"{LOCAL NO YES}"	"LOCAL" ^{Note 3} "NO" ^{Note 4}
TA_DMKEEPALIVE	long	rwxr--r--	-1 <= num <= 2147483647	-1 ^{Note 3} 0 ^{Note 4}
TA_DMKEEPALIVEWAIT	long	rwxr--r--	0 <= num <= 2147483647	0
TA_DMLACCESSPOINT(r)(k)(*)	string	rw-r--r--	string[1..30]	"*"
TA_DMFAILOVERSEQ	short	rw-r--r--	-1 <= num <= 32767	-1

(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} Maximum string length for this attribute is 78 bytes for BEA Tuxedo 8.0 or earlier.

^{Note 2} Link-level encryption value of 40 bits is provided for backward compatibility.

^{Note 3} Default for remote domain access points.

^{Note 4} Default for local domain access points.

^{Note 5} Default TA_DMCONNECTION_POLICY value for a local domain access point is the TA_DMCONNECTION_POLICY value specified in the T_DM_LOCAL class.

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 Domains link-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..256] (**up to 78 bytes for BEA Tuxedo 8.0 or earlier**)

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_DMCPLIMIT: 0 <= num <= MAXLONG

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

TA_DMMINENCRYPTBITS: "{0 | 40 | 56 | 128 | 256}"

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", "128" and "256" specify the encryption length (in bits). If this minimum level of encryption is not met, link establishment fails. The default is "0".

The value of 40 bits is provided for backward compatibility. 256-bit encryption is currently possible only when using SSL.

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

TA_DM_MAXENCRYPTBITS: "{0 | 40 | 56 | 128 | 256}"

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", "128" and "256" specify the encryption length (in bits). The default is "128".

The value of 40 bits is provided for backward compatibility. 256-bit encryption is currently possible only when using SSL.

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

TA_DM_CONNECTION_POLICY = "{LOCAL | ON_DEMAND | ON_STARTUP | INCOMING_ONLY}"

Specifies the conditions under which the TDomain gateway associated with this local or remote domain access point tries to establish connections. Supported values are "LOCAL", "ON_DEMAND", "ON_STARTUP", or "INCOMING_ONLY". "LOCAL" is relevant only to remote domain access points.

The TA_DM_CONNECTION_POLICY attribute is available in the T_DM_TDOMAIN class when running BEA Tuxedo 8.1 or later software. Its value in the T_DM_TDOMAIN class for a particular local or remote domain access point takes precedence over its global value in the T_DM_LOCAL class. The ability to override the global connection policy enables you to configure connection policy on a per remote domain basis.

Specifying no connection policy for a *local domain access point* defaults to the global connection policy specified in the T_DM_LOCAL class. If you choose to specify a global connection policy in the T_DM_TDOMAIN class, do not specify a global connection policy in the T_DM_LOCAL class.

"LOCAL"

A connection policy of "LOCAL" means that a remote domain access point accepts the global connection policy specified in the T_DM_LOCAL class. "LOCAL" is the default connection policy for remote domain access points. Excluding "LOCAL", the connection policy value for a remote domain access point takes precedence over the connection policy value for a local domain access point.

"ON_DEMAND"

A connection policy of "ON_DEMAND" means that the TDomain gateway attempts a connection only when requested by either a client request to a remote service or a `dmadmin(1) connect` command. Connection retry processing is not allowed when the connection policy is "ON_DEMAND".

"ON_STARTUP"

A connection policy of "ON_STARTUP" means that the TDomain gateway attempts to establish a connection at gateway server initialization time. For "ON_STARTUP", the remote services for a particular remote domain (that is, services advertised by the TDomain gateway) are advertised only if a connection is successfully established to the remote domain. Thus, if there is no active connection to the remote domain, the remote services are suspended. By default, this connection policy retries failed connections every 60 seconds, but you can specify a different value for this interval using the `TA_DMRETRY_INTERVAL` attribute in the `T_DM_TDOMAIN` class. Also, see the `TA_DMMAXRETRY` attribute in this class.

"INCOMING_ONLY"

A connection policy of "INCOMING_ONLY" means that the TDomain gateway does not attempt an initial connection upon startup and that remote services are initially suspended. The TDomain gateway is available for incoming connections from a remote domain, and remote services are advertised when the gateway receives an incoming connection or an administrative connection (using the `dmadmin(1)` `connect` command) is made. Connection retry processing is not allowed when the connection policy is "INCOMING_ONLY".

`TA_DMFAILOVERSEQ = -1 <= num <= 32767`

Specifies or requests failover sequences and primary records for a TDomain session record in the `BDMCONFIG` file. If a `DM_MIB SET` request does not specify a `TA_DMFAILOVERSEQ` value or a `DM_MIB SET TA_DMFAILOVERSEQ` request is from Tuxedo releases prior to 9.0, the output TDomain session record in the `BDMCONFIG` file uses the default `FAILOVERSEQ = -1`.

The record with the lowest `FAILOVERSEQ` value is the primary record for that TDomain session. There is only one primary record for a TDomain session, all remaining records for the same TDomain session are called secondary/backup records. With the exceptions of `NWADDR`, `NWDEVICE`, and `FAILOVERSEQ`, the primary record is the source for all TDomain session configuration parameters and attributes. All other parameters and attributes listed in secondary/backup records are ignored.

Based on the `CONNECTION_POLICY` attribute you select, the local domain will try to connect to a TDomain session's primary record. If the primary record has a failover, it will then try to connect to the next sequential secondary/backup record. If all secondary record connections fail, it will retry the primary record information at a later time as determined by `RETRY_INTERVAL` until `MAXRETRY` is exhausted.

`TA_DMLACCESSPOINT: string[1..30]`

Specifies or requests a local domain access point found in the `DM_LOCAL` section for a TDomain session record in the `BDMCONFIG` file. The `TA_DMLACCESSPOINT` parameter is

used exclusively to define TDomain session gateways and can contain only one local domain accesspoint as its value.

If a DM_MIB SET request does not specify a TA_DMLACCESSPOINT value or a DM_MIB SET TA_DMLACCESSPOINT request is from Tuxedo releases prior to 9.0, the output TDomain session record in the BDMCONFIG file uses the default LACCESPOINT = " * " .

Note: DM_MIB *does not* allow regular expression use with TA_DMLACCESSPOINT.

TA_DMMAXRETRY: 0 <= num <= MAXLONG

The number of times that the TDomain gateway associated with this local or remote domain access point tries to establish a connection. This attribute is available in the T_DM_TDOMAIN class when running BEA Tuxedo 8.1 or later software, and is valid when the TA_DMCONNECTION_POLICY attribute for this access point is set to "ON_STARTUP". For other connection policies, automatic retries are disabled.

The minimum value for TA_DMMAXRETRY is 0, and the maximum value is MAXLONG (2147483647). MAXLONG, the default, indicates that retry processing will be repeated indefinitely, or until a connection is established.

TA_DMRETRY_INTERVAL: 0 <= num <= MAXLONG

The number of seconds that the TDomain gateway associated with this local or remote domain access point waits between automatic attempts to establish a connection. This attribute is available in the T_DM_TDOMAIN class when running BEA Tuxedo 8.1 or later software, and is valid when the TA_DMCONNECTION_POLICY attribute for this access point is set to "ON_STARTUP". For other connection policies, automatic retries are disabled.

The minimum value for TA_DMRETRY_INTERVAL is 0, and the maximum value is MAXLONG (2147483647). The default is 60. If TA_DMMAXRETRY is set to 0, setting TA_DMRETRY_INTERVAL is not allowed.

TA_DMNW_PROTOCOL = "{LLE | SSL | SSL_ONE_WAY}"

Specifies SSL, LLE, or one-way SSL encryption. The default value is "LLE." The SSL option requires the domains at both end of the connection to authenticate each other; the SSL_ONE_WAY option does not.

If SSL_ONE_WAY is set, the domain that *accepts* an SSL connection needs to authenticate itself to the domain that *initiates* the connection using an SSL certificate. The initiating domain *does not* need to authenticate itself to the other domain. This value is mainly intended for use with a CONNECTION_POLICY to INCOMING_ONLY, and should only be set when the domain that accepts incoming connections *does not* need to authenticate connecting domains.

Note: If `TA_DMNW_PROTOCOL` is not set or set to `LLE` and `TA_DMSSL_RENEGOTIATION` is set to a non-zero value, the MIB call prints a warning message; however, the requested values are still set. The MIB operation then returns `TAUPDATED` or `TAOK` (unless some other error occurs).

`TA_DMSSL_RENEGOTIATION = 0 <= num <= 2147483647`

Specifies the renegotiation interval (in seconds) for SSL information. It must be greater than or equal to 0 and less than or equal to 2,147,483,647. The default value is 0 (which indicates that no renegotiation takes place).

Changes made to this parameter for a running `GWTDOMAIN` will take effect during the next renegotiation interval.

Note: If `TA_DMNW_PROTOCOL` is not set or set to `LLE` and `TA_DMSSL_RENEGOTIATION` is set to a non-zero value, the MIB call prints a warning message; however, the requested values are still set. The MIB operation then returns `TAUPDATED` or `TAOK` (unless some other error occurs).

`TA_DMTCPKEEPALIVE = "{LOCAL | NO | YES}"`

Enables TCP-level keepalive for this local or remote domain access point. Supported values are "LOCAL", "NO", or "YES". "LOCAL" is relevant only to remote domain access points.

The `TA_DMTCPKEEPALIVE` attribute applies only to domain gateways of type `TDOMAIN` running BEA Tuxedo 8.1 or later software. Its value for a remote domain access point takes precedence over its value for a local domain access point. The ability to override the local domain access point value enables you to configure TCP-level keepalive on a per remote domain basis.

A value of "LOCAL" means that a remote domain access point accepts the TCP-level keepalive value defined for the local domain access point. "LOCAL" is the default TCP-level keepalive value for remote domain access points.

A value of "NO" means that TCP-level keepalive is disabled for this access point. "NO" is the default TCP-level keepalive value for local domain access points.

A value of "YES" means that TCP-level keepalive is enabled for this access point. When TCP-level keepalive is enabled for a connection, the keepalive interval used for the connection is the system-wide value configured for the operating system's TCP keepalive timer. This interval is the maximum time that the TDomain gateway will wait without receiving any traffic on the connection. If the maximum time is exceeded, the gateway sends a TCP-level keepalive request message. If the connection is still open and the remote TDomain gateway is still alive, the remote gateway responds by sending an acknowledgement. If the local TDomain gateway does not receive an acknowledgement

within a fixed period of time of sending the request message, it assumes that the connection is broken and releases any resources associated with the connection.

Not only does TCP-level keepalive keep BEA Tuxedo interdomain connections open during periods of inactivity, but it also enable TDomain gateways to quickly detect connection failures.

Note: The `TA_DMTCPKEEPALIVE` and `TA_DMKEEPALIVE` attributes are *not* mutually exclusive, meaning that you can configure an interdomain connection using both attributes.

`TA_DMKEEPALIVE = -1 <= num <= 2147483647`

Controls application-level keepalive for this local or remote domain access point. This value must be greater than or equal to -1 and less than or equal to 2147483647. The value -1 is relevant only to remote domain access points.

The `TA_DMKEEPALIVE` attribute applies only to domain gateways of type `TDOMAIN` running BEA Tuxedo 8.1 or later software. Its value for a remote domain access point takes precedence over its value for a local domain access point. The ability to override the local domain access point value enables you to configure application-level keepalive on a per remote domain basis.

A value of -1 means that a remote domain access point accepts the application-level keepalive value defined for the local domain access point. -1 is the default application-level keepalive value for remote domain access points.

A value of 0 means that application-level keepalive is disabled for this access point. 0 is the default application-level keepalive value for local domain access points.

A value greater than or equal to 1 and less than or equal to 2147483647, in milliseconds, currently rounded up to the nearest second by the Domains software, means that application-level keepalive is enabled for this access point. The time that you specify is the maximum time that the TDomain gateway will wait without receiving any traffic on the connection. If the maximum time is exceeded, the gateway sends an application-level keepalive request message. If the connection is still open and the remote TDomain gateway is still alive, the remote gateway responds by sending an acknowledgement. If the local TDomain gateway does not receive an acknowledgement within a configurable period of time (see the `TA_DMKEEPALIVEWAIT` attribute) of sending the request message, it assumes that the connection is broken and releases any resources associated with the connection.

Not only does application-level keepalive keep BEA Tuxedo interdomain connections open during periods of inactivity, but it also enable TDomain gateways to quickly detect connection failures.

Note: The `TA_DMKEEPALIVE` and `TA_DMTCPKEEPALIVE` attributes are *not* mutually exclusive, meaning that you can configure an interdomain connection using both attributes.

`TA_DMKEEPALIVEWAIT = 0 <= num <= 2147483647`

Specifies the maximum time for this local or remote domain access point that the TDomain gateway will wait without receiving an acknowledgement to a sent keepalive message. This value must be greater than or equal to 0 and less than or equal to 2147483647, in milliseconds, currently rounded up to the nearest second by the Domains software. The default is 0. This attribute applies only to domain gateways of type `TDOMAIN` running BEA Tuxedo 8.1 or later software.

If `TA_DMKEEPALIVE` is 0 (keepalive disabled) for this access point, setting `TA_DMKEEPALIVEWAIT` has no effect.

If `TA_DMKEEPALIVE` is enabled for this access point and `TA_DMKEEPALIVEWAIT` is set to a value greater than `TA_DMKEEPALIVE`, the local TDomain gateway will send more than one application-level keepalive message before the `TA_DMKEEPALIVEWAIT` timer expires. This combination of settings is allowed.

If `TA_DMKEEPALIVE` is enabled for this access point and `TA_DMKEEPALIVEWAIT` is set to 0, receiving an acknowledgement to a sent keepalive message is unimportant: any such acknowledgement is ignored by the TDomain gateway. The gateway continues to send keepalive messages every time the `TA_DMKEEPALIVE` timer times out. *Use this combination of settings to keep an idle connection open through a firewall.*

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_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_DMTPTTRANID, TA_DMTXACCESSPOINT and TA_DMTXNETTRANID may be supplied to select a particular transaction.

Attribute Table

Table 29 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_DMTPTTRANID(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}" SET: "INV"	N/A 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
Per branch attributes:				
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

Table 29 DM_MIB(5): T_DM_TRANSACTION Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
TA_DMBRANCHSTATE	string	r--r--r--	GET: "{ABD ABY ACT COM DEC DON HAB HCO HHZ HMI REA UNK}"	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_DMLACCESSPOINT: *string*[1..30]

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.
"UNKnown"	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 or later, 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 run-time 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.
"UNKknown"	The state of the transaction could not be determined.

Note: This attribute is available only to gateways running BEA Tuxedo release 7.1 or later, and is set to "UNKknown" 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.

DM_MIB(5) Additional Information

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 ATMI Application Using C

Programming a BEA Tuxedo ATMI Application Using FML

EVENTS(5)

Name

EVENTS—List of system-generated events

Description

The System Event Monitor feature detects and reports certain predefined 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.

The BEA Tuxedo system periodically checks system capacities. If the system finds that a resource is exhausted or near capacity, it posts a system `WARN` or `ERROR` event. The system will continue to post these events until the condition subsides.

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 (`.`).

Limitations

Event reporting is currently limited to classes defined in [TM_MIB\(5\)](#) and the `T_DM_CONNECTION` class defined in [DM_MIB\(5\)](#). Event reporting uses the MIB information base. See [MIB\(5\)](#) and [TM_MIB\(5\)](#) for a definition and the availability of “local attributes,” and be aware that the availability of a local attribute depends on the state of communication within the application's network.

It is possible that the system will not post an event related to a system capacity limit (for example, `.SysMachineFullMaxgtt`) if the condition only exists for a very short period of time.

Generic Event Reporting Fields

`TA_OPERATION`: *string*

The literal string `EVT`, which identifies this buffer as an event report notification.

`TA_EVENT_NAME`: *string*

A string that uniquely identifies this event. All system-generated events begin with `.Sys`.

`TA_EVENT_SEVERITY`: *string*

The string `ERROR`, `WARN`, or `INFO`, to indicate the severity of this event.

`TA_EVENT_LMID`: *string*

A string identifying the machine where the event was detected.

TA_EVENT_TIME: *long*

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

TA_EVENT_USEC: *long*

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

TA_EVENT_DESCRIPTION: *string*

A one-line string summarizing the event.

TA_CLASS: *string*

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

TA_ULOGCAT: *string*

Catalog name from which the message was derived, if any.

TA_ULOGMSGNUM: *num*

Catalog message number, if the message was derived from a catalog.

Event Lists

T_ACLPERM Event List

.SysAclPerm

INFO: .SysAclPerm: system ACL permission change

T_DOMAIN Event List

.SysResourceConfig

INFO: .SysResourceConfig: system configuration change

.SysLicenseInfo

INFO: .SysLicenseInfo: reached 100% of Tuxedo System Licensed User Count, DBBL/BBL lockout canceled Binary

.SysLicenseInfo: reached 90% of Tuxedo System Licensed User Count Binary

.SysLicenseInfo: reached 90% of Tuxedo System Licensed User Count, DBBL/BBL lockout canceled Binary

.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_DM_CONNECTION Event List

.SysConnectionSuccess
    INFO: .SysConnectionSuccess: Connection successful between
    %TA_DMLACCESSPOINT and %TA_DMRACCESSPOINT

.SysConnectionConfig
    INFO: .SysConnectionConfig: Configuration change for connection
    between %TA_DMLACCESSPOINT and %TA_DMRACCESSPOINT

.SysConnectionDropped
    INFO: .SysConnectionDropped: Connection dropped between
    %TA_DMLACCESSPOINT and %TA_DMRACCESSPOINT

.SysConnectionFailed
    INFO: .SysConnectionFailed: Connection failed between
    %TA_DMLACCESSPOINT and %TA_DMRACCESSPOINT

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

.SysServerTpxit
ERROR: .SysServerTpxit: %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\(5\)](#), [DM_MIB\(5\)](#)

EVENT_MIB(5)

Name

EVENT_MIB—Management Information Base for EventBroker

Synopsis

```
#include <tpadm.h>
#include <fm132.h>
#include <evt_mib.h>
```

Description

The BEA Tuxedo EventBroker MIB defines the set of classes through which the EventBroker 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. For additional information pertaining to all EVENT_MIB(5) class definitions, see [“EVENT_MIB\(5\) Additional Information” on page 185](#).

EVENT_MIB consists of the following classes.

Table 30 EVENT_MIB Classes

Class Name	Attributes
T_EVENT_CLIENT	Subscriptions that trigger unsolicited notification
T_EVENT_COMMAND	Subscriptions that trigger system commands
T_EVENT_QUEUE	Subscriptions for queue-based notification
T_EVENT_SERVICE	Subscriptions for server-based notification
T_EVENT_USERLOG	Subscriptions for writing <code>userlog</code> 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 `EventBroker` 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`, the event buffer is sent to the specified client's unsolicited message handling routine.

Attribute Table

Table 31 T_EVENT_CLIENT Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_EVENT_EXPR(r) (*)	string	R--R--R--	<i>string</i> [1..255]	N/A
TA_EVENT_FILTER(k)	string	R--R--R--	<i>string</i> [1..255]	none
TA_EVENT_FILTER_BINARY(k)	carray	R--R--R--	<i>carray</i> [1..64000]	none
TA_STATE(r)	string	R-xR-xR-x	GET: ACT SET: {NEW INV}	N/A N/A
TA_CLIENTID(r) (*)	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.

Attribute Semantics

TA_EVENT_EXPR: *string*[1..255]

Event pattern expression. This expression, in regular expression format, controls which event names match this subscription.

TA_EVENT_FILTER: *string*[1..255]

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

TA_EVENT_FILTER_BINARY: *carray*[1..64000]

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

TA_STATE:

GET: ACTive

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

SET: {NEW | INValid}

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

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

TA_CLIENTID: *string*[1..78]

Send an unsolicited notification message to this client when a matching event is detected.

T_EVENT_COMMAND Class Definition

Overview

The T_EVENT_COMMAND class represents a set of subscriptions registered with the EventBroker 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, the event buffer is formatted and passed to the system's command interpreter.

Attribute Table

Table 32 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 SET: {NEW INV}	N/A N/A

Table 32 T_EVENT_COMMAND Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
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.

Attribute Semantics

TA_EVENT_EXPR: *string*[1..255]

Event pattern expression. This expression, in regular expression format, controls which event names match this subscription.

TA_EVENT_FILTER: *string*[1..255]

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

TA_EVENT_FILTER_BINARY: *carray*[1..64000]

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

TA_STATE:

GET: Active

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

SET: {NEW | INVALID}

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

NEW	Create <code>T_EVENT_COMMAND</code> object. Successful return leaves the object in the <code>ACTIVE</code> state.
INVALID	Delete <code>T_EVENT_COMMAND</code> object. Successful return leaves the object in the <code>INVALID</code> 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 `EventBroker` 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`, the event buffer is stored in the specified reliable queue.

Attribute Table

Table 33 T_EVENT_QUEUE Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
<code>TA_EVENT_EXPR(r)</code> (*)	string	R-----	<i>string</i> [1..255]	N/A
<code>TA_EVENT_FILTER(k)</code>	string	R-x-----	<i>string</i> [1..255]	none
<code>TA_EVENT_FILTER_BINARY(k)</code>	carray	R-x-----	<i>carray</i> [1..64000]	none
<code>TA_STATE(r)</code>	string	R-x-----	GET: ACT SET: {NEW INV}	N/A N/A

Table 33 T_EVENT_QUEUE Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
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 Semantics

TA_EVENT_EXPR: *string*[1..255]

Event pattern expression. This expression, in regular expression format, controls which event names match this subscription.

TA_EVENT_FILTER: *string*[1..255]

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

TA_EVENT_FILTER_BINARY: *carray*[1..64000]

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

TA_STATE:

GET: *ACTive*

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

SET: {*NEW | INValid*}

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

<i>NEW</i>	Create <code>T_EVENT_QUEUE</code> object. Successful return leaves the object in the <i>ACTive</i> state.
<i>INValid</i>	Delete <code>T_EVENT_QUEUE</code> object. Successful return leaves the object in the <i>INValid</i> state.

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

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

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

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

TA_QCTL_QTOP: *short*

This value, if present, is passed in to `tpenqueue()`'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 `tppost()` 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 EventBroker 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`, the event buffer is sent to the specified BEA Tuxedo service routine.

Attribute Table

Table 34 T_EVENT_SERVICE Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_EVENT_EXPR(r) (*)	string	R--R--R--	<i>string</i> [1..255]	N/A
TA_EVENT_FILTER(k)	string	R--R--R--	<i>string</i> [1..255]	none
TA_EVENT_FILTER_BINARY(k)	carray	R--R--R--	<i>carray</i> [1..64000]	none
TA_STATE(r)	string	R-xR-xR-x	GET: ACT SET: {NEW INV}	N/A N/A
TA_SERVICENAME(r) (*)	string	R--R--R--	<i>string</i> [1..15]	N/A
TA_EVENT_PERSIST	short	R-xR-xR-x	<i>short</i>	0
TA_EVENT_TRAN	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 regular expression 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 `tpost()` 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 EventBroker 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, the event buffer is formatted and passed to the BEA Tuxedo `userlog(3c)` function.

Attribute Table

Table 35 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 SET: {NEW INV}	N/A 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.

Attribute Semantics

TA_EVENT_EXPR: *string*[1..255]

Event pattern expression. This expression, in regular expression format, controls which event names match this subscription.

TA_EVENT_FILTER: *string*[1..255]

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

TA_EVENT_FILTER_BINARY: *carray*[1..64000]

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

TA_STATE:

GET: ACTIVE

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

SET: {NEW | INValid}

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

NEW	Create T_EVENT_USERLOG object. Successful return leaves the object in the ACTIVE state.
-----	---

INValid	Delete T_EVENT_USERLOG object. Successful return leaves the object in the INValid state.
---------	--

TA_USERLOG: *string*[1..255]

Write a `userlog(3c)` message when a matching event is detected.

EVENT_MIB(5) Additional Information

Files

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

See Also

[EVENTS\(5\)](#), [TM_MIB\(5\)](#)

factory_finder.ini(5)

Name

`factory_finder.ini`—FactoryFinder Domains configuration file

Description

`factory_finder.ini` is the FactoryFinder configuration file for Domains. This text (ASCII) file is parsed by the `TMFFNAME` service when it is started as a Master NameManager. The file contains information used by NameManagers to control the import and the export of object references for factory objects with other domains. To use the information in the `factory_finder.ini` file, you must specify the `factory_finder.ini` file in the `-f` option of the `TMFFNAME` server process.

The FactoryFinder Domains configuration file may have any name as long as the content of the file conforms to the format described on this reference page.

Definitions

A BEA Tuxedo domain is defined as the environment described in a single `TUXCONFIG` file. A BEA Tuxedo domain can communicate with another BEA Tuxedo domain or with another TP application—an application running on another TP system—via a domain gateway group. In BEA Tuxedo terminology, a *domain* is the same as an *application*—a business application.

A Remote Factory is a factory object that exists in a remote domain that is made available to the application through a BEA Tuxedo FactoryFinder.

A Local Factory is a factory object that exists in the local domain that is made available to remote domains through a BEA Tuxedo FactoryFinder.

File Format

The file is made up of two specification sections. Allowable section names are:

`DM_REMOTE_FACTORIES` and `DM_LOCAL_FACTORIES`.

- Formatting Guidelines

Parameters are generally specified by: `KEYWORD = value`, which sets `KEYWORD` to `value`. Valid keywords are described within each section. `KEYWORDS` are reserved; they cannot be used as values, unless they are quoted.

If a value is an identifier, standard C rules are used. An identifier must start with an alphabetic character or underscore and must contain only alphanumeric characters or underscores. An identifier cannot be the same as any `KEYWORD`.

A value that is not an identifier must be enclosed in double quotes.

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

The # character introduces a comment. A newline ends a comment.

Blank lines and comments are ignored.

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

- **DM_LOCAL_FACTORIES** section

This section provides information about the factories exported by each local domain. This section is optional; if it is not specified, all local factory objects can be exported to remote domains. If this section is specified, it should be used to restrict the set of local factory objects that can be retrieved from a remote domain. The reserved *factory_id.factory_kind* identifier of NONE can be used to restrict any local factory from being retrieved from a remote domain.

Lines within this section have the form:

```
factory_id.factory_kind
```

where *factory_id.factory_kind* is the local name (identifier) of the factory. This name must correspond to the identifier of a factory object registered by one or more BEA Tuxedo server applications with the BEA Tuxedo FactoryFinder.

The *factory_kind* must be specified for TMFFNAME to locate the appropriate factory. An entry that does not contain a *factory_kind* value does not default to a value of `FactoryInterface`.

- **DM_REMOTE_FACTORIES** section

This section provides information about factory objects imported and available on remote domains. Lines within this section have the form:

```
factory_id.factory_kind required_parameters
```

where *factory_id.factory_kind* is the name (identifier) of the factory object used by the local BEA Tuxedo domain for a particular remote factory object. Remote factory objects are associated with a particular remote domain.

Note: If you use the `TobjFactoryFinder` interface, the *factory_kind* must be `FactoryInterface`.

The required parameter is:

```
DOMAINID = domain_id
```

This parameter specifies the identity of the remote domain in which the factory object is to be retrieved. The *domain_id* must not be greater than 32 octets in length. If the value is a string, it must be 32 characters or fewer (counting the trailing `NULL`). The value of *domain_id* can be a sequence of characters or a sequence of hexadecimal digits preceded by `0x`.

The optional parameter is:

```
RNAME = string
```

This parameter specifies the name exported by remote domains. This value will be used by a remote domain to request this factory object. If this parameter is not specified, the remote factory object name is the same as the named specified in *factory_id.factory_kind*.

Multiple entries with the same name can be specified as long as the values associated with either the `DOMAINID` or `RNAME` parameter results in the identification of a unique factory object.

Examples

- Example 1

The following FactoryFinder Domains configuration file defines two entries for a factory object that will be known in the local domain by the identifier `Teller.FactoryIdentity` that is imported from two different remote domains:

```
# BEA Tuxedo FactoryFinder Domains
# Configuration File
#
*DM_REMOTE_FACTORIES
  Teller.FactoryIdentity
    DOMAINID="Northwest"
    RNAME=Teller.FactoryType
  Teller.FactoryIdentity
    DOMAINID="Southwest"
```

In the first entry, a factory object is to be imported from the remote domain with an identity of `Northwest` that has been registered with a factory identity of `Teller.FactoryType`.

In the second entry, a factory object is to be imported from the remote domain with an identity of `Southwest` that has been registered with a factory identity of `Teller.FactoryIdentity`. Note that because no `RNAME` parameter was specified, the name of the factory object in the remote domain is assumed to be the same as the factory's name in the local domain.

- Example 2

The following FactoryFinder Domains configuration file defines that only factory objects registered with the identity of `Teller.FactoryInterface` in the local domain are allowed to be exported to any remote domain. Requests for any other factory should be denied.

```
# BEA Tuxedo FactoryFinder Domains
# Configuration File
#
*DM_LOCAL_FACTORIES
    Teller.FactoryInterface
```

- Example 3

The following FactoryFinder Domains configuration file defines that none of the factory objects registered with the BEA Tuxedo FactoryFinder are to be exported to a remote domain.

```
# BEA Tuxedo FactoryFinder Domains
# Configuration File
#
*DM_LOCAL_FACTORIES
    NONE
```

See Also

[UBBCONFIG\(5\)](#), [DMCONFIG\(5\)](#), [TMFFNAME\(5\)](#), [TMIFRSVR\(5\)](#)

Error, Error32(5)

Name

Error, Error32—FML error codes

Synopsis

```
#include "fml.h"
#include "fml32.h"
```

Description

The numerical value represented by the symbolic name of an error condition is assigned to `Error` 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 FTYPERR 6   /* illegal field type */
#define FEUNIX 7    /* unix system call error */
```

```

#define FBADNAME 8 /* unknown field name */
#define FMALLOC 9 /* malloc failed */
#define FSYNTAX 10 /* bad syntax in boolean expression */
#define FFTOPEN 11 /* cannot find or open field table */
#define FFTSYNTAX 12 /* syntax error in field table */
#define FEINVAL 13 /* invalid argument to function */
#define FBADTBL 14 /* destructive concurrent access to field table */
#define FBADVIEW 15 /* cannot find or get view */
#define FVFSYNTAX 16 /* bad viewfile */
#define FVFOOPEN 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 `Error` to zero, an application can set `Error` to zero, call a routine and then check `Error` 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-to-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

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

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

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

Field Identifiers

FML functions allow field values to be typed. Currently the following types are supported: `char`, `string`, `short`, `long`, `float`, `double`, `carray` (character array), `ptr` (pointer to a buffer), `FML32` (embedded FML32 buffer), and `VIEW32` (embedded VIEW32 buffer). The `ptr`, `FML32`, and `VIEW32` types are supported only for the FML32 interface. Constants for field types are defined in `fm1.h` (`fm132.h` for FML32). So that fielded buffers can be truly self-describing, the type of a field is carried along with the field by encoding the field type in the `FLDID`. Thus, a `FLDID` is composed of two elements: a field type, and a field number. In 32-bit FML, field numbers must be between 10,001 and 30,000,000. The numbers 1-10,000 and 30,000,001-33,554,431 are reserved for system use. In 16-bit FML, field numbers must be between 101 and 8,191. The numbers 1-100 are reserved for system use.

Field Mapping

For efficiency, it is desirable that the field name to field identifier mapping be available at compile time. For utility, it is also desirable that these mappings be available at run time. To accommodate both these goals, FML represents field tables in text files, and provides commands to generate corresponding C header files. Thus, compile time mapping is done by the C preprocessor, `cpp`, by the usual `#define` macro. Run-time mapping is done by the function

`Fldid()` (or `Fldid32()` for FML32), which maps its argument, a field name, to a field identifier by consulting the source field table files.

Field Table Files

Files containing field tables have the following format:

- Blank lines and lines beginning with `#` are ignored.
- Lines beginning with `$` are ignored by the mapping functions but are passed through (without the `$`) to header files generated by `mkfldhdr()` (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 the C preprocessor identifier restrictions (that is, it should contain only alphanumeric characters and the underscore character). Internally, the name is truncated to 30 characters, so names must be unique within the first 30 characters.
- `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 filenames. 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 pathname. (The search for field tables is very similar to the search for executable commands using the `PATH` variable.) If `FLDTBLDIR` is not defined, it is taken to be the current directory. Thus, if `FIELDTBLS` and `FLDTBLDIR` are not set, the default is to take `fld.tbl` from the current directory.

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

Example

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

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

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

# address fields are based at 700

*base 700

EMPADDR 1      string street address
EMPCITY 2      string city
```

field_tables(5)

```
EMPSTATE 3      string state
EMPZIP  4      long  zip code
```

The associated header file would be:

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

See Also

[mkfldhdr](#), [mkfldhdr32\(1\)](#)

Programming a BEA Tuxedo ATMI Application Using FML

GAUTHSVR(5)

Name

GAUTHSVR—General LDAP-based authentication server

Synopsis

```
GAUTHSVR SRVGRP="identifier" SRVID=number other_parms CLOPT="-A -- -f  
filename"
```

Description

GAUTHSVR is a System/T 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 buffer as a user password and validates it against the configured password that is stored in LDAP Server. If the request passes validation, then an application key is returned with a successful return as the ticket for the client to use.

By default, the file `$TUXDIR/udataobj/tpgauth` is used for obtaining LDAP configuration information. The file can be overridden by specifying the file name, using a `"-f filename"` option in the server command line option. For example, `CLOPT="-A -- -f /usr/tuxedo/myapp/myldap"`.

There is no automatic propagation of this configuration file from the master machine to other machines in the Tuxedo UBBCONFIG file. To use multiple GAUTHSVRs, you must provide separate configurations on the various machines.

For additional information pertaining to GAUTHSVR, see [“GAUTHSVR Additional Information” on page 197](#).

SECURITY USER_AUTH

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

An authentication request is authenticated against only the first matching user name in the LDAP database. It does not support authentication against multiple entries.

SECURITY ACL or MANDATORY_ACL

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

The application key that is returned by the `GAUTHSVR` is the user identifier in the low-order 17 bits. The group identifier is the next 14 bits (the high-order bit is reserved for the administrative keys).

See Also

- [Setting up GAUTHSVR as the Authentication Server](#) in *Implementing Single Point Security Administration*.

GAUTHSVR Additional Information

Portability

`GAUTHSVR` is supported as a Tuxedo System/T-supplied server on non-Workstation platforms.

Examples

```
# Using GAUTHSVR
*RESOURCES
AUTHSVC    ". .AUTHSVC"
SECURITY  ACL

*SERVERS
GAUTHSVR  SRVGRP="AUTH"  SRVID=100
CLOPT="-A -- -f /usr/tuxedo/udataobj/tpgauth"
```

GWADM(5)

Name

GWADM—Domains gateway administrative server

Synopsis

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

Description

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

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

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

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

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

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

`-a {on | off}`

This option turns `off` or `on` the audit log feature for this local domain access point. The default is `off`. The `dmadmin` program can be used to change this setting while the gateway group is running (see [dmadmin\(1\)](#)).

`-t {on | off}`

This option turns `off` or `on` the statistics gathering feature for the local domain access point. The default is `off`. The `dmadmin` program can be used to change this setting while the gateway group is running (see [dmadmin\(1\)](#)).

The GWADM server must be booted before the corresponding gateways.

Portability

GWADM is supported as a BEA Tuxedo system-supplied server on all supported server 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.

```
#
*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

[dmadmin\(1\)](#), [tmboot\(1\)](#), [DMADM\(5\)](#), [DMCONFIG\(5\)](#), [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 = value RESTART = Y [MAXGEN = value] [GRACE = value]  
CLOPT = "-A -- [-s][-U inbound-message-size-limit-in-bytes ]"
```

Description

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

Note: From Tuxedo release 9.0 and later, the GWTDOMAIN default is *multithread mode*. This default mode is only useful for machines with multiple CPUs.

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 gateway 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 may be set to `Y` or `N`.

The `CLOPT` option is a string of command-line options that is passed to GWTDOMAIN when it is booted. The following run-time parameter is recognized for a gateway process:

`-s`
This optional parameter turns off the default multithread mode. On a single CPU machine, turning off multithread mode helps to avoid possible negative performance impact.

`-U inbound-message-size-limit-in-bytes`
This option specifies the upper-size limit of incoming network message for GWTDOMAIN. The message size includes internal data items for Tuxedo (should be less than 1024 bytes) and user data. The limit also takes effect when message is compressed, i.e., it also checks the original message size.

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 CLOPT="-A -r -- -U 65536"
```

Additional examples are available in the “EXAMPLES” sections 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

ISL(5)

Name

Enables access to BEA Tuxedo objects by remote BEA Tuxedo clients using IIOP.

Synopsis

```
ISL SRVGRP="identifier"
SRVID="number"
CLOPT="[-A ] [ servopts options ] -- -n netaddr
[-C {detect|warn|none} ]
[-d device ]
[-E principal_name]
[-K {client|handler|both|none} ]
[-m minh ]
```

```

[-M maxh ]
[-T Client-timeout]
[-x mpx-factor ]
[-H external-netaddr]
#options for Outbound IIOP
[-O]
[-o outbound-max-connections]
[-s Server-timeout]
[-u out-mpx-users]
#options for SSL
[-a]
[-R renegotiation-interval]
[-S secure port]
[-v {detect|warn|none} ]
[-z [0|40|56|128|256]]
[-Z [0|40|56|128|256]]"

```

Description

The IIO Server Listener (ISL) is a BEA Tuxedo-supplied server command that enables access to BEA Tuxedo objects by remote BEA Tuxedo clients using IIO. The application administrator enables access to the application objects by specifying the IIO Server Listener as an application server in the `SERVERS` section. The associated command-line options are used to specify the parameters of the IIO Server Listener and IIO Server Handlers.

The location, server group, server ID, and other generic server-related parameters are associated with the ISL using the standard configuration file mechanisms for servers. ISL command-line options allow for customization.

Each ISL booted as part of an application facilitates application access for a large number of remote BEA Tuxedo clients by providing access via a single, well-known network address. The IIO Server Handlers are started and stopped dynamically by the ISL, as necessary, to meet the incoming load.

For joint client/servers, if the remote joint client/server ORB supports bidirectional IIO connections, the ISL can use the same inbound connection for outbound invokes to the remote joint client/server. The ISL also allows outbound invokes (outbound IIO) to objects located in a joint client/server that is not connected to an ISH. This capability is enabled when the `-o` option is specified. The associated command-line options allow configuration of outbound IIO support:

Parameters

-A

Indicates that the ISL is to be booted to offer all its services. This is a default, but it is shown to emphasize the distinction between system-supplied servers and application servers. The latter can be booted to offer only a subset of their available services. The double-dash (--) marks the beginning of parameters that are passed to the ISL after it has been booted.

You specify the following options in the CLOPT string after the double-dash (--) in the CLOPT parameters:

-n netaddr

Specifies the network address to be used by a server listener to accept connections from remote CORBA clients. The remote client must set the environment variable (TOBJADDR) to this value, or specify the value in the Bootstrap object constructor. See the *C++ Programming Reference* for details. This is the only required parameter.

TCP/IP addresses must be specified in one of the following two formats:

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

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

Note: The hostname must begin with a letter character.

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

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

Note: The Java `Tobj_Bootstrap` object uses a `short` type to store the `port_number`. Therefore, you must use a `port_number` in the range of 0 to 32767 if you plan to support connections from Java clients.

Note: The network address that is specified by programmers in the Bootstrap constructor or in `TOBJADDR` must exactly match the network address in the application's `UBBCONFIG` file. The format of the address as well as the capitalization must match. If the addresses do not match, the call to the Bootstrap constructor will fail with a seemingly

unrelated error message:

```
ERROR: Unofficial connection from client at
<tcp/ip address>/<port-number>:
```

For example, if the network address is specified as `//TRIXIE:3500` in the `ISL` command line option string, specifying either `//192.12.4.6:3500` or `//trixie:3500` in the Bootstrap constructor or in `TOBJADDR` will cause the connection attempt to fail.

On UNIX systems, use the `uname -n` command on the host system to determine the capitalization used. On Windows NT systems, see the host system's Network control panel to determine the capitalization used.

Note: Unlike the BEA Tuxedo system Workstation Listener (WSL), the format of the network addresses is limited to `//host:port`. The reason for this limitation is that the host name and port number are used by BEA Tuxedo servers; the host name does not appear as such in the hexadecimal format, and it could only be passed to the servers using the dotted IP address format.

`[-a]`

Specifies that certificate-based authentication should be enabled when accepting an SSL connection from a remote application.

`[-C detect|warn|none]`

Determines how the IIOP Listener/Handler will behave when unofficial connections are made to it. The default value is `detect`.

The official way for the CORBA client to connect to the IIOP Listener/Handler is via a Bootstrap object. The unofficial connection is established directly from an IOR. For example, a client could connect to one IIOP Listener/Handler via a Bootstrap object and then, perhaps inadvertently, connect to a second IIOP Listener/Handler by using an IOR that contains the host and port of the second IIOP Listener/Handler. Typically, this is not the case. Usually, the client uses IORs that contain the host and port of the IIOP Listener/Handler that the client connected to via a Bootstrap object. Use of such IORs does not cause an additional connection to be made.

Caution: The use of unofficial connections can cause problems for remote client applications that use transactions. The application may have the notion that invocations on both the official and unofficial connections within the same transaction have succeeded; however, in reality, only invocations on the official connection are ACID (Atomicity, Consistency, Isolation, and Durability).

A value of `detect` causes the ISL/ISH to raise a `NO_PERMISSION` exception when an unofficial connection is detected. A value of `warn` causes the ISL/ISH to log a message to the user log exception when an unofficial connection is detected; no exception will be raised. A value of `none` causes the ISL/ISH to ignore unofficial connections.

`[-d device]`

Specifies the device filename used for network access by the server listener and its server handlers. This parameter is optional because some transport providers (for example, sockets) do not require a device name. However, other providers (for example, TLI) do require a device name. In the case of TLI, this option is mandatory. There is no default for this parameter. (This does not apply to Windows 2003 systems.)

`[-E principal_name]`

An optional parameter that indicates the identity of the principal that is required in order to establish a trusted connection pool. A trusted connection pool can only be established if a CORBA application is configured to require users to be authenticated.

If a remote client application attempts to propagate per-request security information over a connection that is not part of a trusted connection pool, the accompanying propagated security information will be ignored.

`[-K {client|handler|both|none}]`

Directs the client, or the ISH process, or both, to activate the network provider's `KEEPALIVE` option. This option improves the speed and reliability of network failure detection by actively testing an idle connection's state at the protocol stack level. The availability and timeout thresholds for this feature are determined by operating system tunable parameters.

A value of `client` configures this option for the client; a value of `handler` configures this option for the ISL; and a value of `both` will configure both sides of the connection. The default value is `none`, in which case neither side has the `KEEPALIVE` option configured.

Note: The `KEEPALIVE` interval is an operating system parameter, so changing the value affects any other applications that enable `KEEPALIVE`. Many platforms have a two-hour default value that may be longer than desired.

This option is not available on all platforms. A userlog warning message is generated if the `KEEPALIVE` option is specified but is not available on the ISH's machine. If `KEEPALIVE` is requested but is not available on the client's machine, the setting is ignored.

`[-m minh]`

Specifies the minimum number of handlers that should be available in conjunction with this ISL at any given time. The default is 0. The ISL will start this many ISHs immediately

upon being booted and will not deplete the supply of ISHs below this number until the administrator issues a shutdown to the ISL. The default value for this parameter is 0. The legal range is between 0 and 255.

[*-M maxh*]

Specifies the maximum number of handlers that should be available in conjunction with this ISL at any given time. The Handlers are started as necessary to meet the demand of remote BEA Tuxedo clients attempting to access the system. The default value for this parameter is equal to the setting for `MAXWSCLIENTS` on the logical machine, divided by the multiplexing factor for this ISL (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`.

[*-T Client-timeout*]

Specifies the inactive client timeout option. The inactive client timeout is the time (in minutes) allowed for a client to stay idle. If a client does not make any requests within this time period, the IIOP Listener/Handler disconnects the client. If this argument is not given or is set to 0, the timeout is infinite.

[*-x mpx-factor*]

This is an optional parameter used to control the degree of multiplexing desired within each ISH. The value for this parameter indicates the number of remote BEA Tuxedo clients that can be supported simultaneously by each ISH. The ISH ensures that new handlers are started as necessary to handle new remote BEA Tuxedo clients. This value must be greater than or equal to 1 and less than or equal to 4096. The default value for this parameter is 10.

[*-H external netaddr*]

Specifies the external network address to be set as the host and port in interoperable object references returned to clients of the ISL. It has the same format as the `ISL CLOPT -n netaddr` option. This feature is useful when an IIOP, or remote, client needs to connect to an ISL through a firewall.

[*-O*]

This option (uppercase letter O) enables outbound IIOP to objects that are not located in a client that is connected to an ISH. Since the *-O* option requires a small amount of extra resources, the default is to not allow outbound IIOP.

[*-o outbound-max-connections*]

This option (lowercase letter o) specifies the maximum number of outbound connections that each ISH may have. In effect, it limits the number of simultaneous Outbound IIOP sockets that any single ISH under the control of this ISL will have active at one time.

This option requires that the *-O* (uppercase letter O) option is also specified. The value of this option must be greater than 0, but not more than 4096. An additional requirement is

that the value of this option, (`outbound-max-connections`) times the maximum number of handlers, must be less than 32767. The default for this option is 20.

`[-R renegotiation-interval]`

Specifies the renegotiation interval in minutes. If a connection does not have a renegotiation in the specified interval, the IIOp Listener/Handler will request that the client renegotiate the session for inbound connections or actually perform the renegotiation in the case of outbound connections. The default is 0 minutes which results in no periodic session renegotiations.

`[-S secure-port]`

Specifies the port number that the IIOp Listener/Handler should use to listen for secure connections using the SSL protocol. You can configure the IIOp Listener/Handler to allow only secure connections by setting the port numbers specified by the `-s` and `-n` options to the same value.

`[-s Server-timeout]`

`Server-timeout` is the time, in minutes, allowed for a remote server to remain idle. If a remote server does not receive any requests within this time period, the ISL disconnects the outbound IIOp connection to the server. The ISH reconnects to the remote server on subsequent requests. This option can be used for server platforms that are unstable. Note that this is a best-attempt value in that the ISL does not disconnect the connection before this time is up, but does not guarantee to disconnect the connection once the exact time has elapsed. This option requires that the `-O` (uppercase letter O) option is also specified. The value must be greater than or equal to 1. If this option is not specified, the default is 60 (one hour).

`[-u out-mpx-users]`

An optional parameter used to control the degree of outbound multiplexing desired within each ISH. The value for this option indicates the number of outbound IIOp users (native clients or servers) that can be supported simultaneously by each outbound IIOp connection in the ISH. The ISL ensures that new ISHs are started, as necessary, to handle new users up to the value of this option (`out-mpx-users`). This option requires that the `-O` (uppercase letter O) option is also specified. This option must be greater than 0 (zero), but not more than 1024; the default value is 10.

`[-v {detect|warn|none}]`

Determines how the IIOp Listener/Handler will behave when a digital certificate for a peer of an outbound connection initiated by the BEA object request broker (ORB) is received as part of the Secure Sockets Layer (SSL) protocol handshake. The validation is only performed by the initiator of a secure connection and confirms that the peer server is actually located at the same network address as specified by the domain name in the

server's digital certificate. This validation is not technically part of the SSL protocol but is similar to the check done in web browsers.

A value of `detect` causes the BEA ORB to verify that the host specified in the object reference used to make the connection matches the domain name specified in the peer server's digital certificate. If the comparison fails, the BEA ORB refuses to authenticate the peer and drops the connection. The `detect` value is the default value.

A value of `warn` causes the BEA ORB to verify that the host specified in the object reference used to make the connection matches the domain name specified in the peer's digital certificate. If the comparison fails, the BEA ORB logs a message to the user log but continues to process the connection.

A value of `none` causes the BEA ORB to not perform the peer validation and to continue to process the connection.

The `-v` parameter is only available if licenses for SSL and LLE (link level encryption) are installed.

`[-z [|0|40|56|128|256]]`

Specifies the minimum level of encryption when establishing a network connection between a client and the IIOP Listener/Handler. 0 means no encryption while 40, 56, 128, and 256 specify the length (in bits) of the encryption key. If this minimum level of encryption cannot be met, a connection will not be established. This option is only available if licenses for SSL and LLE (link level encryption) are installed.

`[-z [|0|40|56|128|256]]`

Specifies the maximum level of encryption when establishing a network connection between a client and the IIOP Listener/Handler. 0 means no encryption while 40, 56, 128, and 256 specify the length (in bits) of the encryption key. The default is whatever capability is specified by the license. This option is only available if licenses for SSL and LLE (link level encryption) are installed.

Portability

The IIOP Server Listener is supported as a BEA Tuxedo-supplied server on UNIX and Microsoft Windows NT operating systems.

Interoperability

The ISL works with any IIOP compliant ORB.

Depending on the type of remote object and the desired outbound IIOP configuration, you may have to perform additional programming tasks. [Table 36](#) lists the requirements for each type of object and outbound IIOP configuration.

Table 36 Programming Requirements for Using Outbound IIOP

Types of Objects	Asymmetric Requirements	Paired-connection Requirements	Bidirectional Requirements
Remote joint client/servers	Set ISL CLOPT -O option.	Use the <code>Tobj_Bootstrap::register_callback_port</code> method to register the callback port.	Use the <code>CORBA::ORB::create_policy</code> method to set <code>BiDirPolicy</code> on the POA.
Foreign (non-CORBA) ORBs	Set ISL CLOPT -O option.	Not applicable.	If the foreign ORB supports the POA and <code>BiDirPolicy</code> , use the <code>CORBA::ORB::create_policy</code> method to set <code>BiDirPolicy</code> on the POA.
Remote clients	Remote clients are not servers, so outbound IIOP is not possible.		
Native joint client/servers	Outbound IIOP is not used.		
Native clients	Outbound IIOP is not used.		

Network Addresses

Suppose the local machine on which the ISL 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 ISL should accept requests is `2334`. The address specified by the `-l` option could be:

```
//155.2.193.18:2334
```

```
//backus.company.com:2334
```

Examples

```
*SERVERS
ISL SRVGRP="ISLGRP" SRVID=1002 RESTART=Y GRACE=0
CLOPT="-A -- -n //piglet:1900 -d /dev/tcp"
```

KAUTHSVR(5)

Name

KAUTHSVR—Tuxedo Kerberos-based network authentication server

Synopsis

```
KAUTHSVR SRVGRP=SECGRP SRVID=100 GRACE=0 MAXGEN=2 CLOPT="-A -- -k  
/etc/krbauth.kt -p krbauth@host.yourcompany.com"
```

Description

KAUTHSVR is a Kerberos-based Tuxedo authentication server. Its purpose is two fold:

- it authenticates a user with the kerberos system, and
- it uses the accompanying user information to create a user APPKEY to handle Tuxedo Access Control.

KAUTHSVR must be manually configured in the UBBCONFIG file in order to complete Tuxedo user authentication if you want to use Kerberos the default authentication mechanism. There is a slight difference in how KAUTHSVR is configured for UNIX and Windows platforms. For more information, see [“Using the Kerberos Authentication Plug-in.”](#)

Principal Name and UNIX Key Table Configuration

Kerberos allows you to store principal names and service keys in a local file based database called a Key Table. This key table allows services running on a host to authenticate themselves to the Key Distribution Center. KAUTHSVR does not replace Kerberos Key Distribution Center authentication; however, it does replace AUTHSVR(5) and LAUTHSVR(5) when you want to use Kerberos-based authentication.

Principal Name Configuration

KAUTHSVR must have its own principal name associated with it. To specify which principal name KAUTHSVR uses, you must configure it in the UBBCONFIG file. The CLOPT option uses the -p parameter to establish its principal name. For example, -p <principal name>. The principal name and its password must be configured in the Kerberos database and the local key table.

Note: The principal name can also be configured by using the KAUTHSVRPRINC parameter or the same name environment variable. For more information, see [“Using the Kerberos Authentication Plug-in.”](#)

UNIX Key Table Configuration

Before a server can be setup to use Kerberos, you must setup a key table on a host running the server. KAUTHSVR must access the server Key Table (KTAB) when it is booted. There are two ways to specify the server key table:

- you can use the CLOPT option to tell KAUTHSVR where its KTAB is. The KAUTHSVR CLOPT option uses the -k parameter to specify where KTAB is located. For example, "-k <key table full path name>".
- you can also specify the KTAB location as an environmental variable. For example, "KRB5_KTNAME=<key table full path name>".

Note: Any updates made to the key table do not affect the Kerberos database. If you change the keys in the key table, you must also make the corresponding changes to the Kerberos database.

Account Password on Windows Platform

When KAUTHSVR is configured on a Windows platform a key table is not needed. However, it must have an account password. There are two ways to setup a KAUTHSVR password:

- Configure in the UBBCONFIG file SERVERS section related with KAUTHSVR. For example:

Listing 2 KAUTHSVR Example in UBBCONFIG

```
*SERVERS
... ..
KAUTHSVR
SRVGRP=AUTHGRP SRVID=10
SEC_PRINCIPAL_NAME="kauthsvc " SEC_PRINCIPAL_PASSVAR=passvar
CLOPT="-A -- -p kauthsvc/bjwin2k3.bea.com@KRB.BEA.COM"
... ..
```

When TUXCONFIG is created, you must input the password at the command prompt.

Note: The name `kauthsvc` in `SEC_PRINCIPAL_NAME` is used as an example only.

- Specify in the KAUTHSVR environment variable KAUTHSVRPASS

See Also

[AUTHSVR\(5\)](#)

[LAUTHSVR\(5\)](#)

“Using the Kerberos Authentication Plug-in,” in the *Using Security in ATMI Applications*
Kerberos Introduction from MIT (<http://web.mit.edu/kerberos/www/>)

langinfo(5)

Name

`langinfo`—Language information constants

Synopsis

```
#include <langinfo.h>
```

Description

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

`DAY_1`
Locale's equivalent of “sunday”

`DAY_2`
Locale's equivalent of “monday”

`DAY_3`
Locale's equivalent of “tuesday”

`DAY_4`
Locale's equivalent of “wednesday”

`DAY_5`
Locale's equivalent of “thursday”

`DAY_6`
Locale's equivalent of “friday”

`DAY_7`
Locale's equivalent of “saturday”

`ABDAY_1`
Locale's equivalent of “sun”

`ABDAY_2`
Locale's equivalent of “mon”

`ABDAY_3`
Locale's equivalent of “tue”

ABDAY_4
Locale's equivalent of "wed"

ABDAY_5
Locale's equivalent of "thur"

ABDAY_6
Locale's equivalent of "fri"

ABDAY_7
Locale's equivalent of "sat"

MON_1
Locale's equivalent of "january"

MON_2
Locale's equivalent of "february"

MON_3
Locale's equivalent of "march"

MON_4
Locale's equivalent of "april"

MON_5
Locale's equivalent of "may"

MON_6
Locale's equivalent of "june"

MON_7
Locale's equivalent of "july"

MON_8
Locale's equivalent of "august"

MON_9
Locale's equivalent of "september"

MON_10
Locale's equivalent of "october"

MON_11
Locale's equivalent of "november"

MON_12
Locale's equivalent of “december”

ABMON_1
Locale's equivalent of “jan”

ABMON_2
Locale's equivalent of “feb”

ABMON_3
Locale's equivalent of “mar”

ABMON_4
Locale's equivalent of “apr”

ABMON_5
Locale's equivalent of “may”

ABMON_6
Locale's equivalent of “jun”

ABMON_7
Locale's equivalent of “jul”

ABMON_8
Locale's equivalent of “aug”

ABMON_9
Locale's equivalent of “sep”

ABMON_10
Locale's equivalent of “oct”

ABMON_11
Locale's equivalent of “nov”

ABMON_12
Locale's equivalent of “dec”

RADIXCHAR
Locale's equivalent of “.”

THOUSEP
Locale's equivalent of “,”

YESSTR
Locale's equivalent of “yes”

NOSTR
Locale's equivalent of “no”

CRNCYSTR
Locale's currency symbol

D_T_FMT
Locale's default format for date and time

D_FMT
Locale's default format for the date

T_FMT
Locale's default format for the time

AM_STR
Locale's equivalent of “AM”

PM_STR
Locale's equivalent of “PM”

This information is retrieved by [nl_langinfo\(3c\)](#).

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\(5\)](#)

LAUTHSVR(5)

Name

LAUTHSVR—WebLogic Server embedded LDAP-based authentication server

Synopsis

```
LAUTHSVR SRVGRP="identifier" SRVID=number other_parms CLOPT="-A -- -f
filename"
```

Description

LAUTHSVR is a System/T provided server that offers the authentication service while the user security information is located in WebLogic Server. 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 buffer as a user password and validates it against the configured password that is stored in WebLogic Server. If the request passes validation, then an application key is returned with a successful return as the ticket for the client to use.

- If a user belongs to the “Administrators” group in WebLogic Server, then LAUTHSVR will return TPSYSADM as its application key.
- If a user belongs to the “Operators” group in WebLogic Server, then LAUTHSVR will return TPSYSOP as its application key.

Note: The application keys that correspond to `tpsysadm` and `tpsysop` must be 0x80000000 and 0xC0000000, respectively.

By default, the file `$TUXDIR/udataobj/tpldap` is used for obtaining LDAP configuration information. The file can be overridden by specifying the file name, using a `-f filename` option in the server command line option. For example, `CLOPT="-A -- -f /usr/tuxedo/myapp/myldap"`. There is no automatic propagation of this configuration file from the master machine to other machines in the Tuxedo UBBCONFIG file. To use multiple LAUTHSVRs, you must provide separate configurations on the various machines.

For additional information pertaining to LAUTHSVR, see [“LAUTHSVR Additional Information” on page 218](#).

SECURITY USER_AUTH

If `SECURITY` is set to `USER_AUTH` or higher, per-user authentication is enforced. The name of the authentication service can be configured for the application. If not specified, it defaults to `AUTHSVCS` which is the default service advertised for `LAUTHSVR`.

An authentication request is authenticated against only the first matching user name in the LDAP database. It does not support authentication against multiple entries.

SECURITY ACL or MANDATORY_ACL

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

The application key that is returned by the `LAUTHSVR` is the user identifier in the low-order 17 bits. The group identifier is the next 14 bits (the high-order bit is reserved for the administrative keys).

LAUTHSVR Additional Information

Portability

`LAUTHSVR` is supported as a Tuxedo System/T-supplied server on non-Workstation platforms.

Examples

```
# Using LAUTHSVR
*RESOURCES
AUTHSVCS    ". .AUTHSVCS"
SECURITY    ACL

*SERVERS
LAUTHSVR    SRVGRP="AUTH" SRVID=100
CLOPT="-A -- -f /usr/tuxedo/udataobj/tpldap"
```

METAREPOS(5)

Name

METAREPOS - Tuxedo service metadata repository buffer format

```
#include <fml32.h>
#include <fml1632.h> /* optional */
#include <tpadm.h>
```

Description

This reference page describes the interfaces through which an administrator, operator or user interacts with the defined components of the Tuxedo metadata repository. The service metadata repository can be programmatically accessed and updated through the `.TMMETAREPOS` service offered by `TMMETADATA(5)` server or can be accessed and updated directly using `tpgetrepos(3c)` and `tpsetrepos(3c)`.

Programmatic access to the Tuxedo service metadata repository is accomplished through the use of `FML32` buffers very similar in format to those used by the Tuxedo MIB. In fact, the Tuxedo service metadata repository uses and supports the same kind of generic `MIB(5)` `FML32` input and output buffer fields:

Input buffer fields

```
TA_OPERATION, TA_CLASS, TA_CURSOR, TA_OCCURS, TA_FLAGS, TA_FILTER,
TA_MIBTIMEOUT, and TA_CURSORHOLD
```

Output buffer fields

```
TA_CLASS, TA_OCCURS, TA_MORE, TA_CURSOR, and TA_ERROR
```

Note: The METAREPOS has the following generic `MIB(5)` field limitations:

- Only `MIB_PREIMAGE` (a `TA_FLAGS` field flag) is used in metadata repository operation. The other two flags, `MIB_LOCAL` and `MIB_SELF`, are omitted.
- `TA_MIBTIMEOUT` is ignored by `.TMMETAREPOS` service, `tpsetrepos(3c)` and `tpgetrepos(3c)`.
- `TA_CURSORHOLD` does not work with `tpgetrepos()`.
- `TA_ERROR` applies specific definitions to generic return codes when initiated with a metadata repository setting operation. The generic codes are defined as follows:

TAOK - No service updates were made to the metadata repository

TAUPDATED - All service updates were made to the metadata repository

TAPARTIAL - Partial service updates were made to the metadata repository

FML32 fields related to specific metadata repository attributes use the prefix `TA_REPOS` followed by the name of the repository keyword in upper case. For more information on Metadata Repository service and parameter key words, see [tmloadrepos\(1\)](#).

METAREPOS Attribute Fields

Service-Level Attribute Fields

METAREPOS service-level attribute fields are used to describe services. The `TA_REPOSERVICE` attribute is a key field that is used to name services and uniquely identify them for retrieval or `get` operations. `TA_REPOSERVICE` can accept regular expressions as defined in [rex\(1\)](#). For example, using the regular expression value `"*"` with `TA_REPOSERVICE` retrieves all service information in a metadata repository.

For `set` operations, `TA_REPOSERVICE` must include a Tuxedo service name and *cannot* be interpreted as a regular expression.

For more information on service-level keywords, see [Managing The Tuxedo Service Metadata Repository](#), Creating The Tuxedo Service Metadata Repository.

Parameter-Level Attribute Fields

METAREPOS parameter-level attribute fields are used to describe service parameters. Common *occurrence* numbers are used to associate different attribute fields as part of a common parameter. The *n*th service parameter is described by the occurrence number *n-1* of all parameter-level attribute fields.

For example, the *first service parameter* is described by the *first occurrence of the attribute field* as "0"; the *second service parameter* is described by the *second occurrence of the attribute field* as "1", and so on.

If a specific attribute field occurrence is required by a later numbered parameter, but not by one or more earlier numbered parameters, you must specify a value for the earlier attribute field occurrences so that the later occurrences are properly numbered.

Sub-Parameter Values

`TA_REPOSEMBED` is used to provide information about service parameters that have *sub-parameter* values, or in other words, embedded data.

Because the Tuxedo service metadata repository requests input and output in FML32 format, when `TA_REPOSEMBED` is specified with sub-parameter values (other than the default empty record), it must contain an FML32 record. This FML32 record consists of parameter-level fields corresponding to each

sub-parameter (FML field or VIEW element) in the record described by the associated TA_REPOSPARAM field.

The TA_REPOSEMBED parameter value corresponds to the information contained between matching parentheses "(" and ")" in the repository_input file or the unloaded -t repository_file. For more information on the repository_input file and repository_file, see [tmloadrepos\(1\)](#) and [tmunloadrepos\(1\)](#).

Table 37 METAREPOS Attribute Field Table

Attribute Field	Level	Type	Permissions	Values	Default
TA_REPOSSERVICE (x)(r)(*)	service	string	rwxr--r--	string[1...255]	N/A
TA_STATE(k)	N/A	string	rwxr-xr--	GET: "VAL" SET: "{NEW unset INV}"	N/A N/A
TA_REPOSTUXSERVICE	service	string	rwxr--r--	string[1...15]	N/A
TA_REPOSSEVICETYPE	service	string	rwxr--r--	"{service oneway queue}"	"service"
TA_REPOSSERVICEMODE	service	string	rwxr--r--	"{tuxedo webservi ce}"	"tuxedo"
TA_REPOSEXPORT	service	string	rwxr--r--	"{ Y N }"	"Y"
TA_REPOSINBUF	service	string	rwxr--r--	string[1...8]	N/A
TA_REPOSOUTBUF	service	string	rwxr--r--	string[0...8]	N/A
TA_REPOSERRBUF	service	string	rwxr--r--	string[0...8]	N/A
TA_REPOSINVIEW	service	string	rwxr--r--	string[0..32]	N/A
TA_REPOSOUTVIEW	service	string	rwxr--r--	string[0..32]	N/A
TA_REPOSERRVIEW	service	string	rwxr--r--	string[0..32]	N/A
TA_REPOSINBUFSHEMA	service	string	rwxr--r--	string[0..1023]	N/A
TA_REPOSOUTBUFSHEMA	service	string	rwxr--r--	string[0..1023]	N/A
TA_REPOSERRBUFSHEMA	service	string	rwxr--r--	string[0..1023]	N/A

Table 37 METAREPOS Attribute Field Table (Continued)

Attribute Field	Level	Type	Permissions	Values	Default
TA_ REPOSSVCDESCRIPTION	service	string	rwxr--r--	string[0..1024]	N/A
TA_REPOSENDQSPACE	service	string	rwxr--r--	string[0..15]	N/A
TA_REPOSENDQUEUE	service	string	rwxr--r--	string[0..15]	N/A
TA_REPOSRLYQUEUE	service	string	rwxr--r--	string[0..15]	N/A
TA_REPOSERRQUEUE	service	string	rwxr--r--	string[0..15]	N/A
TA_REPOSRCVQSPACE	service	string	rwxr--r--	string[0..15]	N/A
TA_REPOSRCVQUEUE	service	string	rwxr-r--	string[0..15]	N/A
TA_REPOSVERSION	service	string	rwxr--r--	string[0..1024]	N/A
TA_REPOSATTRIBUTES	service	string	rwxr--r--	string[0..1024]	N/A
TA_REPOSFIELDTBLS	service	string	rwxr--r--	string[0..1024]	N/A
TA_REPOSPARAM	parameter	string	rwxr-r--	string[1..32]	N/A
TA_REPOSTYPE	parameter	string	rwxr--r--	"{ byte short integer float double string carray dec_t xml ptr fml32 view32 mbstring }	N/A
TA_REPOSSUBTYPE	parameter	string	rwxr--r--	string[0..32]	N/A
TA_REPOSACCESS	parameter	string	rwxr--r--	"{ in out err inout inerr outerr inouterr noaccess }	N/A
TA_REPOSCOUNT	parameter	long	rwxr--r--	0<=num<=32767	1
TA_REPOSPARAMDES CRPTION	parameter	string	rwxr--r--	string[0..1024]	N/A
TA_REPOSSIZE	parameter	long	rwxr--r--	0<=num	N/A

Table 37 METAREPOS Attribute Field Table (Continued)

Attribute Field	Level	Type	Permissions	Values	Default
TA_REPOSREQUIRED COUNT	parameter	long	rwxr--r--	0<=num<=32767	N/A
TA_REPOSFLDNUM	parameter	long	rwxr--r--	0<=num	N/A
TA_REPOSFLDID	parameter	long	r--r--r--	0<=num	N/A
TA_REPOSVFBNAME	parameter	string	rwxr--r--	string[0..30]	N/A
TA_REPOSVFLAG	parameter	string	rwxr--r--	string[0..6]	N/A
TA_REPOSVNULL	parameter	string	rwxr--r--	string[0..32]	N/A
TA_REPOSPARAMSCHEMA	parameter	string	rwxr--r--	string[0..1023]	N/A
TA_REPOSPRIMETYPE	parameter	string	rwxr--r--	XML primitive data type, e.g. "anyType", "boolean", "token", "unsignedByte", etc.	N/A
TA_REPOSEMBED	parameter	FML32	rwxr--r--		Empty record

(x) - Regular expression GET key field

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

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

(k) - GET key

METAREPOS Attribute Semantics

TA_REPOSSERVICE: string[1..255]

Service name. This attribute accepts regular expressions as defined in [rex\(1\)](#) for metadata repository service information retrieval. Regular expressions cannot be used to update metadata repository service information.

Note: This field string length was [1..15] in Tuxedo 9.1 release.

TA_STATE:

GET: "{ VALid }"

A GET operation retrieves information for the selected service object(s). The following state(s) define TA_STATE returned in response to a GET request.

VALid	Service object is defined. Note that this is the only valid state for service metadata repository.
-------	--

SET: "{ NEW | unset | INValid }"

A SET operation updates information for the selected service object(s). The following state(s) define TA_STATE set in a set request. States not listed cannot be set

NEW	Create new service object. Successful return leaves the object in the VALid state.
-----	--

unset	Modify an existing service object. This combination is not allowed in the INValid state. Successful return leaves the object state unchanged.
-------	---

INValid	Delete service object. State change allowed only when in the VALid state. Successful return leaves the object in the INValid state.
---------	---

TA_REPOSTUXSERVICE: string[1...15]

Actual tuxedo service name. By default, it has the same value as TA_REPOSSERVICE.

TA_REPOSSERVICETYPE: "{service|oneway|queue}"

Service invocation type. This term comes from the Tuxedo Control.

· "service" supports synchronous request/response.

· "oneway" supports request without response.

· "queue" supports tpenqueue and tpdequeue.

TA_REPOSSERVICEMODE: "{tuxedo|webservice}"

Type of Service origination. This term comes from BEA SALT.

"tuxedo" stands for a Tuxedo originated service definition.

"webservice" stands for a SALT proxy service definition generated by BEA SALT wsdlcvt utility.

TA_REPOSEXPORT: "{ Y | N }"

Indicates whether a service object is available or not. This attribute is for Jolt Repository compatibility only. The default value is "Y".

TA_REPOSINBUF: string[1..8]

The service(s) input buffer type. Valid values: FML, FML32, VIEW, VIEW32, STRING, CARRAY, XML, X_OCTET, X_COMMON, X_C_TYPE, MBSTRING or a custom-defined type. Only one type is allowed.

Note: Limitation: A string of custom type may contains up to 8 characters. See "Managing Typed Buffers" in *Programming a BEA Tuxedo ATMI Application Using C*

TA_REPOSOUTBUF: string[0..8]

The service(s) output buffer type (buffer returned from `treturn()` with TPSUCCESS status). Valid value is same as TA_REPOSINBUF. Note that this attribute can be null.

TA_REPOSERRBUF: string[0..8]

The service(s) error buffer type (buffer returned from `treturn()` with TPFailure status). Valid value is same as TA_REPOSINBUF. Note that this attribute can be null.

TA_REPOSINVIEW: string[0..32]

View name for input parameters. This information is optional only if one of the following buffer types is used: VIEW, VIEW32, X_COMMON, X_C_TYPE.

TA_REPOSOUTVIEW: string[0..32]

View name for output parameters. Similar with TA_REPOSINVIEW.

TA_REPOSERRVIEW: string[0..32]

View name for output parameters. Similar with TA_REPOSINVIEW.

TA_REPOSINBUFSHEMA: string[0..1023]

Customized message schema association for input buffer. This information is optional.

TA_REPOSOUTBUFSHEMA: string[0..1023]

Customized message schema association for output buffer. Similar with TA_REPOSINBUFSHEMA.

TA_REPOSERRBUFSHEMA: string[0..1023]

Customized message schema association for error buffer. Similar with TA_REPOSINBUFSHEMA.

TA_REPOSSVCDESCRIPTION: string[0..1024]

String value for service description.

TA_REPOSENDQSPACE: string[0..15]

String value for send queue space name. Optional only when TA_REPOSSERVICETYPE is "queue".

TA_REPOSENDQUEUE: string[0...15]
String value for send queue name. Optional only when TA_REPOSSERVICETYPE is "queue".

TA_REPOSRLYQUEUE: string[0...15]
String value for reply queue name. Optional only when TA_REPOSSERVICETYPE is "queue".

TA_REPOSERRQUEUE: string[0...15]
String value for error queue name. Optional only when TA_REPOSSERVICETYPE is "queue".

TA_REPOSRCVQSPACE: string[0...15]
String value for receive queue space name. Optional only when TA_REPOSSERVICETYPE is "queue".

TA_REPOSRCVQUEUE: string[0...15]
String value for receive queue name. Optional only when TA_REPOSSERVICETYPE is "queue".

TA_REPOSVERSION: string[0...1024]
Any string defined by the user. Tuxedo does not interpret this attribute.

TA_REPOSATTRIBUTES: string[0...1024]
Any string defined by the user. Tuxedo does not interpret this attribute.

TA_REPOSFIELDTBLS: string[0...1024]
Optionally specifies a comma-separated list of field tables where the FML or FML32 fields used by this service can be found. Use the absolute path to describe each field table file.

TA_REPOSPARAM: string[0...32]
Parameter name.

TA_REPOSTYPE: "{ byte | short | integer | float | double | string | carray | dec_t | xml | ptr | fml32 | view32 | mbstring }"
Parameter type.

TA_REPOSSUBTYPE : string[0...32]
A view name for view32 typed parameter.

TA_REPOSACCESS: '{ in | out | err | inout | inerr | outerr | inouterr | noaccess }'
Parameter access method.

TA_REPOSCOUNT: 0<=num<=32767
Maximum number of parameter occurrences. Default value is 1.

TA_REPOSPARAMDESC: string[0...1024]
Parameter description string.

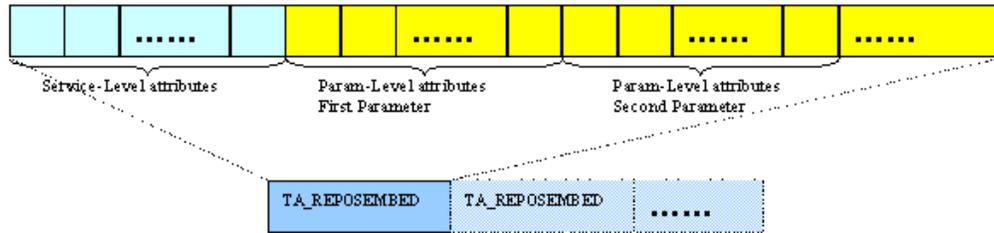
- TA_REPOSSIZE:** 0<=num
Optional only if the following parameter types are used: `carray`, `string`, `xml`, `mbstring`.
- TA_REPOSREQUIREDCOUNT:** 0<=num<=32767
Minimum number of parameter occurrences.
- TA_REPOSFLDNUM:** 0<=num
Optional only for `FML/FML32` field parameter, field number definition.
- TA_REPOSFLDID:** 0<=num
Optional only for `FML/FML32` field parameter, field id. Note that this field cannot be written or updated.
- TA_REPOSVFBNAME:** `string[0...30]`
Optional only when parameter type is `view/view32`. It is used to specify the corresponding field name for views mapped to `FML` buffers.
- TA_REPOSVFLAG:** `string[0...6]`
Optional only when parameter type is `view/view32`. Using this field by following the rules of the "Flag" option defined in `viewfile(5)`.
- TA_REPOSVNULL:** `string[0...32]`
Optional only when parameter type is `view/view32`. It is used to define user-specified NULL as the default NULL value for that parameter.
- TA_REPOSPARAMSCHEMA:** `string[0...1023]`
Optional for SALT proxy service (service with `TA_REPOSSERVICEMODE=webservice`). This value is generated by BEA SALT `wsdlcvt` utility from converting an external WSDL file.
- TA_REPOSPRIMETYPE:** `xml primitive data type`
Optional for SALT proxy service (service with `TA_REPOSSERVICEMODE=webservice`). This value is generated by BEA SALT `wsdlcvt` utility from converting an external WSDL file.
Valid values are: `anyType`, `boolean`, `base64Binary`, `hexBinary`, `float`, `double`, `anyURI`, `QName`, `NOTATION`, `duration`, `dateTime`, `time`, `date`, `gYearMonth`, `gYear`, `gMonthDay`, `gDay`, `gMonth`, `string`, `normalizedString`, `token`, `language`, `Name`, `NCName`, `NMTOKEN`, `NMTOKENS`, `ID`, `IDREF`, `IDREFS`, `ENTITY`, `ENTITIES`, `decimal`, `integer`, `nonPositiveInteger`, `negativeInteger`, `nonNegativeInteger`, `positiveInteger`, `long`, `unsignedLong`, `int`, `unsignedInt`, `short`, `unsignedShort`, `byte`, `unsignedByte`
- TA_REPOSEMBED**
Optional only if the parameter is one of following types: `fml32`, `view32`. It is an embedded `FML32` field to describe sub-parameters of the parameter.

Note: TA_REPOSEMBED field also is used to encapsulate attributes of each service once there might be multiple services in one FML32 buffer. Please see figure 6-1 and 6-2 for more information.

METAREPOS Buffer Format Diagram

Currently, METAREPOS input and output is in FML32 buffer format and is used to describe one or more instances of service metadata information. This FML32 typed buffer format is define in two modes: Standard Mode and Single Mode.

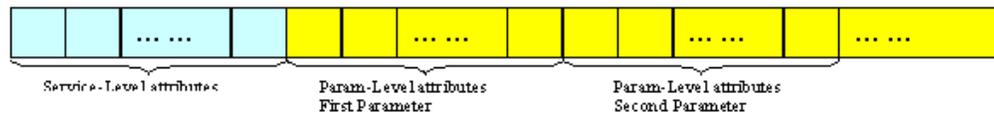
Figure 1 Standard Mode



In standard mode, each service is encapsulated into one embedded TA_REPOSEMBED FML32 field. Users fill in METAREPOS attributes by following the restrictions defined in the METAREPOS service-level and parameter-level attribute tables.

- Output Buffers: All output buffers returned by the Tuxedo service metadata repository must use standard mode, no matter if one or more service objects are returned.
 - Note:** For SET operations, TA_ERROR and TA_STATUS are included in each TA_REPOSEMBED buffer to indicate the set result of each service.
- Input Buffers: When using input buffer to specify services, standard mode is applied under the following conditions:
 1. SET operations (adding or updating) to outstanding multiple services
 2. When a user wants to use standard mode instead of single mode

Figure 2 Single Mode



Single mode can only be used in METAREPOS input buffers that specify one service only.
Single mode can be applied under the following conditions:

1. SET operations to only one outstanding particular service
2. GET operations

METAREPOS Request Examples

1. Adding a service deposit to repository

Figure 3 Single Mode Request

```

TA_OPERATIONSET
TA_CLASST_REPOSITORYTA_REOSPARAM USERNAME
TA_STATENEWTA_REPOSTYPE string
TA_REPOSACCESS-
TA_REPOSSERVICEdepositTA_REPOSCOUNT1
TA_REPOSEXPORTYTA_REOSPARAMDESC"username"
TA_REPOSINBUFFML32TA_REPOSSIZE8
TA_REPOSOUTBUFFML32TA_REPOSREQUIREDCOUNT1
TA_REPOSFLDNUM20001
TA_REOSPARAMUSER_INFO
TA_REPOSTYPEFML32TA_REOSPARAMSEX
TA_REPOSACCESSinTA_REPOSTYPEstring
TA_REPOSCOUNT1TA_REPOSACCESS-
TA_REOSPARAMDESC"user accountTA_REPOSCOUNT1
information"TA_REOSPARAMDESC-
TA_REPOSSIZE-(null)TA_REPOSSIZE6
TA_REPOSREQUIREDCOUNT1TA_REPOSREQUIREDCOUNT1
TA_REPOSFLDNUM20000TA_REPOSFLDNUM20002
TA_REPOSEMBED

```

These parameters
are embedded in
TA_REPOSEMBED

```

TA_REOSPARAMACCOUNT_ID
TA_REPOSTYPEinteger
TA_REPOSACCESSin
TA_REPOSCOUNT1
TA_REOSPARAMDESC"deposit account"
TA_REPOSSIZE-

TA_REPOSREQUIREDCOUNT1
TA_REPOSFLDNUM20003
TA_REPOSEMBED-(null)

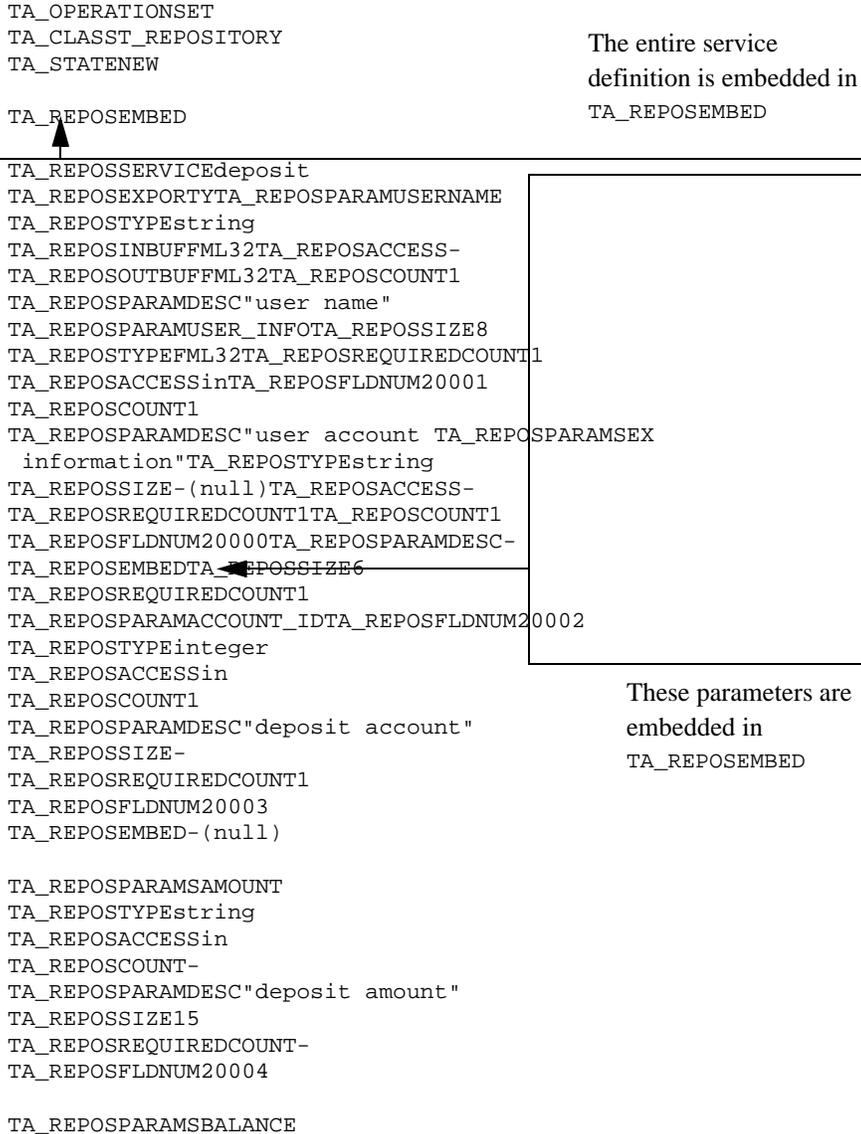
TA_REOSPARAMSAMOUNT
TA_REPOSTYPEstring
TA_REPOSACCESSin
TA_REPOSCOUNT-
TA_REOSPARAMDESC"deposit amount"
TA_REPOSSIZE15
TA_REPOSREQUIREDCOUNT-
TA_REPOSFLDNUM20004

```

METAREPOS(5)

TA_REPOSPARAM	SBALANCE
TA_REPOSTYPE	string
TA_REPOSACCESS	out
TA_REPOSCOUNT	-
TA_REPOSPARAMDESC	"account balance"
TA_REPOSSIZE	15
TA_REPOSREQUIREDCOUNT	-
TA_REPOSFLDNUM	20005
TA_REPOSPARAM	STATLIN
TA_REPOSTYPE	string
TA_REPOSACCESS	out
TA_REPOSCOUNT	-
TA_REPOSPARAMDESC	-
TA_REPOSSIZE	15
TA_REPOSREQUIREDCOUNT	-
TA_REPOSFLDNUM	20006

Figure 4 Standard Mode Request



```

TA_REPOSTYPEstring
TA_REPOSACCESSout
TA_REPOSCOUNT-
TA_REPOSPARAMDESC"account balance"
TA_REPOSSIZE15
TA_REPOSREQUIREDCOUNT-
TA_REPOSFLDNUM20005

```

```

TA_REPOSPARAMSTATLIN
TA_REPOSTYPEstring
TA_REPOSACCESSout
TA_REPOSCOUNT-
TA_REPOSPARAMDESC-
TA_REPOSSIZE15
TA_REPOSREQUIREDCOUNT-
TA_REPOSFLDNUM20006

```

2. Delete service deposit and transfer

Listing 3

TA_OPERATION	SET
A_CLASS	T_REPOSITORY
TA_STATE	DEL
TA_REPOSSERVICE	deposit, transfer

See Also

[tmloadrepos\(1\)](#), [tpgetrepos\(3c\)](#), [tpsetrepos\(3c\)](#), [MIB\(5\)](#), [TMMETADATA\(5\)](#).

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. For additional information pertaining to all `MIB(5)` class definitions, see “[MIB\(5\) Additional Information](#)” on page 259. 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 `tpsadm` or `tpsop`. 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 `tpsadm` or `tpsop` 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 `<fm132.h>` header file instead of the standard `<fm1.h>` header file and then to use the FML32 version of each relevant FML interface specified in the *BEA Tuxedo ATMI FML Function Reference*. For example, one would use `Fchg32()` instead of using `Fchg()`.

Another method for interfacing with FML32 is to include both the `<fm132.h>` header file and the `<fm11632.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, 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, 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, the object is updated using occurrence 1

of any MIB specific class attributes. Attributes occurring less than two times are not considered for pre-image checking. Multiply occurring fields are checked if their associated count attribute is specified twice.

MIB_SELF

This flag is used as a shorthand to indicate that identification attributes for the client or server originating the request should be added to the request buffer prior to processing. For clients, TA_CLIENTID is added and for servers, TA_GRPNO and TA_SRVID are added.

TA_FILTER

Long valued FML32 field that may be specified with up to 32 occurrences to indicate the specific class attributes that should be returned. An occurrence with the value 0 may be specified to end the list but is not required. A list with an initial attribute value of 0 will return no class specific attributes but will return a count of class objects matched.

TA_MIBTIMEOUT

Long valued FML32 field identifying the time, in seconds, that should be allowed within the component MIB service to satisfy the request. A value less than or equal to 0 indicates that the component MIB service should not undertake any blocking operation. If unspecified, this value defaults to 20.

TA_CURSORHOLD

Long valued FML32 field identifying the time, in seconds, that a system snapshot generated from an initial GET operation should be held after the current GET or GETNEXT operation is satisfied before disposing of it. A value less than or equal to 0 indicates that the snapshot should be disposed of after satisfying the current request. If unspecified, this value defaults to 120.

In the following table, R indicates a required INPUT attribute, O an optional INPUT attribute, and — an unused INPUT attribute.

Table 38 Input Table

Attribute	Type	GET	GETNEXT	SET
TA_OPERATION	string	R	R	R
TA_CLASS	string	R	—	R
TA_CURSOR	string	—	R	—
TA_OCCURS	long	O	O	—
TA_FLAGS	long	O	O	O
TA_FILTER	long	O	—	—
TA_MIBTIMEOUT	long	O	O	O
TA_CURSORHOLD	long	O	O	—

Output

Output from successful administrative requests consists of one or more MIB specific objects and one occurrence of the generic output fields. In general, multiple MIB specific objects are reflected in the output buffer by multiple occurrences of each class attribute returned. Occurrence 0 of each attribute relates to the first object, occurrence 1 to the second object, and so on. Exceptions to this guideline are noted in the component MIB reference pages. Intermediate occurrences without values for certain attributes may have FML32-defined `NULL` field values inserted as place holders. A successful `SET` operation returns a single object reflecting the object after the operation was performed. A successful `GET` or `GETNEXT` operation may return 0 or more occurrences depending on how many occurrences were requested (see `TA_OCCURS` below), how many occurrences were matched by the specified key fields and space limitations within the MIB specific system service.

It is important to note that not all attributes defined for any class may necessarily be returned for any request depending on object state, interoperating release environments and/or input request filters. Administrative programmers should avoid implicit dependencies on the presence of certain attributes in output buffers and should instead explicitly check for the presence of attribute values.

To repeat, the reply to a successfully processed administrative request includes certain generic fields that apply to all MIBs. The fields are defined in the header file `<tpadm.h>`. The corresponding field table file can be found in `#{TUXDIR}/udataobj/tpadm`. The generic reply

fields are added to a the reply buffer and returned with the component MIB specific fields. The generic reply fields are described below.

TA_CLASS

String valued field identifying the class represented in the reply buffer. Class names are defined within component MIB specific reference pages.

TA_OCCURS

Long valued FML32 field identifying how many objects are in the reply buffer.

TA_MORE

Long valued FML32 field identifying how many additional objects matching the request key fields are being held in a system snapshot for later retrieval. This field is not returned for SET operations.

TA_CURSOR

String valued FML32 field identifying the position within a system held snapshot. This field must be added to the request buffer for a subsequent GETNEXT operation. The value of this field should not be interpreted or modified by the application user. This field is not returned for SET operations.

TA_ERROR

Long valued FML32 field identifying a non-negative return code characterizing the successful return. Generic return codes and their meaning are defined below.

TAOK

The operation was successfully performed. No updates were made to the application.

TAUPDATED

An update was successfully made to the application.

TAPARTIAL

A partial update was successfully made to the application.

Administrative requests that fail within MIB specific system service processing return an application service failure to the application including the original request and generic fields used to characterize the error. Application service failures are indicated by a TPESVCFailure error return from `tpcall()` or `tpgetrply()`. 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 = tmalloc(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, 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`, 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

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, 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, rpbuf, rplen, flags);
/* Send request and get descriptor back */
flags = TPNOTRAN | TPSIGRSTRT;
cd = tpacall(".TMIB", rqbuf, 0, flags);
/* Enqueue request, assumes qctl already setup */
flags = TPSIGRSTRT;
rval = tpenqueue("queue", ".TMIB", qctl, rqbuf, 0, flags);
```

Receiving MIB Replies

Replies from component MIBs may be received in one of three ways depending on how the original request was generated. If the original request was generated using `tpcall()`, a successful return from `tpcall()` indicates that the reply has been received. If the original request was generated using `tpacall()`, 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, 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 `ACTive` or `ACTive` equivalent. See the description of the `TA_STATE` attribute value for each class to determine which states qualify as `ACTive` equivalent. This attribute represents transient information that is not persistent across distinct activations of the object.

- k Attribute may be specified only as a key field for retrieval or update.

- K Attribute may be specified only as a key field for retrieval or update and then only when the object state is `ACTive` or `ACTive` equivalent. See the description of the `TA_STATE` attribute value for each class to determine which states qualify as `ACTive` equivalent.

 Position 2—Inactive update permissions

- w Attribute may be updated when the object is in an `INActive` or `INActive` equivalent state. See the description of the `TA_STATE` attribute value for each class to determine which states qualify as `INActive` equivalent.

- u Attribute may be updated as described for the `w` permissions value. In addition, the combination of all attribute values identified with the `u` permissions character must be unique within the class.

- U Attribute may be updated as described for the `w` permissions value. In addition, the attribute value must be unique for the attribute within the class.

Position 3—Active update permissions

x	Attribute may be updated when the object is in an <code>ACTive</code> or <code>ACTive</code> equivalent state. See the description of the <code>TA_STATE</code> attribute value for each class to determine which states qualify as <code>ACTive</code> equivalent.
X	Attribute may be updated when the object is in an <code>ACTive</code> or <code>ACTive</code> equivalent state. 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 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 <code>ACTive</code> or <code>ACTive</code> 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 <code>TA_STATE</code> attribute value for each class to determine which states qualify as <code>ACTive</code> 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.
<code>low = num</code>	Numeric value greater than or equal to <code>low</code> .
<code>low = num high</code>	Numeric value greater than or equal to <code>low</code> and less than <code>high</code> .

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 lowercase. Input `TA_STATE` values may be in either shorthand or long notation and are case insensitive. Output `TA_STATE` values are always full length uppercase. 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

Table 39 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 SET: N/A	GET: N/A 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 state indicates 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

Table 40 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 SET: {NEW INV}	GET: N/A 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

Table 40 T_CLASSATT Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_ATTFLAGS	long	r--r--r--	long	N/A
TA_DEFAULT	string	r--r--r--	string	N/A
TA_VALIDATION	string	r--r--r--	string	N/A

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

Attribute Semantics

TA_CLASSNAME: *string*

Class name. Only class names known to the system are accessible.

TA_ATTRIBUTE: *long*

Attribute field identifier as defined in the system provided header file, for example, tpadm.h.

TA_STATE:

GET: VALid

A GET operation will retrieve information for the selected T_CLASSATT object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request.

VALid	T_CLASSATT object is defined. All objects of this class exist in this state. This state is INActive equivalent for the purposes of permissions checking.
-------	--

SET: {NEW | INValid}

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

NEW	Create T_CLASSATT object for application. State change allowed only when in the INValid state. Successful return leaves the object in the VALid state.
<i>unset</i>	Modify T_CLASSATT object. Allowed only when in the VALid state. Successful return leaves the object state unchanged.
INValid	Delete or reset T_CLASSATT object for application. State change allowed only when in the VALid state. Successful return leaves the object in either the INValid state or the VALid state. Objects of this class that are built-in, that is, explicitly known to the system, will revert to their default permissions on this state change and continue to exist in the VALid state. Objects of this class that belong to add-on components for which the class attributes are not explicitly known will be deleted on this state change and transition to the INValid state.

TA_PERM: 0000 <= num <= 0777

Access permissions for this class attribute combination. When setting permissions, the actual value set may be automatically reset if the requested setting exceeds the permissions available for the attribute. The maximum permissions available for an attribute are the permissions documented for the administrator repeated in the operator and other permissions positions. For example, the TA_TYPE attribute of the T_MACHINE class is documented with permissions `rw-r--r--` and has maximum permissions of `rw-rw-rw-`.

TA_FACTPERM: 0000 <= num <= 0777

Permissions for this class attribute combination as set on delivery of the BEA Tuxedo system from the factory. These permissions will apply after a SET operation changing the TA_STATE of an object to INValid.

TA_MAXPERM: 0000 <= num <= 0777

Maximum permissions for this class attribute combination.

TA_ATTFLAGS: *long*

Bitwise or of none, some or all of the following flags indicating special characteristics of this attribute.

MIBATT_KEYFIELD

Attribute is a key field for this class.

MIBATT_LOCAL

Attribute represents local information.

MIBATT_REGEXKEY

Attribute is a regular expression key field for this class.

MIBATT_REQUIRED

Attribute is required when creating a NEW object in this class.

MIBATT_SETKEY

Attribute is a SET key for this class.

MIBATT_NEWONLY

Attribute is writable for inactive equivalent objects in this class only when creating a NEW object by changing the TA_STATE from INValid to NEW.

TA_DEFAULT: *string*

Default for this attribute when creating a NEW object in this class. Note that for classes where NEW objects may not be created through the Admin API, this attribute will always be returned as a 0 length string. Attributes that may not be SET when creating a NEW object are also returned as 0 length strings. Attributes which have *long* values will have defaults returned as the string representing the long value. Some attributes have special characteristics indicated by the special values indicated below that may be returned here.

Inherited: *Classname[:Attribute]*

Attribute default is inherited from the attribute of the same name in the indicated class. If *Attribute* is specified, 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`

Read-only 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.

MIB(5) Additional Information

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, 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.

[TAESYSTEM]

A BEA Tuxedo system error occurred while attempting to satisfy the request. `TA_STATUS` is updated with more information on the error condition.

[TAEUNIQ]

A `SET` operation did not specify class keys identifying a unique object to be updated.

[other]

Other error return codes specific to particular component MIBs are specified in the component MIB reference pages. These error codes are guaranteed to be mutually exclusive both amongst all component MIBs and with generic codes defined here.

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

[TAOK]

The operation succeeded. No updates were done to the component MIB object(s).

[TAUPDATED]

The operation succeeded. Updates were made to the component MIB object.

[TAPARTIAL]

The operation partially succeeded. Updates were made to the component MIB object.

Interoperability

Access to the FML32 interfaces, and therefore to the component MIBs available for administration of a BEA Tuxedo system application, are available on BEA Tuxedo release 4.2.2 and later. The header files and field tables defining generic MIB attributes are available on 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\(3c\)](#),
[tpgetrply\(3c\)](#), [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 ATMI Application Using C

Programming a BEA Tuxedo ATMI Application Using FML

nl_types(5)

Name

nl_types—Native language data types

Synopsis

```
#include <nl_types.h>
```

Description

The `nl_types.h` header file contains the following definitions:

`nl_catd`

Used by the message catalog functions `catopen()`, `catgets()` and `catclose()` to identify a catalogue.

`nl_item`

Used by `nl_langinfo()` to identify items of `langinfo()` data. Values for objects of type `nl_item` are defined in `langinfo.h`.

`NL_SETD`

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.

`NL_MGSMAX`

Maximum number of messages per set.

`NL_SETMAX`

Maximum number of sets per catalogue.

`NL_TEXTMAX`

Maximum size of a message.

`DEF_NLSPATH`

The default search path for locating catalogues.

See Also

[gencat\(1\)](#), [catgets\(3c\)](#), [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][-h][-l locktype][-n prio]
[-o stdout_file][-P][-p [L][low_water][,[terminate_time]]
[:[high_water][,create_time]][-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 `tmbboot(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 run-time 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`.

-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 (`TXLOCK`), data (`DATLOCK`), or the entire process (text and data—`PROCLOCK`), should be locked. See `plock(2)` for details. The lock fails if the server is not run as root. There is no way to unlock a server once it is locked.

-n *prio*

nice the server according to the *prio* argument. Giving the process better priority (a negative argument) requires it to be run with the UID of root. See `nice(2)` for details.

-o *stdout_file*

Specifies the name of a file to be opened as the server's standard output file. Providing this option ensures that a restarted server has the same standard output file as its predecessors. If this option is not used, a default diversion file called `stdout` is created in the directory specified by `$APPDIR`.

-P

Specifies that services advertise running status as:

- **SUSP** (suspended) when booting and `tpsvrinit ()` is running. Requests to a suspended service will fail and return `TPNOENT` immediately.

If `tpsvrinit ()` runs for an extended period of time, the `-P` option helps avoid service requests timeout at the booting stage.

- **AVAIL** (available) after `tpsvrinit ()` has completed and the server is ready to receive requests.

Note: It is highly recommended to use this `CLOPT` with application servers only. Do not use it as the default `CLOPT`, since it may affect all system servers, for example, `TMUSREVT`, `TMSYSEVT`, `GWTDOMAIN`, `GWADM`, `TMS`, `TMQUEUE`, etc.

The `"-P"` option can also be used with CORBA application servers.

`-p [L][low_water][,terminate_time]]:[high_water][,create_time]]`

This option can be used to support the automatic spawning and decaying of servers, both single-threaded RPC servers and conversational servers. For RPC servers, this option must be used on an `MSSQ` set with `MAX` greater than 1. For conversational servers, the `MAX` must be greater than 1.

The decision to spawn/decay servers is based on the number of requests *per server* on the queue. However, if the load `[L]` argument is used with RPC servers, then the load factor of each request is also considered.

If the `-p` option is specified with the `L` argument, then, if the load meets or exceeds a threshold (specified by the `high_water` argument) for a specified amount of time (in seconds), the system will spawn additional servers. If, however, the value of `high_water` is 1, then the single server responsible for spawning another server will not do so as long as it is handling messages.

This problem will persist as long as there is only one request waiting on the queue: the server will process it once it finishes its current request and it will not need to start a new server.

However, when additional requests start arriving and waiting on the queue, then you should eventually see new servers getting started. Again, the new servers will be started when the currently running server finishes processing the current request and starts checking for the next one.

Every time a server returns to its queue to get a new message to process, it checks the conditions governing the need for new servers. If those conditions are met, the server spawns exactly one new server.

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.

Depending on which type of server is being used, arguments to the `-p` option have the following meanings:

RPC Servers

L

The load argument works only with RPC servers. It also only works in SHM mode with load balancing turned on. The decision to spawn more servers is based on the request load, rather than the number of messages per server. If `SHM/LDBAL=Y` is not set, a user log message (`LIBTUX_CAT:1542`) is printed and no spawning or decaying occurs.

low_water, terminate_time, high_water, and create_time

These arguments are used to control when RPC servers are spawned or deactivated based on the number of messages per server. 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. *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.

Conversational Servers

L

The load option is not applicable to conversational servers.

Note: For BEA Tuxedo 8.0 or later, there are no restrictions for the automatic spawning of multi-threaded or non-MSSQ conversational servers. However, the automatic decay feature will *not* be implemented for these types of servers.

low_water, terminate_time, high_water, and create_time

These arguments are used to control when conversational servers are spawned or deactivated. Since conversational servers typically run for a longer time than RPC servers, a conversational server checks the minimum *low_water* percentage and

the maximum *high_water* percentage of other servers that are currently engaged in conversations. If the percentage exceeds the value set for the related time parameters, *terminate_time* and *create_time* respectively, a server may be decayed or spawned, provided that the minimum or maximum number of servers has not been reached.

Also, you can specify a value of 0 seconds for the time parameters so that either a spawn or decay action will occur as soon as the server detects that the percentage has been exceeded. *low_water* percentage defaults to 0% and the *high_water* percentage defaults to 80%. *terminate_time* defaults to 60 seconds and *create_time* defaults to 0 seconds.

-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\(1\)](#), [tmadmin\(1\)](#), [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. For additional information pertaining to all TM_MIB(5) class definitions, see [“TM_MIB\(5\) Additional Information” on page 411](#).

TM_MIB(5) consists of the following classes.

Table 41 TM_MIB Classes

Class Name	Controls . . .
T_BRIDGE	Network connections
T_CLIENT	Clients
T_CONN	Conversations
T_DEVICE	Devices
T_DOMAIN	Global application attributes
T_FACTORY	Factories
T_GROUP	Server groups

Table 41 TM_MIB Classes (Continued)

Class Name	Controls . . .
T_IFQUEUE	Server queue interfaces
T_INTERFACE	Interfaces
T_MACHINE	Machine specific attributes
T_MSG	Message queues
T_NETGROUP	Network groups
T_NETMAP	Machines to Netgroups
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	User log

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

[MIB\(5\)](#) defines the generic `TA_FLAGS` attribute, which is a `long` containing both generic and component MIB specific flag values. The following are the `TM_MIB(5)` specific flag values supported. These flag values should be or'd with any generic MIB flags.

`TMIB_ADMONLY`

A flag used to indicate that only administrative processes should be activated when changing the state of a `T_MACHINE` object from `INActive` to `ACTive`.

`TMIB_APPONLY`

A flag used to indicate that only application processes should be considered when activating or deactivating a `T_MACHINE` object. It may also be used on `T_SERVER` and `T_SERVERTEXT` retrievals to restrict the retrieval to application servers only.

`TMIB_CONFIG`

A flag used to indicate that only configured groups and servers should be considered in satisfying the request.

`TMIB_NOTIFY`

A flag used when activating or deactivating `T_MACHINE`, `T_GROUP`, or `T_SERVER` objects to cause unsolicited notification messages to be sent to the originating client just prior to and just after the activation or deactivation of each server object selected.

FML32 Field Tables

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

Limitations

Access to the header files and field tables for this MIB is being provided only on BEA Tuxedo release 6.1 sites and later, both native and Workstation.

Workstation access to this MIB is limited to run-time only access; the function `tpadmcall(3c)` is not supported on workstations.

For the purpose of pre-image processing (MIB_PREIMAGE flag bit set), local attributes for classes that have global attributes are not considered. Additionally, indexed fields and the indexes that go with them are not considered, for example, T_TLOG class, TA_TLOGCOUNT, TA_TLOGINDEX, TA_GRPNO, TA_TLOGDATA attributes.

T_BRIDGE Class 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

Table 42 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

Table 42 TM_MIB(5): T_BRIDGE Class Definition Attribute Table

Attribute ¹	Type	Permissions	Values	Default
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

¹ All attributes in the T_BRIDGE class are local attributes.

² TA_LMID attribute must be fully specified for SET operations, that is, LMID1, LMID2.

³ SET 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.

⁴ TA_SUSPTIME may be SET only if the TA_STATE is currently SUSPENDED or is being SET to SUSPENDED.

⁵ 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.

<code>ACTive</code>	The connection is established and active.
<code>INActive</code>	The connection is inactive. This state is only returned when status is requested on a particular connection, that is, both <code>Lands</code> specified in the <code>TA_LMID</code> attribute and the source logical machine is reachable.
<code>SUSPended</code>	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 <code>TA_SUSPTIME</code> attribute value. This state is <code>ACTive</code> equivalent for the purpose of determining permissions.
<code>PENding</code>	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.

<code>unset</code>	Modify an existing <code>T_BRIDGE</code> object. This combination is allowed only when in the <code>ACTive</code> or <code>SUSPended</code> state. Successful return leaves the object state unchanged.
<code>ACTive</code>	Activate the <code>T_BRIDGE</code> 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 <code>T_BRIDGE</code> object is establishing the asynchronous connection, the Bridge process will do other work. Using the state change to <code>PENding</code> is recommended. State change allowed in the <code>INActive</code> and <code>SUSPended</code> states. For the purpose of determining permissions for this state transition, the active object permissions are considered (that is, <code>--x--x--x</code>). Successful return leaves the object in the <code>PENding</code> 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 attribute as indicated. State change allowed only when in the ACTIVE state. Successful return leaves the object in the SUSPended state. Limitation: Note that since the statistics reported are from the viewpoint of the source logical machine, resetting those statistics will cause them to be out of sync with the statistics reported by the destination logical machine for the same connection.
-----------	---

PENding	Activate the T_BRIDGE object by establishing an asynchronous connection between the indicated logical machines. This operation will fail if only one logical machine is specified, if either of the two machines is not active, or if the source machine is not reachable. When in the PENding state, the success or failure of the connection request has not yet been determined. However, the Bridge process may continue to process other events and data while the connection is outstanding. State change allowed in the INActive and SUSPended states. For the purpose of determining permissions for this state transition, the active object permissions are considered (that is, --x--x--x). Successful return leaves the object in the PENding state.
---------	--

TA_CURTIME: 0 <= num

Current time, in seconds, since 00:00:00 UTC, January 1, 1970, as returned by the time(2) system call on T_BRIDGE:TA_LMID. This attribute can be used to compute elapsed time from the following attribute value.

TA_CONTIME: 0 <= num

Time, in seconds, since 00:00:00 UTC, January 1, 1970, as returned by the time(2) system call on T_BRIDGE:TA_LMID, when this connection was first established. Elapsed open time in seconds can be computed using TA_CURTIME - TA_CONTIME.

TA_SUSPTIME: 0 <= *num*

Time, in seconds, remaining in the suspension of this connection. After this amount of time, the connection will automatically change to a `TA_STATE` of `INACTIVE` and may be activated by normal application traffic.

TA_RCVDBYT: 0 <= *num*

Number of bytes sent from the destination logical machine to the source logical machine.

TA_SENTBYT: 0 <= *num*

Number of bytes sent from the source logical machine to the destination logical machine.

TA_RCVDNUM: 0 <= *num*

Number of messages sent from the destination logical machine to the source logical machine.

TA_SENTNUM: 0 <= *num*

Number of messages sent from the source logical machine to the destination logical machine.

TA_FLOWCNT: 0 <= *num*

Number of times flow control has been encountered over this connection.

TA_CURENCRYPTBITS: "{0 | 40 | 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

Table 43 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
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_TPBLK_ALL	long	R--R--R--	0 <= num	0
TA_LMID(k)	string	R--R--R--	<i>LMID</i>	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
TA_NUMREQ	long	R-XR-XR--	0 <= num	N/A

Table 43 TM_MIB(5): T_CLIENT Class Definition Attribute Table (Continued)

Attribute 1	Type	Permissions	Values	Default
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..256] ³	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

¹ All attributes in the T_CLIENT class are local attributes.² Link-level encryption value of 40 bits is provided for backward compatibility.³ Maximum string length for this attribute is 78 bytes for BEA Tuxedo 8.0 or earlier.

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.

ACTIVE	Activate a SUSPENDED T_CLIENT object. State change allowed only when in the SUSPENDED state. Successful return leaves the object in the ACTIVE state.
unset	Modify an existing T_CLIENT object. This combination is allowed only when in the ACTIVE or SUSPENDED state. Successful return leaves the object state unchanged.

SUSPENDED Suspend the T_CLIENT object from making service requests (tpcall() or tpacall()), initiating conversations (tpconnect()), beginning transactions (tpbegin()), and enqueueing new requests (tpenqueue()). Clients within a transaction will be permitted to make these calls until they abort or commit the current transaction, at which time they will become suspended. Invocations of these routines will result in a TPESYSTEM error return and a system log message being generated indicating the situation. State change allowed only when in the ACTIVE state. Successful return leaves the object in the SUSPENDED state.

DEAD Abortively deactivate the T_CLIENT object. State change allowed only when in the ACTIVE or SUSPENDED state. The recommended method for deactivating clients is to first broadcast a warning message (tpbroadcast()), then to suspend them (see SET SUSPENDED above), and finally to abortively deactivate them by setting the state to DEAD. Successful return leaves the object in the DEAD state.

Limitation: Workstation handlers (T_CLIENT:TA_WSH == Y) may not be set to a state of DEAD.

The system may not be able to *kill* the client due to platform or signaling restrictions. In this case, a native client will be abortively terminated at its next access to ATMI, and a Workstation client's connection to a WSH will be preemptively torn down.

TA_CLIENTID: *string*[1..78]

Client identifier. The data in this field should not be interpreted directly by the end user except for equality comparison.

TA_CLTNAME: *string*[0..30]

Client name associated with client at tpinit() time via the cltname element of the TPINIT structure.

TA_IDLETIME: 0 <= num

Approximate amount of time, in seconds, since this client last interacted with the system via an ATMI call. This value is accurate to within TA_SCANUNIT (see the T_DOMAIN class) seconds. When specified as a key field, a positive value indicates that all clients with idle times of at least the indicated value match, a negative value indicates that all clients with no more than the indicated value match, and a 0 value matches all clients.

TA_TPBLK_ALL: 0 <= num
 Reports the current `tpsblktime(TPBLK_ALL)` blocktime value per client. If `TPBLK_ALL` has not been set, then the `TA_TPBLK_ALL` value is 0.

TA_LMID: LMID
 Logical machine where client is running (native clients) or where client is connected (Workstation clients).

TA_PID: 1 <= num
 Process identifier of client. Note that for Workstation clients, this identifier indicates the workstation handler through which the Workstation client is connected. A negative number may be specified on a `GET` operation for the purpose of retrieving client information for the calling process. If the calling process is not a client, an error will be returned.

TA_CONTEXTID: -2 <= 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", 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", the indicated client is a workstation handler process.

TA_WSHCLIENTID: string[1..78]
 Client identifier for the associated workstation handler (WSH) if this client is a Workstation client (`TA_WSH == Y`); otherwise, this attribute will be returned as a 0-length string.

TA_RELEASE: 0 <= num
 The BEA Tuxedo system major protocol release number for the machine where the client is running. This may be different from the `TA_SWRELEASE` for the same machine. Note that for Workstation clients (`TA_WSC == Y`), this value may be different than the major release

associated with the application administered machine through which the Workstation client accesses the application.

TA_WSPROTO: 0 <= *num*

The BEA Tuxedo system Workstation protocol version number for a Workstation client. This value is changed with each update to the Workstation protocol. A value of 0 is returned for this attribute when associated with non-Workstation clients (`TA_WSC == N`).

TA_NUMCONV: 0 <= *num*

Number of conversations initiated by this client via `tpconnect()`.

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 \leq *num*

Number of requests initiated by this client via `tpcall()` or `tpacall()` that are still active.

TA_CURTIME: 1 \leq *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 \leq *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..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

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 time out 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

Table 44 TM_MIB(5): T_CONN Class Definition Attribute Table

Attribute ¹	Type	Permissions	Values	Default
TA_LMID(k)	string	R--R--R--	<i>LMID</i>	N/A
TA_STATE(k)	string	R--R--R--	GET: "ACT" SET: N/A	N/A N/A
TA_SERVICENAME	string	R--R--R--	<i>string</i> [1..15]	N/A
TA_CLIENTID(k)	string	R--R--R--	<i>string</i> [1..78]	N/A
TA_CONNOGRPNO	long	R--R--R--	1 <= num < 30,001	N/A
TA_CONNOLMID	string	R--R--R--	<i>LMID</i>	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--	<i>LMID</i>	N/A
TA_CONNSPID	long	R--R--R--	1 <= num	N/A
TA_CONNSSNDCNT	long	R--R--R--	0 <= num	N/A
TA_CONSSRVID	long	R--R--R--	1 <= num < 30,001	N/A

(k)—GET key field

¹ All attributes in the T_CONN class 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, 30,000 is returned as the value for this attribute.

TA_CONNOLMID: *LMID*

Logical machine identifier indicating where the originator is running or is accessing the application (in the case of Workstation clients).

TA_CONNOPID: **1** <= *num*

Process identifier for the originator of the conversation.

TA_CONNOSNDCNT: **0** <= *num*

Number of `tpsend()` calls done by the originator.

TA_CONNOSRVID: **1** <= *num* < **30,001**

Server identifier for the originator of the conversation.

TA_CONNSGRPNO: **1** <= *num* < **30,001**

Server group number for the subordinate of the conversation.

TA_CONNSLMID: *LMID*

Logical machine identifier indicating where the subordinate is running or is accessing the application (in the case of Workstation clients).

TA_CONNSPID: **1** <= *num*

Process identifier for the subordinate in the conversation.

TA_CONSSNDCNT: **0** <= num

Number of tpsend() calls done by the subordinate.

TA_CONSSRVID: **1** <= 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

Table 45 TM_MIB(5): T_DEVICE Class Definition Attribute Table

Attribute ¹	Type	Permissions	Values	Default
TA_LMID(*)	string	ru-r--r--	LMID	"local_lmld"
TA_CFGDEVICE(r)(*)	string	ru-r--r--	string[2..64]	N/A
TA_DEVICE(*)	string	ru-r--r--	string[2..64]	"TA_CFGDEVICE"
TA_DEVOFFSET(*)	long	ru-r--r--	0 <= num	0
TA_DEVSIZE(r)	long	rw-r--r--	0 <= num	1000 ³
TA_DEVINDEXT(*) ²	long	r--r--r--	0 <= num	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

¹ All attributes in the T_DEVICE class are local attributes.

² TA_DEVINDEX is required for SET operations to identify the particular device list entry except when setting the state to NEW for the purpose of creating a new device list entry. In the latter case, TA_DEVINDEX must not be set; a value will be assigned by the system and returned after a successful creation.

³ TA_DEVSIZE may only be SET on object creation.

Attribute Semantics

TA_LMID: *LMID*

Logical machine identifier where the device is located. Note that this attribute may be used as a key field in both unbooted and booted applications as long as they are already configured (that is, at least one T_MACHINE entry is defined). It is required as a key field on SET operations when accessing a booted application. If specified when accessing the T_DEVICE class in an unconfigured application, this attribute is ignored.

TA_CFGDEVICE: *string*[2..64]

Absolute pathname of the file or device where the BEA Tuxedo filesystem is stored or is to be stored.

TA_DEVICE: *string*[2..64]

Absolute pathname of the device list entry.

TA_DEVOFFSET: **0** <= *num*

The offset, in blocks, at which space on this TA_DEVICE begins for use within the BEA Tuxedo System VTOC specified by TA_CFGDEVICE. Limitation: This attribute must be set to 0 for the first device list entry (TA_DEVICE) on the BEA Tuxedo filesystem (TA_CFGDEVICE).

TA_DEVSIZE: **0** <= *num*

The size in pages of the disk area to be used for the device list entry. Limitation: This attribute may be set only in conjunction with a state change to NEW.

TA_DEVINDEX: **0** <= *num*

Device index for TA_DEVICE within the device list addressed by TA_CFGDEVICE. This attribute value is used for identification purposes only in getting and setting attribute values relating to particular devices within a BEA Tuxedo filesystem.

TA_STATE:

GET: "{VALid}"

A GET operation will retrieve run-time information for the selected T_DEVICE object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request.

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 re-installed 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, the object is created; otherwise, it is re-initialized. 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

Table 46 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
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

Table 46 TM_MIB(5): T_DOMAIN Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
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_MAXSPDATA	long	rw-r--r--	1 <= num <= 2147483640	TA_MAXQUEUES*257*2 + 8224
				0
TA_MAXTRANTIME	long	rwyr--r--	1 <= num <= 2147483647	50
TA_MAXSERVERS	long	rw-r--r--	1 <= num < 8,192	100
TA_MAXSERVICES	long	rw-r--r--	1 <= num < 1,048,575	16,384
TA_MAXACLGROUPS	long	rw-r--r--	1 <= num < 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 SSL MIGRATE ACCSTATS NO_XA NO_AA],*}"	" "

Table 46 TM_MIB(5): T_DOMAIN Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
TA_USIGNAL	string	rw-r--r--	"{SIGUSR1 SIGUSR2}"	"SIGUSR2"
TA_SECURITY	string	rw-r--r--	"{NONE APP_PW USER_AUTH ACL MANDATORY_ACL}"	"NONE"
TA_PASSWORD	string	-wx-----	string[0..30]	N/A
TA_AUTHSVC	string	rwxr--r--	string[0..15]	" "
TA_SSL_RENEGOTIATION	long	rwxr--r--	0 <= num <= 2147483647	0
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

Table 46 TM_MIB(5): T_DOMAIN Class Definition Attribute Table (Continued)

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_HWRTDATA	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..1023]	""
TA_SEC_PRINCIPAL_PASSVAR	string	rwxr--r--	string[0..31]	""
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)

¹ UID and GID as known to the UNIX system

² num must be a multiple of 2 or 5

³ Specify num so that num times TA_SCANUNIT is approximately "Default"

Attribute Semantics

TA_IPCKEY: 32,769 <= num < 262,144

Numeric key for the well-known address in a BEA Tuxedo system bulletin board. In a single processor environment, this key “names” the bulletin board. In a multiple processor or LAN environment, this key names the message queue of the DBBL. In addition, this key is used as a basis for deriving the names of resources other than the well-known address, such as the names for bulletin boards throughout the application.

TA_MASTER: "LMID1[,LMID2]"

Master (*LMID1*) and backup (*LMID2*) logical machine identifiers. The master identifier (*LMID1*) must correspond to the local machine for *INActive* applications. *SHM* mode applications (see *TA_MODEL* below) may set only the master logical machine identifier. Modifications to this attribute value in an *ACTive* *MP* application (see *TA_MODEL* below) have the following semantics:

Assuming current active master LMID *A*, current backup master LMID *B*, and secondary LMIDs *C*, *D*, . . . , the following scenarios define the semantics of permitted changes to the *TA_MASTER* attribute in a running *MP* mode application.

A,B -> *B,A* - Master migration from *A* to *B*.

A,B -> *A,C* - Change backup master LMID designation to *C*.

Note that master migration may be either orderly or partitioned. Orderly migration takes place when the master machine is *ACTive* and reachable. Otherwise, partitioned migration takes place. All newly established or reestablished network connections will verify that the two sites connecting share a common view of where the master machine is. Otherwise, the connection will be refused and an appropriate log message generated. The master and backup machines in an *ACTive* application must always have a BEA Tuxedo 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.
<i>INActive</i>	<i>T_DOMAIN</i> 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_P MID 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, it will default to the current directory.
unset	Modify T_DOMAIN object. Allowed only when in the ACTive or INActive state. Successful return leaves the object state unchanged.
INValid	Delete T_DOMAIN object for application. State change allowed only when in the INActive state. Successful return leaves the object in the INValid state.
ACTive	Activate administrative processes (DBBL, BBL, etc.) on the master machine. For the purpose of determining permissions for this state transition, the active object permissions are considered (that is, --x--x--x). State change allowed only when in the INActive state. Successful return leaves the object in the INActive state.
INActive	Deactivate administrative processes (DBBL, BBL, etc.) on the master machine. State change allowed only when in the ACTive state. Successful return leaves the object in the INActive state.
FINActive	Forcibly deactivate administrative processes (DBBL, BBL, etc.) on the master machine. Attached clients will be ignored for the purpose of determining if shutdown should be allowed. State change allowed only when in the ACTive state. Successful return leaves the object in the INActive state.

TA_DOMAINID: *string*[0..30]

Domain identification string.

TA_PREFERENCES: *string*[0..1023]

Application defined field. This field is used by the BEA Tuxedo system /Admin GUI product to store and save GUI display preferences.

TA_UID: 0 <= *num*

Default attribute setting for newly configured objects in the T_MACHINE class. Limitation: Changes to this attribute do not affect active or already configured T_MACHINE objects.

TA_GID: 0 <= *num*

Default attribute setting for newly configured objects in the T_MACHINE class. Limitation: Changes to this attribute do not affect active or already configured T_MACHINE objects.

TA_PERM: 0001 <= *num* <= **0777**

Default attribute setting for newly configured objects in the T_MACHINE class. Limitation: Changes to this attribute do not affect active or already configured T_MACHINE objects.

TA_LICEXPIRE: *string*[**0..78**]

Expiration date for the binary on that machine or a 0-length string if binary is not a BEA Tuxedo system master binary.

TA_LICMAXUSERS: 0 <= *num* < **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 <= *num* <= **0777**

Attribute access mask. User type/access mode combinations specified by this attribute value will no longer be allowed for all class/attribute combinations defined in this reference page. For example, a setting of 0003 disallows all updates to users other than the administrator or the operator.

TA_MAXACCESSERS: 1 <= *num* < **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, 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 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 <= 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 <= 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 <= 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 <= num < 32,761

Maximum string pool space in bytes to be accommodated in the bulletin board string pool table. Strings and arrays specified within `TA_RANGES` values are stored in the string pool. Additional space should be allocated to allow for run-time growth.

TA_MAXSPDATA 0 <= num <= 2147483640

Maximum string pool space in bytes to be accommodated in the bulletin board common string pool. This value must be greater than or equal to 0 and less than or equal to 2147483640. This attribute applies only to applications running BEA Tuxedo 8.1 or later software.

BEA Tuxedo system needs the bulletin board string pool size (`MAXQUEUES * 257 * 2 + 8224`) at a minimum, where `MAXQUEUES` is the `TA_MAXQUEUES` attribute value in the same `T_DOMAIN` class. If this attribute is set with value that smaller than the minimum required size, BEA Tuxedo system automatically changes the value to the minimum required size.

For applications for which extensive dynamic configuration is anticipated (for example, anticipating the addition of six more machines to a BEA Tuxedo application), administrators can use the `TA_MAXSPDATA` attribute to increase the size of the common string pool. Note that adjusting the size of the common string pool has no effect on the size of the of the routing string pool controlled by the `TA_MAXRTDATA` attribute. The two string pools are separate.

Regardless of the value specified for `TA_MAXSPDATA`, the BEA Tuxedo system will *not* allocate an amount of string pool space outside of a system-calculated range based on (1) the strings actually specified in the `TUXCONFIG` file and (2) the amount of space that would be required if all 256-byte capable strings were specified. The `tmloadcf(1)` command will report a warning if the user-specified value is outside of this range and then set the value to the closest acceptable value.

Note that of the `TUXCONFIG` parameters whose maximum allowable length has been increased to 256 bytes, only the `GROUPS` section `TMSNAME` parameter and the `SERVERS` section `AOUT` and `RCMD` parameters are actually stored in the bulletin board. The others are read in at process startup time and stored in process memory.

TA_MAXTRANTIME 0 <= num <= 2147483647

Maximum timeout in seconds allowed for transactions started in or received by this BEA Tuxedo application. This value must be greater than or equal to 0 and less than or equal to 2147483647. The default is 0, which indicates that no global transaction timeout limit

is in effect. This attribute applies only to applications running BEA Tuxedo 8.1 or later software.

If the `TA_MAXTRANTIME` timeout value is less than the `TRANTIME` timeout value specified for an `AUTOTRAN` service or the timeout value passed in a `tpbegin(3c)` call to start a transaction, the timeout for a transaction is reduced to the `TA_MAXTRANTIME` value. `TA_MAXTRANTIME` has no effect on a transaction started on a machine running BEA Tuxedo 8.0 or earlier software, except that when a machine running BEA Tuxedo 8.1 or later software is infected by the transaction, the transaction timeout value is capped—reduced if necessary—to the `TA_MAXTRANTIME` value configured for that machine.

Even if the `TRANTIME` value specified in the `SERVICES` section of the `UBBCONFIG` file is greater than the `TA_MAXTRANTIME` value, the `tmloadcf(1)` command loads the configuration without error. Any BEA Tuxedo 8.1 or later machine infected with the `AUTOTRAN` transaction will automatically reduce the transaction timeout to the `TA_MAXTRANTIME` value configured for that machine.

Limitation: Run-time modifications to this attribute do not affect transactions started before the update takes place.

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 < 1,048,575

Maximum number of services to be accommodated in the bulletin board service table. This value must be greater than 0 and less than 1,048,575. If not specified, the default is 100.

To calculate an adequate value, be sure to count the number of services used by both application servers and system servers, such as the `BBL`, `DBBL`, `BRIDGE`, `TMS`, and any other system-supplied servers needed for administrative purposes. For each BEA Tuxedo system site, add approximately five services to accommodate administration for the site.

You should also include any administrative services that are added to support administrative components such as Workstation, /Q, and Domains.

TA_MAXACLGROUPTS: 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_MAXACLGROUPTS - 1.

TA_CMTRET: "{COMPLETE | LOGGED}"

Initial setting of the TP_COMMIT_CONTROL characteristic for all client and server processes in a BEA Tuxedo system application. LOGGED initializes the TP_COMMIT_CONTROL characteristic to TP_CMT_LOGGED; otherwise, it is initialized to TP_CMT_COMPLETE. See the description of the BEA Tuxedo System ATMI function `tpscmt()` 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 | SSL | MIGRATE | ACCSTATS | NO_XA | NO_AA], *}"`

Comma-separated list of application options in effect. Valid options are defined below:

`LAN`—networked application.

`SSL`—initiates SSL security. If not entered, LLE security is used.

`MIGRATE`—allow server group migration.

`ACCSTATS`—exact statistics (SHM mode only).

NO_XA—do not allow XA transactions.

NO_AA—the auditing and authorization plugin functions will not be called.

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.

Note: If the NO_AA value is enabled in the TA_OPTIONS attribute, the security values NONE, APP_PW, and USER_AUTH will continue to work properly—except that no authorization or auditing will take place. The remaining modes of security, ACL and MANDATORY_ACL will continue to work properly—but will only use the default BEA security mechanism.

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_SSL_RENEGOTIATION: 0 <= num <= 2147483647

Specifies the renegotiation interval (in seconds) for SSL information. It must be greater than or equal to 0 and less than or equal to 2,147,483,647. The default value is 0 (which indicates that no renegotiation takes place).

Changes made to this parameter while the system is running will take effect during the next renegotiation interval.

TA_SCANUNIT: 0 <= num <= 60 (multiple of 5)

Interval of time (in seconds) between periodic scans by the system. Periodic scans are used to detect old transactions and timed-out blocking calls within service requests. The TA_BBLQUERY, TA_BLOCKTIME, TA_DBBLWAIT, and TA_SANITYSCAN attributes are multipliers of this value. Passing a value of 0 for this attribute on a SET operation will cause the attribute to be reset to its default.

TA_BBLQUERY: 0 <= num < 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 <= 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 <= 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 <= 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 <= num < 32,768

Current number of in use bulletin board routing table entries.

TA_CURGROUPS: 0 <= num < 32,768

Current number of in use bulletin board server group table entries.

TA_CURMACHINES: 0 <= num < 32,768

Current number of configured machines.

TA_CURQUEUES: 0 <= num < 32,768
Current number of in use bulletin board queue table entries.

TA_CURRFT: 0 <= num < 32,768
Current number of in use bulletin board routing criteria range table entries.

TA_CURRTDATA: 0 <= num < 32,768
Current size of routing table string pool.

TA_CURSERVERS: 0 <= num < 32,768
Current number of in use bulletin board server table entries.

TA_CURSERVICES: 0 <= num < 32,768
Current number of in use bulletin board service table entries.

TA_CURSTYPE: 0 <= num < 32,768
Current number of in use bulletin board subtype table entries.

TA_CURTYPE: 0 <= num < 32,768
Current number of in use bulletin board type table entries.

TA_HWDRT: 0 <= num < 32,768
High water number of in use bulletin board routing table entries.

TA_HWGROUPS: 0 <= num < 32,768
High water number of in use bulletin board server group table entries.

TA_HWMACHINES: 0 <= num < 32,768
High water number of configured machines.

TA_HWQUEUES: 0 <= num < 32,768
High water number of in use bulletin board queue table entries.

TA_HWRFT: 0 <= num < 32,768
High water number of in use bulletin board routing criteria range table entries.

TA_HWRDATA: 0 <= num < 32,768
High water size of routing table string pool.

TA_HWSERVERS: 0 <= num < 32,768
High water number of in use bulletin board server table entries.

TA_HWSERVICES: 0 <= 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..1023]

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 1023 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..31]

Variable in which the password for the principal specified in TA_SEC_PRINCIPAL_NAME is stored. This attribute may contain a maximum of 31 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 applies only 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 applies only 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 applies only 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 applies only 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_FACTORY MIB

Overview

The T_FACTORY MIB class represents occurrences of factories registered with the FactoryFinder. The available factories for the application are reflected in this MIB and can be shown to the administrator via the Administration Console or command-line tools. The scope is global.

Attribute Table

Table 47 TM_MIB(5): T_FACTORY Attributes

Attribute	Usage	Type	Permissions	Values	Default
TA_STATE	k	string	R--R--R--	GET: "{ACT}"	N/A
TA_FACTORYID	k	string	R--R--R--	<i>string</i> [1..25]	N/A
TA_INTERFACENAME	k	string	R--R--R--	<i>string</i> [1..128]	N/A

(k) - GET key field

Attributes Semantics

TA_STATE

GET: {ACTive }

A GET operation will retrieve configuration and run-time information for the selected T_FACTORY objects.

The following state indicates the meaning of a TA_STATE returned in response to a GET request:

ACTive	The T_FACTORY object is registered with the FactoryFinder.
--------	--

TA_FACTORY

The registered ID for the factory.

TA_INTERFACENAME

The fully qualified interface name for the factory. The interface repository ID for the factory. The format of this name is dependent on the options specified in the IDL which generates the interface implementation. See CORBA 2.1 Specification Section 7.6 for details.

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

Table 48 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) ¹	string	rwyr--r--	"LMID1[,LMID2]"	N/A
TA_STATE(k)	string	rwxr--xr--	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..256] ²	" "
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..256] ²	" "

Table 48 TM_MIB(5): T_GROUP 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..1023]	""
TA_SEC_PRINCIPAL_PASSVAR	string	rwxr--r--	<i>string</i> [0..31]	""
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

¹ TA_LMID must be unique within this class.

² Maximum string length for this attribute is 78 bytes for BEA Tuxedo 8.0 or earlier.

Attribute Semantics

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

Logical name of the server group. The group name must be unique within all group names in the T_GROUP class and TA_LMID values in the T_MACHINE class. Server group names cannot contain an asterisk (*), comma, or colon.

TA_GRPNO: **1** <= *num* < **30,000**

Group number associated with this server group.

TA_LMID: "*LMID1*[,*LMID2*]"

Primary machine logical machine identifier for this server group (*LMID1*) and optional secondary logical machine identifier (*LMID2*). The secondary LMID indicates the machine to which the server group can be migrated (if the MIGRATE option is specified in the T_DOMAIN:TA_OPTIONS attribute). A single LMID specified on a GET operation will match either the primary or secondary LMID. Note that the location of an active group is available in the TA_CURLMID attribute. Logical machine identifiers specified with the TA_LMID attribute must be already configured. Limitation: Modifications to this attribute for an active object may only change the backup LMID designation for the group.

TA_STATE:

GET: "{ACTIVE | INACTIVE | MIGRATING}"

A GET operation will retrieve configuration and run-time information for the selected T_GROUP object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request.

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.
unset	Modify an existing T_GROUP object. This combination is allowed only when in the ACTIVE or INACTIVE state. Successful return leaves the object state unchanged.
INVALID	Delete T_GROUP object for application. State change allowed only when in the INACTIVE state. Successful return leaves the object in the INVALID state.

ACTive	<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, 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, the request fails.</p> <p>If the group is currently in the MIGrating state, 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: *LMD*

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..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)**

Environment file for servers running in this group. If the value specifies an invalid filename, no values are added to the environment. the value of *string* is placed in the environment.

When booted, 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 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 does not 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, *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 #
OPENINFO = "TUXEDO/QM: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 Windows, the separator after `qmconfig` must be a semicolon (;).

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 <= num < 11

If a non `NULL` string is specified for the `TA_TMSNAME` attribute, the `TA_TMSCOUNT` 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..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

Transaction manager server `a.out` associated with this group. This attribute 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, 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..1023]

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 1023 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..31]

Variable in which the password for the principal specified in TA_SEC_PRINCIPAL_NAME is stored. This attribute may contain a maximum of 31 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 applies only 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 applies only 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_IFQUEUE Class

Overview

The T_IFQUEUE MIB class represents run-time attributes of an interface as it pertains to a particular server queue (T_QUEUE) in a CORBA environment. This is primarily a read-only class providing access to the inherited configuration attributes of an interface as well as statistics relating to the interface on the queue. Additionally, this class gives administrators finer granularity in suspending and activating interfaces. This class provides the link between an interface name and the server processes capable of processing method invocations on the interface, that is, TA_RQADDR can be used as a key search field on the T_SERVER class.

Attribute Table

Table 49 TM_MIB(5): T_IFQUEUE Class Definition Attribute Table

Attribute	Usage	Type	Permissions	Values	Default
TA_INTERFACENAME	*	string	R--R--R--	string[1..128]	N/A
TA_SRVGRP	*	string	R--R--R--	string[1..30]	N/A
TA_RQADDR	*	string	R--R--R--	string[1..30]	N/A
TA_STATE	k	string	R-XR-XR--	GET: "{ACT SUS PAR}" SET: "{ACT SUS}"	N/A
TA_AUTOTRAN		string	R--R--R--	"{Y N}"	N/A

Table 49 TM_MIB(5): T_IFQUEUE Class Definition Attribute Table (Continued)

TA_LOAD		long	R--R--R--	1 <= num < 32K	N/A
TA_PRIO		long	R--R--R--	1 <= num < 101	N/A
TA_TIMEOUT		long	R--R--R--	0 <= num	N/A
TA_TRANTIME		long	R--R--R--	0 <= num	N/A
TA_FBRROUTINGNAME		string	R--R--R--	string[1..15]	N/A
TA_LMID	k	string	R--R--R--	LMID	N/A
TA_NUMSERVERS		long	R--R--R--	0 <= num	N/A
TA_RTPOLICY		string	R--R--R--	"{always never}"	never
TA_TPPOLICY		string	R--R--R--	"{method transaction process}"	N/A
TA_TXPOLICY		string	R--R--R--	"{always never optional ignore}"	N/A
TA_NCOMPLETED	l	long	R-XR-XR--	0 <= num	N/A
TA_NQUEUED	l	long	R--R--R--	0 <= num	N/A
TA_CUROBJECTS	l	long	R--R--R--	0 <= num	N/A
TA_CURTRANSACTIONS	l	long	R--R--R--	0 <= num	N/A

(k) - GET key field

(l) - local Field

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

Attribute Semantics

TA_INTERFACENAME: *string*[1..128]

The fully qualified interface name. The interface repository id for the interface. The format of this name is dependent on the options specified in the IDL which generates the interface implementation. See CORBA 2.1 Specification Section 7.6 for details.

TA_SRVGRP: *string*[0..30]

Server group name. Server group names cannot contain an asterisk, comma or colon.

TA_RQADDR: *string*[1..30]

Symbolic address of the request queue for an active server offering this interface. See T_SERVER:TA_RQADDR for more information on this attribute.

TA_STATE:

GET: "{ACTIVE | SUSPENDED | PARTITIONED}"

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

ACTIVE	T_IFQUEUE object represents an available interface in the running system.
SUSPENDED	T_IFQUEUE object represents a currently suspended interface in the running system.
PARTITIONED	T_IFQUEUE object represents a currently partitioned interface in the running system.

SET: "{ACTIVE | SUSPENDED}"

The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

ACTIVE	Activate the T_IFQUEUE object. State change only allowed when in the SUSPENDED state. Successful return leaves object in ACTIVE state.
SUSPENDED	Suspend the T_IFQUEUE object. State change only allowed when in the ACTIVE state. Successful return leaves object in SUSPENDED state.

Limitation: Dynamic advertisement of interfaces (i.e., state change from INACTIVE or INVALID to ACTIVE) is not supported, nor is unadvertisement (i.e., state change from ACTIVE to INACTIVE).

TA_AUTOTRAN: "{Y | N}"

Signifies whether a transaction will be automatically started for invocations made outside a transaction context. See T_INTERFACE description of this attribute for discussion of limitations regarding this attribute.

TA_LOAD: 1 <= num <= 32K

This `T_INTERFACE` object imposes the indicated load on the system. Interface 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_INTERFACE` object has the indicated dequeuing priority. If multiple interface requests are waiting on a queue for servicing, the higher priority requests will be handled first.

TA_TIMEOUT: 0 <= num

Time limit (in seconds) for processing individual method invocations for this interface. Servers processing method invocations for this interface will be abortively terminated if they exceed the specified time limit in processing the request. A value of 0 for this attribute indicates that the server should not be abortively terminated.

TA_TRANTIME: 0 <= num

Transaction timeout value in seconds for transactions automatically started for this `T_INTERFACE` object. Transactions are started automatically when a request not in transaction mode is received and the `T_INTERFACE:TA_AUTOTRAN` attribute value for the interface is "Y".

TA_FBROUTINGNAME: string[1..15]

The factory-based routing criteria associated with this interface.

TA_LMID: LMID

Current logical machine on which the queue offering this interface is located.

TA_NUMSERVERS: 0 <= num

Number of corresponding servers offering this interface on this queue.

TA_RTPOLICY: "{always | never}"

Used to mark an interface implementation as idempotent in the `implementation configuration file (ICF)`. An idempotent implementation can be repeated without any negative side-effects. For example, `SET BALANCE`.

TA_TPPOLICY: "{method | transaction | process}"

The TP framework deactivation policy. This reflects the policy registered with the framework at server startup. The first server to register the interface sets the value in `T_INTERFACE`. This value cannot be changed.

TA_TXPOLICY: "{optional | always | never | ignore}"

The transaction policy for the interface. The setting in this attribute affects the effect of the `TA_AUTOTRAN` attribute. See `TA_AUTOTRAN` for further explanation. This attribute is

always read-only. It is set by the developer when the server is built and registered at server startup.

TA_NCOMPLETED: $0 \leq num$

Number of interface method invocations completed since the interface was initially offered.

TA_NQUEUED: $0 \leq num$

Number of requests currently enqueued for this interface.

TA_CUROBJECTS: $0 \leq num$

Number of active objects for this interface for associated queue. This number represents the number of entries in the active object table for this queue on the associated machine. This includes objects that are not in memory but that were invoked within an active transaction.

TA_CURTRANSACTIONS: $0 \leq num$

Number of active global transactions associated with this interface for its associated queue.

T_INTERFACE Class

Overview

The **T_INTERFACE** MIB class represents configuration and run-time attributes of CORBA interfaces at both the domain and server group levels.

A domain-level **T_INTERFACE** object is one that is not associated with a Server Group. Its **TA_SRVGRP** attribute contains a **NULL** string (string of length 0, "").

A server group level **T_INTERFACE** object is one that has an associated server group (i.e., its **TA_SRVGRP** attribute contains a valid server group name for the domain). This Server Group level representation of an interface also provides a container for managing interface state (**TA_STATE**) and for collecting accumulated statistics.

An associated server group level **T_INTERFACE** object must exist for any CORBA Interfaces that are activated in a server. The activation of interfaces in a server is controlled by the state of a **T_IFQUEUE** object for the interface. Activation of a **T_IFQUEUE** object causes its attributes to be initialized with the values specified for the associated server group level **T_INTERFACE** object. If such an object does not exist, one will be dynamically created. This dynamically-created server group level **T_INTERFACE** object will be initialized with the attributes of the domain level **T_INTERFACE** object for the interface if one exists. If an associated domain level **T_INTERFACE**

object does not exist, system specified default configuration values will be applied. Once activated, interfaces are always associated with a server group level `T_INTERFACE` object.

The specification of configuration attributes for interfaces at any level is completely optional, system defined defaults will be provided and run-time server group level `T_INTERFACE` objects will be created. Interfaces to be offered by a server are identified via the ICF file used to generate server skeletons and advertised automatically by the system at server activation time.

Attribute Table

Table 50 TM_MIB(5): T_INTERFACE Class Definition Attribute Table

Attribute	Usage	Type	Permissions	Values	Default
TA_INTERFACENAME	r*	string	ru-r--r--	<i>string</i> [1..12]	N/A
TA_SRVGRP	r*	string	ru-r--r--	<i>string</i> [0..30]	N/A
TA_STATE	k	string	rwxr-xr--	GET: "{ACT INA SUS PAR}" SET: "{NEW INV ACT REA SUS}"	N/A
TA_AUTOTRAN		string	rwxr-xr--	"{Y N}"	"N"
TA_LOAD		long	rwxr-xr--	1 <= num < 32K	50 ¹
TA_PRIO		long	rwxr-xr--	1 <= num < 101	50
TA_TIMEOUT		long	rwxr-xr--	0 <= num	0
TA_TRANTIME		long	rwxr-xr--	0 <= num	30
TA_FBROUTINGNAME		string	rwyr-yr--	<i>string</i> [1..15]	(?)
TA_LMID	k	string	R--R--R--	<i>LMID</i>	N/A
TA_NUMSERVERS		long	R--R--R---	0 <= num	N/A
TA_RTPOLICY		string	R--R--R--	"{always never}"	never
TA_TPPOLICY		string	R--R--R--	"{method transaction process}"	N/A

Table 50 TM_MIB(5): T_INTERFACE Class Definition Attribute Table (Continued)

Attribute	Usage	Type	Permissions	Values	Default
TA_TXPOLICY		string	R--R--R--	"{always never optional ignore}"	N/A
TA_NCOMPLETED	1	long	R-XR-XR--	0 <= num	N/A ³
TA_NQUEUED	1	long	R--R--R--	0 <= num	N/A

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

1. Group level T_INTERFACE objects (TA_SRVGRP != "") determine their defaults from the domain level T_INTERFACE object with a matching TA_INTERFACENAME setting if one exists. The listed defaults apply if no domain level object exists or if a domain level object is being created.

2. All T_INTERFACE objects with the same TA_INTERFACENAME must have matching TA_FBROUTINGNAME values. Therefore, the default for a newly configured object is the 0 length string ("") if there are currently no matching objects with the same TA_INTERFACENAME. Otherwise, the default (and in fact only legal value) is the currently configured TA_FBROUTINGNAME value for the existing matched objects.

3. TA_NCOMPLETED and TA_IMPLID (locals) require TA_LDBAL="Y" in the T_DOMAIN MIB class.

Attribute Semantics

TA_INTERFACENAME: *string*[1..128]

The fully qualified interface name. The interface repository ID for the interface. The format of this name is dependent on the options specified in the IDL which generates the interface implementation. See CORBA 2.1 Specification Section 7.6 for details.

TA_SRVGRP: *string*[0..30]

Server group name. Server group names cannot contain an asterisk, comma or colon. An explicitly specified 0 length string for this attribute is used to specify and query domain level configuration and run-time information for an interface. There are certain limitations and semantic differences noted in other attributes with respect to domain and group level objects in this class.

TA_STATE:

Following are the semantics for GET and SET TA_STATE values on the T_INTERFACE class. Where semantics differ between group and domain level objects, those differences are noted.

GET: "{Active | INActive | SUSPended | PARTitioned}"

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

ACTive	T_INTERFACE object is defined and at least one corresponding T_IFQUEUE entry is in the ACTive state. Note: For a group level T_INTERFACE object, corresponding T_IFQUEUE entries are those with matching TA_INTERFACENAME and TA_SRVGRP attributes. For a domain level T_INTERFACE object, corresponding T_IFQUEUE entries are those with matching TA_INTERFACENAME attributes regardless of their TA_SRVGRP value.
INACTive	T_INTERFACE object is defined and there are no corresponding T_IFQUEUE entries in any ACTive equivalent state.
SUSPended	T_INTERFACE object is defined and amongst all corresponding T_IFQUEUE entries there are none in the ACTive state and at least one in the SUSPended state. This state is ACTive equivalent for the purpose of determining permissions.
PARTitioned	T_INTERFACE object is defined and amongst all corresponding T_IFQUEUE entries there are: <ol style="list-style-type: none">1. None in the ACTive state2. None in the SUSPended state and3. At least one in the PARTitioned state. This state is ACTive equivalent for the purpose of determining permissions.

SET: "{NEW | INValid | ACTive | REACTivate | SUSPended}"

A SET operation will update configuration and run-time information for the selected T_INTERFACE object. Note that modifications may affect more than one server group when making domain level changes and run-time modifications may affect more than one server if multiple servers are currently offering an interface. 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_INTERFACE object for application. State change only allowed when in the INValid state. Successful return leaves object in INActive state. Creation of a domain level T_INTERFACE object will affect existing group level objects with the same TA_INTERFACENAME value by resetting all TA_FBROUTINGNAME values if a new value is explicitly specified. All other configuration attribute settings will not affect existing group level T_INTERFACE objects.
INValid	Delete T_INTERFACE object for application. State change only allowed when in the INActive state. Successful return leaves object in INValid state.
ACTive	Activate the T_INTERFACE object. Setting this state on the domain level object has the effect of activating all corresponding T_IFQUEUE entries that are currently SUSPended throughout the domain. Setting this state on the group level object will affect only servers within the group offering the interface. State change only allowed when in the SUSPended state. Successful return leaves object in ACTive state.
REACTivate	Reactivate the T_INTERFACE object. Setting this state on the domain level object has the effect of activating all corresponding T_IFQUEUE entries that are currently SUSPended throughout the domain. Setting this state on the group level object will affect only servers within the group offering the interface. State change only allowed when in the ACTive or SUSPended states. Successful return leaves object in ACTive state. This state permits global activation of T_IFQUEUE entries suspended at the group level without having to individually activate each group level T_INTERFACE object.
SUSPended	Suspend the T_INTERFACE object. Setting this state on the domain level object has the effect of suspending all corresponding T_IFQUEUE entries that are currently ACTive throughout the domain. Setting this state on the group level object will affect only servers within the group offering the interface. State change only allowed when in the ACTive state. Successful return leaves object in SUSPended state.

Limitation: Dynamic advertisement of interfaces (i.e., state change from INActive or INValid to ACTive) is not supported, nor is unadvertisement (i.e., state change from ACTive to INActive).

TA_AUTOTRAN: "{Y | N}"

Signifies whether a transaction will be automatically started for invocations made outside a transaction context.

Limitations: Run-time updates to this attribute are not reflected in active equivalent T_INTERFACE objects and TA_TXPOLICY may override the value specified for this attribute in the UBBCONFIG file. If TA_TXPOLICY is:

always	A value of N will have no effect at run time. Behavior will be as though the setting was Y.
never	A value of Y will have no effect at run time. The interface will never be involved in a transaction.
ignore	A value of Y will have no effect at run time. The interface will never be involved in a transaction.

TA_LOAD: 1 <= num <= 32K

This T_INTERFACE object imposes the indicated load on the system. Interface loads are used for load balancing purposes, that is, queues with higher enqueued workloads are less likely to be chosen for a new request.

Limitation: Run-time updates to this attribute for domain level objects will not affect corresponding group level objects for the same interface.

TA_PRIO: 1 <= num <= 101

This T_INTERFACE object has the indicated dequeuing priority. If multiple interface requests are waiting on a queue for servicing, the higher priority requests will be handled first.

Limitation: Run-time updates to this attribute for domain level objects will not affect corresponding group level objects for the same interface.

TA_TIMEOUT: 0 <= num

Time limit (in seconds) for processing individual method invocations for this interface. Servers processing method invocations for this interface will be abortively terminated if they exceed the specified time limit in processing the request. A value of 0 for this attribute indicates that the server should not be abortively terminated.

Limitation: Run-time updates to this attribute for domain level objects will not affect corresponding group level objects for the same interface.

TA_TRANTIME: 0 <= num

Transaction timeout value in seconds for transactions automatically started for this T_INTERFACE object. Transactions are started automatically when a requests not in transaction mode is received and the T_INTERFACE: TA_AUTOTRAN attribute value for the interface is "Y".

Limitation: Run-time updates to this attribute for domain level objects will not affect corresponding group level objects for the same interface.

Note: Updating this value at run-time for domain level objects should cause a warning, since the only use would be to set the default for a subsequent boot of the application.

TA_FBROUTINGNAME: *string*[1..15]

The factory-based routing criteria associated with this interface. The name `FBROUTINGNAME` is used to allow for the future possibility of other routing criteria for message-based routing. This will be less confusing than trying to overload `ROUTINGNAME`.
Limitation: This attribute may be set only for a domain level `T_INTERFACE` object, i.e., `TA_SRVGRP` is "".

TA_LMID: *LMID*

Current logical machine with which the active equivalent group level `T_INTERFACE` object is associated. This attribute is blank, i.e., "" for domain level objects unless a local query is performed, i.e., `TA_FLAGS` has the `MIB_LOCAL` bit set. In the local case, multiple domain level objects will be returned for the same interface, one per machine, with the local values retrieved from each machine represented in the separate objects.

TA_NUMSERVERS: **0** <= *num*

Number of corresponding servers offering this interface.

TA_RTPOLICY: "{always | never}"

Used to mark an interface implementation as idempotent in the `implementation` configuration file (ICF). An idempotent implementation can be repeated without any negative side-effects. For example, `SET BALANCE`.

TA_TPPOLICY: "{method | transaction | process}"

The TP framework deactivation policy. This reflects the policy registered with the framework at server startup. The first server to register the interface sets the value in `T_INTERFACE`. This value cannot be changed.

TA_TXPOLICY: "{optional | always | never | ignore}"

The transaction policy for the interface. The setting in this attribute affects the effect of the `TA_AUTOTRAN` attribute. See `TA_AUTOTRAN` for further explanation. This attribute is always read-only. It is set by the developer when the server is built and registered at server startup.

TA_NCOMPLETED: **0** <= *num*

Number of interface method invocations completed with respect to the corresponding `T_IFQUEUE` objects since they were initially offered. Local queries (`TA_FLAGS` `MIB_LOCAL` bit set) on domain level objects will return one object per machine with the statistics for the indicated interface on that machine.

TA_NQUEUED: **0** <= *num*

Number of requests currently enqueued for this interface. Local queries (`TA_FLAGS` `MIB_LOCAL` bit set) on domain level objects will return one object per machine with the statistics for the indicated interface on that machine.

Implementation Hint

The `T_INTERFACE` MIB is a mapping from an interface to a BEA Tuxedo service. The MIB server can implement some of the get/set operations for an interface by calling the existing logic for the associated `T_SERVICE` object.

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_PMID(r)(*) ¹	string	rU-r--r--	<i>string</i> [1..30]	N/A
TA_TUXCONFIG(r)	string	rw-r--r--	<i>string</i> [2..256] ⁵	N/A
TA_TUXDIR(r)	string	rw-r--r--	<i>string</i> [2..256] ⁶	N/A
TA_APPDIR(r)	string	rw-r--r--	<i>string</i> [2..256] ⁶	N/A
TA_STATE(k)	string	rwyr--r--	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..256] ⁶	" "
TA_PERM	long	rwyr--r--	0001 <= num <= 0777	(²)
TA_ULOGPFX	string	rwyr--r--	<i>string</i> [0..256] ⁶	(³)

TM_MIB(5): T_MACHINE Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
TA_TYPE	string	rw-r--r--	<i>string</i> [0..15]	""
TA_MAXACCESSERS	long	rw-r--r--	1 <= <i>num</i> < 32,768	(²)
TA_MAXCONV	long	rw-r--r--	0 <= <i>num</i> < 32,768	(²)
TA_MAXGTT	long	rw-r--r--	0 <= <i>num</i> < 32,768	(²)

TM_MIB(5): T_MACHINE Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
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)

TM_MIB(5): T_MACHINE Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
TA_MAXWSCLIENTS	long	rw-r--r--	0 <= num < 32,768	0
TA_MAXACLCACHE	long	rw-r--r--	10 <= num <= 32,000	100
TA_TLOGDEVICE	string	rw-r--r--	string[0..256] ⁵	" "
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_BRTHREADS	string	rw-r--r--	"{Y N}"	"N"
TA_NADDR	string	rw-r--r--	string[0..256] ⁶	N/A
TA_NLSADDR	string	rw-r--r--	string[0..256] ⁶	N/A
TA_FADDR	string	rw-r--r--	string[0..256] ⁶	" "
TA_FRANGE	long	rw-r--r--	1 <= num <= 65,535	1
TA_CMPLIMIT	string	rwyr-yr--	"remote[, local]"	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
TA_SICACHEENTRIESMAX	string	rw-r--r--	"0"- "32767"	"500"

TM_MIB(5): T_MACHINE Class Definition Attribute Table (Continued)

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..1023]	""
TA_SEC_PRINCIPAL_PASSVAR	string	rwxr--r--	<i>string</i> [0..31]	""
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

TM_MIB(5): T_MACHINE Class Definition Attribute Table (Continued)

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

¹ TA_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, they must match the same object.

² Default is same as value set for this attribute in the T_DOMAIN class.

³ Default is TA_APPDIR for this machine followed by /U LOG.

⁴ Link-level encryption value of 40 bits is provided for backward compatibility.

⁵ Maximum string length for this attribute is 64 bytes for BEA Tuxedo 8.0 or earlier.

⁶ Maximum string length for this attribute is 78 bytes for BEA Tuxedo 8.0 or earlier.

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..256] (up to 64 bytes for BEA Tuxedo 8.0 or earlier)

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..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

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..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

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>
CLEaning	<p>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, 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.</p>

Limitation: State change to ForceINActive or INActive allowed only for non-master machines. The master site administrative processes are deactivated via the T_DOMAIN class.

TA_UID: 0 <= num

UNIX system user identifier for the BEA Tuxedo system application administrator on this machine. Administrative commands such as `tmbboot(1)`, `tmshutdown(1)`, and `tmadmin(1)` must run as the indicated user on this machine. Application and administrative servers on this machine will be started as this user.

Limitation: This is a UNIX system-specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_GID: 0 <= num

UNIX system group identifier for the BEA Tuxedo system application administrator on this machine. Administrative commands such as `tmbboot(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..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

Environment file for clients and servers running on this machine. If the value specifies an invalid filename, 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, *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 $\leq num \leq$ **0777**

UNIX system permissions associated with the shared memory bulletin board created on this machine. Default UNIX system permissions for system and application message queues.

Limitations: Modifications to this attribute for an active object will not affect running servers or clients.

This is a UNIX system-specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_ULOGPFX: string[0..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

Absolute pathname prefix of the path for the `userlog()` file on this machine. The `userlog()` filename is formed by appending the string `.mmddy` to the `TA_ULOGPFX` attribute value. `mmddy` 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 $\leq num <$ **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 attribute) 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 <= 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 <= 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 <= num < 32,768

Number of accesser 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 accesser slots specified with TA_MAXACCESSERS, meaning that the accesser 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 <= num <= 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 attribute helps to conserve on shared memory resources and yet reduce the number of disk access to do ACL checking.

TA_TLOGDEVICE: *string*[0..256] (up to 64 bytes for BEA Tuxedo 8.0 or earlier)

The device (raw slice) or UNIX system file containing the BEA Tuxedo filesystem that holds the DTP transaction log for this machine. The DTP transaction log is stored as a BEA Tuxedo system VTOC table on the device. This device or file may be the same as that specified for the `TA_TUXCONFIG` attribute for this machine.

TA_TLOGNAME: *string*[0..30]

The name of the DTP transaction log for this machine. If more than one DTP transaction log exists on the same `TA_TLOGDEVICE`, they must have unique names. `TA_TLOGNAME` must be different from the name of any other table on the `TA_TLOGDEVICE` where the DTP transaction log table is created.

TA_TLOGSIZE: 1 <= num < 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 for this logical machine to access the network. This value is required 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_BRTHREADS: "{Y | N}"

The Bridge process for this logical machine is configured for multithreaded execution ("Y") or single-threaded execution ("N"). The default is "N". This attribute applies only to applications running BEA Tuxedo 8.1 or later software.

Setting `TA_BRTHREADS` to "Y" makes sense only if a machine has multiple CPUs. However, having multiple CPUs is not a prerequisite for setting `TA_BRTHREADS` to "Y".

Configurations with `TA_BRTHREADS` set to "Y" on the local machine and `TA_BRTHREADS` set (or defaulted) to "N" on the remote machine are allowed, but the throughput between the machines will not be greater than that for the single-threaded Bridge process.

A Bridge process configured for single-threaded or multithreaded execution can interoperate with a Bridge process running in an earlier release of BEA Tuxedo or WebLogic Enterprise: BEA Tuxedo release 8.0 or earlier, WebLogic Enterprise release

5.1 or earlier. In general, a threaded Bridge can interoperate with an unthreaded Bridge because there are no external functional or behavioral changes due to the threading.

Note: If `BRTHREADS=Y` and the Bridge environment contains `TMNOTHREADS=Y`, the Bridge starts up in threaded mode and logs a warning message to the effect that the Bridge is ignoring the `TMNOTHREADS` setting. The `TMNOTHREADS` environment variable was added to the BEA Tuxedo product in release 8.0.

TA_NADDR: *string*[0..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

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..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

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..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

Specifies the complete network address to which local processes such as `tmbboot`, `tmloadcf`, and Bridge 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 attempts to bind before making an outbound connection. If this attribute is set to the `NULL` or empty string, 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 <= num <= 65,535`

Specifies the range of TCP/IP ports to which local processes 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 <= 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 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_SPINCOUNT: `0 <= num`

Spincount used on this machine for pre-ticket user level semaphore access. Defaults are built into the BEA Tuxedo system binaries on each machine. These defaults may be overridden at run time 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 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_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_SICACHEENTRIESMAX: "0" - "32767"

The number of service and interface cache entries kept on this machine. If not specified, the value will be set to "500". A value of "0" implies that service caching is not used on this machine.

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..1023]

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 1023 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..31]

Variable in which the password for the principal specified in TA_SEC_PRINCIPAL_NAME is stored. This attribute may contain a maximum of 31 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 applies only 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 applies only 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 <= 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 <= num < 32,768

Number of clients, both native and Workstation, currently logged in to this machine.

TA_CURCONV: 0 <= num < 32,768

Number of active conversations with participants on this machine.

TA_CURGTT: 0 <= num < 32,768

Number of in use transaction table entries on this machine.

TA_CURRLOAD: 0 <= num

Current service load queued on this machine. Limitation: If the `T_DOMAIN:TA_LDBAL` attribute is "N" or the `T_DOMAIN:TA_MODEL` attribute is "MP", an FML32 NULL value is returned (0).

TA_CURWSCLIENTS: 0 <= num < 32,768
 Number of Workstation clients currently logged in to this machine.

TA_HWACCESSERS: 0 <= 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 <= num < 32,768
 High water number of clients, both native and Workstation, logged in to this machine.

TA_HWCONV: 0 <= num < 32,768
 High water number of active conversations with participants on this machine.

TA_HWGTT: 0 <= num < 32,768
 High water number of in use transaction table entries on this machine.

TA_HWWSCLIENTS: 0 <= num < 32,768
 High water number of Workstation clients currently logged in to this machine.

TA_NUMCONV: 0 <= num
 Number of `tpconnect()` operations performed from this machine.

TA_NUMDEQUEUE: 0 <= num
 Number of `tpdequeue()` operations performed from this machine.

TA_NUMENQUEUE: 0 <= num
 Number of `tpenqueue()` operations performed from this machine.

TA_NUMPOST: 0 <= num
 Number of `tppost()` operations performed from this machine.

TA_NUMREQ: 0 <= num
 Number of `tpacall()` or `tpcall()` operations performed from this machine.

TA_NUMSUBSCRIBE: 0 <= 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` attribute) applications can have only one `T_MACHINE` object. MP mode (see `T_DOMAIN:TA_MODEL` attribute) applications with the `LAN` option set (see `T_DOMAIN:TA_OPTIONS` attribute) 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 must at least be active in a minimally 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

Table 51 TM_MIB(5): T_MSG Class Definition Attribute Table

Attribute ¹	Type	Permissions	Values	Default
TA_LMID(k)	string	R--R--R--	<i>LMID</i>	N/A
TA_MSGID(k)	long	R--R--R--	1 <= <i>num</i>	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 <= <i>num</i>	N/A
TA_MSG_CBYTES	long	R--R--R--	1 <= <i>num</i>	N/A
TA_MSG_CTIME	long	R--R--R--	1 <= <i>num</i>	N/A
TA_MSG_LRPID	long	R--R--R--	1 <= <i>num</i>	N/A
TA_MSG_LSPID	long	R--R--R--	1 <= <i>num</i>	N/A
TA_MSG_QBYTES	long	R--R--R--	1 <= <i>num</i>	N/A
TA_MSG_QNUM	long	R--R--R--	1 <= <i>num</i>	N/A
TA_MSG_RTIME	long	R--R--R--	1 <= <i>num</i>	N/A
TA_MSG_STIME	long	R--R--R--	1 <= <i>num</i>	N/A

(k)—GET key field

¹All attributes in the `T_MSG` class 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 state indicates 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

Table 52 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 <= 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_LMIDs 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

Table 53 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-----	LMID	N/A
TA_STATE	string	RW-----	GET: "VAL" SET: "{NEW INV}"	N/A N/A
TA_NADDR	string	rw-r--r--	string[1..256] ¹	" "
TA_FADDR	string	rw-r--r--	string[0..256] ¹	" "

Table 53 TM_MIB(5): T_NETMAP Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_FRANGE	long	rw-r--r--	1 <= num <= 65,535	1
TA_MINENCRYPTBITS	string	rw-rwx---	"{0 40 56 128}" ²	"0"
TA_MAXENCRYPTBITS	string	rw-rwx---	"{0 40 56 128}" ²	"128"
(r)—required field for object creation (SET TA_STATE NEW)				
(*)—GET/SET key, one or more required for SET operations				

¹ Maximum string length for this attribute is 78 bytes for BEA Tuxedo 8.0 or earlier.

² Link-level encryption value of 40 bits is provided for backward compatibility.

Attribute Semantics

TA_NETGROUP: *string*[1..30]

This is the name of the associated network group found in the T_NETGROUP class.

TA_LMID: *LMID*

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*[1..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

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 "*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_FADDR: *string*[0..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

Specifies the complete network address to which local processes such as *tmbboot*, *tmloadcf*, and Bridge and 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 attempts to bind before making an outbound connection. If this attribute is set to the NULL or empty string, 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 <= num <= 65,535`

Specifies the range of TCP/IP ports to which local processes 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

Table 54 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..256] ¹	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
TA_NQUEUED	long	R--R--R--	0 <= <i>num</i>	N/A
TA_WKQUEUED	long	R--R--R--	0 <= <i>num</i>	N/A
(k)—GET key field				
(*)—GET/SET key, one or more required for SET operations				

¹ Maximum string length for this attribute is 78 bytes for BEA Tuxedo 8.0 or earlier.

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 <code>T_QUEUE</code> object is active.
MIGrating	The server(s) associated with this <code>T_QUEUE</code> object is currently in the <code>MIGrating</code> state. See the <code>T_SERVER</code> class for more details on this state. This state is <code>ACTIVE</code> equivalent for the purpose of determining permissions.
SUSPended	The server(s) associated with this <code>T_QUEUE</code> object is currently in the <code>SUSPended</code> state. See the <code>T_SERVER</code> class for more details on this state. This state is <code>ACTIVE</code> equivalent for the purpose of determining permissions.
PARTitioned	The server(s) associated with this <code>T_QUEUE</code> object is currently in the <code>PARTitioned</code> state. See the <code>T_SERVER</code> class for more details on this state. This state is <code>ACTIVE</code> 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..256**] (up to **78 bytes for BEA Tuxedo 8.0 or earlier**)

Application specified command to be executed in parallel with the system restart of application servers associated with this queue.

TA_RESTART: "{Y | N}"

Servers associated with this queue are restartable ("Y") or non-restartable ("N").

TA_CONV: "{Y | N}"

Servers associated with this queue are conversational-based ("Y") or request/response-based ("N").

TA_LMID: *LMID*

Logical machine on which servers associated with this queue are active.

TA_RQID: **1** <= num

UNIX system message queue identifier.

Limitation: This is a UNIX system specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_SERVERCNT: **1** <= num < **8,192**

Number of active servers associated with this queue.

TA_TOTNQUEUED: **0** <= num

The sum of the queue lengths of this queue while it has been active. This sum includes requests enqueued to and processed by servers that are no longer active on the queue. Each

time a new request is assigned to the queue, the sum is incremented by the length of the queue immediately before the new request is enqueued.

Limitation: If the T_DOMAIN:TA_LDBAL attribute is "N" or the T_DOMAIN:TA_MODEL attribute is "MP", TA_TOTNQUEUED is not returned. In the same configuration, updates to this attribute are ignored. Consequently, when this attribute is returned TA_LMID and TA_SOURCE have the same value.

TA_TOTWKQUEUED: 0 <= num

The sum of the workloads enqueued to this queue while it has been active. This sum includes requests enqueued to and processed by servers that are no longer active on the queue. Each time a new request is assigned to the queue, the sum is incremented by the workload on the queue immediately before the new request is enqueued.

Limitation: If the T_DOMAIN:TA_LDBAL attribute is "N" or the T_DOMAIN:TA_MODEL attribute is "MP", TA_TOTWKQUEUED is not returned. In the same configuration, updates to this attribute are ignored. Consequently, when this attribute is returned TA_LMID and TA_SOURCE have the same value.

TA_SOURCE: LMID

Logical machine from which local attribute values are retrieved.

TA_NQUEUED: 0 <= num

Number of requests currently enqueued to this queue from the TA_SOURCE logical machine. This value is incremented at enqueue time and decremented when the server dequeues the request.

Limitation: If the T_DOMAIN:TA_LDBAL attribute is "N" or the T_DOMAIN:TA_MODEL attribute is "MP", 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", the TA_WKQUEUED 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

Table 55 TM_MIB(5): T_ROUTING Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_ROUTINGNAME(r)(*)	string	ru-r--r--	string[1..15]	N/A
TA_ROUTINGTYPE(r)	string	ru-r--r--	SERVICE or FACTORY	"SERVICE"
TA_BUFTYPE(r)(*)	string	ru-r--r--	string[1..256]	N/A ¹
TA_FIELD(r)(k)(*)	string	ru-r--r--	string[1..30]	N/A ¹
TA_FIELDTYPE	string	ru-r--r--	[char short long float double string]	"string"
TA_FIELDTYPE(r) (factory-based routing only)	string	rw-r--r--	string[1..30]	N/A
TA_RANGES(r)	carray	rw-r--r--	carray[1..2048]	N/A
TA_TYPE	string	ru-r--r--	string[1..15]	"SERVICE"
TA_STATE(k)	string	rw-r--r--	GET: "VAL" SET: "{NEW INV}"	N/A N/A

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

¹TA_BUFTYPE applies only to ATMI data-dependent routing criteria. TA_FIELDTYPE applies only to CORBA factory-based 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, and TA_ROUTINGNAME, TA_TYPE and TA_FIELD must be unique for TA_TYPE=FACTORY.

The TA_TYPE attribute determines the permissible attributes for the TA_ROUTING object. TYPE=SERVICE corresponds to ATMI data-dependent routing criteria. TYPE=FACTORY corresponds to CORBA factory-based routing. The default is SERVICE. SET operations are assumed to be for data-dependent routing if no TA_TYPE is specified. Specification of TA_FIELDTYPE is invalid for data-dependent routing. Specification of TA_BUFTYPE is invalid for factory-based routing.

Attribute Semantics

TA_ROUTINGNAME: *string*[1..15]

Routing criteria name.

TA_ROUTINGTYPE: *type*

Specifies the routing type. The default is TYPE=SERVICE to ensure that existing UBBCONFIG files used in ATMI environments continue to work properly. Use TYPE=FACTORY if you are implementing factory-based routing for a CORBA interface.

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]

The routing field name. When TA_TYPE=FACTORY, this is assumed to be a field that is specified in an NVList parameter to PortableServer::POA::create_reference_with_criteria for an interface that has this factory routing criteria associated with it. See section on factory-based routing for more details.

When TA_TYPE=SERVICE, 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` attribute 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_FIELDTYPE (factory-based routing only)

Routing field type. This field is only valid if `TA_TYPE=FACTORY`. Valid types are: `SHORT`, `LONG`, `FLOAT`, `DOUBLE`, `CHAR`, `STRING`. Specification of this attribute is only valid for factory-based routing criteria.

```
TA_RANGES: carray[1..2048]
```

The ranges and associated server groups for the routing field. The format of *string* is a comma-separated, ordered list of range/group name pairs. A range/group name pair has the following format:

```
lower[-upper]:group
```

lower and *upper* are signed numeric values or character strings in single quotes. *lower* must be less than or equal to *upper*. To embed a single quote in a character string value, it must be preceded by two backslashes (for example, 'O\\'Brien'). The value `MIN` can be used to indicate the minimum value for the data type of the associated field on the machine. The value `MAX` can be used to indicate the maximum value for the data type of the associated field on the machine. Thus, "`MIN--5`" is all numbers less than or equal to -5, and "`6-MAX`" is all numbers greater than or equal to 6.

The meta-character “*” (wildcard) in the position of a range indicates any values not covered by the other ranges previously seen in the entry; only one wildcard 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.

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 state indicates the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

VALid	T_ROUTING object is defined. Note that this is the only valid state for this class. Routing criteria are never ACTIVE; rather, they are associated through the configuration with service names and are acted upon at run-time 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.

TA_TYPE

Routing criteria type. Valid values are “FACTORY” or “SERVICE”. “FACTORY” specifies that the routing criteria applies to factory-based routing for a CORBA interface. The specification of TYPE=FACTORY is mandatory for a factory-based routing criteria. “SERVICE” specifies that the routing criteria applies to data-dependent routing for an ATMI service. Default is “SERVICE”. Specification of this attribute is optional for data-dependent routing criteria. Note that the type specified affects the validity and possible values for other fields defined for this MIB class. These are noted for each field. TA_TYPE is required for SET operations for factory-based routing criteria.

Limitations

None.

T_SERVER Class 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_SERVERCXTXT class. The T_SERVERCXTXT 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

Table 56 TM_MIB(5): T_SERVER Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_SRVGRP(r)(*)	string	ru-r--r--	<i>string</i> [1..30]	N/A
TA_SRVID(r)(*)	long	ru-r--r--	1 <= num < 30,001	N/A
TA_SERVERNAME(k)(r)	string	rw-r--r--	<i>string</i> [1..78]	N/A
TA_GRPNO(k)	long	r--r--r--	1 <= num < 30,000	N/A
TA_STATE(k)	string	rwxr--r--	GET: "{ACT INA MIG CLE RES SUS PAR DEA}" SET: "{NEW INV ACT INA DEA}"	N/A N/A
TA_BASESRVID	long	r--r--r--	1 <= num < 30,001	N/A
TA_CLOPT	string	rwyr--r--	<i>string</i> [0..256]	"-A"
TA_ENVFILE	string	rwyr--r--	<i>string</i> [0..256] ²	" "
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 <= 2147483647	0
TA_CURDISPATCHTHREADS	long	R-XR-XR--	0 <= num	N/A

Table 56 TM_MIB(5): T_SERVER Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
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..256] ²	" "
TA_RESTART	string	rwyr--r--	"{Y N}"	N
TA_SEQUENCE(k)	long	rwxr--r--	1 <= num < 10,000	>= 10,000
TA_SYSTEM_ACCESS	string	rwyr--r--	"{FASTPATH PROTECTED}"	(¹)
TA_CONV(k)	string	rw-r--r--	"{Y N}"	"N"
TA_REPLYQ	string	rw-r--r--	"{Y N}"	"N"
TA_RPPERM	long	rw-r--r--	0001 <= num <= 0777	(¹)
TA_RQADDR(k)	string	rw-r--r--	string[0..30]	"GRPNO. SRVID"
TA_RQPERM	long	rw-r--r--	0001 <= num <= 0777	(¹)
TA_LMID(k)	string	R--R--R--	LMID	N/A
TA_GENERATION	long	R--R--R--	1 <= num < 32,768	N/A

Table 56 TM_MIB(5): T_SERVER Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
TA_PID(k)	long	R--R--R--	1 <= num	N/A
TA_RPID	long	R--R--R--	1 <= num	N/A
TA_RQID	long	R--R--R--	1 <= num	N/A
TA_TIMERESTART	long	R--R--R--	1 <= num	N/A
TA_TIMESTART	long	R--R--R--	1 <= num	N/A
TA_SEC_PRINCIPAL_NAME	string	rwxr--r--	string[0..511]	" "
TA_SEC_PRINCIPAL_LOCATION	string	rwxr--r--	string[0..1023]	" "
TA_SEC_PRINCIPAL_PASSVAR	string	rwxr--r--	string[0..31]	" "
TA_SICACHEENTRIESMAX	string	rw-r--r--	{ "0"-"32767" "DEFAULT" }	"DEFAULT"
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_CLTMID	string	R--R--R--	LMID	N/A

Table 56 TM_MIB(5): T_SERVER Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
TA_CLTPID	long	R--R--R--	1 <= num	N/A
TA_CLTREPLY	string	R--R--R--	"{Y N}"	N/A
TA_CMTRET	string	R--R--R--	"{COMPLETE LOGGED}"	N/A
TA_CURCONV	long	R--R--R--	0 <= num	N/A
TA_CUROBJECTS	long	R--R--R--	0 <= num	N/A
TA_CURINTERFACE	string	R--R--R--	string[0..128]	N/A
TA_CURREQ	long	R--R--R--	0 <= num	N/A
TA_CURRSERVICE	string	R--R--R--	string[0..15]	N/A
TA_CURTIME	long	R--R--R--	1 <= num	N/A
TA_LASTGRP	long	R--R--R--	1 <= num < 30,000	N/A
TA_SVCTIMEOUT	long	R--R--R--	0 <= num	N/A
TA_TIMELEFT	long	R--R--R--	0 <= num	N/A
TA_TRANLEV	long	R--R--R--	0 <= num	N/A

(k)—GET key field

(r)—required field for object creation (SET TA_STATE NEW)

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

¹ Defaults to value set for this attribute in the T_DOMAIN class.

² Maximum string length for this attribute is 78 bytes for BEA Tuxedo 8.0 or earlier.

Attribute Semantics

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

Logical name of the server group. Server group names cannot contain an asterisk (*), comma, or colon.

TA_SRVID: **1 <= num < 30,001**

Unique (within the server group) server identification number.

TA_SERVERNAME: *string*[1..78]

Name of the server executable file. The server identified by `TA_SERVERNAME` will run on the machine(s) identified by the `T_GROUP:TA_LMID` attribute for this server's server group. If a relative pathname is given, 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 the 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 | EXItting | 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	<code>T_SERVER</code> 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 <code>TA_CURRSERVICE</code> attribute should be interpreted as a busy server, that is, one that is processing a service request.
INActive	<code>T_SERVER</code> object defined and inactive.
MIGrating	<code>T_SERVER</code> 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 <code>T_GROUP:TA_LMID</code> attribute that does not match the <code>T_GROUP:TA_CURLMID</code> attribute. This state is ACTIVE equivalent for the purpose of determining permissions.
CLEaning	<code>T_SERVER</code> 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 <code>TA_MAXGEN</code> starts/restarts within their <code>TA_GRACE</code> 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.
EXItIng	T_SERVER object defined and currently pending on exit(). This state is ACTIVE equivalent for the purpose of determining permissions.
DEAD	T_SERVER object defined, identified as active in the bulletin board, but currently not running due to an abnormal death. This state will exist only until the BBL local to the server notices the death and takes action (REStarting CLEaning). Note that this state will only be returned if the MIB_LOCAL TA_FLAGS value is specified and the machine where the server was running is reachable. This state is ACTIVE equivalent for the purpose of determining permissions.

SET: "{NEW | INValid | ACTIVE | INActive | DEAD}"

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

NEW	Create T_SERVER object for application. State change allowed only when in the INValid state. Successful return leaves the object in the INActive state.
unset	Modify an existing T_SERVER object. This combination is allowed only when in the ACTIVE or INActive state. Successful return leaves the object state unchanged.

INValid	Delete T_SERVER object for application. State change allowed only when in the INActive state. Successful return leaves the object in the INValid state.
ACTive	Activate the T_SERVER object. State change allowed only when in the INActive state. (Servers in the MIGRating state must be restarted by setting the T_GROUP:TA_STATE to ACTive.) For the purpose of determining permissions for this state transition, the active object permissions are considered (that is, --x--x--x). Successful return leaves the object in the ACTive state. The TMIB_NOTIFY TA_FLAG value should be used when activating a server if status on the individual server is required.
INActive	Deactivate the T_SERVER object. State change allowed only when in the ACTive state. Successful return leaves the object in the INActive state. The TMIB_NOTIFY TA_FLAG value should be used when deactivating a server if status on the individual server is required.
DEAd	Deactivate the T_SERVER object by sending the server a SIGTERM signal followed by a SIGKILL signal if the server is still running after the appropriate timeout interval (see TA_MIBTIMEOUT in MIB (5)). Note that by default, a SIGTERM signal will cause the server to initiate orderly shutdown and the server will become inactive even if it is restartable. If a server is processing a long running service or has chosen to disable the SIGTERM signal, 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 REStArting 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 reference page [servopts\(5\)](#) for details. Limitation: Run-time modifications to this attribute will not affect a running server.

TA_ENVFILE: *string*[0..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

Server specific environment file. See `T_MACHINE:TA_ENVFILE` for a complete discussion of how this file is used to modify the environment. Limitation: Run-time modifications to this attribute will not affect a running server.

TA_GRACE: **0** <= *num*

The period of time, in seconds, over which the `T_SERVER:TA_MAXGEN` limit applies. This attribute is meaningful only for restartable servers, that is, if the `T_SERVER:TA_RESTART` attribute is set to "Y". When a restarting server would exceed the `TA_MAXGEN` limit but the `TA_GRACE` period has expired, the system resets the current generation (`T_SERVER:TA_GENERATION`) to 1 and resets the initial boot time (`T_SERVER:TA_TIMESTART`) to the current time. A value of 0 for this attribute indicates that a server should always be restarted.

Note that servers sharing a request queue (that is, equal values for `T_SERVER:TA_RQADDR`) should have equal values for this attribute. If they do not, the first server activated will establish the run-time value associated with all servers on the queue.

Limitation: Run-time modifications to this attribute will affect a running server and all other active servers with which it is sharing a request queue. However, only the selected server's configuration parameter is modified. Thus, the behavior of the application depends on the order of boot in subsequent activations unless the administrator ensures that all servers sharing a queue have the same value for this attribute.

TA_MAXGEN: **1** <= *num* < **256**

Number of generations allowed for a restartable server (`T_SERVER:TA_RESTART == "Y"`) over the specified grace period (`T_SERVER:TA_GRACE`). The initial activation of the server counts as one generation and each restart also counts as one. Processing after the maximum generations is exceeded is discussed above with respect to `TA_GRACE`.

Note that servers sharing a request queue (that is, equal values for `T_SERVER:TA_RQADDR`) should have equal values for this attribute. If they do not, the first server activated will establish the run-time value associated with all servers on the queue.

Limitation: Run-time modifications to this attribute will affect a running server and all other active servers with which it is sharing a request queue. However, only the selected server's configuration parameter is modified. Thus, the behavior of the application depends on the order of boot in subsequent activations unless the administrator ensures that all servers sharing a queue have the same value for this attribute.

TA_MAX: **1** <= *num* < **1,001**

Maximum number of occurrences of the server to be booted. Initially, `tmboot()` boots `T_SERVER:TA_MIN` objects of the server, and additional objects may be started

individually (by starting a particular server ID) or through automatic spawning (conversational servers only). Run-time modifications to this attribute will affect all running servers in the set of identically configured servers (see `TA_BASESRVID` above) as well as the configuration definition of the server.

TA_MIN: 1 <= num < 1,001

Minimum number of occurrences of the server to be booted by. If a `T_SERVER:TA_RQADDR` is specified and `TA_MIN` is greater than 1, 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 <= num < 1,000

Specifies the number of server dispatch threads started on initial server boot. This attribute 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 <= num < 1,000

Specifies the maximum number of concurrently dispatched threads which each server process may spawn. This attribute is effective only if the server has been built with the `buildserver -t` command.

If `TA_MAXDISPATCHTHREADS > 1`, 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: 0 <= num <= 2147483647

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 if the value specified is 0, a default thread stack size is used. If the value specified is bigger than 0 and less than a minimum thread stack size, the

minimum thread stack size is used. If the value specified is bigger than that minimum thread stack size, value specified 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. The purpose of the minimum thread stack size is to allow customer to specify less thread stack size than Tuxedo default thread stack size. Currently, the BEA Tuxedo default thread stack size is 1,024,000, and minimum thread stack size is 100,000.

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..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

Application specified command to be executed in parallel with the system restart of an application server.

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, 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.

Note: If you choose to do redirection or piping on a Windows 2003 system, you must use one of the following methods:

- Do redirection or piping from within a command file or script.

- To do redirection from within the queue manager administration program, precede the command with `cmd`. For example:
`cmd /c ipconfig > out.txt`
- If you choose to create a binary executable, you must allocate a console within the binary executable using the Windows `AllocConsole()` API function

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` attribute and `T_GROUP:TA_LMID` attribute with alternate site), `TA_RESTART` 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, 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 (`tmbboot(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 `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_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 the `T_DOMAIN:TA_SYSTEM_ACCESS` attribute 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".

Note: If you choose to do redirection or piping on a Windows 2003 system, you must use one of the methods listed in the description of the TA_RCMD attribute.

TA_RPPERM: **0001** <= num <= **0777**

UNIX system permissions for the server's reply queue. If a separate reply queue is not allocated (T_SERVER:TA_REPLYQ == "N"), TA_RPPERM is ignored.

Note: If you choose to do redirection or piping on a Windows 2003 system, you must use one of the methods listed in the description of the TA_RCMD attribute.

TA_RQADDR: *string*[**0..30**]

Symbolic address of the request queue for the server. Specifying the same TA_RQADDR attribute value for more than one server is the way Multiple Server, Single Queue (MSSQ) sets are defined. Servers with the same TA_RQADDR attribute value must be in the same server group.

TA_RQPERM: **0001** <= num <= **0777**

UNIX system permissions for the server's request queue.

Limitation: This is a UNIX system specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_LMID: *LMID*

Current logical machine on which the server is running.

TA_GENERATION: **1** <= num < **32,768**

Generation of the server. When a server is initially booted via `tmbboot(1)` or activated through the `TM_MIB(5)`, its generation is set to 1. Each time the server dies abnormally and is restarted, its generation is incremented. Note that when `T_SERVER:TA_MAXGEN` is exceeded and `T_SERVER:TA_GRACE` has expired, the server will be restarted with the generation reset to 1.

TA_PID: **1** <= num

UNIX system process identifier for the server. Note that this may not be a unique attribute since servers may be located on different machines allowing for duplication of process identifiers.

Limitation: This is a UNIX system specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_RPID: 1 <= *num*

UNIX system message queue identifier for the server's reply queue. If a separate reply queue is not allocated (`T_SERVER:TA_REPLYQ == "N"`), the `TA_RPID` 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"`) the `TA_RQID` 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_SICACHEENTRIESMAX: { "0" - "32767" | "DEFAULT" }

The number of service and interface cache entries kept on this machine. A value of "0" implies that service caching is not used on this machine. If the value is "DEFAULT", the value for this server will come from the corresponding `T_MACHINE` class entry.

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..1023]

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 1023 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..31]

Variable in which the password for the principal specified in `TA_SEC_PRINCIPAL_NAME` is stored. This attribute may contain a maximum of 31 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 <= *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_CLTLMID: L MID**
 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_CUROBJECTS: **0** <= *num*

The number of entries in use in the bulletin board object table for this server. Scope is local.

TA_CURINTERFACE: *string*[**0..128**]

The interface name of the interface currently active in this server. Scope is local.

TA_CURREQ: 0 <= *num*

Number of requests initiated by this server via `tpcall()` 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 time out.

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 run-time 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 run-time tracking of statistics and resources associated with each server dispatch context.

Attribute Table

Table 57 TM_MIB(5): T_SERVERCTXT Class Definition Attribute Table

Attribute ¹	Type	Permissions	Values	Default
TA_SRVGRP(k)	string	r--r--r--	string[1..30]	N/A
TA_SRVID(k)	long	r--r--r--	1 <= num < 30,001	N/A
TA_CONTEXTID(k)	long	r--r--r--	-2 <= num < 30,000	N/A
TA_CLTLMID	string	r--r--r--	LMID	N/A

Table 57 TM_MIB(5): T_SERVERCTXT Class Definition Attribute Table

Attribute ¹	Type	Permissions	Values	Default
TA_CLTPID	long	r--r--r--	1 <= num	N/A
TA_CLTREPLY	string	r--r--r--	"{Y N}"	N/A
TA_CMTRET	string	R--R--R--	"{COMPLETE LOGGED}"	N/A
TA_CURCONV	long	r--r--r--	0 <= num	N/A
TA_CURREQ	long	r--r--r--	0 <= num	N/A
TA_CURRSERVICE	string	r--r--r--	string[0..15]	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				

¹All attributes in the T_SERVERCTXT class are local attributes.

Attribute Semantics

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

Logical name of the server group. Server group names cannot contain an asterisk (*), comma, or colon.

TA_SRVID: **1 <= num < 30,001**

Unique (within the server group) server identification number.

TA_CONTEXTID: **0 <= num < 30000**

Identifier of this particular server context.

TA_CLTMLID: *LMID*

Logical machine for the initiating client or server. The initiating client or server is the process that made the service request that the server is currently working on.

TA_CLTPID: **1 <= num**

UNIX system process identifier for the initiating client or server.

Limitation: This is a UNIX system-specific attribute that may not be returned if the platform on which the application is being run is not UNIX-based.

TA_CLTREPLY: "{Y | N}"

The initiating client or server is expecting a reply ("Y") or is not expecting a reply ("N").

TA_CMTRET: "{COMPLETE | LOGGED}"

Setting of the TP_COMMIT_CONTROL characteristic for this server. See the description of the BEA Tuxedo ATMI function `tpsamt(3c)` for details on this characteristic.

TA_CURCONV: 0 <= num

Number of conversations initiated by this server via `tpconnect()` that are still active.

TA_CURREQ: 0 <= num

Number of requests initiated by this server via `tpcall()` or `tpacall()` that are still active.

TA_CURRSERVICE: string[0..15]

Service name that the server is currently working on, if any.

TA_LASTGRP: 1 <= num < 30,000

Server group number (T_GROUP:TA_GRPNO) of the last service request made or conversation initiated from this server outward.

TA_SVCTIMEOUT: 0 <= num

Time left, in seconds, for this server to process the current service request, if any. A value of 0 for an active service indicates that no timeout processing is being done. See T_SERVICE:TA_SVCTIMEOUT for more information.

TA_TIMELEFT: 0 <= num

Time left, in seconds, for this server to receive the reply for which it is currently waiting before it will timeout. This timeout may be a transactional timeout or a blocking timeout.

TA_TRANLEV: 0 <= num

Current transaction level for this server. 0 indicates that the server is not currently involved in a transaction.

Limitations

None.

T_SERVICE Class 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), 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), 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, the first one found is used to initially configure the advertised service.
4. If none of the preceding cases is used, 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 a description of how attribute values are identified at service activation time). Service names to be offered by a server are built in at run time (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

Table 58 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_BLOCKTIME	long	rwyr--r--	0 <= num < 32,768	0
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"
TA_BUFTYPECONV	string	rwyr--r--	XML2FML, XML2FML32, NOCONVERT	NOCONVE RT

(k)—GET key field

(r)—required field for object creation (SET TA_STATE NEW)

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

Attribute Semantics

TA_SERVICENAME: *string*[1..15]
Service name.

TA_STATE:

GET: "{ACTIVE | INACTIVE}"

A GET operation will retrieve configuration information for the selected T_SERVICE object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request.

ACTIVE	T_SERVICE object is defined and at least one T_SVCGRP object with a matching TA_SERVICENAME value is active.
INACTIVE	T_SERVICE object is defined and no T_SVCGRP object with a matching TA_SERVICENAME value is active.

SET: "{NEW | INVALID}"

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

NEW	Create T_SERVICE object for application. State change allowed only when in the INVALID state. Successful return leaves the object in the INACTIVE state. Limitation: Unconfigured services may still be active by virtue of a server advertising them. In this case, the creation of a new T_SERVICE object is not allowed.
unset	Modify an existing T_SERVICE object. This combination is not allowed in the INVALID state. Successful return leaves the object state unchanged.
INVALID	Delete T_SERVICE object for application. State change allowed only when in the INACTIVE state. Successful return leaves the object in the INVALID state.

TA_AUTOTRAN: "{Y | N}"

Automatically begin a transaction ("Y") when a service request message is received for this service if the request is not already in transaction mode. Limitation: Run-time updates to this attribute are not reflected in active T_SVCGRP objects.

TA_LOAD: 1 <= num < 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` attribute value is set to "Y".

Limitation: Run-time updates to this attribute are not reflected in active `T_SVCGRP` objects.

TA_PRIO: 1 $\leq num < 101$

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_BLOCKTIME: 0 $\leq num < 32,768$

Blocktime limit, in seconds, indicating the minimum amount of time a blocking API call will delay before timing out for a particular service. This attribute lets the client know that (after a specified time in seconds), no reply has been received by the server while the service request is still processing.

If not specified, the default is 0 which indicates that the system-wide `BLOCKTIME` value specified in the `UBBCONFIG RESOURCES` section is used for the service.

Limitations: Run-time updates to this attribute are not reflected in active `T_SVCGRP` objects.

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.

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 $\leq num$

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 [tuxtypes\(5\)](#)). For a buffer type that has subtypes, "*" can be specified for the subtype to indicate that the service accepts all subtypes for the associated buffer type.

A single service can only interpret a fixed number of buffer types, namely those found in its buffer type switch (see [tuxtypes\(5\)](#)). If the TA_BUFTYPE attribute value is set to ALL, that service accepts 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 applies only 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 applies only 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_BUFTYPECONV`: *string*[`XML2FML32`, `XML2FML`, `NOCONVERT`]

Converts an input XML buffer to an FML/FML32 buffer before being delivered to the service and converts the output FML/FML32 buffer to an XML buffer before being returned.

The `XML2FML32` value initiates XML to FML32 conversion. The `XML2FML` value initiates XML to FML conversion. The `NOCONVERT` value indicates no conversion takes place.

Limitation: Run-time updates to this attribute are not reflected in active `T_SVCGRP` objects.

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), 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), 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, the first one found is used to initially configure the advertised service.
4. If none of the preceding cases is used, 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 a description of how attribute values are identified at service activation time). Service names to be offered by a server are built in at run time (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 run time if there are multiple servers within the group offering the service.

Attribute Table

Table 59 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_BLOCKTIME	long	rwyr--r--	0 <= num < 32,768	0
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]	(²)

Table 59 TM_MIB(5): T_SVCGRP Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_BUFTYPE	string	r--r--r--	string[1..256]	N/A
TA_ROUTINGNAME	string	r--r--r--	string[0..15]	N/A
TA_SVCTYPE(k)	string	r--r--r--	"{APP CALLABLE SYSTEM}"	"APP"
T_SVCGRP Class: LOCAL Attributes				
TA_NCOMPLETED	long	R-XR-XR--	0 <= num	N/A
TA_NQUEUED	long	R--R--R--	0 <= num < 32,768	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				

¹SET operations on this class must specify sufficient key fields to uniquely identify the object being addressed. If the object is active, 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	<p>Create T_SVCGRP object for application. State change allowed only when in the INValid state. Successful return leaves the object in the INActive state.</p> <p>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.</p>
unset	<p>Modify an existing T_SVCGRP object. This combination is not allowed in the INValid state. Successful return leaves the object state unchanged.</p>
INValid	<p>Delete T_SVCGRP object for application. State change allowed only when in the INActive state. Successful return leaves the object in the INValid state.</p>
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_BLOCKTIME: 0 <= *num* < 32,768

Blocktime limit (in seconds) indicating the minimum amount of time a blocking ATMI call will block before timing out for this service name. The per service blocktime is applicable only when receiving a reply to the service.

If not specified, the default is 0 which indicates that the system-wide `BLOCKTIME` value specified in the `UBBCONFIG RESOURCES` section is used for the service.

TA_SVCTIMEOUT: 0 <= *num*

Time limit (in seconds) for processing requests for this service name. Servers processing service requests for this service will be abortively terminated (kill -9) if they exceed the specified time limit in processing the request. A value of 0 for this attribute indicates that the service should not be abortively terminated.

Limitation: This attribute value is not enforced on BEA Tuxedo release 4.2.2 sites or earlier.

TA_TRANTIME: 0 <= *num*

Transaction timeout value in seconds for transactions automatically started for this `T_SVCGRP` object. Transactions are started automatically when a request not in transaction mode is received and the `T_SVCGRP:TA_AUTOTRAN` attribute value for the service is "Y".

TA_LMID: *LMID*

Current logical machine on which an active server offering this service is running.

TA_RQADDR: *string*[1..30]

Symbolic address of the request queue for an active server offering this service. See `T_SERVER:TA_RQADDR` for more information on this attribute.

TA_SRVID: 1 <= num < **30,001**

Unique (within the server group) server identification number for an active server offering this service. See T_SERVER:TA_SRVID for more information on this attribute.

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

Function name within the associated server assigned to process requests for this service. On a SET request, the server must be able to map the function name to a function using its symbol table to successfully advertise the service. In some situations (for example, direct calls to tpadvertise() 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 value 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_LDBAL attribute value 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 value 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

Table 60 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

Table 61 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

¹ All attributes in the T_TLOG class are local attributes

² One or more TA_GRPNO and TA_TLOGDATA attribute values may be returned with each object of the T_TLOG class. The attribute values for each of these attributes belonging to the particular object are the TA_TLOGCOUNT number of occurrences beginning with the TA_TLOGINDEX.

Attribute Semantics

TA_LMID: *LMID*

Transaction log logical machine identifier.

TA_STATE:

GET: "{ACTIVE | INActive | WARMstart}"

A GET operation will retrieve log configuration and run-time information for the selected T_TLOG object(s). The following states indicate the meaning of a TA_STATE returned in response to a GET request.

ACTIVE	The transaction log exists and is actively logging commit records for transactions coordinated on the site. This corresponds to the associated T_MACHINE object being active.
INActive	The transaction log exists but is currently inactive. This state corresponds to the associated T_MACHINE object being inactive and can only be returned if the site has a <code>tlisten(1)</code> process running; otherwise, the site is unreachable and a object will not be returned.
WARMstart	The transaction log exists, is currently active, and is marked for warmstart processing. Warmstart processing will occur when the next server group is started on the site. This state is ACTIVE equivalent for the purposes of determining permissions.

SET: "{WARMstart}"

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

<i>unset</i>	Modify T_TLOG object. Allowed only when in the ACTIVE state. Successful return leaves the object state unchanged. The only object modifications permitted on this class are additions to the transaction log. In this case, TA_TLOGINDEX and TA_TLOGCOUNT indicate the objects of TA_TLOGDATA to be added.
<i>WARMstart</i>	Initiate warmstart for the T_TLOG object. State change allowed only when in the ACTIVE state. Successful return leaves the object in the WARMstart state.

TA_TLOGCOUNT: 1 <= num

Number of transaction log data records (TA_TLOGDATA) counted, retrieved, or to be added. This attribute is ignored for SET operations with a state change indicated. For valid SET operations with no state change, this attribute indicates the number of log records to be added to an active transaction log. A GET operation with neither TA_GRPNO nor TA_TLOGDATA specified returns a count of in-use log records. A GET operation with only TA_GRPNO set will return a count of in use log records with a coordinator group matching the indicated group. A GET operation with only TA_TLOGDATA set ("") will return a count of in use log records and populate arrays of TA_TLOGDATA and TA_GRPNO attribute values corresponding to the in use log records. A GET operation with both TA_GRPNO and TA_TLOGDATA set ("") will return a count of in use log records with a coordinator group matching the indicated group and populate arrays of TA_TLOGDATA and TA_GRPNO attribute values corresponding to the in use log records.

TA_TLOGINDEX: 0 <= num

Index of the first object specific attribute values (TA_GRPNO and TA_TLOGDATA) corresponding to this object.

TA_GRPNO: 1 <= num < **30,000**

Transaction coordinator's group number.

TA_TLOGDATA: string[1..256]

Formatted transaction log entry. This attribute value should not be interpreted directly. Rather, it should be used solely as a means of migrating log records as part of server group migration.

Limitations

None

T_TRANSACTION Class Definition

Overview

The T_TRANSACTION class represents run-time attributes of active transactions within the application.

Attribute Table

Table 62 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_GRPINDEX	long	R--R--R--	1 <= num	N/A
TA_GRPNO	long	R--R--R--	0 <= num	N/A
TA_GRPNO	long	R--R--R--	1 <= num < 30,000	(²)
TA_GSTATE	long	R-XR-XR--	GET: "PREP PABT PCOM" SET: "{HCO HAB}"	N/A N/A

(k)—GET key field

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

¹ All attributes in the T_TRANSACTION class are local attributes.

² One or more TA_GRPNO and TA_GSTATE attribute values may be returned with each object of the T_TRANSACTION class. The attribute values for each of these attributes belonging to the particular object are the TA_GRPINDEX 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.

PREPrepare	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> .
PostABOrT	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.
PostCOMmit	Not yet implemented.

SET: "{ABOrteD}"

A SET operation will update run-time information for the selected `T_TRANSACTION` object. The following states indicate the meaning of a `TA_STATE` set in a SET request. States not listed may not be set.

<i>unset</i>	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 (see <code>TA_GSTATE</code> below). Successful return leaves the object state unchanged.
ABOrteD	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_GRPCount: 1 <= num

Number of groups identified as participants in the transaction by the information returned from the retrieval site.

TA_GRPINDEX: 1 <= num

Index of the first group specific attribute values (TA_GRPNO and TA_GSTATE) corresponding to this object.

TA_GRPNO: 1 <= num < 30,000

Group number of the participating group.

TA_GSTATE:

GET: "{ACTive | ABorted | ReaDOnly | REAdy | HCOmmit | HABort | DONE}"

A GET operation will retrieve run-time information for the selected T_TRANSACTION object(s) pertaining to the indicated group. The following states indicate the meaning of a TA_GSTATE returned in response to a GET request. States not listed will not be returned. Note that distinct objects pertaining to the same global transaction (equivalent transaction identifiers) may indicate differing states for individual groups. In general, the state indicated on the group's site indicates the true state of the group's participation in the transaction. The exception is when the coordinator site determines that the transaction should abort and sets each participant group state to ABorted. This transition will be propagated to the group's site and result in the rollback of the group's work in the transaction but may not be reflected immediately.

ACTive	The transaction is active in the indicated group.
ABorted	The transaction has been identified for rollback and rollback has been initiated for the indicated group.
ReaDOnly	The group has successfully completed the first phase of two-phase commit and has performed only read operations on the resource manager, thus making it unnecessary to perform the second phase of commit for this group.
REAdy	The group has successfully completed the first phase of two-phase commit and is ready to be committed.
HCOmmit	The group has been heuristically committed. This may or may not agree with the final resolution of the transaction.

HABort	The group has been heuristically rolled back. This may or may not agree with the final resolution of the transaction.
DONe	This group has completed the second phase of the two-phase commit.

SET: "{HCOmmit | HABort}"

A SET operation will update run-time information for the first group in the originating request within the selected T_TRANSACTION object. The following states indicate the meaning of a TA_GSTATE set in a SET request. States not listed may not be set. State transitions are allowed only when performed within the object representing the group's site (TA_LMID).

HCOmmit	Heuristically commit the group's work as part of the indicated transaction. State change allowed only when TA_GSTATE is REAdy, TA_STATE is REAdy, and the indicated group is not on the coordinator's site. Successful return leaves the object in the HCOmmit state.
HABort	Heuristically rollback the group's work as part of the indicated transaction. State change allowed only when TA_GSTATE is ACTive or REAdy, TA_STATE is REAdy, and the indicated group is not on the coordinator's site. Successful return leaves the object in the HABort state.

Limitations

None.

T_ULONG Class Definition

Overview

The T_ULONG class represents run-time attributes of `userlog()` files within an application.

Attribute Table

Table 63 TM_MIB(5): T_ULOG 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>mmddyy</i>	Current date
TA_STATE	string	R--R--R--	GET: "ACT" SET: N/A	N/A N/A
TA_ULOGTIME(k)	long	R--R--R--	<i>hhmmss</i>	000000
TA_ENDTIME(k)	long	K--K--K--	<i>hhmmss</i>	235959
TA_ULOGLINE(k)	long	R--R--R--	1 <= <i>num</i>	1
TA_ULOGMSG(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_ULOGCAT(x)	string	R--R--R--	<i>string</i> [1..30]	N/A
TA_ULOGMSGNUM(k)	long	R--R--R--	1 <= <i>num</i>	N/A
TA_ULOGPROCNM(x)	string	R--R--R--	<i>string</i> [1..30]	N/A

(k)—GET key field
(x)—regular expression GET key field

¹ All attributes in the T_ULOG class 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 filesystem, multiple TA_LMID values may be returned even though a specific value has been provided as a *key* field. For the same reasons, TA_PMIID 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_PMIID as a regular expression key field.

Attribute Semantics

TA_LMID: *LMID*

Retrieval machine logical machine identifier.

TA_PMIID: *string*[1..30]

Physical machine identifier.

TA_MMDDYY: *mmddyy*

Date of user log 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 user log file on the indicated logical machine.
--------	--

SET:

SET operations are not permitted on this class.

TA_ULOGTIME: *hhmmss*

The time of the user log 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_ULOGLINE: **1** <= num

The line number of the user log message returned/requested within the user log file. When used as a key field for retrieval, this value indicates the starting line within the log file.

TA_ULOGMSG: *string*[**1..256**]

The entire text of the user log message as it appears in the user log 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** <= num

Process identifier of the client or server that generated the user log message.

TA_THREADID: **0** <= num

Identifier for the thread that wrote this user log message.

TA_CONTEXTID: **-2** <= 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 user log 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.

TM_MIB(5) Additional Information

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()`, `tpgetreply()`, 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, failure may be returned in the form of an application level service failure. In these cases, `tpcall()` and `tpcall()` will return an error with `tpgetreply()` 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 6.1 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 6.1, are interoperating, 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 pathname 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 `tpadmcall()` for processing. This example also shows interpretation of `tpadmcall()` 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 tpadmcall() */
if (tpadmcall(ibuf, obuf, 0) 0) {
fprintf(stderr, "tpadmcall failed: %s\n", tpsterror(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);

tpadmcall(...); /* See earlier example for detailed error processing */

/* Set TM_MIB(5) attributes defining second group */
Fchg32(ibuf, TA_SRVGRP, 0, "GRP2", 0);
Fchg32(ibuf, TA_GRPNO, 0, (char *)&grpno[1], 0);
Fchg32(ibuf, TA_LMID, 0, "LMID2,LMID1", 0);

tpadmcall(...); /* See earlier example for detailed error processing */
```

Add One Server Per Group

Reuse the allocated buffers to add one server per group to the configured application.

```
/* Clear the request buffer */ Finit32(ibuf, Fsizeof32(ibuf));

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

```

/* Set TM_MIB(5) attributes defining first server */
Fchg32(ibuf, TA_SRVGRP, 0, "GRP1", 0);
Fchg32(ibuf, TA_SRVID, 0, (char *)&srvid[0], 0);
Fchg32(ibuf, TA_SERVERNAME, 0, "ECHO", 0)

tpadmcall(...); /* See earlier example for detailed error processing */

/* Set TM_MIB(5) attributes defining second server */
Fchg32(ibuf, TA_SRVGRP, 0, "GRP2", 0);
Fchg32(ibuf, TA_SRVID, 0, (char *)&srvid[1], 0);

tpadmcall(...); /* See earlier example for detailed error processing */

```

Add Routing Criteria

Add a routing criteria definition. Note that routing criteria may be dynamically added to a running application using a similar operation via the `tpcall()` interface.

```

/* Clear the request buffer */ Finit32(ibuf, Fsizeof32(ibuf));

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

/* Set TM_MIB(5) attributes defining routing criteria */
Fchg32(ibuf, TA_ROUTINGNAME, 0, "ECHOROUTE", 0);
Fchg32(ibuf, TA_BUFTYPE, 0, "FML", 0);
Fchg32(ibuf, TA_FIELD, 0, "LONG_DATA", 0);
Fchg32(ibuf, TA_RANGES, 0, "MIN-100:GRP1,100-MAX:GRP2", 26);

tpadmcall(...); /* See earlier example for detailed error processing */

```

Add Service Definition

Define a service object that maps the advertised service name to the routing criteria defined above.

```

/* Clear the request buffer */ Finit32(ibuf, Fsizeof32(ibuf));

/* Set MIB(5) attributes defining request type */

```

```

Fchg32(ibuf, TA_OPERATION, 0, "SET", 0);
Fchg32(ibuf, TA_CLASS, 0, "T_SERVICE", 0);
Fchg32(ibuf, TA_STATE, 0, "NEW", 0);

/* Set TM_MIB(5) attributes defining service entry */
Fchg32(ibuf, TA_SERVICENAME, 0, "ECHO", 0);
Fchg32(ibuf, TA_ROUTINGNAME, 0, "ECHOROUTE", 0);

tpadmcall(...); /* See earlier example for detailed error processing */

```

Activate Master Site Admin

Activate the master site administrative processes (DBBL, BBL, Bridge) by setting the T_DOMAIN class object state to ACTIVE.

```

/* Clear the request buffer */ Finit32(ibuf, Fsizeof32(ibuf));

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

tpadmcall(...); /* See earlier example for detailed error processing */

```

Switch to Active Application Administration

Now that the application is active, we need to join the application and make our AdminAPI requests via the tpcall() 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);
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);

tpadmcalls(...); /* 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 ATMI Application Using C

Programming a BEA Tuxedo ATMI Application Using FML

TMFFNAME(5)

Synopsis

Server that runs the FactoryFinder and supporting NameManager services.

Syntax

```
TMFFNAME SRVGRP="identifier" SRVID="number"
      [CLOPT="[-A] [servopts options]
      [-- [-F ] [-N | -N -M [-f filename]]]"
```

Description

TMFFNAME is a server provided by BEA Tuxedo that runs the FactoryFinder and supporting NameManager services which maintain a mapping of application-supplied names to object references.

Parameters

-A
Advertise all services built into the server

-F
FactoryFinder service

-N
Slave NameManager service; this is the default.

-M
Master NameManager service

-f *filename*
Location of FactoryFinder import/export file

The FactoryFinder service is a CORBA-derived service that provides client applications with the ability to find application factories that correspond to application-specified search criteria. Consult the [BEA Tuxedo CORBA Programming Reference](#) for a complete description on the FactoryFinder API and [Creating CORBA Server Applications](#) for a description of registering and unregistering factories. The FactoryFinder service is the “default” service if no services are specified in the CLOPT.

The NameManager service is a BEA Tuxedo-specific service that maintains a mapping of application-supplied names to object references. One usage of this service is to maintain the application factory name-to-object reference list. The NameManager service can be booted with

an `-M` option that designates a Master role. If the `-M` option is not specified, the NameManager is assumed to be a Slave. Slave NameManagers obtain updates from the Master. Only one Master NameManager can be specified in an application.

The master NameManager can be configured to make factory objects residing in remote domains accessible in the local domain. It can also be configured to make factory objects residing in the local domain accessible from remote domains. Either or both of these configuration options can be specified in the FactoryFinder Domains configuration file, `factory_finder.ini`.

The location of the `factory_finder.ini` file is specified with the `-f` command-line option for the master NameManager. If the `-f` option is specified and the `factory_finder.ini` file is not found, the initialization of the master NameManager fails. If the `-f` option is not specified, only locally registered factory objects are accessible to the local application, and none of the local factory objects are accessible to applications in remote domains.

Note: It is possible to boot one or more `TMFFNAME` processes running the same service. To provide increased reliability, at least two NameManager services must be configured, preferably on different machines.

Interoperability

The `TMFFNAME` servers run on BEA Tuxedo version 4.0 software and later.

Notes

If there are less than two NameManager services configured in the application's `UBBCONFIG` (`TMFFNAME -N`), the server terminates itself during boot and writes an error message to the user log.

If a Master NameManager service is not configured in the application's `UBBCONFIG` file and is running when a Slave NameManager service starts, the server terminates itself during boot and writes an error message to the user log. Additionally, if the Master is down, registration and unregistration of factories is disabled until the Master restarts.

If a `TMSYSEVT` server is not configured in the application's `UBBCONFIG` file and is not running when a NameManager service is being started, the server terminates itself during boot and writes an error message to the user log.

If a NameManager service is not configured in the application's `UBBCONFIG` file and a FactoryFinder service is being started, the server terminates itself during boot and writes an error message to the user log.

If running an MP configuration, all name managers (`TMFFNAME -N`) should be configured to boot before any slave event service servers (`TMSYSEVT -S`). The master `TMSYSEVT` must still be booted

before any name managers. If slave TMSYSEVT booted before name managers, a slave name manager could miss update events sent by the master name manager, this can result in some clients seeing NoFactory exceptions when trying to find a factory, or the factory finder returning a factory object which is no longer registered (resulting in NO_IMPLEMENT or other exceptions when invoked on), or unexpected load balancing behavior.

If a slave name manager was shutdown and boot again, and if any customer written CORBA servers that register (or unregister) a factory are booting (or shutting down), then either boot (or shutdown) the customer written servers first or wait for the longest polling interval set on all the slave TMSYSEVT's. Default for those with no -p option is 30 seconds.

Example

```
*SERVERS
TMSYSEVT SRVGRP=ADMIN1 SRVID=44 RESTART=Y
        CLOPT="-A"

TMFFNAME SRVGRP=ADMIN1 SRVID=45 RESTART=Y
        CLOPT="-A -- -F"

TMFFNAME SRVGRP=ADMIN1 SRVID=46 RESTART=Y
        CLOPT="-A -- -N -M -f c:\appdir\import_factories.ini"
TMFFNAME SRVGRP=ADMIN2 SRVID=47 RESTART=Y
        CLOPT="-A -- -N"
TMFFNAME SRVGRP=ADMIN3 SRVID=48 RESTART=Y
        CLOPT="-A -- -F"
TMFFNAME SRVGRP=ADMIN4 SRVID=49 RESTART=Y
        CLOPT="-A -- -F"
```

See Also

[factory_finder.ini\(5\)](#), [TMSYSEVT\(5\)](#), [UBBCONFIG\(5\)](#), [userlog\(3c\)](#), "TP Framework" in *BEA Tuxedo CORBA Programming Reference*.

TMIFRSVR(5)

Name

The Interface Repository server

Synopsis

```
TMIFRSVR SRVGRP="identifier" SRVID="number" RESTART=Y GRACE=0  
CLOPT="[servopts options] -- [-f repository_file_name]"
```

Description

The TMIFRSVR server is a server provided by BEA for accessing the Interface Repository. The API is a subset of the CORBA-defined Interface Repository API. For a description of the Interface Repository API, see [BEA Tuxedo CORBA Programming Reference](#).

Parameter

```
[-f repository_file_name]  
Interface Repository filename. This file must have been generated previously using the  
id12ir command. If this parameter is not specified, the default repository filename  
repository.ifr located in the application directory (APPDIR) for the machine is used.  
If the repository file cannot be read, the server fails to boot.
```

Examples

```
*SERVERS  
  
#This server uses the default repository TMIFRSVR  
SRVGRP="IFRGRP" SRVID=1000 RESTART=Y GRACE=0  
  
#This server uses a non-default repository TMIFRSVR  
SRVGRP="IFRGRP" SRVID=1001 RESTART=Y GRACE=0  
CLOPT="-- -f /nfs/repository.ifr"
```

See Also

[ir2idl\(1\)](#), [UBBCONFIG\(5\)](#), [servopts\(5\)](#)

TMMETADATA(5)

Name

TMMETADATA - Tuxedo service metadata repository server

Synopsis

```
TMMETADATA SRVGRP="identifier" SRVID="number"
CLOPT="[-A] [servopts options] -- -f repository_file [-r]
```

Description

TMMETADATA is a Tuxedo system server that processes requests to retrieve and/or update Tuxedo service metadata repository information.

TMMETADATA provides and supports just one service, `.TMMETAREPOS`, which uses FML32 input and output buffers similar to those used by the Tuxedo MIB. The TMMETADATA FML32 buffer format is described in [MIB\(5\)](#).

Note: Metadata information retrieval and updating are handled through a service independent from `.TMIB` in order to avoid burdening the BBL with metadata request processing overhead since the metadata repository is stored separately from the Tuxedo configuration.

The CLOPT option is a string of command link options that is passed to TMMETADATA when it is booted. The following run-time parameters are recognized by TMMETADATA:

- f This option is mandatory and specifies the location of the metadata *repository_file*.
- r If this option is specified, TMMETADATA only allows retrieve information requests from the metadata repository and disallows any metadata repository update requests. The TMMETADATA default permissions setting is `read/write`.

Limitation

Because TMMETADATA provides only one service, `.TMMETAREPOS`, multiple TMMETADATA servers running on a particular Tuxedo domain must all be configured for the same permission access. That is, they either should all be read only or they should all be read and write.

Each TMMETADATA server must be configured to access the same metadata repository file or an exact copy of the file to provide consistent request results. Therefore, it is strongly recommended

that a stable version of the metadata repository is made available for multiple TMMETADATA server access.

Interoperability

TMMETADATA must run on a Tuxedo 9.0 release or later.

Tuxedo Jolt Repository

If invoked on a Tuxedo Jolt repository file with the `-r` option, TMMETADATA can read and return records from that file just as it would for a Tuxedo metadata repository file.

If invoked on a Tuxedo Jolt repository file without the `-r` option, TMMETADATA fails upon server initialization.

Example(s)

Listing 4 Single TMMETADATA Server Configuration

```
*SERVERS
  TMMETADATA SRVGRP=ADMIN1 SRVID=137 RESTART=Y MAXGEN=5
  GRACE=3600 CLOPT="-A -- -f /usr/tuxadmin/METAREPOS"
```

Listing 5 Multiple TMMETADATA Server Configuration

```
*SERVERS
TMMETADATA   SVRGRP=ADMIN1 SVRID=101 RESTART=N
CLOPT="-A -- -f /usr/tuxadmin/metarepos1 -r"
TMMETADATA   SVRGRP=ADMIN1 SVRID=102 RESTART=Y MAXGEN=5
GRACE=3600 CLOPT="-A -- -f /usr/tuxadmin/metarepos1 -r"
TMMETADATA   SVRGRP=ADMIN1 SVRID=103 RESTART=Y MAXGEN=5
GRACE=3600 CLOPT="-A -- -f /usr/tuxadmin/metarepos1 -r"
```

See Also

[tpgetrepos\(3c\)](#), [tpsetrepos\(3c\)](#), [MIB\(5\)](#).

TMQFORWARD(5)

Name

TMQFORWARD—Message Forwarding server

Synopsis

```
TMQFORWARD SRVGRP="identifier" SRVID="number" REPLYQ=N CLOPT="
[-A] [servopts options] -- -q queuename[,queuename...]
[-t trantime] [-i idletime] [-b timeout] [-e] [-d] [-n] [-f delay] "
```

Description

The message forwarding server is a BEA Tuxedo system-supplied server that forwards messages that have been stored using `tqueue()` for later processing. The application administrator enables automated message processing for the application servers by specifying this server as an application server in the `SERVERS` section.

The location, server group, server identifier and other generic server related parameters are associated with the server using the already defined configuration file mechanisms for servers. The following is a list of additional command-line options that are available for customization.

`-q queuename[,queuename...]`

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`

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`

Used to indicate the amount of time (in seconds) that the server remains idle after draining the queue(s) that it is reading. A negative value indicates an amount of time in milliseconds. For example if you specify `-i -10`, the idle time will be 10 milliseconds.

If a value of zero is specified, the server will read the queue(s) continually, which can be inefficient if the queues do not continually have messages. If no value is specified, the default is 30 seconds.

`-b timeout`

Used to limit nontransaction block waiting time, in seconds, for a forwarded service to complete. The `-b` option can only be used with the `-f` option.

`-e`

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`

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`

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`

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, 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, 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, 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, a customized version of `TMQFORWARD` needs to be built using `buildserver(1)` with a customized type switch. See the description in *Using the ATMI/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 filename 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 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, filenames 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 Windows, :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\(3c\)](#), [topenqueue\(3c\)](#), [servopts\(5\)](#), [TMQUEUE\(5\)](#), [UBBCONFIG\(5\)](#)

Setting Up a BEA Tuxedo Application

Programming a BEA Tuxedo ATMI 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 additional command-line option is available for customization.

`-t timeout`

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, a customized version of `TMQUEUE` needs to be built using `buildserver(1)`. See the description in *Using the ATMI/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 filename 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 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, filenames 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 Windows, :myqueue becomes ;myqueue
TMQUEUEGRP1 GRPNO=1 TMSNAME=TMS_QM
  OPENINFO="TUXEDO/QM:/dev/device1:myqueue"
# For Windows, :myqueue becomes ;myqueue
TMQUEUEGRP2 GRPNO=2 TMSNAME=TMS_QM
  OPENINFO="TUXEDO/QM:/dev/device2:myqueue"

*SERVERS
# The queue space name, myqueue, is aliased as ACCOUNTING in this example
TMQUEUE SRVGRP="TMQUEUEGRP1" SRVID=1000 RESTART=Y GRACE=0
  CLOPT="-s ACCOUNTING:TMQUEUE"
TMQUEUE SRVGRP="TMQUEUEGRP2" SRVID=1000 RESTART=Y GRACE=0
  CLOPT="-s ACCOUNTING:TMQUEUE"
```

```

TMQFORWARD SRVGRP="TMQUEUEGRP1" SRVID=1001 RESTART=Y GRACE=0 REPLYQ=N
  CLOPT=" -- -qservice1"
TMQFORWARD SRVGRP="TMQUEUEGRP2" SRVID=1001 RESTART=Y GRACE=0 REPLYQ=N
  CLOPT=" -- -qservice1"
*SERVICES
ACCOUNTING ROUTING="MYROUTING"
*ROUTING
MYROUTING FIELD=ACCOUNT BUFTYPE="FML"
RANGES="MIN - 60000:TMQUEUEGRP1,60001-MAX:TMQUEUEGRP2"

```

In this example, two queues spaces are available. Both TMQUEUE servers offer the same services and routing is done via the ACCOUNT field in the application typed buffer.

See Also

[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 ATMI Application Using C

TMSYSEVT(5)

Name

TMSYSEVT—System event reporting process

Synopsis

```
TMSYSEVT SRVGRP="identifier" SRVID="number"
  [CLOPT="[-A] [servopts options]
  [-- [-S] [-p poll-seconds] [-f control-file]]"]
```

Description

TMSYSEVT is a BEA Tuxedo system provided server that processes event reports related to system failure or potential failure conditions. The event reports are filtered, and may trigger one or more notification actions.

Filtering and notification rules are stored in *control-file*, which defaults to `${APPDIR}/tmsysevt.dat`. Control file syntax is defined in [EVENT_MIB\(5\)](#); specifically, the attributes of the classes in `EVENT_MIB` can be set to activate subscriptions under the full range of notification rules.

It is possible to boot one or more secondary TMSYSEVT processes for increased availability. Additional servers must be booted with the `-S` command-line option, which indicates a “secondary” server.

When the [EVENT_MIB\(5\)](#) configuration is updated, the primary TMSYSEVT server writes to its control file. Secondary servers poll the primary server for changes and update their local control file if necessary. The polling interval is controlled by the `-p` option, and is 30 seconds by default.

Note: If you are setting up an MP configuration that includes more than one release of the BEA Tuxedo system and you want to run the `TMUSREVT` and/or `TMSYSEVT` server, you must run these servers on the node with the highest available release of the system.

Interoperability

TMSYSEVT must run on a BEA Tuxedo release 6.0 or later machine.

Notices

To migrate the primary TMSYSEVT server to another machine, the system administrator must provide a current copy of *control-file*. Each secondary TMSYSEVT server automatically maintains a recent copy.

TMSYSEVT needs access to the system's FML32 field table definitions for system events. FLDTBLDIR32 should include \$TUXDIR/udataobj, and FIELDTBLS32 should include evt_mib. These environment variables may be set in the machine's or server's environment file.

Example

```
*SERVERS
TMSYSEVT SRVGRP=ADMIN1 SRVID=100 RESTART=Y GRACE=900 MAXGEN=5
  CLOPT="-A --"
TMSYSEVT SRVGRP=ADMIN2 SRVID=100 RESTART=Y GRACE=900 MAXGEN=5
  CLOPT="-A -- -S -p 90"
```

See Also

[tpssubscribe\(3c\)](#), [EVENTS\(5\)](#), [EVENT_MIB\(5\)](#), [TMUSREVT\(5\)](#)

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 user log, if this was not being done already.

Dyeing can be explicitly turned on or off by the `dye` and `undy` 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 -, 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, 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.

Legal *receiver* values are:

`ulog`

Outputs trace record information to the user log.

`utrace`

Outputs trace record information as defined by the user. The `utrace` receiver calls user-defined `tputrace(3c)` for atmi trace category records only. Users can customize trace record information and output location.

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 user log.

system(*command*)

Execute the *command* using `system(3)` (this is not supported for Windows 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 Windows 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 user log, the line looks like this:

hhmmss.system-name!process-name.pid: TRACE:cc:data

Notices

Match patterns cannot be specified for the receiver and trigger for Workstation clients running on MAC platforms; the regular expressions will be ignored.

The `tmadmin changetrace` command cannot be used to affect the tracing level for Workstation clients.

Examples

To trace a client, as well as to trace all ATMI calls made by an application server on behalf of that client, set and export `TMTRACE=on` in the environment of the client. This specification will

cause all explicit ATMI trace points in the client to be logged and message dyeing to be turned on. Any application server process that performs a service on behalf of the client will automatically log all explicit ATMI trace points.

To see all client trace points, both explicit and implicit, for the previous example, set and export:

```
TMTRACE="*:u log: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/u log: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/u log
```

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=*:u log: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\(3c\)](#), [tputrace\(3c\)](#)

TMUSREVT(5)

Name

TMUSREVT—User event reporting process

Synopsis

```
TMUSREVT SRVGRP="identifier" SRVID="number"
  [CLOPT="[-A] [servopts options]
  [-- [-S] [-p poll-seconds] [-f control-file]]"]
```

Description

TMUSREVT is a BEA Tuxedo system provided server that processes event report message buffers from [tppost\(3c\)](#), and acts as an EventBroker to filter and distribute them.

Filtering and notification rules are stored in *control-file*, which defaults to `${APPDIR}/tmusrevt.dat`. Control file syntax is defined in [EVENT_MIB\(5\)](#); specifically, the attributes of the classes in `EVENT_MIB` can be set to activate subscriptions under the full range of notification rules.

It is possible to boot one or more secondary TMUSREVT processes for increased availability. Additional servers must be booted with the `-S` command-line option, which indicates a “secondary” server.

When the [EVENT_MIB\(5\)](#) configuration is updated, the primary TMUSREVT server writes to its control file. Secondary servers poll the primary server for changes and update their local control file if necessary. The polling interval is controlled by the `-p` option, and is 30 seconds by default.

Note: If you are setting up an MP configuration that includes more than one release of the BEA Tuxedo system and you want to run the TMUSREVT and/or TMSYSEVT server, you must run these servers on the node with the highest available release of the system.

Interoperability

TMUSREVT must run on a BEA Tuxedo release 6.0 or later machine.

Notices

To migrate the primary TMUSREVT server to another machine, the system administrator must provide a current copy of *control-file*. Each secondary TMUSREVT server automatically maintains a recent copy.

If `tppost()` will be called in transaction mode, all TMUSREVT server groups must have transactional capability (a TMS process).

The TMUSREVT server's environment variables must be set so that FML field tables and viewfiles needed for message filtering and formatting are available. They could be set in the machine's or server's environment file.

Example

```
*SERVERS
TMUSREVT SRVGRP=ADMIN1 SRVID=100 RESTART=Y MAXGEN=5 GRACE=3600
CLOPT="-A --"
TMUSREVT SRVGRP=ADMIN2 SRVID=100 RESTART=Y MAXGEN=5 GRACE=3600
CLOPT="-A -- -S -p 120"
```

See Also

[tppost\(3c\)](#), [tpssubscribe\(3c\)](#), [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 `tperrno` for errors that occur when executing a BEA Tuxedo system library routine.

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

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

The following list describes the general meaning of each error:

TPEABORT

A transaction could not commit because either the work performed by the initiator or by one or more of its participants could not commit.

TPEBADDESC

A call descriptor is invalid or is not the descriptor with which a conversational service was invoked.

TPEBLOCK

A blocking condition exists and `TPNOBLOCK` was specified.

TPEIDIAGNOSTIC

The enqueueing of a message on the specified queue failed. The reason for failure can be determined by the diagnostic returned via *ctl*.

TPEEVENT

An event occurred; the event type is returned in *revent*.

TPEGOTSIG

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

TPEHAZARD

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

TPEHEURISTIC

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

TPEINVAL

An invalid argument was detected.

TPEITYPE

The type and subtype of the input buffer is not one of the types and subtypes that the service accepts.

TPELIMIT

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

TPEMATCH

svcname is already advertised for the server but with a function other than *func*.

TPEMIB

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\(5\)](#).

TPENOENT

Cannot send to *svc* because it does not exist or is not the correct type of service.

TPEOS

An operating system error has occurred.

TPEOTYPE

The type and subtype of the reply are not known to the caller.

TPPEPERM

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.

TPPEPROTO

A library routine was called in an improper context.

TPPERELEASE

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

This error code indicates that either a timeout has occurred or a transactional ATMI function has been attempted, in spite of the fact that the current transaction is already marked rollback only.

If the caller is in transaction mode, then either the transaction is already rollback only or a transaction timeout has occurred. The transaction is marked abort-only. If the caller is not in transaction mode, a blocking timeout has occurred. (A blocking timeout cannot occur if **TPNOBLOCK** and/or **TPNOTIME** is specified.) In either case, no changes are made to `*odata`, its contents, or `*olen`.

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

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

tperrno(5)

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

`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);  
. . .
```

`tpurcode(5)`

```
(void) fprintf(stdout, "Returned string is: %s\n", rcvbuf);  
(void) fprintf(stdout, "Returned tpurcode is: %d\n", tpurcode);
```

If we call the sample client, `simpl`, with the value of “My String,” the output will look like this:

```
%simpl "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 Viewfiles
- Filesystem 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 username 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)`, `tadmin(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 other BEA Tuxedo code. When a call is made to `tmpnam()`, the BEA Tuxedo system ignores the `TMPDIR` variable.

Note that for BEA Tuxedo release 6.5 or earlier, the BEA Tuxedo code responsible for transferring message files from clients to service queues would write a message file to the temporary location specified with the `tmpnam()` function *if the service queue was too full to hold the message file*; the code would then place the pathname of the temporary location on the service queue. For BEA Tuxedo release 7.1 or later, this code operates just like it did in previous releases except that the temporary location, if needed, is the pathname of the directory specified by `TMPDIR` *assuming that the variable is set*; if `TMPDIR` is not set, the temporary location becomes the one specified by the underlying operating system.

TZ

On systems where the ANSI C `mkttime` functions does not exist, `TZ` must be set to use the BEA Tuxedo `gp_mkttime(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)`.

TUXDIR

Specifies the base directory where the BEA Tuxedo system software is installed.

ULOGPFX

Prefix of the filename of the central event log; default, `ULOG`.

TPMBENC

Specifies the code-set encoding name that the application server or client running BEA Tuxedo 8.1 or later includes in an allocated `MBSTRING` typed buffer. When an application server or client process allocates and sends an `MBSTRING` buffer, the code-set encoding name defined in `TPMBENC` is automatically added as an attribute to the buffer and sent with the buffer data to the destination process.

When the application server or client process receives an `MBSTRING` buffer, and assuming another environment variable named `TPMBACONV` is set, the code-set encoding name defined in `TPMBENC` is automatically compared to the code-set encoding name in the received buffer; if the names are *not* the same, the `MBSTRING` buffer data is automatically converted to the encoding defined in `TPMBENC` before being delivered to the server or client process.

TPMBENC has no default value. For an application server or client using MBSTRING typed buffers, TPMBENC must be defined.

Note: TPMBENC is used in a similar way for FLD_MBSTRING fields in an FML32 typed buffer.

TPMBACONV

Specifies whether the application server or client running BEA Tuxedo 8.1 or later automatically converts the data in a received MBSTRING buffer to the encoding defined in TPMBENC. By default, the automatic conversion is turned off, meaning that the data in the received MBSTRING buffer is delivered to the destination server or client process *as is*—no encoding conversion. Setting TPMBACONV to any non-NULL value, say Y (yes), turns on the automatic conversion.

Note: TPMBACONV is used in a similar way for FLD_MBSTRING fields in an FML32 typed buffer.

URLENTITYCACHING

Specifies whether the application server or workstation machine running BEA Tuxedo 8.1 or later software caches Document Type Definition (DTD), XML schema, and entity files; specifically, whether the Apache Xerces-C++ parser running on the application server or workstation machine caches the DTD and XML schema files when validation is required, or caches external entity files called out in the DTD. By default, the caching is turned on (Y). Setting URLENTITYCACHING to N (no) turns off the caching.

URLENTITYCACHEDIR

Applies only if URLENTITYCACHING=Y (yes) or is not set; for details, see the description of URLENTITYCACHING in this list.

Specifies the directory in which the application server or workstation machine running BEA Tuxedo 8.1 or later software caches DTD, schema, and entity files; specifically, where the Apache Xerces-C++ parser running on the application server or workstation machine caches the DTD, XML schema, and entity files. The URLENTITYCACHEDIR variable specifies the absolute pathname for the cached files. If URLENTITYCACHEDIR is not specified, the default directory becomes URLEntityCachedir, which will be created in the current working directory of the application server or Workstation client process provided that the appropriate write permissions are set.

More information about these variables can be found in *Programming a BEA Tuxedo ATMI Application Using C*, *Setting Up a BEA Tuxedo Application*, and *Administering a BEA Tuxedo Application at Run Time*.

Variables for Field Table Files and Viewfiles

The following environment variables are used by FML and VIEWS:

FIELDTBLS

Comma-separated list of field table files.

VIEWFILES

Comma-separated list of binary viewfiles.

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 ATMI Application Using C*, and *Programming a BEA Tuxedo ATMI Application Using FML*.

Filesystem and TLOG Variables

The following variables are used by the BEA Tuxedo system filesystem and the transaction log.

FSCONFIG

The pathname for the Universal Device List.

FSMAXCOMMIT

Sets the maximum size of the commit buffer.

FSMAXUPDATE

Sets the size of the update list and the maximum number of updates.

FSMSGREP

Sets the message repetition interval.

FSOFFSET

Specifies an offset into the Universal Device List.

Workstation Variables

The following variables are used on Workstation client machines:

TPMBENC

See [“Key BEA Tuxedo System Variables” on page 454](#).

TPMBA CONV

See [“Key BEA Tuxedo System Variables” on page 454](#).

URLENTITYCACHING

See [“Key BEA Tuxedo System Variables” on page 454](#).

URLENTITYCACHEDIR

See “Key BEA Tuxedo System Variables” on page 454.

WSINTOPPRE71

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 (`WSINTOPPRE71=Y`) allows interoperability.

WSBUFFERS

The number of packets per application.

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.

WSENVFILE

Pathname of a file containing Workstation client environment variables.

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 attempts to bind before making an outbound connection.

WSFRANGE

The range of TCP/IP ports to which a native process attempts 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 ATMI 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.

ISSANE

Used to enable /Q to continue to work even though `TMQFORWARD` is terminated abnormally without holding locks. When this variable is set to `yes/YES`, if `TMQFORWARD` is shutdown abnormally, (for example, hung due to application server hanging), when shutdown request is sent to `TMQFORWARD`, and `TMQFORWARD` does not hold any /Q locks, the /Q can work normally and `TMQFORWARD` can be restarted later. Otherwise, if `TMQFORWARD` terminated abnormally, /Q will be marked as insane and must be restarted.

There is more information on this in *Using the ATMI /Q Component*.

COBOL Variables

The following environment variables are used with COBOL:

ALTCC

Specifies the compiler for use with COBOL compilations.

Note: If using Fujitsu's NetCOBOL compiler, you must set this variable to `cobcc85`, regardless of the platform.

ALTCFLAGS

Flags to be passed to the COBOL compiler.

Note: On a Windows 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.

Note: If using Fujitsu's NetCOBOL compiler, you may not set this variable. Refer to the NetCOBOL manuals for specific information about COBOL environment variables.

COBDIR

Specifies the directory where COBOL compiler software is located.

COBOPT

Contains command-line arguments for the COBOL compiler.

Note: If using Fujitsu's NetCOBOL compiler, you may not set this variable. Refer to the NetCOBOL manuals for specific information about COBOL environment variables.

TM_ORB_CLTMAXRTY

Used to specify the maximum time an ORB client should retry for a request. Valid values are the integers 0 through 32767. No setting or setting to 0 results in no retry limit. For other invalid values, system will reset interval value to 0 silently.

There is more information on these variables in the *Programming a BEA Tuxedo ATMI 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. In addition, in a high availability (HA) environment, PMID can be used to replace the machine name specified in the UBBCONFIG file with an alternate machine name. This allows for moving a master machine from master to backup in an HA cluster.

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.

TMNOTHEADS

To turn off multithreaded processing, set this variable to *yes*. For applications that do not use threads, turning them off should significantly improve performance by reducing the amount of calls to mutexing functions.

TMSICACHEENTRIESMAX

Used to specify the amount of service and interface caching on a per-process basis. Valid values are the integers 0 through 32767. The value set for this variable will override any value in the `UBBCONFIG` file.

TM_ENGINE_TMSHMSEGSZ

Through this environment variable, Tuxedo 9.x allows you to adjust the maximum size of a shared memory segment accessible by Tuxedo to something other than the Tuxedo imposed default for the operating system. The Tuxedo system imposes a default shared memory segment maximum on its use of shared memory. This Tuxedo imposed limit can be lower than the maximum memory segment size tuned in the users kernel.

This environment variable should be set to the number of Mega-bytes that a shared memory segment should be set to. For example, if you want to set the segment size to 500MB, you would issue the following command in the tuxedo environment:

```
export TM_ENGINE_TMSHMSEGSZ=500
```

The logic is implemented such that the maximum shared memory segment used by Tuxedo will be set to $500 * (1024 * 1024)$ for a total of 524,288,000 bytes. If the environment variable is not set, then the Tuxedo default size for the given operating system will be used.

TM_ICU_COMPATIBILITY

This environment variable is used in `MBSTRING` codeset conversion to keep compatible with ICU standard. The default is GNU standard. `TM_ICU_COMPATIBILITY` can be set as follows:

```
export TM_ICU_COMPATIBILITY=yes
```

TM_GWT_OLDSECHECK

This environment variable is for *old-style security check*. This should be set in the environment of `GWTDOMAIN`. This variable will not affect any other Tuxedo processes even if set for them. It is used to make interdomain transactional requests between Tuxedo 6.5 and other Tuxedo releases work when the Tuxedo domain running on Tuxedo 6.5 cannot upgrade to patch level 446.

Note: When the Tuxedo 6.5 domain is upgraded to patch level 446 or later, this environment variable should be removed.

If `TM_GWT_OLDSECHECK=Y`, the old-style security check is used by `GWTDOMAIN`. This is necessary to interoperate with Tuxedo 6.5 patches before patchlev 446. However, this implies weaker security. If `TM_GWT_OLDSECHECK=N`, the `GWTDOMAIN` process writes an informational `ULOG` message indicating that fact, when it receives the first incoming data/connection from the network.

If `TM_GWT_OLDSECHECK=N` or if `TM_GWT_OLDSECHECK` is *not* set, the latest security check is used. This implies that all the interoperating Tuxedo 6.5 domains should be at least at patchlev 446.

TM_LOG_ESYS

This environment variable is used to add detailed information to `ULOG` when `TPSYSTEM` error occurs. `TM_LOG_ESYS` is limited to ATMI calls and can be set as follows:

```
export TM_LOG_ESYS=all
export TM_LOG_ESYS=native
export TM_LOG_ESYS=native:ws
export TM_LOG_ESYS=native:ws:domain (same as "all")
```

Note: `native=native` ATMI calls, `ws=workstation` ATMI calls, `domain=ATMI` calls across a domain gateway.

TUX_BLOCKLICIW

Used to help reduce the amount of error messages sent to the user log and event broker.

Note: User should `set` an arbitrary string to turn `TUX_BLOCKLICIW` on and `unset` to turn off.

`TUX_BLOCKLICIW` blocks the following error messages:

1) User Log:

```
CMDTUX_CAT:4749 WARN: Reached 100% of TUXEDO System Binary Licensed
User Count
```

```
CMDTUX_CAT:4753 INFO: Reached 90% of TUXEDO System Binary Licensed
User Count
```

```
CMDTUX_CAT:4729 WARN: Reached 100% of TUXEDO System Binary Licensed
User Count
```

```
CMDTUX_CAT:4731 INFO: Reached 90% of TUXEDO System Binary Licensed
User Count
```

2) Event Broker:

```
CMDTUX_CAT:4750 WARN: .SysLicenseWarn: Reached 100%% of TUXEDO System
Binary Licensed User Count
```

```
CMDTUX_CAT:4754 INFO: .SysLicenseInfo: Reached 90%% of TUXEDO System  
Binary Licensed User Count
```

```
CMDTUX_CAT:4730 WARN: .SysLicenseWarn: Reached 100%% of TUXEDO System  
Binary Licensed User Count
```

```
CMDTUX_CAT:4732 INFO: .SysLicenseInfo: Reached 90%% of TUXEDO System  
Binary Licensed User Count
```

TUX_SSL_ENFORCECONSTRAINTS

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 <stdio.h>
#include <tmtypes.h>

/*
 * Initialization of the buffer type switch.
 */

struct tmttype_sw_t tm_typesw[] = {
{
    "CARRAY",      /* type */
    "",            /* subtype */
    0              /* dfltsize */
    NULL,          /* initbuf */
    NULL,          /* reinitbuf */
    NULL,          /* uninitbuf */
    NULL,          /* presend */
    NULL,          /* postsend */
    NULL,          /* postrecv */
    NULL,          /* encdec */
    NULL,          /* route */
    NULL,          /* filter */
    NULL,          /* format */
    NULL,          /* presend2 */
    NULL          /* multibyte code-set encoding conversion */
},
{

```

```

    "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 */
    NULL,         /* presend2 */
    NULL          /* multibyte code-set encoding conversion */
},
{
    "FML",        /* type */
    "",           /* subtype */
    1024,        /* dfltsize */
    _finit,      /* initbuf */
    _freinit,    /* reinitbuf */
    _funinit,    /* uninitbuf */
    _fpresend,   /* presend */
    _fpostsend, /* postsend */
    _fpostrecv, /* postrecv */
    _fencdec,    /* encdec */
    _froute,     /* route */
    _ffilter,    /* filter */
    _fformat,    /* format */
    NULL,        /* presend2 */
    NULL         /* multibyte code-set encoding conversion */
},
{
    "VIEW",      /* type */
    "",           /* subtype */
    1024,        /* dfltsize */
    _vinit,      /* initbuf */

```

```

    _vreinit,      /* reinitbuf */
    NULL,          /* uninitbuf */
    _vpresent,    /* present */
    NULL,         /* postsend */
    NULL,         /* postrecv */
    _vencdec,     /* encdec */
    _vroute,     /* route */
    _vfilter,    /* filter */
    _vformat,    /* format */
    NULL,        /* present2 */
    NULL         /* multibyte code-set encoding conversion */
},
{
    /* 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 */
    _vpresent, /* present */
    NULL,     /* postsend */
    NULL,     /* postrecv */
    _vencdec, /* encdec */
    _vroute,  /* route */
    _vfilter, /* filter */
    _vformat, /* format */
    NULL,     /* present2 */
    NULL      /* multibyte code-set encoding conversion */
},
{
    /* XATMI - identical to VIEW */
    {'X','_','C','O','M','M','O','N'}, /* type */

```

```

    "",          /* subtype */
    1024,       /* dfltsize */
    _vinit,     /* initbuf */
    _vreinit,   /* reinitbuf */
    NULL,      /* uninitbuf */
    _vpresent, /* present */
    NULL,      /* postsend */
    NULL,      /* postrecv */
    _vencdec,  /* encdec */
    _vroute,   /* route */
    _vfilter,  /* filter */
    _vformat,  /* format */
    NULL,      /* present2 */
    NULL       /* multibyte code-set encoding conversion */
},
{
    "FML32",    /* type */
    "",        /* subtype */
    1024,      /* dfltsize */
    _finit32,  /* initbuf */
    _freinit32, /* reinitbuf */
    _funinit32, /* uninitbuf */
    _fpresent32, /* present */
    _fpostsend32, /* postsend */
    _fpostrecv32, /* postrecv */
    _fencdec32, /* encdec */
    _froute32, /* route */
    _ffilter32, /* filter */
    _fformat32, /* format */
    _fpresent232 /* present2 */
    _fmbconv32 /* multibyte code-set encoding conversion */
},
{
    "VIEW32",  /* type */
    "",        /* subtype */
    1024,      /* dfltsize */
    _vinit32,  /* initbuf */
    _vreinit32, /* reinitbuf */

```

```

        NULL,          /* uninitbuf */
        _vpresent32,   /* present */
        NULL,          /* postsend */
        NULL,          /* postrecv */
        _vencdec32,    /* encdec */
        _vrout32,      /* route */
        _vfilter32,    /* filter */
        _vformat32,    /* format */
        NULL,          /* present2 */
        NULL           /* multibyte code-set encoding conversion */
    },
    {
        "XML",          /* type */
        "*",            /* subtype */
        0,              /* dfltsize */
        NULL,           /* initbuf */
        NULL,           /* reinitbuf */
        NULL,           /* uninitbuf */
        NULL,           /* present */
        NULL,           /* postsend */
        NULL,           /* postrecv */
        NULL,           /* encdec */
        _xrout,         /* route */
        NULL,           /* filter */
        NULL,           /* format */
        NULL,           /* present2 */
        NULL           /* multibyte code-set encoding conversion */
    },
    {
        "MBSTRING",    /* type */
        "*",            /* subtype */
        0,              /* dfltsize */
        _mbsinit,      /* initbuf */
        NULL,           /* reinitbuf */
        NULL,           /* uninitbuf */
        _mbspresent,   /* present */
        NULL,           /* postsend */
        NULL,           /* postrecv */
    }

```

```

        NULL,          /* encdec */
        NULL,          /* route */
        NULL,          /* filter */
        NULL,          /* format */
        NULL,          /* presend2 */
        _mbsconv       /* multibyte code-set encoding conversion */
    },
    {
        ""
    }
};

struct tmtypesw_t _TM_FAR *
_TMDLLENTY
_tmtypeswaddr(void)
{
    return(tm_typesw);
}

```

Description

The following table lists the 11 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
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
XML	Buffer for XML documents
MBSTRING	Character array for multibyte characters

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 subtype 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/tmypesw.c`, and may be used as a starting point. The recommended procedure for installing a new buffer type switch is to compile `tmypesw.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 tmttype_sw_t {
    char type[TMTYPELEN];      /* type of buffer */
    char subtype[TMSTYPELEN]; /* subtype of buffer */
    long dfltsize;            /* default size of buffer */
    /* buffer initialization function pointer */
    int (_TMDLLENTY *initbuf) _((char _TM_FAR *, long));

    /* buffer reinitialization 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));

    /* XDR 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,
    long, char _TM_FAR *, long, long _TM_FAR *));

/* Multibyte code-set encoding conversion function pointer*/
long (_TMDLLENTY *mbconv) _((char _TM_FAR *, long,
    char _TM_FAR *, char _TM_FAR *, long, long _TM_FAR *));

/* this space reserved for future expansion */
void (_TMDLLENTY *reserved[8]) _((void));
};
/*
 * application types switch pointer
 * always use this pointer when accessing the table
 */
extern struct tmtypes_sw_t *tm_typeswp;

```

Description

Each buffer type and subtype 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

When a BEA Tuxedo application is booted, the `tmboot` command refers to a binary configuration file called `TUXCONFIG` 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 `UBBCONFIG`. To configure an application, an administrator creates a `UBBCONFIG` file (with a text editor) and loads the file into a binary version (`TUXCONFIG`) by running the `tmloadcf(1)` command. During the life of the application, the `TUXCONFIG` file is used by various BEA Tuxedo administrative tools. `tadmin(1)` uses the configuration file (or a copy of it) in its monitoring activity. `tmshutdown(1)` references the configuration file for information needed to shut the application down.

A BEA Tuxedo `UBBCONFIG` file may be given any name as long as the content of the file conforms to the format described on this reference page. In addition, the `TUXCONFIG` file may be given any name; the actual name is the device or system filename specified in the `TUXCONFIG` environment variable.

For additional information pertaining to the entire `UBBCONFIG` file, see [“UBBCONFIG\(5\) Additional Information” on page 519](#).

Definitions

A server is a process that accepts requests and sends replies for clients and other servers. A client originates requests and gets replies.

A resource manager is an interface and associated software providing access to a collection of information and/or processes. An example of a resource manager is a database management system; a resource manager instance is a particular instantiation of a database controlled by a DBMS. A distributed transaction is a transaction that spans multiple resource manager instances, is started with `tpbegin()`, and ended with `tpcommit()` or `tpabort()`.

A server group is a resource manager instance and the collection of servers and/or services providing access to that resource manager instance on a particular machine. The XA interface associated with the group is used for transaction management. If a server does not access a resource manager instance or does not access it as part of a distributed transaction, it must be in a server group with a `NULL` XA interface. Similarly, clients run in a special client group that does

not have to be specified in the `GROUPS` section. The client group is not associated with a resource manager.

A remote domain is defined to be an environment for which the bulletin board for this BEA Tuxedo system configuration is not available. Remote domains are not specified in the `UBBCONFIG` file, but rather through host-specific environment variables that are specified in host-specific reference pages.

Configuration File Format

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`
- `INTERFACES`
- `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 (bytes), 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 `TUXCONFIG`, `TUXDIR`, `APPDIR`, `TLOGDEVICE`, `ULOGPFX`, `ENVFILE`, `TMSNAME`, `RCMD`, `NADDR`, `NLSADDR`, `FADDR`, and `AOUT` (in `SERVERS` section) parameters, which can be 256 characters in length as of BEA Tuxedo release 8.1; string values for these parameters are limited to 78 characters in length for BEA Tuxedo 8.0 or earlier.
- The `SEC_PRINCIPAL_NAME`, `SEC_PRINCIPAL_LOCATION`, and `SEC_PRINCIPAL_PASSVAR` parameters, which can be 51, 1023, and 31 characters in length respectively (not including the terminating `NULL`)
- The `RANGES` parameter, which can be 2048 characters in length (except in Domains, where it can be 4096 characters in length)

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.”

`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\(1\)](#)).

`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`, `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 1,048,575. To calculate an adequate value, be sure to count the number of services used by both application servers and system servers, such as the BBL, DBBL, BRIDGE, TMS, and any other system servers needed by the application. 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.

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 32,768. 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 in bytes the maximum string pool space to be accommodated in the bulletin board string pool table for data-dependent routing range strings. This value 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.

Strings and carrays specified within the `RANGES` values in the `ROUTING` section are stored in the string pool. Additional space should be allocated to allow for run-time growth.

MAXSPDATA *numeric_value*

Specifies in bytes the maximum string pool space to be accommodated in the bulletin board common string pool. This value must be greater than or equal to 0 and less than or equal to 2147483640. ~~The default is 0.~~ This parameter applies only to applications running BEA Tuxedo 8.1 or later software.

~~[In most cases, accepting the default for this parameter will result in the BEA Tuxedo system allocating sufficient string pool space for the following TUXCONFIG parameter strings whose maximum allowed length has been increased to 256 bytes in BEA Tuxedo 8.1: TUXCONFIG, TUXDIR, APPDIR, TLOGDEVICE, ULOGPFX, ENVFILE, TMSNAME, RCMD, NADDR, NLSADDR, FADDR, and the SERVERS section AOUT.]~~

BEA Tuxedo system needs the bulletin board string pool size ($\text{MAXQUEUES} * 257 * 2 + 8224$) at a minimum, where `MAXQUEUES` is the configured or default value of parameter `MAXQUEUES` in the same `*RESOURCE` section. If this parameter is not configured or the specified value is smaller than the minimum required size, BEA Tuxedo system automatically changes the value to the minimum required size.

For applications for which extensive dynamic configuration is anticipated (for example, anticipating the addition of six more machines to a BEA Tuxedo application), administrators can use the `MAXSPDATA` parameter to increase the size of the common string pool. Note that adjusting the size of the common string pool has no effect on the size of the of the routing string pool controlled by the `MAXRTDATA` parameter. The two string pools are separate.

Regardless of the value specified for `MAXSPDATA`, the BEA Tuxedo system will *not* allocate an amount of string pool space outside of a system-calculated range based on (1) the strings actually specified in the `TUXCONFIG` file and (2) the amount of space that would be required if all 256-byte capable strings were specified. The `tmloadcf(1)` command will report a warning if the user-specified value is outside of this range and then set the value to the closest acceptable value.

Note that of the `TUXCONFIG` parameters whose maximum allowable length has been increased to 256 bytes, only the `GROUPS` section `TMSNAME` parameter and the `SERVERS` section `AOUT` and `RCMD` parameters are actually stored in the bulletin board. The others are read in at process startup time and stored in process memory.

Note: If TSAM Event Plug-in is used in Tuxedo application, `MAXSPDATA` should be configured explicitly for additional TSAM Plug-in event rules storage. For more information, see [BEA TSAM Agent](#) in the *BEA TSAM Administration Guide*.

`MAXTRANTIME` *numeric_value*

Specifies in seconds the maximum timeout allowed for transactions started in or received by this BEA Tuxedo application. This value must be greater than or equal to 0 and less than or equal to 2147483647. The default is 0, which indicates that no global transaction timeout limit is in effect. This parameter applies only to applications running BEA Tuxedo 8.1 or later software.

If the `MAXTRANTIME` timeout value is less than the `TRANTIME` timeout value specified for an `AUTOTRAN` service or the timeout value passed in a `tpbegin(3c)` call to start a transaction, the timeout for a transaction is reduced to the `MAXTRANTIME` value. `MAXTRANTIME` has no effect on a transaction started on a machine running BEA Tuxedo 8.0 or earlier software, except that when a machine running BEA Tuxedo 8.1 or later software is infected by the transaction, the transaction timeout value is capped—reduced if necessary—to the `MAXTRANTIME` value configured for that machine.

Even if the `TRANTIME` value specified in the `SERVICES` section of the `UBBCONFIG` file is greater than the `MAXTRANTIME` value, the `tmloadcf(1)` command loads the configuration without error. Any BEA Tuxedo 8.1 or later machine infected with the `AUTOTRAN` transaction will automatically reduce the transaction timeout to the `MAXTRANTIME` value configured for that machine.

`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`, 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.

`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, set `LDBAL` to `N` because load balancing is automatic.

If you set `LDBAL` to `Y`, server load balancing is performed automatically. Each interface request is routed to the server with the smallest total load. The routing of a request to a server causes the server's total to be increased by the `LOAD` factor of the `CORBA` interface requested.

When load balancing is not activated and multiple servers offer the same `CORBA` interface, the first available queue receives the request.

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.

OPTIONS {[LAN | SSL | MIGRATE | NO_XA | NO_AA], *}

Specifies options that are used. If two or more 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 SSL is not specified, LLE is used. In order for SSL to take effect, the `tlisten(1) -s` option must be specified.

Note: If the UBBCONFIG *RESOURCES Section and tlisten SSL settings are not in sync, the application will not boot.

If MIGRATE is specified, LAN should also be specified (except for the case where the configuration runs on a single multiprocessor computer).

The identifier NO_XA indicates that XA transactions are not allowed. The identifier NO_AA indicates that the auditing and authorization functions will not be called. This parameter is optional, and the default is no options.

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.

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.

`SSL_RENEGOTIATION` *numeric_value*

Specifies the renegotiation interval (in seconds) for SSL information. It must be greater than or equal to 0 and less than or equal to 2,147,483,647. The default value is 0.

`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.

`SCANUNIT` *numeric_value*

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 2 or 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 ($\text{SCANUNIT} * \text{SANITYSCAN}$) 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 ($\text{SCANUNIT} * \text{DBBLWAIT}$) 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 ($\text{SCANUNIT} * \text{BBLQUERY}$) 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 ($\text{SCANUNIT} * \text{BLOCKTIME}$) 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()`, 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.

`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..1023**]

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 1023 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..31**]

Specifies the variable in which the password for the principal specified in `SEC_PRINCIPAL_NAME` is stored. This parameter may contain a maximum of 31 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 applies only 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 applies only 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 applies only 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 applies only 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 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[2..256]` **(up to 64 bytes for BEA Tuxedo 8.0 or earlier)**

This is the absolute pathname of the file or device where the binary `TUXCONFIG` file is found on this machine. 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, the BEA Tuxedo filesystem 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 pathname specified for this parameter must match exactly (including case) the pathname specified for the `TUXCONFIG` environment variable. Otherwise, `tmloadcf(1)` cannot be run successfully.

`TUXDIR = string_value[2..256]` **(up to 78 bytes for BEA Tuxedo 8.0 or earlier)**

This is the absolute pathname of the directory where the BEA Tuxedo system software is found on this machine. This parameter must be specified for each machine and the pathname should be local to each machine; in other words, `TUXDIR` should not be on a remote filesystem. 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[2..256]` **(up to 78 bytes for BEA Tuxedo 8.0 or earlier)**

The value specified for this parameter is the absolute pathname of the application directory and is the current directory for all application and administrative servers booted on this machine. The absolute pathname can optionally be followed by a colon-separated list of other pathnames. In a configuration where `SECURITY` is set, each application must

have its own distinct APPDIR. See ENVFILE in the MACHINES section for a discussion of how this value is used in the environment.

Optional parameters are:

UID = *number*

Specifies the numeric user ID to be associated with the IPC structures created for the bulletin board. The valid range is 0-2147483647. If not specified, the default is the value specified in the RESOURCES section.

GID = *number*

Specifies the numeric group ID to be associated with the IPC structures created for the bulletin board. The valid range is 0-2147483647. If not specified, the default is the value specified in the RESOURCES section.

PERM = *number*

Specifies the numeric permissions associated with the IPC structures that implement the bulletin board. It is used to specify the read/write permissions for processes in the usual UNIX system fashion (that is, with an octal number such as 0600). The value can be between 0001 and 0777, inclusive. If not specified, the default is the value specified in the RESOURCES section.

BRTHREADS = {Y | N}

Specifies whether the Bridge process for this machine is configured for multithreaded execution (Y) or single-threaded execution (N). The default is N. This parameter applies only to applications running BEA Tuxedo 8.1 or later software.

Setting BRTHREADS to Y makes sense only if a machine has multiple CPUs. However, having multiple CPUs is not a prerequisite for setting BRTHREADS to Y.

Configurations with BRTHREADS set to Y on the local machine and BRTHREADS set (or defaulted) to N on the remote machine are allowed, but the throughput between the machines will not be greater than that for the single-threaded Bridge process.

A Bridge process configured for single-threaded or multithreaded execution can interoperate with a Bridge process running in an earlier release of BEA Tuxedo or WebLogic Enterprise: BEA Tuxedo release 8.0 or earlier, WebLogic Enterprise release 5.1 or earlier. In general, a threaded Bridge can interoperate with an unthreaded Bridge because there are no external functional or behavioral changes due to the threading.

Note: If BRTHREADS=Y and the Bridge environment contains TMNOTHEADS=Y, the Bridge starts up in threaded mode and logs a warning message to the effect that the Bridge is ignoring the TMNOTHEADS setting. The TMNOTHEADS environment variable was added to the BEA Tuxedo product in release 8.0.

`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`, `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 `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`

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`.

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***[0..256] (up to 64 bytes for BEA Tuxedo 8.0 or earlier)**

Specifies the BEA Tuxedo filesystem 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, the machine is assumed to not have a TLOG.

TLOGOFFSET = *offset*

Specifies the numeric offset in pages (from the beginning of the device) to the start of the BEA Tuxedo filesystem 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 filesystem. If not specified, the default is 100 pages.

ULOGPFX = *string_value***[0..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)**

Specifies the absolute pathname prefix of the path for the `userlog(3c)` message file on this machine. The value of ULOGPFX for a given machine is used to create the `userlog(3c)` message file for all servers, clients, and administrative processes executed on that machine. If this parameter is not specified, `$APPDIR/ULOG` is used. “mmddy” (month, day, year) is appended to the prefix to get the actual log filename.

TUXOFFSET = *offset*

Specifies the numeric offset in pages (from the beginning of the device) to the start of the BEA Tuxedo filesystem 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* [0..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

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 filename, no values are added to the environment. Lines must be of the form *ident=value* where *ident* begins with an underscore or alphabetic character, and contains only underscore or alphanumeric characters. Within the *value*, strings of the form $\${env}$ are expanded when the file is processed using variables already in the environment. (Forward referencing is not supported and if a value is not set, the variable is replaced with the empty string). Backslash (\) may be used to escape the dollar sign and itself. All other shell quoting and escape mechanisms are ignored and the expanded *value* is placed into the environment.

Client programs process only the MACHINES ENVFILE during `tpinit()`.

When booting servers, local servers inherit the environment of `tmboot(1)` and remote servers (not on the MASTER) inherit the environment of `tlisten(1)`. TUXCONFIG, TUXDIR, and APPDIR are also put into the environment when a server is booted based on the information in the associated MACHINES entry. An attempt to reset these three variables to another value will not be allowed and will result in a warning. `tmboot` and `tlisten` process the machine ENVFILE before starting the server, allowing for the environment to indicate necessary pathnames for finding executable and dynamically loaded files. Once the server is running, as part of server initialization (before the application gets control in `tpsvrinit()`), a server will read and export variables from both the machine and server ENVFILE files. If a variable is set in both the machine and server ENVFILE, the value in the server ENVFILE will override the value in the machine ENVFILE.

PATH and LD_LIBRARY_PATH are treated specially. Before a server is activated, the machine ENVFILE is scanned to find the first occurrence of a PATH or LD_LIBRARY_PATH variable; embedded environment variables within either PATH variable are not expanded. PATH and LD_LIBRARY_PATH are used to find pathnames for executable and dynamically loaded files. PATH will always be prefixed with:

```
 $\${APPDIR}:\${TUXDIR}/bin:/bin:$ 
```

if the value doesn't already begin with this string. This `PATH` will be used as a search path for servers that are specified with a simple or relative pathname. `LD_LIBRARY_PATH` will always be prefixed with:

```
${APPDIR}:${TUXDIR}/lib:/lib:/usr/lib:
```

if the value doesn't already begin with this string. `SHLIB_PATH` is set on HPUX and `LIBPATH` is set on AIX instead of `LD_LIBRARY_PATH`.

`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..1023]**

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 1023 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..31]`

Specifies the variable in which the password for the principal specified in `SEC_PRINCIPAL_NAME` is stored. This parameter may contain a maximum of 31 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 applies only 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 applies only 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.

`SICACHEENTRIESMAX = string_value`

represents the maximum number of service cache entries any process is to hold on this machine. It must be greater than or equal to 0 and less than 32,768. If a value is not specified, the default is 500. If the value is set to 0, no service caching will be performed by any process on this machine. The maximum value this attribute can take is 32,767. All clients on this machine will use this value.

Note: Unlike the corresponding attribute in the `SERVERS` section, this parameter cannot take the string `DEFAULT` as a valid value.

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[0..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)
```

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, 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*[0..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

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 filename, 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..1023]

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 1023 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..31]

Specifies the variable in which the password for the principal specified in `SEC_PRINCIPAL_NAME` is stored. This parameter may contain a maximum of 31 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 applies only 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 applies only 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

```
OPENINFO = "TUXEDO/QM: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 Windows, the separator after *qmconfig* must be a semicolon (;).

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/myubbcconfig
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 failback situations. If this entry describes `DEFAULTNET`, 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, 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*[0..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

Specifies the complete network listening address for the Bridge process on this *LMID*. 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 "0x*hex-digits*" or "\\x*hex-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:

```
//hostname: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 for this *LMID* to access the network. This value is required for participation in a networked application via a TLI-based BEA Tuxedo system binary. This parameter is not needed for sockets-based BEA Tuxedo system binaries.

NLSADDR = *string_value*[0..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

Specifies the network address used by the `tlisten(1)` process servicing the network for this *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*[0..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

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 attempts 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, 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 attempts 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, 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, 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 pathname for a server is given, the search for *AOUT* is done first in `APPDIR`, then in `TUXDIR/bin`, then in `/bin`, and then in *path*, where *path* is the value of the last `PATH=` line appearing in the machine environment file, if one exists. The values for `APPDIR` and `TUXDIR` are taken from the appropriate machine entry in the `TUXCONFIG` file. See `ENVFILE` in the `MACHINES` section for a more detailed discussion.

For BEA Tuxedo 8.1 or later, the maximum length of *AOUT* in the `SERVERS` section is 256 bytes. For BEA Tuxedo 8.0 or earlier, the maximum length of *AOUT* in the `SERVERS` section is 78 bytes.

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 bytes 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, 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. Servers with a `MIN=0` value will not be booted when `tmboot -y` is executed. This gives users in a multi-server environment the flexibility to boot servers as needed, and therefore reduce boot time.

`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[0..256]` (up to 78 bytes for BEA Tuxedo 8.0 or earlier)
 requests the addition of the values in this file to the environment of the server during its initialization. If a server is associated with a server group that can be migrated to a second machine, the `ENVFILE` must be in the same location on both machines.

Note that this file is processed after the server starts. Therefore, it cannot be used to set the pathnames used to find executable or dynamically loaded files needed to execute the server; use the machine `ENVFILE` instead. See `ENVFILE` in the `MACHINES` section for a discussion of how this file is used to modify the environment.

`CONV = {Y | N}`
 Specifies whether or not the server is a conversational server. Connections can only be made to conversational servers, and `rpc` requests (via `tpacall()` 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, 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`.

Note: The value of `REPLYQ` for conversational servers is always forced to `Y`, regardless of the value assigned to it in the `UBBCONFIG` file.

`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 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*[0..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

If `AOUT` is restartable, this parameter specifies the command that should be executed when `AOUT` abnormally terminates. The string, up to the first space or tab, must be the name of an executable UNIX file, either a full pathname or relative to `APPDIR` (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`, 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`

Specifies the size of the stack created for each dispatch thread in a multithreaded server. This value must be greater than or equal to 0 or less than or equal to 2147483647. The default is 0. This parameter has an effect on the server only when a value greater than 1 is specified for `MAXDISPATCHTHREADS`.

If this parameter is not specified or if the value specified is 0, a default thread stack size is used. If the value specified is bigger than 0 and less than a minimum thread stack size, the minimum thread stack size is used. If the value specified is bigger than that minimum thread stack size, value specified 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. The purpose of the minimum thread stack size is to allow customer to specify less thread stack size than Tuxedo default thread stack size. Currently, the BEA Tuxedo default thread stack size is 1,024,000, and minimum thread stack size is 100,000.

Note that if the thread stack size is exceeded, the server will core dump.

`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..1023]`

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 1023 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..31]`

Specifies the variable in which the password for the principal specified in `SEC_PRINCIPAL_NAME` is stored. This parameter may contain a maximum of 31 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.

`SICACHEENTRIESMAX = string_value`

if the string contains only numeric characters, than number specifies the maximum number of service cache entries this server can keep. It must be greater than or equal to 0 and less than 32,768. Otherwise, the string can take the value `DEFAULT`, in which case the number of services to cache will come from the `MACHINE` section entry that corresponds to this server. If a value is not specified, it will take the string `DEFAULT` as a valid value. A value of 0 implies that no service caching will be performed by any process on this machine. The maximum value this parameter can take is 32,767.

`CONCURR_STRATEGY=PER_REQUEST`

`CONCURR_STRATEGY = PER_OBJECT`

Use the `CONCURR_STRATEGY` parameter to specify the threading model to be used by a multithreaded CORBA server application. The `CONCURR_STRATEGY` parameter accepts either of the following values:

`CONCURR_STRATEGY = PER_REQUEST`

`CONCURR_STRATEGY = PER_OBJECT`

When you specify `CONCURR_STRATEGY = PER_REQUEST` to employ the *thread-per-request* model, each invocation on the CORBA server application is assigned to an arbitrary thread from the threads pool.

When you specify `CONCURR_STRATEGY = PER_OBJECT` to employ the *thread-per-object* model, each active object is associated with a single thread at any one time. Each request for an object establishes an association between a dispatch thread and the object.

Note: User-controlled concurrency takes precedence over threading model. Therefore, once user-controlled concurrency is chosen, the threading models behave the same so the behavior is consistent for instances of an object in the same process in multiple threads as it is for instances of an object in separate processes.

For a description of Parallel Objects, refer to “[Parallel Objects](#)” in the *BEA Tuxedo CORBA Programming Reference*.

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 32,767 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[. . .]]] . . . "*

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 accepts 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.
```

```
BUFTYPECONV = {XML2FML | XML2FML32}
```

When you boot a server with this parameter, the input buffer is converted from an XML type buffer to an FML/FML32 type buffer before being delivered to the service. When the [tpreturn\(3c\)](#) function is called, the FML/FML32 buffer is converted to XML before being returned back to the client.

```
ROUTING = string_value
```

Specifies the name of the routing criteria used for this service when doing data-dependent routing. The *string_value*, which is a *ROUTING_CRITERIA_NAME* defined in the ROUTING section, is the name of the routing criteria used for data-dependent routing for this service. 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.

```
BLOCKTIME numeric_value
```

Specifies the nontransactional client blocking time value, in seconds, per service indicating the minimum amount of time a blocking API call will delay before timing out for a particular service.

This parameter lets the client know that (after a specified time in seconds), no reply has been received by the server while the service request is still processing.

numeric_value can be between 0 and 32,767 inclusive. If not specified, the default is 0 which indicates that the system-wide BLOCKTIME value specified in the UBBCONFIG RESOURCES section is used for the service.

```
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 applies only 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 applies only 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.

INTERFACES Section

This section provides information for defining application-wide default parameters for CORBA interfaces used by the application. There are no required parameters for CORBA interfaces unless you are implementing factory-based routing, a feature that allows you to distribute processing to specific server groups. If you are implementing factory-based routing, you must specify the following parameters:

Table 64 Factory-based Routing Parameters

In this section...	You must specify...
INTERFACES	<ul style="list-style-type: none"> Names of the interfaces being used Names of the routing criteria that the system should apply to each interface
ROUTING	Routing criteria
GROUPS	Names of the server groups

For details about factory-based routing and the parameters associated with it, see [“ROUTING Section” on page 516](#).

You do not need to list any CORBA interfaces if you do not want to specify any parameters.

The following optional parameters are available.

`AUTOTRAN = {Y | N}`

Indicates that you want the system to automatically initiate a transaction on every operation invocation and end it upon return from the invocation. The `AUTOTRAN` parameter is only honored for interfaces that have the `optional` transaction policy. Otherwise, this parameter is ignored. The default is `N`.

The transactional policy is specified in an implementation configuration file. This transactional policy will become the transactional policy attribute of the associated `T_IFQUEUE` MIB object at run time.

Before setting the `AUTOTRAN` value, the system administrator must know the value of the transactional policy assigned to the interface by the programmer. Without knowing the policy, the administrator’s expectations of run-time `AUTOTRAN` functionality may be wrong.

If `AUTOTRAN` is set to `Y`, the `TRANTIME` parameter must also be set.

`FACTORYROUTING = criteria_name`

Required if you want to use a routing criteria when creating object references for this interface. The routing criteria is specified in the `ROUTING` section of the `UBBCONFIG` file.

`LOAD = number`

An arbitrary number between 1 and 100 that represents the relative load that the CORBA interface is expected to impose on the system. The numbering scheme is relative to the `LOAD` numbers assigned to other CORBA interfaces used by this application. The default

is 50. The value of `LOAD` is used in a CORBA environment to select the best machine to enqueue a request. The routing of the request causes the server's total load to be increased by the `LOAD` factor of the CORBA interface requested.

`PRIO = number`

Specifies the dequeuing priority number for all methods of the CORBA interface. The value must be greater than 0 and less than or equal to 100. 100 is the highest priority. The default is 50.

`SRVGRP = server-group-name`

Indicates that any parameter defined in this portion of the `INTERFACES` section applies to the interface within the specified server group. This feature lets you define, for a given CORBA interface, different parameter values in different server groups.

`TRANTIME = number`

The length of the time out (in seconds) for the transactions to be computed. If `AUTOTRAN` is set to `Y`, you must set the `TRANTIME` parameter. The value must be greater than or equal to zero and must not exceed 2147483647 ($2^{31} - 1$), or about 68 years. A value of 0 implies there is no time out for the transaction. (The default is 30 seconds.)

`TIMEOUT = number`

Indicates the amount of time (in seconds) to allow for processing of a method for this CORBA interface. The value must be greater than or equal to 0. A value of 0 indicates that the interface cannot time out. A timed-out method causes the server processing the method for the interface to terminate with a `SIGKILL` event. We recommend specifying a timeout value for the longest-running method for the interface.

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:

```
ROUTING_CRITERIA_NAME required_parameters
```

where `ROUTING_CRITERIA_NAME` is the (*string_value*) name assigned to the `ROUTING` parameter for a particular service entry in the `SERVICES` section. `ROUTING_CRITERIA_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 4096 characters). The format of string is a

comma-separated ordered list of range/group_name pairs; for example,
RANGES="0-2:DBG1,3-5:DBG2,6-9:"DBG3".

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 “*” (wildcard) in the position of a range indicates any values not covered by the other ranges previously seen in the entry; only one wildcard 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, 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 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(3): an optional sign, then a string of digits optionally containing a decimal point, then an optional e or E followed by an optional sign or space, followed by an integer.

The group name indicates the associated group to which the request is routed if the field matches the range. The meta-character “*” (wildcard) 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, 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[, ... ]]]..."
```

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 wildcard 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.

UBBCONFIG(5) Additional Information

Files

The TUXCONFIG and TUXOFFSET environment variables are used to find the TUXCONFIG configuration file on the MASTER machine.

Example

```
# The following configuration file defines a 2-site
# configuration with two machine types. Data-dependent
# routing is used.
*RESOURCES
IPCKEY      80952 # key for well known address
DOMAINID   My_Domain_Name
UID        4196 # user id for ipc structures
GID        601  # group id for ipc structures
PERM       0660 # permissions for ipc access
MAXSERVERS 20   # at most 20 simultaneous servers
MAXSERVICES 40  # offering at most 40 services
MAXGTT     20   # at most 20 simultaneous global transactions
MASTER     SITE1
SCANUNIT   10
SANITYSCAN 12
BBLQUERY   180
BLOCKTIME  30
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 Windows, :bankdb: becomes ;bankdb;
BANKB1    LMID=SITE1 GRPNO=1
           OPENINFO="TUXEDO/SQL:/usr2/apps/bank/bankd11:bankdb:readwrite"
# For Windows, :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
           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 ATMI 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 run-time 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

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

The type of the member, and is specified as one of the following: int, short, long, char, float, double, string, carray, mbstring, or dec_t; if type is '-', the type of the member is defaulted to the type of *fbyname* if the view is mapped to FML buffers.

Note: *mbstring* member type is supported by VIEW32 typed buffer only.

cname

The identifier for the structure member, and should be a valid C identifier name, since it is the name of a C structure member. Internally, the *cname* is truncated to 30 characters, so *cnames* must be unique within the first 30 characters. If the view is mapped to FML buffers, it cannot be a valid *fbyname*.

fbyname

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

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

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

The size of the member if the type is either string or carray; must be less than or equal to 65535. For 32-bit FML, the max size is 2 to the 32nd or several gazillion. For the dec_t type, *size* is two numbers separated by a comma, the first being the number of bytes in

the decimal value (it must be greater than 0 and less than 10) and the second being the number of decimal places to the right of the decimal point (it must be greater than 0 and less than two times the number of bytes minus one). For other field types, '-' should be specified, and the view compiler will compute the size.

null

The user-specified NULL value or '-' to indicate the default NULL value for that field; see below for a discussion of NULL values.

Flag Options

The following is a list of the options that can be specified as the `flag` element of a member description in a view description. Note that the `L` and `C` options generate additional structure members even for views that are not FML-based.

C

This option specifies that an additional structure member, called the associated count member (ACM), be generated, in addition to the structure member described in the member description (even for views that are not FML-based). When transferring data from a fielded buffer to a structure, each ACM in the structure is set to the number of occurrences transferred to the associated structure member. A value of 0 in an ACM indicates that no fields were transferred to the associated structure member; a positive value indicates the number of fields actually transferred to the structure member array; a negative value indicates that there were more fields in the buffer than could be transferred to the structure member array (the absolute value of the ACM equals the number of fields not transferred to the structure). During a transfer of data from a structure member array to a fielded buffer, the ACM is used to indicate the number of array elements that should be transferred. For example, if a member's ACM is set to N, 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, 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), 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.

Note: For MBSTRING field in VIEW32 buffer, the `viewc32(1)` command automatically adds the L option and the corresponding ALM is declared. The size of the MBSTRING data prepared by `Fmbpack32()` must be set in the ALM by the application and it is used for `Fmbunpack32()`.

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, carray, and mbstring 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, mbstring 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, carray, or mbstring 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 or mbstring, and the `NULL` value is of length `N`, the first `N` characters in the carray or mbstring 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 pathnames 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, its value is taken to be the current directory.

For `VIEW32`, the environment variable `VIEWFILES32` and `VIEWDIR32` are used.

Examples

```
# BEGINNING OF AN FML-BASED VIEWFILE
VIEW custdb
$/* This is a comment */
#
#type    cname    ffname 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   ffname 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 ATMI Application Using FML

WS_MIB(5)

Name

WS_MIB—Management Information Base for Workstation

Synopsis

```
#include <fm132.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. For additional information pertaining to all WS_MIB(5) class definitions, see “[WS_MIB\(5\) Additional Information](#)” on page 545.

WS_MIB(5) consists of the following classes.

Table 65 WS_MIB Classes

Class Name	Attributes
T_WSH	Workstation Handler
T_WSL	Workstation Listener

Each class description section has four subsections:

Overview

High level description of the attributes associated with the class.

Attribute Table

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

Attribute Semantics

Tells how each attribute should be interpreted.

Limitations

Limitations in the access to and interpretation of this class.

Attribute Table Format

As described above, each class that is a part of this MIB is defined below in four parts. One of these parts is the attribute table. The attribute table is a one-page reference guide to the attributes within a class and how they may 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 `udataobj/tpadm` relative to the root directory of the BEA Tuxedo system software installed on the system. The directory `#{TUXDIR}/udataobj` should be included by the application in the colon-separated list specified by the `FLDTBLDIR` environment variable and the field table name `tpadm` should be included in the comma-separated list specified by the `FIELDTBLS` environment variable.

Limitations

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

T_WSH Class Definition

Overview

The `T_WSH` class represents run-time attributes of WSH client processes. These attribute values characterize Workstation statistics specific to a particular WSH client process. This class is linked to the `T_WSL` class by the common key fields, `TA_SRVGRP` and `TA_SRVID`. It is also linked to the `T_CLIENT` class (see [TM_MIB\(5\)](#)) by the common key field `TA_WSHCLIENTID`.

Attribute Table

Table 66 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..256] ²	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
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

Table 66 WS_MIB(5): T_WSH Class Definition Attribute Table (Continued)

Attribute ¹	Type	Permissions	Values	Default
TA_NUMBLOCKQ	long	R--R--R--	0 <= num	N/A
TA_RCVDBYT	long	R--R--R--	0 <= num	N/A
TA_RCVDNUM	long	R--R--R--	0 <= num	N/A
TA_SENTBYT	long	R--R--R--	0 <= num	N/A
TA_SENTNUM	long	R--R--R--	0 <= num	N/A

(k)—GET key field
 (*)—GET/SET key, one or more required for SET operations

¹ All attributes in the T_WSH class are local attributes.

² Maximum string length for this attribute is 78 bytes for BEA Tuxedo 8.0 or earlier.

Attribute Semantics

TA_CLIENTID: *string*[1..78]

Client identifier for this WSH. The data in this field should not be interpreted directly by the end user except for equality comparison.

TA_WSHCLIENTID: *string*[1..78]

Client identifier for this WSH. The data in this field should not be interpreted directly by the end user except for equality comparison. This field can be used to link the WSH to its associated Workstation client T_CLIENT objects. This field value is always equal to the value for the TA_CLIENTID attribute for this class.

TA_SRVGRP: *string*[1..30]

Logical name of the server group for the associated WSL.

TA_SRVID: 1 <= num < 30,001

Unique (within the server group) server identification number for the associated WSL.

TA_STATE:

State for the WSH client within the application. Any state defined for the T_CLIENT class in [TM_MIB\(5\)](#) may be returned or set as indicated on that reference page. State changes to the SUSPended state are transitive to all clients associated with this WSH as is the resetting of a SUSPended WSH to ACTIVE. Additionally, SUSPended WSH clients will

not be assigned any additional incoming clients by the WSL. Note that the state of a WSH client may not be set to DEAD when accessing the T_CLIENT class; however, the state transition to DEAD is allowed via the T_WSH class and will result in all connections being handled by the 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..256**] (up to **78 bytes for BEA Tuxedo 8.0 or earlier**)

Network address of workstation handler. Hexadecimal addresses are converted to an ascii format with a leading "0x". TCP/IP addresses are reported in the "*//#. #. #. #:port*" format.

TA_HWCLIENTS: **1** <= *num* <**32,767**

High water number of clients accessing application via this WSH.

TA_MULTIPLEX: **1** <= *num* <**32,767**

Maximum number of clients that may access the application via this WSH.

TA_CURCLIENTS: **1** <= *num* <**32,767**

Current number of clients accessing application via this WSH.

TA_TIMELEFT: **0** <= *num*

A non-0 value for this attribute indicates that the WSH has been assigned a newly connecting Workstation client that has the indicated amount of time, in seconds, to complete the initialization process with the WSH.

TA_ACTIVE: {*Y* | *N*}

A value of *Y* indicates that the WSH is currently performing work on behalf of one of its associated Workstation clients. A value of *N* indicates that the WSH is currently waiting for work to perform on behalf of one of its associated Workstation clients.

TA_TOTACTTIME: **0** <= *num*

Time, in seconds, that the WSH has been active since it started processing.

TA_TOTIDLTIME: **0** <= *num*

Time, in seconds, that the WSH has been idle since it started processing.

TA_CURWORK: 0 <= *num*

Amount of work processed by this WSH since the last WSH assignment by the WSL. This value is used by the WSL to load balance new incoming connections amongst a set of WSH processes.

TA_FLOWCNT: 0 <= *num*

Number of times flow control has been encountered by this WSH. This attribute should be considered only in relation to recent past values as it may wrap around during the lifetime of the WSH.

TA_NUMBLOCKQ: 0 <= *num*

Number of times this WSH has been unable to enqueue a message to a local UNIX system message queue due to queue blocking conditions. This attribute should be considered only in relation to recent past values as it may wrap around during the lifetime of the WSH.

TA_RCVDBYT: 0 <= *num*

Number of bytes received from the network by this WSH from all of its present and past Workstation clients. This attribute should be considered only in relation to recent past values as it may wrap around during the lifetime of the WSH.

TA_RCVDNUM: 0 <= *num*

Number of BEA Tuxedo system messages received from the network by this WSH from all of its present and past Workstation clients. This attribute should be considered only in relation to recent past values as it may wrap around during the lifetime of the WSH.

TA_SENTBYT: 0 <= *num*

Number of bytes sent to the network by this WSH to all of its present and past Workstation clients. This attribute should be considered only in relation to recent past values as it may wrap around during the lifetime of the WSH.

TA_SENTNUM: 0 <= *num*

Number of BEA Tuxedo system messages sent to the network by this WSH to all of its present and past Workstation clients. This attribute should be considered only in relation to recent past values as it may wrap around during the lifetime of the WSH.

Limitations

This class represents a specialization of the `T_CLIENT` class and as such represents certain attributes that are duplicated in the corresponding `T_CLIENT` objects. Attributes not listed that are included in the `T_CLIENT` class must be accessed via that class and are not available through the `T_WSH` class.

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

Table 67 WS_MIB(5): T_WSL Class Definition Attribute Table

Attribute	Type	Permissions	Values	Default
TA_SRVGRP(r)(*)	string	ru-r--r--	<i>string</i> [1..30]	N/A
TA_SRVID(r)(*)	long	ru-r--r--	1 <= num < 30,001	N/A
TA_GRPNO(k)	long	r--r--r--	1 <= num < 30,001	N/A
TA_STATE(k)	string	rwxr-xr--	See T_SERVER Class in TM_MIB (5)	
TA_LMID(k)	string	R--R--R--	LMID	N/A
TA_PID(k)	long	R--R--R--	1 <= num	N/A
TA_DEVICE	string	rw-r--r--	<i>string</i> [0..78]	N/A
TA_NADDR(r)	string	rw-r--r--	<i>string</i> [1..256] ³	N/A
TA_EXT_NADDR	string	rw-r--r--	<i>string</i> [0..78]	" "
TA_WSHNAME	string	rw-r--r--	<i>string</i> [1..78]	"WSH"
TA_MINHANDLERS	long	rwxr-xr--	0 <= num < 256	0
TA_MAXHANDLERS	long	rw-r--r--	0 <= num < 32,767	See note ¹
TA_MULTIPLEX	long	rw-r--r--	1 <= num < 32,767	10
TA_MINENCRYPTBITS	string	rwxrwx---	"{ 0 40 56 128 256 }" ²	"0"

Table 67 WS_MIB(5): T_WSL Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
TA_MAXENCRYPTBITS	string	rw-rwx---	"{0 40 56 128 256}" ²	"128"
TA_CERTIFICATE_AUTHENTICATION	string	rw-r--r--	"{Y N}"	"N"
TA_SECUREPORT	long	rw-r--r--	0 <= num < 32767	0
TA_SSL_RENEGOTIATION	long	rw-r--r--	0 <= num < 35791394	0
TA_MINWSHPORT	long	rw-r-xr--	0 <= num < 65,535	2048
TA_MAXWSHPORT	long	rw-r--r--	0 <= num < 65,535	65,535
TA_MAXIDLETIME	long	rw-r-xr--	0 <= num < 35,204,650	35,204,649
TA_MAXINITTIME	long	rw-r-xr--	1 <= num < 32,767	60
TA_CMPLIMIT	string	rw-r-xr--	<i>threshold</i>	MAXLONG
TA_CLOPT	string	rw-r--r--	<i>string</i> [0..128]	"-A"
TA_ENVFILE	string	rw-r--r--	<i>string</i> [0..256] ³	" "
TA_GRACE	long	rw-r--r--	0 <= num	0
TA_KEEPA_LIVE	string	rw-r-xr--	"{client handler both none}"	"none"
TA_MAXGEN	long	rw-r--r--	0 <= num < 256	1
TA_NETTIMEOUT	long	rw-r-xr--	0 <= num <= MAXLONG	0
TA_RCMD	string	rw-r--r--	<i>string</i> [0..256] ³	" "
TA_RESTART	string	rw-r--r--	"{Y N}"	"Y"
TA_SEQUENCE(k)	long	rw-r--r--	1 <= num < 10,000	>= 10,000
T_WSL Class: Local Attributes				
TA_CURHANDLERS	long	R--R--R--	0 <= num	N/A

Table 67 WS_MIB(5): T_WSL Class Definition Attribute Table (Continued)

Attribute	Type	Permissions	Values	Default
TA_HWHANDLERS	long	R--R--R--	0 <= num	N/A
TA_WSPROTO	long	R--R--R--	0 <= num	N/A
TA_SUSPENDED	string	R-XR-XR--	"{NEW ALL NONE}"	N/A
TA_VIEWREFRESH	string	--X--X---	"Y"	N/A

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

¹ If a value for this attribute is not specified at the time the object is created, 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 Link-level encryption value of 40 bits is provided for backward compatibility.

³ Maximum string length for this attribute is 78 bytes for BEA Tuxedo 8.0 or earlier.

Attribute Semantics

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

Logical name of the server group. Server group names cannot contain an asterisk (*), comma, or colon.

TA_SRVID: **1 <= num < 30,001**

Unique (within the server group) server identification number.

TA_GRPNO: **1 <= num < 30,001**

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: *LMID*

Current logical machine on which the server is running.

TA_PID: 1 = num

Native operating system process identifier for the WSL server. Note that this may not be a unique attribute since servers may be located on different machines allowing for duplication of process identifiers.

TA_DEVICE: string[0..78]

Device name to be used by the WSL process to access the network. This attribute is optional.

TA_NADDR: string[1..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

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 "0x*hex-digits*" or "\\x*hex-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:

```
//hostname: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 is 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 0x00021111fffff 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 <= *num* < 256

The minimum number of handlers that should be available in conjunction with this WSL at any given time. The WSL will start this many WSHs immediately upon being activated and will not deplete the supply of WSHs below this number until the administrator issues a shutdown to the WSL. Modifications to this attribute for a running WSL may cause additional handlers to be activated.

TA_MAXHANDLERS: 0 <= *num* < 32,767

The maximum number of handlers that should be available in conjunction with this WSL at any given time. Handlers are started as necessary to meet the demand of Workstation clients attempting to access the system. This attribute must be greater than or equal to the setting for the minimum number of handlers.

TA_MULTIPLEX: 1 <= *num* < 32,767

Maximum number of clients that are to be supported by any one handler process concurrently.

TA_MINENCRYPTBITS: { 0 | 40 | 56 | 128 | 256 }

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 | 256 }

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_CERTIFICATE_AUTHENTICATION:** "{Y | N}"
 Specifies that certificate-based authentication should be enabled when accepting an SSL connection from a remote application. the default value is N.
- TA_SECUREPORT:** 0 <= *num* < 32,767
 Specifies the port number that WS_MIB should use to listen for secure connections using the SSL protocol. The default value is 0 (which indicates that no secure port is in use).
- TA_SSL_RENEGOTIATION:** 0 <= *num* < 35,791,394
 Specifies the renegotiation interval in minutes. If a connection does not have a renegotiation in the specified interval, WS_MIB will request that the client renegotiate the session for inbound connections or actually perform the renegotiation in the case of outbound connections. The default is 0 minutes which results in no periodic session renegotiations.
- TA_MINWSHPORT:** 0 <= *num* < 65,535
 The lower end of the range of available port numbers that may be allocated to WSH processes by this listener.
- TA_MAXWSHPORT:** 0 <= *num* < 65,535
 The upper end of the range of available port numbers that may be allocated to WSH processes by this listener.
- TA_MAXIDLETIME:** 0 <= *num* < 35,204,650
 Maximum amount of time, in minutes, that a Workstation client is permitted to be idle before it will be abortively disconnected from the application by the handler. A value of 35,204,650 allows clients to be idle as long as they wish without being timed out. A value of 0 indicates clients may be terminated after any period of inactivity greater than 1 second.
- TA_MAXINITTIME:** 1 <= *num* < 32,767
 The minimum number of seconds that should be allowed for a Workstation client to complete initialization processing through the WSH before being timed out by the WSL.
- TA_CMPLIMIT:** *threshold*
 Threshold message size at which compression will occur for traffic to and from Workstation clients. The *threshold* value may be either non-negative numeric values or 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..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

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_KEEPAALIVE: "{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..256] (up to 78 bytes for BEA Tuxedo 8.0 or earlier)

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), this attribute must be set to "Y".

TA_SEQUENCE: 1 <= num < 10,000

Specifies when this server should be booted (`tmboot(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 `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_CURHANDLERS: 0 <= num

Number of currently active handlers associated with this WSL.

TA_HWHANDLERS: 0 <= num

Maximum number of currently active handlers associated with this WSL at any one time.

TA_WSPROTO: 0 <= num

The BEA Tuxedo 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.

WS_MIB(5) Additional Information

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, 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, 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 log off.

```

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 ATMI Application Using C

Programming a BEA Tuxedo ATMI Application Using FML

WSL(5)

Name

WSL—Workstation Listener server

Synopsis

```
WSL SRVGRP="identifier"
  SRVID="number"
  CLOPT="[-A] [servopts options] -- -n netaddr [-d device]
        [-w WSHname] [-t timeout-factor] [-T Client-timeout]
        [-m minh] [-M maxh] [-x mpx-factor]
        [-p minwshport] [-P maxwshport] [-I init-timeout]
        [-c compression-threshold] [-k compression-threshold]
        [-K {client|handler|both|none}]
        [-z bits] [-Z bits] [-H external-netaddr][-N network-timeout]
        [-U inbound-message-size-limit-in-bytes]
        [-a] [-v{detect|warn|none}] [-R renegotiation_interval]
        [-S secure_port]"
```

Description

The workstation listener is a BEA Tuxedo system-supplied server that enables access to native services by Workstation clients. The application administrator enables workstation access to the application by specifying the workstation listener server as an application server in the `SERVERS` section. The associated command-line options are used to parameterize the processing of the workstation listener and workstation handlers.

The location, server group, server ID, and other generic server related parameters are associated with the workstation listener using the already defined configuration file mechanisms for servers. Workstation listener specific command-line options are specified to allow for customization.

Each WSL booted as part of an application facilitates application access for a large number of Workstation clients by providing access via a single well known network address to a set of workstation handlers (WSHs) acting as surrogate clients for the users running on the workstations. The WSHs are started and stopped dynamically by the WSL as necessary to meet the incoming load from the application workstations. The advantages to the application administrator are that a small number of native site processes (WSHs) can support a much larger number of clients, thus reducing the process count on the native site, and that the native site does

not need to incur the overhead of maintaining bulletin board information on the workstation sites, which may be quite numerous.

The following WSL-specific command-line options are available and may be listed after the double-dash (--) in the CLOPT parameter.

`-n netaddr`

Specifies the complete network address to be used by the WSL process as its listening address. 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 "\\x*hex-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:

```
//hostname: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 greater than or equal to *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 are compressed if they are larger than the given value. The default for this parameter is 2147483647, which means no compression is done since the legal range is between 0 and 2147483647.

`[-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 WSL/WSH 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 | 256]]`

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, 128, and 256 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 56-bit or 128/256-bit) is installed.

Note: The link-level encryption value of 40 bits is provided for backward compatibility. 256-bit encryption is currently possible only when using SSL.

`[-z [0 | 40 | 56 | 128|256]]`

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, 128, and 256 specify the length (in bits) of the encryption key.

The default is 128 for LLE and 256 for SSL. This option is available only if BEA Tuxedo Security (either 56-bit or 128/256-bit) is installed.

Note: The link-level encryption value of 40 bits is provided for backward compatibility. 256-bit encryption is currently possible only when using SSL.

[-H *external-netaddr*]

Specifies the complete network address to be used as a well known address template of the WSH process. The address is 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 0x00021111fffff then the well known network address will be 0x00021111ddd 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 timeout; this is the default. Note: setting this value too low may cause an unacceptably high number of disconnects.

[-U *inbound-message-size-limit-in-bytes*]

This option specifies the upper size limit of incoming network message for WSH. The message size includes internal data items for Tuxedo (should be less than 1024 bytes) and user data. The limit also takes effect when message is compressed, i.e., it also checks the original message size.

[-a]

Specifies that certificate-based authentication should be required when accepting an SSL connection from a remote application.

[-v {*detect/warn/none*}]

Determines how the WSL will behave when a digital certificate for a peer of an outbound connection initiated by the BEA object request broker (ORB) is received as part of the Secure Sockets Layer (SSL) protocol handshake. The validation is only performed by the initiator of a secure connection and confirms that the peer server is actually located at the same network address as specified by the domain name in the server's digital certificate.

This validation is not technically part of the SSL protocol but is similar to the check done in web browsers.

A `detect` value causes the BEA Tuxedo to verify that the host specified in the object reference used to make the connection matches the domain name specified in the peer server's digital certificate. If the comparison fails, the BEA Tuxedo refuses to authenticate the peer and drops the connection. The `detect` value is the default value.

A `warn` value causes the BEA Tuxedo to verify that the host specified in the object reference used to make the connection matches the domain name specified in the peer's digital certificate. If the comparison fails, the BEA Tuxedo logs a message to the user log but continues to process the connection.

A `none` value causes the BEA Tuxedo to not perform the peer validation and to continue to process the connection.

The `-v` parameter is only available if licenses for SSL or LLE (link level encryption) are installed.

`[-R renegotiation-interval]`

Specifies the renegotiation interval in minutes. If a connection does not have a renegotiation in the specified interval, the WSL will request that the client renegotiate the session for inbound connections or actually perform the renegotiation in the case of outbound connections. The default is 0 minutes which results in no periodic session renegotiations.

Note: If the `-R` parameter is specified and the `-S` parameter is not specified or set to 0, the WSL sends a warning message to the userlog.

`[-S secure-port]`

Specifies the port number that the WSL should use to listen for secure connections using the SSL protocol. You can configure the WSL to allow only secure connections by setting the port numbers specified by the `-s` and `-n` options to the same value. The value must be greater than or equal to 0 and less than or equal to 32767. The default value is 0.

Note: If the `-R` parameter is specified and the `-S` parameter is not specified or set to 0, the WSL sends a warning message to the userlog.

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 system-supplied server on all supported server platforms.

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"
WSL   SRVGRP=GROUP1 SRVID=1
  CLOPT="-A -r -- -n //hostname:port -m1 -M10 -x20 -U 2048"
  # size limit set to 2048 bytes.
```

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 ATMI Application Using C

