



BEA Tuxedo

Administering a BEA Tuxedo Application at Run Time

BEA Tuxedo Release 7.1
Document Edition 7.1
May 2000

Copyright

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

Restricted Rights Legend

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

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

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

Trademarks or Service Marks

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

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

Administering a BEA Tuxedo Application at Run Time

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

Contents

1. Starting Up and Shutting Down an Application

The Tasks Involved in Starting Up and Shutting Down an Application	1-2
How to Set Your Environment	1-2
How to Create the TUXCONFIG File.....	1-4
How to Propagate the BEA Tuxedo System Software	1-5
How to Create a TLOG Device	1-6
How to Start tlisten at All Sites	1-7
tlisten Command Options	1-7
How to Boot the Application.....	1-9
Sequence of tmboot Tasks for a 2-Machine Configuration	1-10
Sequence of tmboot Tasks for Large Applications (Over 50 Machines) .	1-11
How to Shut Down Your Application	1-11
Running tmsshutdown	1-12
Using the IPC Tool When an Application Fails to Shut Down Properly.	1-13

2. Monitoring Your BEA Tuxedo Application

Ways to Monitor Your Application.....	2-2
System and Application Data that You Can Monitor.....	2-4
Monitoring System Data	2-4
Monitoring Dynamic and Static Administrative Data.....	2-5
Common Startup and Shutdown Problems.....	2-7
Common Startup Problems	2-7
Common Shutdown Problems.....	2-8
Selecting Appropriate Monitoring Tools.....	2-8
Using the BEA Administration Console to Monitor Your Application	2-9
Using the Power Bar to Monitor Activities.....	2-9

Using Command-line Utilities to Monitor Your Application	2-10
Inspecting Your Configuration Using tadmin	2-10
Generating Reports on Servers and Services Using txrpt	2-11
How a tadmin Session Works.....	2-13
Monitoring Your System Using tadmin Commands	2-14
Using EventBroker to Monitor Your Application	2-15
Using Log Files to Monitor Activity	2-16
What Is the Transaction Log (TLOG)	2-17
What Is the User Log (ULOG)	2-17
Detecting Errors Using Logs	2-18
Analyzing tlisten Messages in the ULOG	2-18
Analyzing the Transaction Log (TLOG)	2-19
Analyzing the User Log (ULOG).....	2-20
Estimating Service Workload Using the Application Service Log	2-21
Using the MIB to Monitor Your Application	2-22
Limiting Your MIB Queries	2-22
Querying Global and Local Data.....	2-22
Using tadmincall to Access Information	2-23
Querying and Updating the MIB with ud32	2-24
Using the Run-time Tracing Utility	2-25
Managing Errors Using the DBBL and BBLs	2-26
Using the ATMI to Handle System and Application Errors	2-28
Using Configurable Timeout Mechanisms.....	2-28
Configuring Redundant Servers to Handle Failures.....	2-29
Monitoring Multithreaded and Multicontexted Applications.....	2-30
How to Retrieve Data About a Multithreaded/Multicontexted Application Using the MIB	2-31

3. Dynamically Modifying an Application

Dynamic Modification Methods.....	3-1
Tools for Modifying Your Application	3-2
Using tmconfig to Make Permanent Changes to Your Configuration	3-4
How tmconfig Works	3-5
How Results of a tmconfig Task Are Displayed	3-9

How to Run tmconfig	3-11
How to Set Environment Variables for tmconfig.....	3-11
How to Conduct a tmconfig Walk-through Session.....	3-12
tmconfig Input Buffer Considerations.....	3-14
Making Temporary Modifications to Your Configuration with tmconfig	3-15
How to Add a New Machine	3-16
How to Add a Server	3-19
How to Activate a Newly Configured Machine	3-21
How to Add a New Group.....	3-24
How to Change Data-dependent Routing (DDR) for an Application	3-25
How to Change Application-wide Parameters	3-26
How to Change an Application Password.....	3-29
Limitations on Dynamic Modification Using tmconfig	3-31
Tasks that Cannot Be Performed on a Running System	3-32
Making Temporary Modifications to Your Configuration with tmadmin	3-33
How to Set Environment Variables for tmadmin.....	3-34
How to Suspend Services or Servers.....	3-34
How to Resume Services or Servers	3-35
How to Advertise Services or Servers.....	3-35
How to Unadvertise Services or Servers	3-36
How to Change Service Parameters	3-36
How to Change the Timeout Value	3-37

4. Managing the Network in a Distributed Application

Running a Network for a Distributed Application	4-1
Compressing Data Over a Network.....	4-2
How to Set the Compression Level.....	4-2
Selecting Data Compression Thresholds.....	4-3
Balancing Network Request Loads	4-4
How to Use Data-Dependent Routing	4-5
Example of Data-dependent Routing with a Horizontally-partitioned Database	4-6
Example of Data-dependent Routing with Rule-based Servers	4-7
How to Change Your Network Configuration	4-8

5. About the EventBroker

What Is an Event.....	5-1
Differences Between Application-defined and System-defined Events.....	5-2
What Is the EventBroker	5-3
How the EventBroker Works	5-4
Event Notification Methods	5-5
Severity Levels of System Events	5-6
What Are the Benefits of Brokered Events	5-6

6. Subscribing to Events

Process of Using the EventBroker	6-1
How to Configure EventBroker Servers.....	6-2
How to Set the Polling Interval	6-3
Subscribing, Posting, and Unsubscribing to Events with the ATMI and the EVENT_MIB	6-3
Identifying Event Categories Using eventexpr and filter	6-4
Accessing the EventBroker	6-5
How to Select a Notification Method	6-6
How to Cancel a Subscription to an Event	6-8
How to Use the EventBroker with Transactions	6-8
How Transactions Work with the EventBroker	6-9

7. Migrating Your Application

What Is Migration.....	7-1
Performing a Master Migration	7-2
Migrating a Server Group.....	7-3
Migrating Machines.....	7-3
Performing a Scheduled Migration	7-4
Migration Options.....	7-5
How to Switch the Master and Backup Machines.....	7-6
Examples of Switching MASTER and BACKUP Machines	7-6
How to Migrate Server Groups	7-7
How to Migrate a Server Group When the Alternate Machine Is Accessible from the Primary Machine	7-8

How to Migrate a Server Group When the Alternate Machine Is Not Accessible from the Primary Machine	7-9
Examples of Migrating a Server Group	7-9
How to Migrate Server Groups from One Machine to Another.....	7-11
How to Migrate Machines When the Alternate Machine Is Accessible from the Primary Machine	7-11
How to Migrate Machines When the Alternate Machine Is Not Accessible from the Primary Machine	7-12
Examples of Migrating a Machine	7-13
How to Cancel a Migration	7-14
Example of a Migration Cancellation	7-14
How to Migrate Transaction Logs to a Backup Machine.....	7-15

8. Tuning Your Application

Maximizing Your Application Resources	8-2
When to Use MSSQ Sets.....	8-2
How to Enable Load Balancing.....	8-4
How to Measure Service Performance Time.....	8-5
How to Assign Priorities to Interfaces or Services.....	8-5
Example of Using Priorities	8-6
Using the PRIO Parameter to Enhance Performance.....	8-6
Bundling Services into Servers	8-7
When to Bundle Services	8-7
Enhancing Efficiency with Application Parameters.....	8-8
Setting the MAXACCESSERS, MAXSERVERS, MAXINTERFACES, and MAXSERVICES Parameters	8-8
Setting the MAXGTT, MAXBUFTYPE, and MAXBUFSTY Parameters	8-9
Tuning with the SANITYSCAN, BLOCKTIME, BBLQUERY, and DBBLWAIT Parameters.....	8-9
Recommended Values for Tuning-related Parameters.....	8-10
Determining Your System IPC Requirements	8-10
Measuring System Traffic	8-12
Example of Detecting a System Bottleneck.....	8-13
Detecting Bottlenecks on UNIX Platforms	8-13
Detecting Bottlenecks on Windows NT Platforms	8-15

9. Troubleshooting a BEA Tuxedo Application

Determining Types of Failures	9-2
How to Determine the Cause of an Application Failure	9-2
How to Determine the Cause of a BEA Tuxedo System Failure	9-3
How to Broadcast an Unsolicited Message	9-4
Maintaining Your System Files	9-5
How to Print the Universal Device List (UDL).....	9-5
How to Print VTOC Information	9-6
How to Reinitialize a Device.....	9-6
How to Create a Device List	9-7
How to Destroy a Device List	9-7
Repairing Partitioned Networks	9-8
Detecting a Partitioned Network	9-8
Restoring a Network Connection	9-10
Restoring Failed Machines	9-11
How to Restore a Failed MASTER Machine	9-11
How to Restore a Failed Nonmaster Machine.....	9-12
How to Replace System Components	9-13
How to Replace Application Components	9-13
Cleaning Up and Restarting Servers Manually	9-14
How to Clean Up Resources Associated with Dead Processes.....	9-14
How to Clean Up Other Resources	9-15
Aborting or Committing Transactions.....	9-15
How to Abort a Transaction	9-15
How to Commit a Transaction	9-16
How to Recover from Failures When Transactions Are Used	9-17
How to Use the IPC Tool When an Application Fails to Shut Down Properly	9-18
Troubleshooting Multithreaded/ Multicontexted Applications	9-19
Debugging Multithreaded/Multicontexted Applications.....	9-19
Limitations of Protected Mode in a Multithreaded Application	9-19

1 Starting Up and Shutting Down an Application

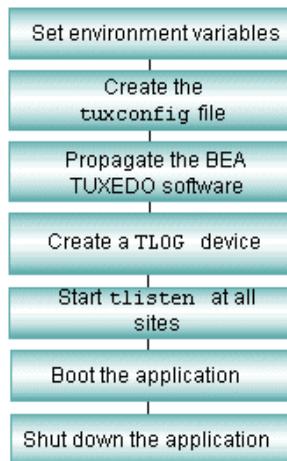
- The Tasks Involved in Starting Up and Shutting Down an Application
- How to Set Your Environment
- How to Create the TUXCONFIG File
- How to Propagate the BEA Tuxedo System Software
- How to Create a TLOG Device
- How to Start tlisten at All Sites
- How to Boot the Application
- How to Shut Down Your Application

The Tasks Involved in Starting Up and Shutting Down an Application

The following flowchart illustrates the tasks required to start up and shut down your BEA Tuxedo application.

Click on each of the following tasks for instructions on completing that task.

Figure 1-1 Startup and Shutdown Tasks



How to Set Your Environment

Being able to access the BEA Tuxedo executables and data libraries is essential to the job of managing a BEA Tuxedo application. For example, the commands needed to start up or shut down an application are located in `$TUXDIR/bin` on a UNIX host machine, and in `%TUXDIR%\bin` on a Windows NT host machine.

On a UNIX host machine, set and export the following environment variables to set up your environment.

```
TUXCONFIG=path_name_of_TUXCONFIG_file
TUXDIR=path_name_of_BEA_Tuxedo_system_root_directory
APPDIR=path_name_of_BEA_Tuxedo_application_root_directory
PATH=$APPDIR:$TUXDIR/bin:/bin:$PATH
LD_LIBRARY_PATH=$APPDIR:$TUXDIR/lib:/lib:/usr/lib:$LD_LIBRARY_PATH
export TUXCONFIG TUXDIR APPDIR PATH LD_LIBRARY_PATH
```

On this platform . . .	Make this change . . .
-------------------------------	-------------------------------

SunOS	Add /usr/5bin as the first directory in your PATH, as shown in the following example: PATH=/usr/5bin:\$APPDIR:\$TUXDIR/bin:/bin:\$PATH
-------	---

AIX on the RS/6000	Use LIBPATH instead of LD_LIBRARY_PATH
--------------------	--

HP-UX on the HP 9000	Use SHLIB_PATH instead of LD_LIBRARY_PATH
----------------------	---

Replace the substitutable strings (italicized) with the absolute path names appropriate for your installation.

Note: The application administrator defines the TUXCONFIG, TUXDIR, and APPDIR environment variables in the MACHINES section of the UBBCONFIG file or the T_MACHINE class of the TM_MIB for each machine in an application. See the UBBCONFIG(5) or TM_MIB(5) reference page for a description of these environment variables.

On a Windows NT host machine, enter the following commands at the MS-DOS prompt to set up your environment.

```
set TUXCONFIG=path_name_of_TUXCONFIG_file
set TUXDIR=path_name_of_BEA_Tuxedo_system_root_directory
set APPDIR=path_name_of_BEA_Tuxedo_application_root_directory
set PATH=%APPDIR%;%TUXDIR%\bin;%PATH%
```

Replace the substitutable strings (italicized) with the absolute path names appropriate for your installation.

Windows NT accesses the required dynamically loadable library files through its PATH variable setting. Specifically, Windows NT searches for dynamically loadable library files in the following order:

1. The directory from which the BEA Tuxedo application was loaded
2. The current directory
3. The Windows system directory (for example, C:\winnt\System32)
4. The Windows directory (for example, C:\winnt)
5. The directories listed in the `PATH` environment variable

How to Create the TUXCONFIG File

The `tmloadcf(1)` command converts the text configuration file to a binary file called `tuxconfig` and writes the new file to the location given in the `TUXCONFIG` variable. Run the command as follows.

```
$ tmloadcf [-n] [-y] [-c] [-b blocks] {ubconfig_file | - }
```

Note: You must be logged in on the `MASTER` machine and have the effective user ID of the configuration file owner.

The options shown here perform the following functions:

- `-n` performs a syntax check only; reports errors.
- `-y` overwrites the existing `TUXCONFIG` file without asking.
- `-c` calculates minimum interprocess communication (IPC) resources of the configuration.
- `-b` limits the size of the `TUXCONFIG` file.

The `-c` and `-n` options do not load the `TUXCONFIG` file. IPC resources are platform specific. If you use the `-c` option, check the data sheet for your platform in the *BEA Tuxedo Installation Guide* to judge whether you must make changes. If you do want to change IPC resources, check the administration documentation for your platform. If the `-n` option checks for syntax errors in the configuration file, correct the errors before you proceed. (For `ubconfig_file`, substitute the fully qualified name of your configuration file.)

The `-b` option takes an argument that limits the number of blocks used to store the `TUXCONFIG` file. Use it if you are installing `TUXCONFIG` on a raw disk device that has not been initialized. The option is not recommended if `TUXCONFIG` is stored in a regular UNIX system file.

How to Propagate the BEA Tuxedo System Software

TUXCONFIG is propagated automatically to all machines in your configuration by the BEA Tuxedo system when you run `tmbboot(1)`. There are, however, other files that you need to propagate manually. Following is a list of the files and directories that you need to create for a networked application. First, install the BEA Tuxedo system on the machine.

Figure 1-2 Directories and Files to Propagate

Directory/File	Description
APPDIR	You must create the directory named in the APPDIR variable must be created on each node. It is easier if this directory has the same path name on all nodes.
Executables	You must build one set of application servers for each platform, and manually propagate the appropriate set to all machines running on each platform (that is, the BEA Tuxedo system does not do this automatically). Store the executables in APPDIR, or in a directory specified in a PATH variable in ENVFILES in the MACHINES section of your configuration file.
Field tables VIEW tables	If FML or VIEWS buffer types are used, field tables and VIEW description files must be manually propagated to the machines where they are used, and then recompiled. Use <code>mkfldhdr</code> , <code>mkfldhdr32(1)</code> to make a header file out of a field table file; use <code>viewc</code> , <code>viewc32(1)</code> to compile a VIEW file. The FML field tables and VIEW description files should be available through the environment variables FLDTBLDIR, FIELDTBLS, VIEWDIR, and VIEWFILES, or their 32-bit equivalents.
tlisten	The tlisten process must be started on each machine of a networked BEA Tuxedo application before the application is booted. Refer to the <code>tlisten(1)</code> reference page. You must define TUXDIR, TUXCONFIG, APPDIR, and other relevant environment variables before starting tlisten.

How to Create a TLOG Device

To create distributed transaction processing, you must have create a global transaction log (TLOG) on each participating machine. To define a TLOG, complete the following steps.

1. You must first set several parameters in the `MACHINES` section of the configuration file: `TLOGDEVICE`, `TLOGOFFSET`, `TLOGNAME`, and `TLOGSIZE`.
2. You must also create a universal device list entry (UDL) for the `TLOGDEVICE` on each participating machine. (You can do this task before or after loading `tuxconfig`, but you must do so before booting the system.) To create an entry in the UDL for the TLOG device, invoke `tmadmin -c` on the MASTER machine with the application inactive. (The `-c` option invokes `tmadmin` in configuration mode.)
3. Enter the command:

```
crdl -z config -b blocks
```

where `-z config` specifies the full path name for the device on which the UDL should be created (that is, where the TLOG will reside) and `-b blocks` specifies the number of blocks to be allocated on the device. The value of `config` should match the value of the `TLOGDEVICE` parameter in the `MACHINES` section. The blocks must be larger than the `TLOGSIZE`. If `-z` is not specified, the value of `config` defaults to the value of the variable `FSCONFIG` (which points to the application's databases).

4. Repeat Steps 1 and 2 on each machine in your application that will use global transactions.

If the `TLOGDEVICE` is mirrored between two machines, Step 4 is not required on the paired machine. To be recoverable, the TLOG should reside on a device that can be mirrored. Because the TLOG is too small (typically, 100 pages) to warrant the allocation of a whole disk partition, the TLOG is commonly stored on the same raw disk slice as the BEA Tuxedo /Q database.

How to Start tlisten at All Sites

For a networked application, a listener process must be running on each machine. A networked application is an application that runs on more than one machine, as established by the `MODEL MP` parameter in the `RESOURCES` section of the application's `UBBCONFIG` file.

Note: You must define `TUXDIR`, `TUXCONFIG`, `APPDIR`, and other relevant environment variables before starting `tlisten`.

The port on which the process is listening must be the same as the port specified for `NLSADDR` in the `NETWORK` section of the configuration file. On each machine, use the `tlisten(1)` command, as follows.

```
tlisten [ -d device ] -l nlsaddr [-u {uid-# | uid-name}] [ -z bits ] [ -Z bits ]
```

tlisten Command Options

- `-d device`—The full path name of the network device. For BEA Tuxedo Release 6.4 or above, this option is not required. For earlier versions of the BEA Tuxedo system (up to Release 6.3), some network providers (for example, TCP/IP) require this information.
- `-l nlsaddr`—Network address at which the process listens for connections. TCP/IP addresses may be specified in the following forms:

```
"/hostname:port_number"
```

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

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

In the second example, the `#. #. #. #` is in dotted decimal format. In dotted decimal format, each `#` should be a number from 0 to 255. This dotted decimal number represents the IP address of the local machine. In both of the above formats, `port_number` is the TCP port number at which the `tlisten` process

1 Starting Up and Shutting Down an Application

listens for incoming requests. `port_number` can either be a number between 0 and 65535 or a name. If `port_number` is a name, then it must be found in the network services database on your local machine. The address can also be specified in hexadecimal format when preceded by the characters `0x`. Each character after the initial `0x` is a number between 0 and 9 or a letter between A and F (case insensitive). The hexadecimal format is useful for arbitrary binary network addresses such as `IPX/SPX` or `TCP/IP`. The address can also be specified as an arbitrary string. The value should be the same as that specified for the `NLSADDR` parameter in the `NETWORK` section of the configuration file.

`tmloadcf(1)` prints an error if `nlsaddr` is missing from any entry but the entry for the `MASTER LMID`, for which it prints a warning. However, if `nlsaddr` is missing from the `MASTER LMID` entry, `tmadmin(1)` cannot be run in administrator mode on remote machines; it is limited to read-only operations. This also means that a backup site is unable to reboot the `MASTER` site after failure.

- `-u uid-# or uid-name`—Used to run the `tlisten` process as the indicated user. This option is required if the `tlisten(1)` command is run by root on a remote machine.
- `-z [bits]`—Specifies the minimum level of encryption required when establishing a network link between a BEA Tuxedo system administrative process and `tlisten`. Zero (0) means no encryption, while 56 and 128 specify the length (in bits) of the encryption key. If this minimum level of encryption cannot be met, link establishment fails. The default is zero.
- `-Z [bits]`—Specifies the maximum level of encryption allowed when establishing a network link between a BEA Tuxedo system administrative process and `tlisten`. Zero (0) means no encryption, while 56 and 128 specify the length (in bits) of the encryption key. The default is 128. The `-z` and `-Z` options are available only if either the International or US and Canada BEA Tuxedo Security license is installed.

How to Boot the Application

Once all prerequisites have been completed successfully, you can bring up the application using `tmbboot`. Only the administrator who created the `TUXCONFIG` file can execute `tmbboot(1)`.

The application is generally booted from the machine designated as `MASTER` in the `RESOURCES` section of the configuration file or the `BACKUP` acting as the `MASTER`. The `-b` option allows some deviation from this rule. For `tmbboot` to find executables, `BEA` Tuxedo system processes such as the `BBL` must be located in `$TUXDIR/bin`. Application servers should be in the directory defined by the `APPDIR` variable, as specified in the configuration file.

When booting application servers, `tmbboot` uses the `CLOPT`, `SEQUENCE`, `SRVGRP`, `SRVID`, and `MIN` parameters from the configuration file. Application servers are booted in the order specified by the `SEQUENCE` parameter, if `SEQUENCE` is used. If `SEQUENCE` is not specified, servers are booted in the order in which they appear in the configuration file. The command line should look something like the following.

```
$ tmbboot [-g grpname] [-o sequence] [-S] [-A] [-y]
```

Figure 1-3 tmbboot Options

This Option	Performs this Function
<code>-g grpname</code>	Boots all TMS and application servers in groups using this <code>grpname</code> parameter.
<code>-o sequence</code>	Boots all servers in the order shown in the <code>SEQUENCE</code> parameter.
<code>-s server-name</code>	Boots an individual server.
<code>-S</code>	Boots all servers listed in the <code>SERVERS</code> section.
<code>-A</code>	Boots all administrative servers for machines listed in the <code>MACHINES</code> section. This option ensures that the <code>DBBL</code> , <code>BBL</code> , and <code>BRIDGE</code> processes are started in the proper order.
<code>-y</code>	Provides an automatic “yes” response to the prompt that asks whether all administrative and application servers should be booted. This prompt is displayed only if no options that limit the scope of the command (<code>-g grpname</code> , for example) are specified.

Note: For a complete list of `tmbboot` options, see the `tmbboot(1)` reference page.

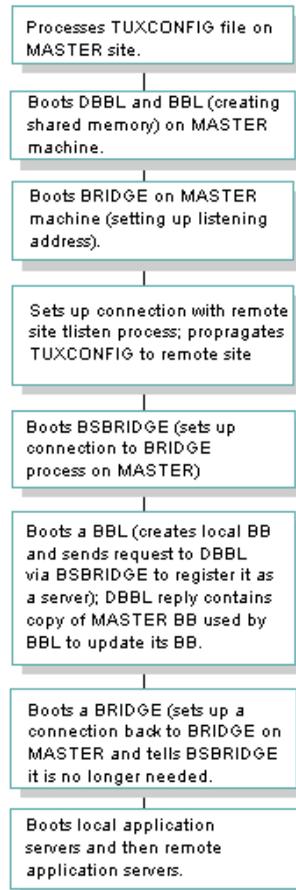
Sequence of tmbboot Tasks for a 2-Machine Configuration

To boot the entire configuration, enter the following command.

```
prompt> tmbboot -y
```

tmbboot performs the following tasks.

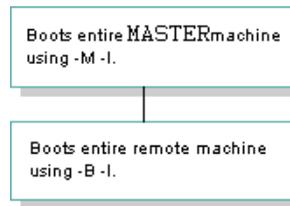
Figure 1-4 Default Boot Sequence for a Small Application



Sequence of tmbboot Tasks for Large Applications (Over 50 Machines)

For relatively large applications (that is, those consisting of over 50 machines), `tmbboot` boots entire machines in a single step rather than performing all the steps used to boot two machines in the default sequence. Following is the optimized sequence of tasks.

Figure 1-5 Boot Sequence for a Large Application



Note: The boot sequence is much faster than the boot sequence for large applications because the number of system messages is far smaller. This method generally reduces boot time by 50%. In a configuration running on a slow network, boot time can be improved by booting machines with higher speed connections to the `MASTER` machine first.

How to Shut Down Your Application

Use the `tmshtutdown(1)` command to shut down all or part of a BEA Tuxedo application. The rules for running this command are similar to those for running `tmbboot(1)`; `tmshtutdown` is the inverse of `tmbboot`.

When the entire application is shut down, `tmshtutdown` removes the interprocess communication (IPC) resources associated with the BEA Tuxedo system. The options used by `tmbboot` for partial booting (`-A`, `-g`, `-I`, `-S`, `-s`, `-l`, `-M`, `-B`) are supported in `tmshtutdown`. The `-b` option (allowing `tmbboot` to be used from a non-`MASTER` machine) is not supported for `tmshtutdown`; you must enter the `tmshtutdown` command from the `MASTER` (or `BACKUP MASTER`) machine.

1 Starting Up and Shutting Down an Application

To migrate servers, use the `-R` option. This option shuts down the servers without removing bulletin board entries for them. If a machine is partitioned, run `tmshutdown` with the `-P LMID` option on the partitioned machine to shut down the servers on that machine.

`tmshutdown` does not shut down the administrative server BBL on a machine to which clients are attached. You can use the `-c` option to override this feature. You need this option for occasions when you must bring down a machine immediately and you cannot contact the clients.

You can use the `-w delay` option to force a hard shutdown after *delay* seconds. This option suspends all servers immediately so that additional work cannot be queued. The value of *delay* should allow time for requests already queued to be serviced. After *delay* seconds, a `SIGKILL` signal is sent to the servers. This option enables the administrator to shut down servers that are looping or blocked in application code.

Running `tmshutdown`

Only the administrator who has written the `TUXCONFIG` file can execute `tmshutdown(1)`. The application can be shut down only from the machine designated as `MASTER` in the configuration file. When the `BACKUP` acts as `MASTER`, it is considered to be the `MASTER` for shutdown purposes. (The only exception to this rule is a partitioned machine. By using the `-p` option, an administrator can run the `tmshutdown` command from a partitioned machine to shut down the application at that site.)

The order in which application servers are shut down is the reverse of the order specified by the `SEQUENCE` parameter for them, or the reverse order in which they are listed in the configuration file. If some servers have `SEQUENCE` numbers and others do not, the unnumbered servers are the first to be shut down, followed by the application servers with `SEQUENCE` numbers (in reverse order). Finally, administrative servers are shut down.

When an application is shut down, all the IPC resources allocated by the BEA Tuxedo system are removed; `tmshutdown` does not remove IPC resources allocated by the DBMS.

Using the IPC Tool When an Application Fails to Shut Down Properly

IPC resources are operating system resources, such as message queues, shared memory, and semaphores. When a BEA Tuxedo application shuts down properly with the `tmshutdown` command, all IPC resources used by the BEA Tuxedo application are removed from the system. In some cases, however, an application may fail to shut down properly and stray IPC resources may remain on the system. When this happens, it may not be possible to reboot the application.

One way to address this problem is to remove IPC resources with a script that invokes the system `IPCS` command and scan for all IPC resources owned by a particular user account. However, with this method, it is difficult to distinguish among different sets of IPC resources; some may belong to a particular BEA Tuxedo application; and others to applications unrelated to the BEA Tuxedo system. It is important to be able to distinguish among these sets of resources; unintentional removal of IPC resources can severely damage an application.

The BEA Tuxedo IPC tool (that is, the `tmipcrm(1)` command) enables you to remove IPC resources allocated by the BEA Tuxedo system (that is, for core BEA Tuxedo and Workstation components only) in an active application.

The command to remove IPC resources, `tmipcrm(1)`, resides in `TUXDIR/bin`. This command reads the binary configuration file (`TUXCONFIG`), and attaches to the bulletin board using the information in this file. `tmipcrm` works only on the local server machine; it does not clean up IPC resources on remote machines in a BEA Tuxedo configuration.

To run this command, enter it as follows on the command line.

```
tmipcrm [-y] [-n] [tuxconfig_file]
```

The IPC tool lists all IPC resources used by the BEA Tuxedo system and gives you the option of removing them.

Note: This command will not work unless you have set the `TUXCONFIG` environment variable correctly or specified the appropriate `TUXCONFIG` file on the command line.

To remove /Q IPC resources, use the `qmadmin(1) ipcrm` command.

1 *Starting Up and Shutting Down an Application*

2 Monitoring Your BEA Tuxedo Application

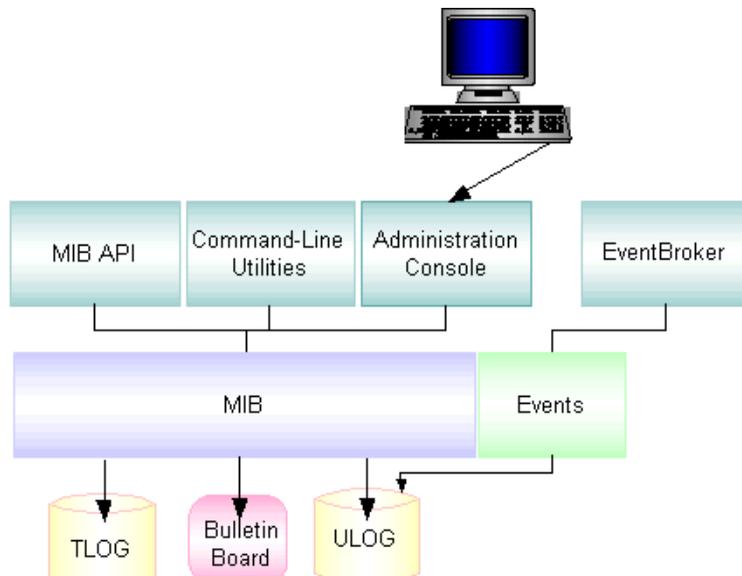
- Ways to Monitor Your Application
- Selecting Appropriate Monitoring Tools
- Using the BEA Administration Console to Monitor Your Application
- Using Command-line Utilities to Monitor Your Application
- Using EventBroker to Monitor Your Application
- Using Log Files to Monitor Activity
- Using the MIB to Monitor Your Application
- Using the Run-time Tracing Utility
- Managing Errors Using the DBBL and BBLs
- Using the ATMI to Handle System and Application Errors
- Monitoring Multithreaded and Multicontexted Applications

Ways to Monitor Your Application

As an administrator, you must ensure that once an application is up and running, it continues to meet the performance, availability, and security requirements set by your company. To perform this task, you need to monitor the resources (such as shared memory), activities (such as transactions), and potential problems (such as security breaches) in your configuration, and take any necessary corrective actions.

To help you meet this responsibility, the BEA Tuxedo system provides several methods for monitoring system and application events, and dynamically reconfiguring your system to improve performance. The BEA Administration Console, command-line utilities, log files, the ATMI, the MIB, and a run-time tracing facility offer an excellent view of how a system is working. They help make your application capable of responding quickly and efficiently to changing business needs or failure conditions. You can use these tools to keep your application performing fast, well, and securely.

Figure 2-1 Monitoring Tools



The BEA Tuxedo system offers the following tools to monitor your application:

- **BEA Administration Console**—a Web-based graphical user interface you can use to observe the behavior of the application, and to dynamically configure its operation. You can display and change configuration information, determine the state of each component of the system, and obtain statistical information about items such as executed requests, and queued requests.
- **Command-line utilities**—a set of commands (for example, `tmboot(1)`, `tmadmin(1)`, and `tmshutdown(1)`) you can use to activate, deactivate, configure, and manage your application.
- **EventBroker**—a mechanism that informs administrators of system faults and exceptional happenings such as network failures. When an event is posted by clients or servers, the EventBroker matches the name of the posted event to a list of subscribers for that event, and takes appropriate action, determined by each subscription.
- **Log files**—a set of files that make up a repository for error and warning messages, debugging messages, and informational messages helpful in tracking and resolving problems in the system.
- **MIB**—an interface to a set of procedures for accessing and modifying information in the MIBs. Using the MIB, you can write programs that enable you to monitor your run-time application.
- **Run-time tracing facility**—software that tracks the execution of an application, thus providing information that is helpful in resolving system problems.

See Also

- “System and Application Data that You Can Monitor” on page 2-4
- “Selecting Appropriate Monitoring Tools” on page 2-8
- “Using the BEA Administration Console to Monitor Your Application” on page 2-9
- “Using the BEA Administration Console” on page 3-4 in *Introducing the BEA Tuxedo System*
- “Using Command-line Utilities to Monitor Your Application” on page 2-10

- “Using EventBroker to Monitor Your Application” on page 2-15
- “Using Log Files to Monitor Activity” on page 2-16
- “Using the ATMI to Handle System and Application Errors” on page 2-28
- “Using the MIB to Monitor Your Application” on page 2-22
- “Managing Operations Using the MIB” on page 3-10 in *Introducing the BEA Tuxedo System*
- “Using the Run-time Tracing Utility” on page 2-25
- `tmsshutdown(1)` in *BEA Tuxedo Command Reference*

System and Application Data that You Can Monitor

The BEA Tuxedo system enables you to monitor system and application data.

Monitoring System Data

To help you monitor a running system, your BEA Tuxedo system maintains parameter settings and generates statistics for the following system components:

- Clients
- Conversations
- Groups
- Message queues
- Networks
- Servers

- Services
- Transactions

You can access these components using the MIB or `tmadmin`. You can set up your system so that it can use the statistics in the bulletin board to make decisions and to modify system components dynamically, without your intervention. With proper configuration, your system can perform the following tasks (when bulletin board statistics indicate that they are required):

- Turn on load balancing
- Start a new copy of a server
- Shut down servers that are not being used

By monitoring the administrative data for your system, you can prevent and resolve problems that threaten the performance, availability, and security of your application.

Where the System Data Resides

To ensure that you have the information necessary to monitor your system, the BEA Tuxedo system provides the following three data repositories:

- *Bulletin Board*—a segment of shared memory (on each machine in your network) to which your system writes statistics about the components and activities of your configuration
- *Log files*—files to which your system writes messages
- *UBBCONFIG*—a text file in which you define the parameters of your system and application

Monitoring Dynamic and Static Administrative Data

You can monitor two types of administrative data that are available on every running BEA Tuxedo system: [static](#) and [dynamic](#).

What Is Static Data

Static data about your configuration consists of configuration settings that you assign when you first configure your system and application. These settings are never changed without intervention (either in real-time or through a program you have provided). Examples include system-wide parameters (such as the number of machines used) and the amount of interprocess communication (IPC) resources (such as shared memory) allocated to your system on your local machine. Static data is kept in the `UBBCONFIG` file and in the bulletin board.

Checking Static Data

At times you may need to check static data about your configuration. For example, you may want to add a large number of machines without exceeding the maximum number of machines allowed in your configuration (or allowed in the machine tables of the bulletin board). You can look up the maximum number of machines allowed by checking the current values of the system-wide parameters for your configuration (one of which is `MAXMACHINES`).

You may be able to improve the performance of your application by tuning your system. To determine whether tuning is required, you need to check the amount of local IPC resources currently available.

What Is Dynamic Data

Dynamic data about your configuration consists of information that changes in real time, that is, while an application is running. For example, the *load* (the number of requests sent to a server) and the *state* of various configuration components (such as servers) change frequently. Dynamic data is kept in the bulletin board.

Checking Dynamic Data

Dynamic configuration data is useful in resolving many administrative problems, as demonstrated by two examples.

In the first example, suppose your throughput is suffering and you want to know whether you have enough servers running to accommodate the number of clients currently connected. Check the number of running servers and connected clients, and the load on one or more servers. These numbers help you determine whether adding more servers will improve performance.

In the second example, suppose you receive multiple complaints about slow response from users when making particular requests of your application. By checking load statistics, you can determine whether increasing the value of the `BLOCKTIME` parameter would improve response time.

Common Startup and Shutdown Problems

When evaluating whether your BEA Tuxedo system is operating normally, you might want to consider the following list of common startup and shutdown problems, and monitor your system periodically.

Common Startup Problems

- Application server failed or dumped core during initialization
- Application server file not found or not executable
- Automatic migration of server group
- Default boot sequence may not be optimal
- Environment variable not set or not set properly
- `IPCKEY` is already in use
- Invalid network address
- Met upper bound limits specified in the `UBBCONFIG` file
- Network port is in use already
- Reached limit on system resources
- Server boot dependency
- `TLOG` file is not created

Common Shutdown Problems

- Clients still attached
- Dead servers
- Shutdown sequence

Selecting Appropriate Monitoring Tools

To monitor a running application, you need to keep track of the [dynamic](#) aspects of your configuration and sometimes check the [static](#) data. In other words, you need to be able to watch the bulletin board on an ongoing basis and consult the `UBBCONFIG` file when necessary. The method you choose depends on the following factors:

- Your BEA Tuxedo system administration experience: If you have a lot of experience as an administrator, as well as shell programming expertise, you may prefer to write programs that automate your most frequently run commands.
- Your operating system experience: If you are inexperienced, you may be more comfortable using the BEA Administration Console.
- Which information you want to view: If you decide to monitor your application by examining the `RESOURCES` section of the `UBBCONFIG` file through the `tmadmin` command, you will have access to only the current values.

The following table describes how to use each monitoring method.

Use this Method	By
BEA Administration Console	Using a graphical interface
Command-line utilities , such as <code>txrpt</code> and <code>tmadmin</code>	Entering commands after a prompt
EventBroker	Subscribing to BEA Tuxedo system events, such as servers dying, and network failures.

Use this Method	By
Log files (for example, ULOG , TLOG)	Viewing the ULOG with any text editor; checking the ULOG for <code>listen</code> messages; and converting the TLOG (a binary file) to a text file by running <code>tmadmin dumptlog</code> which downloads a TLOG to a text file.
MIB	Writing programs that monitor your run-time application
Run-time tracing utility	Specifying a tracing expression that contains a category, a filtering expression, and an action, and enabling the <code>TMTRACE</code> environment variable. For more information, see “Using the Run-time Tracing Utility” on page 2-25.

Using the BEA Administration Console to Monitor Your Application

The BEA Administration Console is a graphical user interface to the MIB that enables you to tune and modify your application. It is accessed through the World Wide Web and used through a Web browser. Any administrator with a supported browser can monitor a BEA Tuxedo application.

Using the Power Bar to Monitor Activities

The Power Bar appears near the top of the main BEA Administration Console window, immediately below the Menu Bar. The Power Bar is a row of 12 buttons that allow you to run tools for frequently performed administrative and monitoring functions. All buttons are labeled with both icons and names. The following buttons are available for monitoring:

- *Logfile*—Displays the ULOG file from a particular machine in the active domain.
- *Event Tool*—Helps you monitor system events. When you click on the Event Tool button, a pop-up window displays four options: *subscribe*—to request notification of specified system events, *unsubscribe*—to reject further notification of specified system

events, *snapshot*—to create a record of the data currently held by the Event Tool, and *select format*—to choose parameters for the information being collected by the Event Tool.

- *Stats*—to display a graphical representation of BEA Tuxedo system activity
- *Search*—to look for a particular object class or object in the Tree.

See Also

- “Using the BEA Administration Console” on page 3-4 in *Introducing the BEA Tuxedo System*

Using Command-line Utilities to Monitor Your Application

To monitor your application through the command-line interface, use the `tmadmin(1)` or `txrpt(1)` command.

Inspecting Your Configuration Using `tmadmin`

The `tmadmin` command is an interpreter for 53 commands that enable you to view and modify a bulletin board and its associated entities. Using the `tmadmin` commands, you can monitor statistical information in the system such as the state of services, the number of requests executed, the number of queued requests, and so on.

Using the `tmadmin` commands, you can also dynamically modify your BEA Tuxedo system. You can, for example, perform the following types of changes while your system is running:

- Suspend and resume services
- Advertise and unadvertise services

- Change service parameters
- Change the `AUTOTRAN` timeout value

Whenever you start a `tmadmin` session, you can choose the following operating modes for that session: the default operating mode, read-only mode, or configuration mode:

- In *default operating mode*, you can view and change bulletin board data during a `tmadmin` session, if you have administrator privileges (that is, if your effective UID and GID are those of the administrator).
- In *read-only mode*, you can view the data in the bulletin board, but you cannot make any changes. The advantage of working in read-only mode is that your administrator process is not tied up by `tmadmin`; the `tmadmin` process attaches to the bulletin board as a client, leaving your administrator slot available for other work.
- In *configuration mode*, you can view the data in the bulletin board and, if you are the BEA Tuxedo application administrator, you can make changes. You can start a `tmadmin` session in configuration mode on any machine, including an inactive machine. On most inactive machines, configuration mode is required in order to run `tmadmin`. (The only inactive machine on which you can start a `tmadmin` session without requesting configuration mode is the `MASTER` machine.)

Note: You can also generate a report of the BEA Tuxedo version and license numbers.

Generating Reports on Servers and Services Using `txrpt`

The `txrpt` command analyzes the standard error output of a BEA Tuxedo server and provides a summary of service processing time within the server. The report shows the number of times each service was dispatched and the average amount of time it took for each service to process a request during the specified period. `txrpt` takes its input from the standard input or from a standard error file redirected as input. To create standard error files, have your servers invoked with the `-r` option from the `servopts(5)` selection; you can name the file by specifying it with the `-e servopts` option. Multiple files can be concatenated into a single input stream for `txrpt`.

Over time, information about service X and server Y (on which service X resides) is accumulated in a file. `txrpt` processes the file and provides you with a report about the service access and timing characteristics of the server.

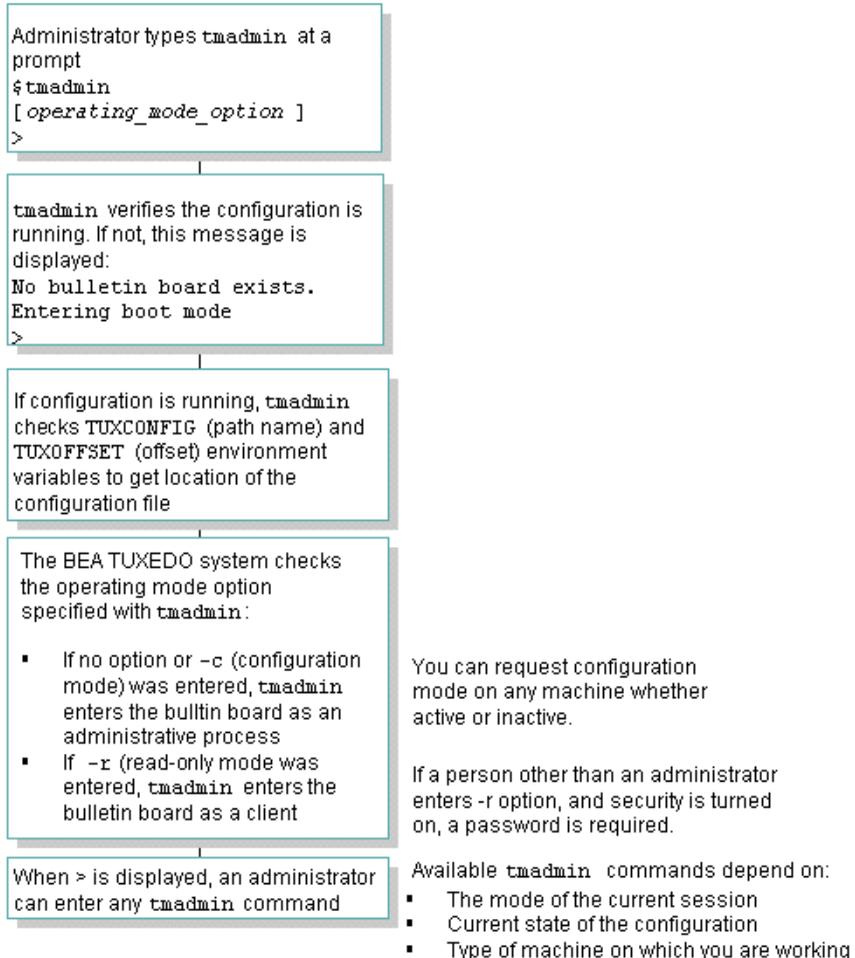
See Also

- “Ways to Monitor Your Application” on page 2-2
- “How a `tmadmin` Session Works” on page 2-13
- “Monitoring Your System Using `tmadmin` Commands” on page 2-14
- “Performing Dynamic Operations Using `tmadmin(1)`” on page 3-26 in *Introducing the BEA Tuxedo System*
- “Using Command-Line Utilities” on page 3-12 in *Introducing the BEA Tuxedo System*

How a `tmadmin` Session Works

The `tmadmin` command is an interpreter for 53 commands that enable you to view and modify a bulletin board and its associated entities. The following illustration shows you how a typical `tmadmin` session works.

Figure 2-2 Typical `tmadmin` Session



Monitoring Your System Using tadmin Commands

Following is a list of run-time system functions that you can monitor with `tadmin` commands:

- Number of servers installed in a service
- Appropriate load distribution
- If a particular service is doing any work
- Inactive clients
- If distribution of work is flowing smoothly through the system
- If a client is tying up a connection and preventing a server from doing any work for another client
- Stability of network
- If you must manually commit or abort a transaction
- Sufficient operating system resources (such as shared memory and semaphores) on a local machine

See Also

- “Performing Dynamic Operations Using `tadmin(1)`” on page 3-26 in *Introducing the BEA Tuxedo System*
- `tadmin(1)` in *BEA Tuxedo Command Reference*

Using EventBroker to Monitor Your Application

The BEA Tuxedo EventBroker monitors a running application for events (for example, a state change in a MIB object, such as the transition of a client from active to inactive). When the EventBroker detects an event, it reports or posts the event, and then notifies relevant subscribers that the event has occurred. You can be informed automatically when events occur in the MIB by receiving FML data buffers representing MIB objects. To post the event and report it to subscribers, the EventBroker uses the `tppost(3c)` function. Both administrators and application processes can subscribe to events.

The EventBroker recognizes over 100 meaningful state transitions to a MIB object as system events. A posting for a system event includes the current MIB representation of the object on which the event occurred, and some event-specific fields that identify the event that occurred. For example, if a machine is partitioned, an event is posted with the following:

- The name of the affected machine, as specified in the `T_MACHINE` class), with all the attributes of that machine
- Some event attributes identifying the event as *machine partitioned*

To use the EventBroker, you simply subscribe to system events.

See Also

- “Managing System Events Using EventBroker” on page 3-14 in *Introducing the BEA Tuxedo System*

Using Log Files to Monitor Activity

To help you identify error conditions quickly and accurately, the BEA Tuxedo system provides the following log files:

- **Transaction log (TLOG)**—A binary file that is not normally read by you (the administrator), but that is used by the Transaction Manager Server (TMS). A TLOG is created only on machines involved in BEA Tuxedo global transactions.
- **User log (ULOG)**—A log of messages generated by the BEA Tuxedo system while your application is running.

These logs are maintained and updated constantly while your application is running.

See Also

- “What Is the Transaction Log (TLOG)” on page 2-17
- “What Is the User Log (ULOG)” on page 3-64 in *Introducing the BEA Tuxedo System*
- “Ways to Monitor Your Application” on page 2-2
- “Detecting Errors Using Logs” on page 2-18
- “Estimating Service Workload Using the Application Service Log” on page 2-21

What Is the Transaction Log (TLOG)

The transaction log (TLOG) keeps track of global transactions during the commit phase. At the end of the first phase of a 2-phase commit protocol, the participants in a global transaction issue a reply to the question of whether to commit or roll back the transaction. This reply is recorded in the TLOG.

The TLOG file is used only by the Transaction Manager Server (TMS) that coordinates global transactions. It is not read by the administrator. The location and size of the TLOG are specified by four parameters that you set in the MACHINES section of the UBBCONFIG file.

We recommend that you create a TLOG on each machine that participates in global transactions.

See Also

- “Detecting Errors Using Logs” on page 2-18

What Is the User Log (ULOG)

The user log (ULOG) is a file to which all messages generated by the BEA Tuxedo system—error messages, warning messages, information messages, and debugging messages—are written. Application clients and servers can also write to the user log. A new log is created every day and there can be a different log on each machine. However, a ULOG can be shared by multiple machines when a remote file system is being used.

The ULOG provides an administrator with a record of system events from which the causes of most BEA Tuxedo system and application failures can be determined. You can view the ULOG, a text file, with any text editor. The ULOG also contains messages generated by the `tlisten` process. The `tlisten` process provides remote service connections for other machines in an application. Each machine, including the master machine, should have a `tlisten` process running on it.

Detecting Errors Using Logs

The BEA Tuxedo log files can help you detect failures in both your application and your system by:

- “Analyzing tlisten Messages in the ULOG” on page 2-18
- “Analyzing the Transaction Log (TLOG)” on page 2-19
- “Analyzing the User Log (ULOG)” on page 2-20

Analyzing tlisten Messages in the ULOG

Part of the ULOG records error messages to the tlisten process. You can view tlisten messages using any text editor. Each machine, including the MASTER machine contains a separate tlisten process. Though separate tlisten logs are maintained in the ULOG on each machine, they can be shared across remote file systems.

The ULOG records tlisten process failures. tlisten is used, during the boot process, by tmboot and, while an application is running, by tadmin. tlisten messages are created as soon as the tlisten process is booted. Whenever a tlisten process failure occurs, a message is recorded in the ULOG.

Note: Application administrators are responsible for analyzing the tlisten messages in the ULOG, but programmers may also find it useful to check these messages.

The *BEA Tuxedo System Messages CMDTUX Catalog* contains the following information about tlisten messages:

- Descriptions of all messages
- Recommended actions that you (or a programmer) can take to resolve the error conditions reported in these messages

tlisten Message Example

Consider the following example of a `tlisten` message in the ULOG.

```
121449.gumby!simpsserv.27190.1.0: LIBTUX_CAT:262: std main starting
```

A ULOG message consists of a tag and text. The tag consists of the following:

- A 6-digit string (hhmmss) representing the time of day (in terms of hour, minute, and second)
- The name of the machine (as returned, on UNIX systems, by the `uname -n` command)
- The name and process identifier of the process that is logging the message. (This process ID can optionally include a transaction ID.) Also included is a thread ID (1) and a context ID (0).

Note: Placeholders are printed in the `thread_ID` and `context_ID` field of entries for single-threaded applications. (Whether an application is multithreaded is not apparent until more than one thread is used.)

The text consists of the following:

- The name of the message catalog
- The message number
- The BEA Tuxedo system message

Note: You can find this message in the *BEA Tuxedo System Messages LIBTUX Catalog*.

Analyzing the Transaction Log (TLOG)

The TLOG is a binary file that contains only messages about global transactions that are in the process of being committed. To view the TLOG, you must first convert it to text format so that it is readable. The BEA Tuxedo system provides two `tmadmin` operations to do this:

- `dumptlog (dl)` downloads (or dumps) the TLOG (a binary file) to a text file.
- `loadtlog` uploads (or loads) an text version of the TLOG into an existing TLOG (a binary file).

2 Monitoring Your BEA Tuxedo Application

The `dumpltlog` and `loadtlog` commands are also useful when you need to move the TLOG between machines as part of a server group migration or machine migration.

Detecting Transaction Errors

You can detect TLOG errors using the MIB to obtain the status of a transaction. You can also run the `tadmin` command `display transaction` to detect any errors in transactions.

Analyzing the User Log (ULOG)

On each active machine in an application, the BEA Tuxedo system maintains a log file that contains BEA Tuxedo system error messages, warning messages, debugging messages, or other helpful information. This file is called the user log or ULOG. The ULOG simplifies the job of finding errors returned by the BEA Tuxedo ATMI, and provides a central repository in which the BEA Tuxedo system and applications can store error information.

You can use the information in the ULOG to identify the cause of system or application failures. Multiple messages about a given problem can be placed in the user log. Generally, earlier messages provide more useful diagnostic information than later messages.

ULOG Message Example

In the following example, message 358 from the LIBTUX_CAT catalog identifies the cause of the trouble reported in subsequent messages, namely, that there are not enough UNIX system semaphores to boot the application.

Listing 2-1 Sample ULOG Messages

```
151550.gumby!BBL.28041.1.0: LIBTUX_CAT:262: std main starting
151550.gumby!BBL.28041.1.0: LIBTUX_CAT:358: reached UNIX limit on semaphore ids
151550.gumby!BBL.28041.1.0: LIBTUX_CAT:248: fatal: system init function ...
151550.gumby!BBL.28040.1.0: CMDTUX_CAT:825: Process BBL at SITE1 failed ...
151550.gumby!BBL.28040.1.0: WARNING: No BBL available on site SITE1.
    Will not attempt to boot server processes on that site.
```

Note: *BEA Tuxedo System Messages* contains complete descriptions of user log messages and recommendations for any actions that should be taken to resolve the problems indicated.

See Also

- “How to Create a TLOG Device” on page 1-6
- “What Is the User Log (ULOG)” on page 3-64 in *Introducing the BEA Tuxedo System*
- “How to Start tlisten at All Sites” on page 1-7
- “Managing Transactions” on page 3-34 in *Introducing the BEA Tuxedo System*
- “Using Transactions” on page 1-18 in *Tutorials for Developing a BEA Tuxedo Application*

Estimating Service Workload Using the Application Service Log

A BEA Tuxedo application server can generate a log of the service requests it handles. The log is displayed on the server’s standard output (`stdout`). Each record contains a service name, start time, and end time.

You can request such a log when a server is activated. The `txrpt` facility produces a summary of the time spent by the server, thus giving you a way to analyze the log output. Using this data, you can estimate the relative workload generated by each service, which will help you set workload parameters appropriately for the corresponding services in the MIB.

Using the MIB to Monitor Your Application

There are essentially two operations you can perform using the MIB: you can *get* information from the MIB (a `get` operation) or you can *update* information in the MIB (a `set` operation) at any time using a set of ATMI functions (for example, `tpalloc(3c)`, `tprealloc(3c)`, `tpcall(3c)`, `tpacall(3c)`, `tpgetrply(3c)`, `tpenqueue(3c)`, and `tpdequeue(3c)`).

When you query the MIB with a `get`, the MIB responds to your reply with a number of matcher, and indicates how many more objects match your request. The MIB returns a handle (that is, the cursor) that you can use to get the remaining objects. The operation you use to get the next set of objects is called `getnext`. The third operation occurs when queries span multiple buffers.

Limiting Your MIB Queries

When you query the MIB, which is a virtual database, you are selecting a set of records from the database table. You can control the size of the database table in two ways: by controlling the number of objects about which you want information, or by controlling the amount of information about each object. Using *key fields* and *filters*, you can limit the scope of your request to data that is meaningful for your needs. The more limits you specify, the less information is requested from the application, and the faster the data is provided to you.

Querying Global and Local Data

Data in the MIB is stored in a number of different places. Some data is replicated on more than one machine in a distributed application. Other data is not replicated, but is local to particular machines based on the nature of the data or the object represented.

What Is Global Data

Global data is information about application components such as servers that is replicated on every machine in an application. Most of the data about a server, for example, such as information about its configuration and state, is replicated globally throughout an application, specifically in every bulletin board. A BEA Tuxedo application can access this information from anywhere.

For example, from any machine in an application called Customer Orders, the administrator can find out that server B6 belongs to Group 1, runs on machine CustOrdA, and is active.

What Is Local Data

Other information is not replicated globally, but is local to an entity, such as statistics for a server. An example of a local attribute is `TA_TOTREQC`, which defines the number of times services have been processed in a specified server. This statistic is stored with the server on its host machine. When the server accepts and processes a service request, the counter is incremented. Because this kind of information is managed locally, replicating it would inhibit your system's performance.

There are also classes in the MIB that are exclusively local, such as clients. When a client logs in, the BEA Tuxedo system creates an entry for it in the bulletin board, and records all tracking information about the client in that entry. The MIB can determine the state of the client at anytime by checking this entry.

Using `tmadmcall` to Access Information

The BEA Tuxedo system provides a programming interface that offers direct access to the MIB while your application is not running. This interface, the `tpadmcall` function, gives the application direct access to the data upon which the MIB is based. `tpadmcall` allows you access to a subset of information that is local to your process.

Use `tpadmcall` when you need to query the system or make administrative changes while your system is *not running*. `tpadmcall` queries the `TUXCONFIG` file on behalf of your request. Data buffers that you put in, and data buffers that you receive (containing your queries and the replies to them) are exactly the same.

See Also

- “Managing Operations Using the MIB” on page 3-10 in *Introducing the BEA Tuxedo System*
- MIB(5) in *BEA Tuxedo File Formats and Data Descriptions Reference*
- “Querying and Updating the MIB with ud32” on page 2-24

Querying and Updating the MIB with ud32

ud32 is a client program delivered with the BEA Tuxedo system that reads input consisting of text representation of FML buffers. You can use ud32 for ad hoc queries and updates to the MIB. It creates an FML32 buffer, makes a service call with the buffer, receives a reply (also in an FML32 buffer) from the service call, and displays the results on screen or in a file in text format.

ud32 builds an FML32-type buffer with the FML fields and values that you represent in text format, makes a service call to the identified service in the buffer, and waits for the reply. The reply then comes back in FML32 format as a report. Now, because the MIB is FML32-based, ud32 becomes the scripting tool for the MIB.

For example, suppose you write a small file that contains the following text.

```
service name=.tmib and ta_operation=get,TACLASSES=T_SERVER
```

When you type this file into ud32, you receive an FML output buffer listing all the data in the system about the servers.

Using the Run-time Tracing Utility

The BEA Tuxedo system provides a run-time tracing facility that enables you to track the execution of distributed business applications. The system has a set of built-in trace points that mark calls to functions in different categories, such as ATMI functions issued by the application or XA functions issued by the BEA Tuxedo system to an X/Open compliant resource manager.

To enable tracing, you must specify a tracing expression that contains a category, a filtering expression, and an action. The category indicates the type of function (such as ATMI) to be traced. The filtering expression specifies which particular functions trigger an action. The action indicates the response to the specified functions by the BEA Tuxedo system. The system may, for example, write a record in the `ULOG`, execute a system command, or terminate a trace process. A client process can also propagate the tracing facility with its requests. This capability is called *dyeing*; the trace dye *colors* all services that are called by the client.

There are two ways to specify a tracing expression: by setting the `TMTRACE` environment variable, or by specifying the expression in a server environment.

- For a simple tracing expression, define `TMTRACE=on` in the environment of the client. This expression enables tracing of ATMI functions on the client and on any server that performs a service on behalf of that client. The trace records are written to the `ULOG` file.
- You can also specify a tracing expression in the environment of a server. For example, you might enter the following: `TMTRACE=atmi:/tpservice/ulog`. If you export this setting within the environment of the server, a record will be generated in the `ULOG` file each time a service is invoked on that server.

You can activate or deactivate the tracing option using the `changetrace` command of `tmadmin`. This command enables you to overwrite the tracing expression on active client or server processes. Administrators can enable global tracing for all clients and servers, or for a particular machine, group, or server.

See Also

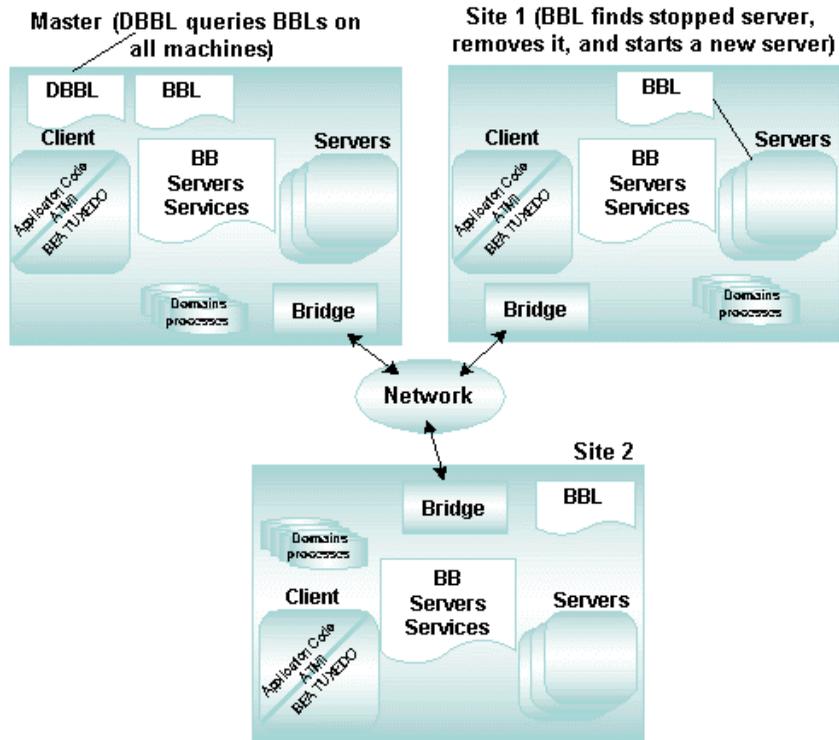
- “Ways to Monitor Your Application” on page 2-2
- `tmtrace(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*

Managing Errors Using the DBBL and BBLs

The BEA Tuxedo system uses the following two administrative servers to distribute the information on the bulletin board to all active machines in the application:

- *DBBL*—The Distinguished Bulletin Board Liaison server propagates global changes to the MIB and maintains the static part of the MIB. Specifically, the DBBL:
 - Resides (only one DBBL per application) on the `MASTER` machine and provides periodic status requests to all BBLs
 - Coordinates bulletin board updates, the state of different machines, and queries with the BBLs
 - Coordinates migration of servers
 - Can be migrated to other machines for fault resiliency
- *BBL*—The Bulletin Board Liaison server maintains the bulletin board on its host machine, coordinating changes to the local MIB, and verifying the integrity of application programs active on its machine. Specifically, the bulletin board:
 - Resides on each BEA Tuxedo machine in an application, carries out requests from the DBBL, and administers timeouts for service requests, replies to requesters, and transactions
 - Detects server failures, initiates user-defined recovery, and automatically restarts servers
 - Detects client failures
 - Cleans up client and server entries, and conversations on the bulletin board
 - Detects and recovers DBBL failures (if it is the BBL residing on the `MASTER` machine)

Figure 2-3 Diagnosis and Repair Using the DBBL and BBLs



Both servers have a role in managing faults. The DBBL coordinates the state of other active machines in the application. Each BBL communicates state changes in the MIB, and sometimes sends a message to the DBBL indicating all is OK on its host machine.

The BEA Tuxedo run-time system records events, along with system errors, warnings, and tracing events, in the user log (ULOG). Programmers can use the ULOG to debug their applications or notify administrators of special conditions or states found (for example, an authorization failure).

Using the ATMI to Handle System and Application Errors

Using the ATMI, a programmer controls some of the more global aspects of communications. The ATMI provides functions for handling both application and system-related errors. When a service routine encounters an *application* error, such as an invalid account number, the client knows the service performed its task but could not fulfill its request because of an application error.

With a *system* failure, such as a server crashing while performing a request, the client knows the service routine did not perform its task because of an underlying system error. The BEA Tuxedo system notifies programs of system errors that occur as it monitors the application's behavior and its own behavior.

Using Configurable Timeout Mechanisms

At times, a service may get stuck in an infinite loop while processing a request. The client waits, but no reply is forthcoming. To protect a client from endless waiting, the BEA Tuxedo system has two types of configurable timeout mechanisms: blocking timeouts and transaction timeouts.

A *blocking timeout* is a mechanism that ensures a blocked program waits no longer than the specified timeout value for something to occur. Once a timeout is detected, the waiting program is alerted with a system error informing it that a blocking timeout has occurred. The blocking timeout defines the duration of service requests, or how long the application is willing to wait for a reply to a service request. The timeout value is a global value defined in the `BLOCKTIME` field of the `RESOURCES` section of the `TUXCONFIG` file.

A *transaction timeout* is another type of timeout that can occur because active transactions tend to be resource-intensive. A transaction timeout defines the duration of a transaction, which may involve several service requests. The timeout value is defined when the transaction is started (with `tpbegin(3c)`). Transaction timeouts are useful when maximizing resources. For example, if database locks are held while a

transaction progresses, an application programmer may want to limit the amount of time that the application's transaction resources are held up. A transaction timeout always overrides a blocking timeout.

Configuring Redundant Servers to Handle Failures

You can handle some failure situations by configuring an application with redundant servers and the automatic restart capability. Redundant servers provide high availability, and can be used to handle large amounts of work, server failures, or machine failures. The BEA Tuxedo system continually checks the status of active servers, and when it detects the failure of a restartable server, the system automatically creates a new instance of that server.

By configuring servers with the automatic restart property, you can handle individual server failures. You can also specify the number of restarts that the system will provide. This capability can prevent a recurring application error by limiting the number of times a server is restarted.

The BEA Tuxedo system frequently checks the availability of each active machine. A machine is marked as *partitioned* when it cannot be reached by the system. If this occurs, a system event is generated. A partition can occur due to a network failure, machine failure, or severe performance degradation.

See Also

- “Development View: What You Can Do Using the ATMI” on page 3-37 in *Introducing the BEA Tuxedo System*
- “System and Application Data that You Can Monitor” on page 2-4
- “Monitoring Dynamic and Static Administrative Data” on page 2-5

Monitoring Multithreaded and Multicontexted Applications

- While monitoring a multithreaded application, keep in mind that individual threads are not visible to an administrator.
- You can get MIB statistical reports for various aspects of your multithreaded and/or multicontexted application by running the `tmadmin(1)` command interpreter. Here are a few examples of the information you can request for a multithreaded application:
 - Count of client contexts per client process and a separate entry for each client context. (obtained by running the `tmadmin pclt` command)
 - Count of dispatched services per server process and, optionally, information about each context (obtained by running `tmadmin/psr`, optionally in verbose mode)
- When the BBL checks clients, it verifies that a process is alive. If a process has died, the BBL detects the process death. If an individual thread within a process has died, however, the death of the thread is not detected by the BBL.

Therefore application programmers should keep in mind the possibility that individual threads within a process may die. If one thread dies and a signal is issued, the whole process to which the thread belongs usually dies, and that death is detected by the BBL.

If a thread dies as the result of an erroneous call to a thread exit function, however, no signal is generated. If this type of death occurs before the thread calls `tpterm()`, then the BBL cannot detect the death and does not deallocate the registry table slot for the context associated with the dead thread. (It would not be proper for the BBL to deallocate this registry table slot even if it could detect the death of the thread because, in some application models, another thread might subsequently choose to associate itself with that context.)

There is no solution for this limitation so it is important for programmers to keep it in mind and design their applications accordingly.

How to Retrieve Data About a Multithreaded/Multicontexted Application Using the MIB

Note: The information presented here applies to all multithreaded and/or multicontexted applications, regardless of which administrative tools are being used. The functionality is discussed from the point of view of an administrator using MIB calls, but is the same for an administrator using an interface to the MIB, whether that interface is `tmadmin(1)` or the BEA Administration Console.

You can obtain information about a multithreaded or multicontexted application by:

- Issuing calls to the MIB
- Issuing selected `tmadmin` commands

Information is available in the following locations:

- The client section of the bulletin board registry provides an entry for each context. (An entry is created automatically by the BEA Tuxedo system whenever a new context is created through a call to `tpinit()` in `TPMULTICONTEXTS` mode.)
- The `T_SERVERCTXT` class of the `TM_MIB` provides multiple instances of 14 fields if multiple server dispatch threads are active simultaneously. Specifically, the `T_SERVERCTXT` section includes an instance of each of the following fields for each active sever dispatch thread:
 - `TA_CONTEXTID` (key field)
 - `TA_SRVGRP` (key field)
 - `TA_SRVID` (key field)
 - `TA_CLTLMID`
 - `TA_CLTPID`
 - `TA_CLTREPLY`
 - `TA_CMTRET`
 - `TA_CURCONV`

2 Monitoring Your BEA Tuxedo Application

- TA_CURREQ
- TA_CURRSERVICE
- TA_LASTGRP
- TA_SVCTIMEOUT
- TA_TIMELEFT
- TA_TRANLEV

For example, if 12 server dispatch threads are active simultaneously, then the T_SERVERCTXT class of the MIB for this application will include 12 occurrences of the TA_CONTEXTID field, 12 occurrences of the TA_SRVGRP field, and so on.

When multiple instances of T_SERVER class fields contain multiple values for different contexts of a multicontexted server, a “dummy” value is specified in the T_SERVER class field and the T_SERVERCTXT field contains an actual value for each context.

See Also

- `tmadmin(1)` in *BEA Tuxedo Command Reference*
- `TM_MIB(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*
- “Programming a Multithreaded and Multicontexted Application” on page 10-1 in *Programming a BEA Tuxedo Application Using C*

3 Dynamically Modifying an Application

- Dynamic Modification Methods
- Using `tmconfig` to Make Permanent Changes to Your Configuration
- How to Run `tmconfig`
- Making Temporary Modifications to Your Configuration with `tmconfig`
- Limitations on Dynamic Modification Using `tmconfig`
- Making Temporary Modifications to Your Configuration with `tmadmin`

Dynamic Modification Methods

As an administrator, you must ensure that once an application is up and running, it continues to meet the performance, availability, and security requirements set by your company. The BEA Tuxedo system allows you to make changes to your configuration without shutting it down. Without inconveniencing your users, you can do the following:

- Modify existing entries in your configuration file, that is, make changes to `TUXCONFIG`.
- Add components to your application by adding entries for them to your configuration file.

3 *Dynamically Modifying an Application*

- Make temporary changes to an application by advertising, unadvertising, suspending, or resuming services, and changing service parameters (such as `LOAD` and `PRIORITY`).

Note: To modify the configuration file for a running application, you must do one of the following:

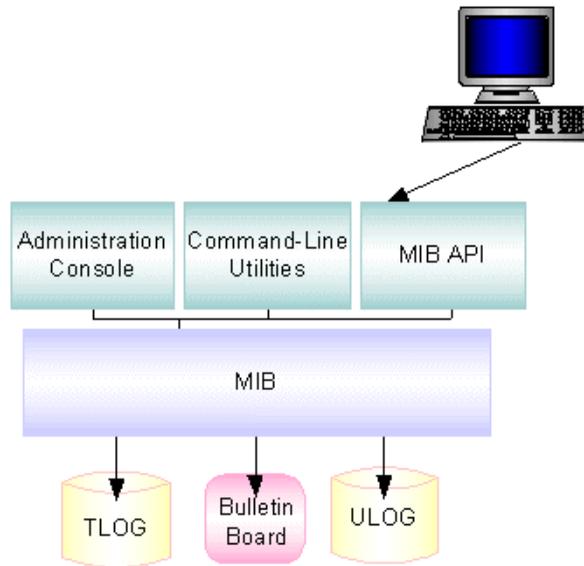
- Shut down your application first (and reboot it after revising the configuration file).
- Run the `tmconfig(1)` command (described on the `tmconfig`, `wtmconfig(1)` reference page), which allows you to modify your configuration file dynamically.

Thus, you can adjust your system to reflect either current or expected conditions by making either permanent or temporary changes to an application. Temporary changes are reflected in the bulletin board only. Permanent changes are made by modifying the `TUXCONFIG` file. Because `TUXCONFIG` is a binary file, however, you cannot edit it through a simple text editor.

Tools for Modifying Your Application

To help you dynamically modify your application, the BEA Tuxedo system provides the following three methods: the BEA Administration Console, command-line utilities, and the Management Information Base (MIB) API. These tools help you respond quickly and efficiently to the need for changes in your application resulting from changing business needs or failure conditions. Use them to keep your application performing fast, well, and securely.

Figure 3-1 Dynamic Modification Tools



- *BEA Administration Console*—a Web-based graphical user interface (GUI) you can use to dynamically configure your application. You can display and change configuration information, determine the state of each component of the system, and obtain statistical information about items such as executed requests and queued requests.
- *Command-line utilities*— Most of the functionality needed for dynamic modification is provided by two commands: `tmadmin` and `tmconfig`. `tmadmin` is a shell-level command with over 70 subcommands for performing various administrative tasks, including dynamic system modification. `tmconfig` is a shell-level command that you can use to add and modify configuration entries while your system is running.
- *MIB API*—a Management Information Base API that enables you to write your own programs to monitor your system and make dynamic changes to your system.

You always have the choice of these three tools for any administrative task. For dynamic modification or reconfiguration, however, we recommend the BEA Administration Console for its ease of use. Full descriptions of all the features in the Administration Console are available through the Help utility provided with the GUI.

If you prefer to work on the command line, however, simply run the `tmadmin` or `tmconfig` command.

Note: For lists of configuration parameters and reconfiguration restrictions, see `tmconfig`, `wtmconfig(1)` in *BEA Tuxedo Command Reference* and `TM_MIB(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*.

See Also

- “Using `tmconfig` to Make Permanent Changes to Your Configuration” on page 3-4
- “Using the BEA Administration Console” on page 3-4 in *Introducing the BEA Tuxedo System*
- “Managing Operations Using the MIB” on page 3-10 in *Introducing the BEA Tuxedo System*
- `APPQ_MIB(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*
- `DM_MIB(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*
- `MIB(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*
- `TM_MIB(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*

Using `tmconfig` to Make Permanent Changes to Your Configuration

The `tmconfig` command enables you to browse and modify your configuration file (`TUXCONFIG` on the `MASTER` machine) and its associated entities, and to add new components (such as machines and servers) to your application while it is running. When you modify your configuration file (`TUXCONFIG` on the `MASTER` machine), `tmconfig` enables you to perform the following tasks:

- Update the `TUXCONFIG` file on all machines that are currently booted in the application.
- Propagate the `TUXCONFIG` file automatically to new machines as they are booted.

Note: The `tmconfig` command runs as a BEA Tuxedo system client.

Because `tmconfig` runs as a BEA Tuxedo client, it is characterized by the following conditions:

- `tmconfig` fails if it cannot allocate a `TPINIT` typed buffer.
- The `username` associated with the client is the login name of the user. (`tmconfig` fails if the user's login name cannot be determined.)
- For a secure application (that is, an application for which the `SECURITY` parameter has been set in the configuration file), `tmconfig` prompts for the application password. If the application password is not provided, `tmconfig` fails.
- If `tmconfig` cannot register as a client, an error message containing `tperrno` is displayed and `tmconfig` exits. If this happens, check the user log to determine the cause. The most likely causes for this type of failure are:
 - The `TUXCONFIG` environment variable was not set correctly.
 - The system was not booted on the machine on which `tmconfig` is being run.
- `tmconfig` ignores all unsolicited messages.
- The client name for the `tmconfig` process that is displayed in the output from `printclient` (a `tadmin` command) is `tpsysadm`.

How `tmconfig` Works

When you type `tmconfig` on a command line, you are launching the display of a series of menus and prompts through which you can request an operation such as the display or modification of a configuration file record. `tmconfig` collects your menu choices, performs the requested operation, and prompts you (by displaying another set of menu choices) to request another operation. It repeatedly offers to perform operations (by repeatedly displaying the menus) until you exit the session by selecting `QUIT` from a menu.

3 *Dynamically Modifying an Application*

The following listing shows the menus and prompts that are displayed once you launch a `tmconfig` command session.

Note: The lines in the listing are numbered in this example for your convenience; during an actual `tmconfig` session, these numbers are not displayed.

Listing 3-1 Menus and Prompts Displayed in a `tmconfig` Session

```
1  $ tmconfig
2  Section: 1) RESOURCES, 2) MACHINES, 3) GROUPS 4) SERVERS
3  5)SERVICES 6) NETWORK 7) ROUTING q) QUIT 9) WSL
4  10) NETGROUPS 11) NETMAPS 12) INTERFACES [1]:
5
6  Operation: 1) FIRST 2) NEXT 3) RETRIEVE 4) ADD 5) UPDATE
7  6) CLEAR BUFFER 7) QUIT [1]:
8  Enter editor to add/modify fields [n]?
9  Perform operation [y]?
```

As shown, you are asked to answer four questions:

- In which section of the configuration file do you want to view, add, or modify a record?
- For the section of the configuration file you have just specified, which operation do you want to perform?
- Do you want to enter a text editor now to add or modify fields for the record?
- Do you want `tmconfig` to perform the requested operation now?

How to Select a Section of the Configuration File

When you start a `tmconfig` session, the following menu is displayed. Each item is a section of `TUXCONFIG`, the configuration file for the application.

```
Section: 1) RESOURCES, 2) MACHINES, 3) GROUPS 4) SERVERS
5)SERVICES 6) NETWORK 7) ROUTING q) QUIT 9) WSL
10) NETGROUPS 11) NETMAPS 12) INTERFACES [1]:
```

Note: For details about these sections (including a list of configurable parameters for each section), see `TM_MIB(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*. `TM_MIB` includes the names of fields that are displayed during a `tmconfig` command session, the range of values for each field, the key fields for each section, and any restrictions or updates to the fields in each section.

- To select a section, enter the appropriate number after the menu prompt. For example, to select the `MACHINES` section, enter 2, as follows.

```
10) NETGROUPS 11) NETMAPS 12) INTERFACES [1]: 2
```

- The default section is the `RESOURCES` section, in which parameters that apply to your entire application are defined. To accept the default selection (which is displayed within square brackets), simply press the Enter key.

```
10) NETGROUPS 11) NETMAPS 12) INTERFACES [1]:
```

How to Select a `tmconfig` Task

A menu of tasks that `tmconfig` can perform is displayed after you select a section of the configuration file.

```
Operation: 1) FIRST 2) NEXT 3) RETRIEVE 4) ADD 5) UPDATE  
6) CLEAR BUFFER 7) QUIT [1]:
```

To select an operation, enter the appropriate number at the menu prompt. For example, to select the `CLEAR BUFFER` section, enter 6, as follows.

```
6) CLEAR BUFFER 7) QUIT [1]: 6
```

The following table defines each task.

3 Dynamically Modifying an Application

Table 3-1 tmconfig tasks

This Menu Item	Called	Performs the Following Activities
1	FIRST	<p>Displays the first record from the specified section. No key fields are needed. If any are in the input buffer, they are ignored.</p> <p>Using the FIRST operation can reduce the amount of typing that is needed. When adding a new record to a section, instead of typing all the required field names and values, use the FIRST operation to retrieve an existing record for the UBBCONFIG section. Then, select the ADD operation and use the text editor to modify the parameter values in the newly created record.</p>
2	NEXT	<p>Displays the next record from the specified section, based on the key fields in the input buffer.</p>
3	RETRIEVE	<p>Displays the requested record (specified with the appropriate key fields) from the specified section.</p>
4	ADD	<p>Adds the indicated record to the specified section. For any optional fields that are not specified, the defaults specified in TM_MIB (5) are used. (All defaults and validations used by <code>tmloadcf(1)</code> are enforced.) The current values for all fields are returned in the output buffer. This operation can be done only by the BEA Tuxedo application administrator.</p>
5	UPDATE	<p>Updates the record specified in the input buffer in the selected section. Any fields not specified in the input buffer remain unchanged. (All defaults and validations used by <code>tmloadcf(1)</code> are enforced.) The current values for all fields are returned in the input buffer. This operation can be done only by the BEA Tuxedo application administrator.</p>
6	CLEAR BUFFER	<p>Clears the input buffer. (All fields are deleted.) After this operation, <code>tmconfig</code> immediately prompts for the specified section again.</p>
7	QUIT	<p>Exits <code>tmconfig</code> gracefully: the client is terminated. You can also exit <code>tmconfig</code> at any time by entering <code>q</code> at any prompt.</p>

How Results of a `tmconfig` Task Are Displayed

After `tmconfig` completes a task, the results—a return value and the contents of the output buffer—are displayed on the screen.

- If the *operation was successful but no update was done*, the following message is displayed.

Return value `TAOK`

The message in the `TA_STATUS` field is

Operation completed successfully.

- If the *operation was successful and an update was done*, the following message is displayed.

Return value `TAUPDATED`

The message in the `TA_STATUS` field is

Update completed successfully.

- If the *operation failed*, an error message is displayed:
 - If there is a problem with permissions or a BEA Tuxedo system communications error (rather than with the configuration parameters), one of the following return values is displayed: `TAEPerm`, `TAEOS`, `TAESystem`, or `TAEtime`.
 - If there is a problem with a configuration parameter of the running application, the name of that parameter is displayed as the value of the `TA_BADFLDNAME` file, and the problem is indicated in the value of the `TA_STATUS` field in the output buffer. If this type of problem occurs, one of the following return values is displayed: `TAERange`, `TAEInconsis`, `TAEConfig`, `TAEDuplicate`, `TAENotFound`, `TAERequired`, `TAESize`, `TAEUpdate`, or `TAENospace`.

`tmconfig` Error Message Conditions

The following list describes the conditions indicated by both sets of error messages.

`TAEPerm`

The `UPDATE` or `ADD` operation was selected but `tmconfig` is not being run by the BEA Tuxedo application administrator.

3 *Dynamically Modifying an Application*

TAESYSTEM

A BEA Tuxedo system error has occurred. The exact nature of the error is recorded in the user log. See `userlog(3c)` in *BEA Tuxedo C Function Reference*.

TAEOS

An operating system error has occurred. The exact nature of the error is written to the user log.

TAETIME

A blocking timeout has occurred. The output buffer is not updated so no information is returned for retrieval operations. The status of update operations can be checked by retrieving the record that was being updated.

TAERANGE

A field value is either out of range or invalid.

TAEINCONSIS

For example, an existing `RQADDR` value or one `SRVGRP/SERVERNAME` entry may be specified for a different `SRVGRP/SERVERNAME` entry.

TAECONFIG

An error occurred while the `TUXCONFIG` file was being read.

TAEDUPLICATE

The operation attempted to add a duplicate record.

TAENOTFOUND

The record specified for the operation was not found.

TAEREQUIRED

A field value is required but is not present.

TAESIZE

A value for a string field is too long.

TAEUPDATE

The operation attempted to do an update that is not allowed.

TAENOSPACE

The operation attempted to do an update but there was not enough space in the `TUXCONFIG` file and/or the bulletin board.

How to Run `tmconfig`

To run `tmconfig` properly, you must set the required environmental variables. Also, if you have not run `tmconfig`, we recommend that you walk through a generic `tmconfig` session, during which you modify entries in your configuration file.

How to Set Environment Variables for `tmconfig`

Before you can start a `tmconfig` session, you must set the required environment variables and permissions. For your convenience, you may also want to select a text editor other than the default editor.

Complete the following procedure to set up your working environment properly before running `tmconfig`.

1. Log in as the BEA Tuxedo application administrator if you want to add entries to `TUXCONFIG`, or modify existing entries. (You do not need to log in as the administrator if you only want to view existing configuration file entries without changing or adding to them.)
2. Assign values to two mandatory environment variables: `TUXCONFIG` and `TUXDIR`.
 - The value of `TUXCONFIG` must be the full path name of the binary configuration file on the machine on which `tmconfig` is being run.
 - The value of `TUXDIR` must be the full path name of the root directory for the BEA Tuxedo system binary files. (`tmconfig` must be able to extract field names and identifiers from `$TUXDIR/udataobj/tpadmin`.)
3. You may also set the `EDITOR` environment variable; this step is optional. The value of `EDITOR` must be the name of the text editor you want to use when changing parameter values; the default is `ed` (a UNIX system command-line editor).

Note: Many full-screen editors do not function properly unless the `TERM` environment variable is also set.

How to Conduct a `tmconfig` Walk-through Session

The following procedure leads you through a sample `tmconfig` session.

1. Enter `tmconfig` after a shell prompt.

```
$ tmconfig
```

Note: You can end a session at any time by entering `q` (short for quit) after the Section menu prompt.

A menu of sections in the `TUXCONFIG` file is displayed.

```
Section: 1) RESOURCES, 2) MACHINES, 3) GROUPS 4) SERVERS
5)SERVICES 6) NETWORK 7) ROUTING q) QUIT 9) WSL
10) NETGROUPS 11) NETMAPS 12) INTERFACES [1]:
```

2. Select the section that you want to change by entering the appropriate menu number, such as 2 for the `MACHINES` section. The default choice is the `RESOURCES` section, represented by [1] at the end of the list of sections shown in Step 1. If you specify a section (instead of accepting the default), that section becomes the new default choice and remains so until you specify another section.

A menu of possible operations is displayed.

```
Operation: 1) FIRST 2) NEXT 3) RETRIEVE 4) ADD 5) UPDATE
6) CLEAR BUFFER 7) QUIT [1]:
```

Each operation listed here is available to be performed on one record at a time of one section of the configuration file. The names of most operations (`FIRST` and `NEXT`) are self-explanatory. When you select `FIRST`, you are asking to have the first record (in the specified section of the configuration file) displayed on the screen. When you select `NEXT`, you are asking to have the contents of the buffer replaced by the second record in the specified section, and to have the new buffer contents displayed on the screen. By repeatedly choosing `NEXT`, you can view all the records in a given section of the configuration file in the order in which they are listed.

3. Select the operation that you want to have performed.

The default choice is the `FIRST` operation, represented by [1] at the end of the list of operations shown in Step 2.

A prompt is displayed, asking whether you want to enter a text editor to start making changes to the `TUXCONFIG` section you specified in Step 2.

```
Enter editor to add/modify fields [n]?
```

4. Select `y` or `n` (for *yes* or *no*, respectively). The default choice (shown at the end of the prompt) is `n`.

If you select *yes* (`y`), the specified editor is invoked and you can start adding or changing fields. The format of each field is

```
field_name<tabs>field_value
```

where the name and value of the field are separated by one or more tabs.

In most cases, the field name is the same as the corresponding *KEYWORD* in the `UBBCONFIG` file, prefixed with `TA_`.

Note: For details about valid input, see “`tmconfig` Input Buffer Considerations” on page 3-14. For descriptions of the field names associated with each section of the `UBBCONFIG` file, see `TM_MIB(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*.

When you finish editing the input buffer, `tmconfig` reads it. If any errors are found, a syntax error is displayed and `tmconfig` prompts you to decide whether to correct the problem.

```
Enter editor to correct?
```

5. Select `n` or `y`.

If you decide not to correct the problem (by entering `n`), the input buffer contains no fields; otherwise, the editor is executed again.

When you finish editing the input buffer, a prompt is displayed, asking whether you want to have the operation you specified (in Step 3) performed now.

```
Perform operation [y]?
```

6. Select `n` or `y`. The default choice (shown at the end of the prompt) is `y`.
 - If you select *no*, the menu of sections is displayed again. Return to Step 2.
 - If you select *yes*, `tmconfig` executes the requested operation and displays the following confirmation message.

```
Return value TAOK
```

The results of the operation are displayed on the screen.

3 *Dynamically Modifying an Application*

You have completed an operation on one section of TUXCONFIG; you may now start another operation on the same section or on another section. To allow you to start a new operation, `tmconfig` displays, again, the menu of TUXCONFIG sections displayed in Step 1.

Note: All output buffer fields are available in the input buffer unless the input buffer is cleared.

7. Continue your `tmconfig` session by requesting more operations, or quit the session.
 - To continue requesting operations, return to Step 2.
 - To end your `tmconfig` session, select `QUIT` from the menu of operations (as shown in Step 3).
8. After you end your `tmconfig` session, you can make a backup copy, in text format, of your newly modified TUXCONFIG file. In the following example, the administrator chooses the default response to the offer of a backup (`yes`) and overrides the default name of the backup file (`UBBCONFIG`) by specifying another name (`backup`).

```
Unload TUXCONFIG file into ASCII backup [y]?
Backup filename [UBBCONFIG]? backup
Configuration backed up in backup
```

tmconfig Input Buffer Considerations

The following considerations apply to the input buffer used with `tmconfig`:

- If the value that you are typing into a field extends beyond one line, you may continue it on the next line if you insert one or more tabs at the beginning of the second line. (The tab characters are dropped when your input is read into TUXCONFIG.)
- A line that contains only a single newline character is ignored.
- If more than one line is provided for a particular field, the first occurrence is used and other occurrences are ignored.

- To enter an unprintable character as part of the value of a field, or to enter a tab as the first character in a field, enter a backslash, followed by the two-character hexadecimal representation of the desired character. (For a mapping of ASCII to hexadecimal characters, see ASCII(5) in a UNIX system reference manual.) Here are a few examples:
 - To insert a blank space, type
`\20`
 - To insert a backslash, type
`\\`

Making Temporary Modifications to Your Configuration with tmconfig

Many aspects of your configuration can be changed dynamically. This section provides instructions for performing the tasks cited in the following list:

- “How to Add a New Machine” on page 3-16
- “How to Add a Server” on page 3-19
- “How to Activate a Newly Configured Machine” on page 3-21
- “How to Add a New Group” on page 3-24
- “How to Change Data-dependent Routing (DDR) for an Application” on page 3-25
- “How to Change Application-wide Parameters” on page 3-26
- “How to Change an Application Password” on page 3-29

How to Add a New Machine

1. Enter `tmconfig`.
2. To specify the `MACHINES` section of the configuration file, enter `2` after the prompt following the list of sections. (Refer to lines 2-4 in the following sample listing.)
3. Press the Enter key to accept the default operation to be performed. The default is `1) FIRST`, an operation that displays the first record in the designated section. In this case, the first record is for the first machine appearing in the `MACHINES` section. (Refer to line 6.)
4. Press the Enter key to accept the default choices regarding whether to enter the text editor (`no`) and whether to have the specified operation performed (`yes`). As requested, the first record in the `MACHINES` section is now displayed, which is the record for a machine named `SITE1` in the following sample listing. (Refer to lines 10-35 in the following listing.)
5. Select the `MACHINES` section again, by pressing the Enter key after the menu of sections. (Refer to lines 36-38.)
6. Select the `ADD` operation by entering `4` after the menu of operations. (Refer to lines 39-40.)
7. Enter the text editor by entering `y` at the prompt. (Refer to line 41.)
8. Change path names as appropriate and specify new values for four key fields:
 - `TA_TLOGSIZE` (refer to lines 50-51)
 - `TA_P MID` (refer to lines 52-53)
 - `TA_L MID` (refer to lines 54-55)
 - `TA_TYPE` (refer to lines 56-57)
9. Write (that is, save) your input and quit the editor. (Refer to lines 58-60).
10. Direct `tmconfig` to perform the operation (add the machine) by entering `y` at the prompt. (Refer to line 61.)

The following sample listing illustrates a `tmconfig` session in which a machine is being added.

Listing 3-2 Adding a Machine

```
1 $ tmconfig
2 Section: 1) RESOURCES, 2) MACHINES, 3) GROUPS 4) SERVERS
3 5)SERVICES 6) NETWORK 7) ROUTING q) QUIT 9) WSL
4 10) NETGROUPS 11) NETMAPS 12) INTERFACES [1]: 2
5 Operation: 1) FIRST 2) NEXT 3) RETRIEVE 4) ADD 5) UPDATE
6 6) CLEAR BUFFER 7) QUIT [1]:
7 Enter editor to add/modify fields [n]?
8 Perform operation [y]?
9 Return value TAOK
10 Buffer contents:
11 TA_OPERATION          4
12 TA_SECTION           1
13 TA_OCCURS             1
14 TA_PERM               432
15 TA_MAXACCESSERS      40
16 TA_MAXGTT            20
17 TA_MAXCONV           10
18 TA_MAXWSCLIENTS     0
19 TA_TLOGSIZE          100
20 TA_UID               4196
21 TA_GID               601
22 TA_TLOGOFFSET        0
23 TA_TUXOFFSET         0
24 TA_STATUS             LIBTUX_CAT:1137: Operation completed successfully
25 TA_PPID              mchnl
26 TA_LMID              SITE1
27 TA_TUXCONFIG         /home/apps/bank/tuxconfig
28 TA_TUXDIR            /home/tuxroot
29 TA_STATE             ACTIVE
30 TA_APPDIR            /home/apps/bank
31 TA_TYPE              3B2
32 TA_TLOGDEVICE        /home/apps/bank/TLOG
33 TA_TLOGNAME          TLOG
34 TA_ULOGPFX           /home/apps/bank/ULOG
35 TA_ENVFILE           /home/apps/bank/ENVFILE
36 Section: 1) RESOURCES, 2) MACHINES, 3) GROUPS 4) SERVERS
37 5)SERVICES 6) NETWORK 7) ROUTING q) QUIT 9) WSL
38 10) NETGROUPS 11) NETMAPS 12) INTERFACES [2]:
39 Operation: 1) FIRST 2) NEXT 3) RETRIEVE 4) ADD 5) UPDATE
40 6) CLEAR BUFFER 7) QUIT [1]: 4
41 Enter editor to add/modify fields [n]? y
42 491
```

3 Dynamically Modifying an Application

```
43 g/home/s//usr/p
44 TA_TUXCONFIG          /usr/apps/bank/tuxconfig
45 TA_TUXDIR             /usr/tuxroot
46 TA_APPDIR             /usr/apps/bank
47 TA_TLOGDEVICE         /usr/apps/bank/TLOG
48 TA_ULOGPFX            /usr/apps/bank/ULOG
49 TA_ENVFILE            /usr/apps/bank/ENVFILE
50 /100/s//150/p
51 TA_TLOGSIZE           150
52 /mchn1/s//mchn2/p
53 TA_PMIID              mchn2
54 /SITE1/s//SITE3/p
55 TA_LMIID              SITE3
56 /3B2/s//SPARC/p
57 TA_TYPE               SPARC
58 w
59 412
60 q
61 Perform operation [y]?
62 Return value TAUPDATED
63 Buffer contents:
64 TA_OPERATION          2
65 TA_SECTION            1
66 TA_OCCURS              1
67 TA_PERM                432
68 TA_MAXACCESSERS       40
69 TA_MAXGTT              20
70 TA_MAXCONV             10
71 TA_MAXWSCLIENTS       0
72 TA_TLOGSIZE           150
73 TA_UID                 4196
74 TA_GID                 601
75 TA_TLOGOFFSET         0
76 TA_TUXOFFSET          0
77 TA_STATUS              LIBTUX_CAT:1136: Update completed successfully
78 TA_PMIID              mchn2
79 TA_LMIID              SITE3
80 TA_TUXCONFIG          /usr/apps/bank/tuxconfig
81 TA_TUXDIR             /usr/tuxroot
82 TA_STATE               NEW
83 TA_APPDIR             /usr/apps/bank
84 TA_TYPE               SPARC
85 TA_TLOGDEVICE         /usr/apps/bank/TLOG
86 TA_TLOGNAME           TLOG
87 TA_ULOGPFX            /usr/apps/bank/ULOG
88 TA_ENVFILE            /usr/apps/bank/ENVFILE
```

How to Add a Server

1. Enter `tmconfig`.
2. To specify the `SERVERS` section of the configuration file, enter 4 after the menu of sections. (Refer to line 3 in the following sample listing.)
3. Request the `CLEAR BUFFER` operation by entering 6 after the menu of operations. (Refer to line 5 in the following sample listing.)
4. Press the Enter key to accept the default section: `SERVERS`. (Refer to lines 7-9 in the following sample listing.)
5. Request the `ADD` operation by entering 4 after the menu of operations. (Refer to lines 10-11 in the listing.)
6. Enter the text editor by entering `y` at the prompt. (Refer to line 12.)
7. Specify new values for three key fields:
 - `TA_SERVERNAME` (refer to line 15)
 - `TA_SRVGRP` (refer to line 16)
 - `TA_SRVID` (refer to line 17)
8. Write (that is, save) your input and quit the editor. (Refer to lines 19-21).
9. Direct `tmconfig` to perform the operation (add the server) by entering `y` at the prompt. (Refer to line 22.)

The following sample listing illustrates a `tmconfig` session in which a server is being added.

Listing 3-3 Adding a Server

```

1 Section: 1) RESOURCES, 2) MACHINES, 3) GROUPS 4) SERVERS
2 5)SERVICES 6) NETWORK 7) ROUTING q) QUIT 9) WSL
3 10) NETGROUPS 11) NETMAPS 12) INTERFACES [1]: 4
4 Operation: 1) FIRST 2) NEXT 3) RETRIEVE 4) ADD 5) UPDATE
5 6) CLEAR BUFFER 7) QUIT [4]: 6
6 Buffer cleared
7 Section: 1) RESOURCES, 2) MACHINES, 3) GROUPS 4) SERVERS

```

3 Dynamically Modifying an Application

```
8 5)SERVICES 6) NETWORK 7) ROUTING q) QUIT 9) WSL
9 10) NETGROUPS 11) NETMAPS 12) INTERFACES [4]:
10 Operation: 1) FIRST 2) NEXT 3) RETRIEVE 4) ADD 5) UPDATE
11 6) CLEAR BUFFER 7) QUIT [6]: 4
12 Enter editor to add/modify fields [n]? y
13 1
14 c
15 TA_SERVERNAME XFER
16 TA_SRVGRP BANKB1
17 TA_SRVID 5
18 .
19 w
20 28
21 q
22 Perform operation [y]?
23 Return value TAOK
24 Buffer contents:
25 TA_OPERATION 3
26 TA_SECTION 3
27 TA_OCCURS 1
28 TA_SRVID 5
29 TA_SEQUENCE 0
30 TA_MIN 1
31 TA_MAX 1
32 TA_RQPERM 432
33 TA_RPPERM 432
34 TA_MAXGEN 5
35 TA_GRACE 86400
36 TA_STATUS LIBTUX_CAT:1137: Operation completed successfully
37 TA_SYSTEM_ACCESS FASTPATH
38 TA_ENVFILE
39 TA_SRVGRP BANKB1
40 TA_SERVERNAME XFER
41 TA_CLOPT -A
42 TA_CONV N
43 TA_RQADDR
44 TA_REPLYQ Y
45 TA_RCMD
46 TA_RESTART Y
```

How to Activate a Newly Configured Machine

1. Enter `tmconfig`.
2. To specify the `MACHINES` section of the configuration file, enter 2 after the menu of sections. (Refer to lines 1-3 in the following sample listing.)
3. In order to select the appropriate record in the `MACHINES` section, you need to toggle through the list of machine records. To view the first machine record, select the `FIRST` operation by pressing the Enter key after the menu of operations. (Refer to lines 4-5 in the following sample listing.) If you do not want the first machine record, select the `NEXT` operation to view the next machine record by entering 2 after the menu of operations.
4. Press the Enter key to accept the default choices regarding whether to enter the text editor (*no*) and whether to have the specified operation performed (*yes*). The requested record in the `MACHINES` section is now displayed, which is the record for a machine named `SITE3` in the following sample listing. (Refer to lines 9-34 in the following listing.)
5. Select the `MACHINES` section again, by pressing the Enter key after the menu of sections. (Refer to lines 35-37.)
6. Select the `UPDATE` operation by entering 5 after the menu of operations. (Refer to lines 38-39.)
7. Enter the text editor by entering `y` at the prompt. (Refer to line 40.)
8. Change the value of the `TA_STATE` field from `NEW` to `ACTIVE`. (Refer to lines 42-45.)
9. Write (that is, save) your input and quit the editor. (Refer to lines 46-48).
10. Direct `tmconfig` to perform the operation (activate the newly configured machine) by entering `y` at the prompt. (Refer to line 49.)

3 *Dynamically Modifying an Application*

11. `tmconfig` displays the revised record for the specified machine so that you can review your change and, if necessary, edit it.
12. If the revised entry is acceptable, select 7 after the menu of operations to end the `tmconfig` session.

The following sample listing illustrates a `tmconfig` session in which a server is being activated.

Listing 3-4 Activating a New Server

```
1 Section: 1) RESOURCES, 2) MACHINES, 3) GROUPS 4) SERVERS
2   5)SERVICES  6) NETWORK 7) ROUTING q) QUIT 9) WSL
3   10) NETGROUPS 11) NETMAPS 12) INTERFACES [1]: 2
4 Operation: 1) FIRST 2) NEXT 3) RETRIEVE 4) ADD 5) UPDATE
5   6) CLEAR BUFFER 7) QUIT [1]:
6 Enter editor to add/modify fields [n]?
7 Perform operation [y]?
8 Return value TAOK
9 Buffer contents:
10 TA_OPERATION           4
11 TA_SECTION            1
12 TA_OCCURS              1
13 TA_PERM                432
14 TA_MAXACCESSERS       40
15 TA_MAXGTT              20
16 TA_MAXCONV            10
17 TA_MAXWSCLIENTS      0
18 TA_TLOGSIZE           150
19 TA_UID                 4196
20 TA_GID                 601
21 TA_TLOGOFFSET         0
22 TA_TUXOFFSET          0
23 TA_STATUS              LIBTUX_CAT:1175: Operation completed successfully
24 TA_PLMID               mchn2
25 TA_LMID                SITE3
26 TA_TUXCONFIG           /usr/apps/bank/tuxconfig
27 TA_TUXDIR              /usr/tuxroot
28 TA_STATE               NEW
29 TA_APPDIR              /usr/apps/bank
30 TA_TYPE                SPARC
31 TA_TLOGDEVICE          /usr/apps/bank/TLOG
32 TA_TLOGNAME            TLOG
33 TA_ULOGPFX             /usr/apps/bank/ULOG
34 TA_ENVFILE             /usr/apps/bank/ENVFILE
35 Section: 1) RESOURCES, 2) MACHINES, 3) GROUPS 4) SERVERS
```

How to Activate a Newly Configured Machine

```
36 5)SERVICES 6) NETWORK 7) ROUTING q) QUIT 9) WSL
37 10) NETGROUPS 11) NETMAPS 12) INTERFACES [2]:
38 Operation: 1) FIRST 2) NEXT 3) RETRIEVE 4) ADD 5) UPDATE
39 6) CLEAR BUFFER 7) QUIT [1]: 5
40 Enter editor to add/modify fields [n]? y
41 491
42 /TA_STATE
43 TA_STATE NEW
44 s/NEW/ACTIVE
45 TA_STATE ACTIVE
46 w
47 412
48 q
49 Perform operation [y]?
50 Return value TAUPDATED
51 Buffer contents:
52 .
53 .
54 .
```

How to Add a New Group

1. Enter `tmconfig`.
2. To specify the `GROUPS` section of the configuration file, enter `3` after the prompt following the list of sections. (Refer to lines 1-3 in the following sample listing.)
3. Request the `CLEAR BUFFER` operation by entering `6` after the menu of operations. (Refer to line 5 in the following sample listing.)
4. Accept the default section, `GROUPS`, by pressing the Enter key. (Refer to lines 7-9 in the following sample listing.)
5. Request the `ADD` operation by entering `4` after the menu of operations. (Refer to lines 10-11 in the listing.)
6. Enter the text editor by entering `y` at the prompt. (Refer to line 12.)
7. Specify new values for three key fields:
 - `TA_LMID` (refer to line 15)
 - `TA_SRVGRP` (refer to line 16)
 - `TA_GRPNO` (refer to line 17)

The following sample listing illustrates a `tmconfig` session in which a group is being added.

Listing 3-5 Adding a Group

```
1 Section: 1) RESOURCES, 2) MACHINES, 3) GROUPS 4) SERVERS
2 5)SERVICES 6) NETWORK 7) ROUTING q) QUIT 9) WSL
3 10) NETGROUPS 11) NETMAPS 12) INTERFACES [1]: 3
4 Operation: 1) FIRST 2) NEXT 3) RETRIEVE 4) ADD 5) UPDATE
5 6) CLEAR BUFFER 7) QUIT [4]: 6
6 Buffer cleared
7 Section: 1) RESOURCES, 2) MACHINES, 3) GROUPS 4) SERVERS
8 5)SERVICES 6) NETWORK 7) ROUTING q) QUIT 9) WSL
9 10) NETGROUPS 11) NETMAPS 12) INTERFACES [3]:
10 Operation: 1) FIRST 2) NEXT 3) RETRIEVE 4) ADD 5) UPDATE
11 6) CLEAR BUFFER 7) QUIT [6]: 4
12 Enter editor to add/modify fields [n]? y
```

```
13 1
14 c
15 TA_LMID                SITE3
16 TA_SRVGRP              GROUP3
17 TA_GRPNO               3
18 .
19 w
20 42
21 q
22 Perform operation [y]?
23 Return value TAUPDATED
24 Buffer contents:
25 TA_OPERATION           2
26 TA_SECTION             2
27 TA_OCCURS              1
28 TA_GRPNO               3
29 TA_TMSCOUNT            0
30 TA_STATUS              LIBTUX_CAT:1136: Update completed successfully
31 TA_LMID                SITE3
32 TA_SRVGRP              GROUP3
33 TA_TMSNAME
34 TA_OPENINFO
35 TA_CLOSEINFO
```

How to Change Data-dependent Routing (DDR) for an Application

1. Enter `tmconfig`.
2. To specify the `ROUTING` section of the configuration file, enter `7` after the prompt following the list of sections.
3. Toggle through the list of entries for the `ROUTING` section by selecting the `FIRST` and `NEXT` operations, which display the first and subsequent entries, respectively. Select the entry for which you want to change the `DDR`.
4. Select `5)UPDATE` from the menu of operations.

3 Dynamically Modifying an Application

5. Enter the text editor by entering `y` at the prompt.

```
Do you want to edit(n)? y
```

6. Change the values of relevant fields to the values shown in the “Sample Value” column of the following table.

Field	Sample Value	Meaning
TA_ROUTINGNAME	account_routing	Name of the routing section
TA_BUFTYPE	FML	Buffer type
TA_FIELD	account_ID	Name of the routing field
TA_RANGES	1-10:group1,***	The routing criteria being used. If, as shown here, the value of <code>account_ID</code> is between 1 and 10 (inclusive), requests are sent to the servers in group 1. Otherwise, requests are sent to any other server in the configuration.

Note: For details, see `tmconfig`, `wtmconfig(1)` in *BEA Tuxedo Command Reference*.

How to Change Application-wide Parameters

Some run-time parameters are relevant to all the components (machines, servers, and so on) of your configuration. These parameters are listed in the `RESOURCES` section of the configuration file.

An easy way to familiarize yourself with the parameters in the `RESOURCES` section is to display the first entry in that section. To do so, complete the following procedure.

1. Enter `tmconfig`.
2. Select the `RESOURCES` section, which is the default, by pressing the Enter key after the list of sections. (Refer to lines 1-3 in the following sample listing.)

3. Request the `FIRST` operation, which is the default, by pressing the Enter key after the menu of operations. (Refer to lines 4-5.)
4. When asked whether you want to edit, accept the default (`n`) by pressing the Enter key.

```
Do you want to edit(n)?
```
5. When asked whether you want the specified operation (`FIRST`) to be performed, accept the default (`y`) by pressing the Enter key.

```
Perform operation [y]?
```

The following sample listing shows a `tmconfig` session in which the first entry in the `RESOURCES` section is displayed.

Listing 3-6 Displaying the First Entry in the RESOURCES Section

```
1 Section: 1) RESOURCES, 2) MACHINES, 3) GROUPS 4) SERVERS
2 5) SERVICES 6) NETWORK 7) ROUTING q) QUIT 9) WSL
3 10) NETGROUPS 11) NETMAPS 12) INTERFACES [1]:
4 Operation: 1) FIRST 2) NEXT 3) RETRIEVE 4) ADD 5) UPDATE
5 6) CLEAR BUFFER 7) QUIT [1]:
6 Enter editor to add/modify fields [n]?
7 Perform operation [y]?
8 Return value TAOK
9 Buffer contents:
10 TA_OPERATION          1
11 TA_SECTION            0
12 TA_STATUS             Operation completed successfully
13 TA_OCCURS             1
14 TA_PERM               432
15 TA_BBLQUERY          30
16 TA_BLOCKTIME         6
17 TA_DBBLWAIT          2
18 TA_GID                10
19 TA_IPCKEY            80997
20 TA_LICMAXUSERS       1000000
21 TA_MAXACCESSERS      100
22 TA_MAXBUFSTYPE       32
23 TA_MAXBUFTYPE        16
24 TA_MAXCONV           10
25 TA_MAXDRT            0
26 TA_MAXGROUPS         100
27 TA_MAXGTT            25
28 TA_MAXMACHINES       256
29 TA_MAXQUEUEUES       36
```

3 *Dynamically Modifying an Application*

```
30 TA_MAXRFT          0
31 TA_MAXRTDATA      8
32 TA_MAXSERVERS     36
33 TA_MAXSERVICES    100
34 TA_MIBMASK        0
35 TA_SANITYSCAN     12
36 TA_SCANUNIT       10
37 TA_UID            5469
38 TA_MAXACLGROUPS  16384
39 TA_MAXNETGROUPS   8
40 TA_MAXINTERFACES  150
41 TA_MAXOBJECTS     1000
42 TA_STATE          ACTIVE
43 TA_SIGNATURE      AHEAD
44 TA_AUTHSVC
45 TA_CMTRET         COMPLETE
46 TA_DOMAINID
47 TA_LDBAL          Y
48 TA_LICEXPIRE      1998-09-15
49 TA_LICSERIAL      1234567890
50 TA_MASTER         SITE1
51 TA_MODEL          SHM
52 TA_NOTIFY         DIPIN
53 TA_OPTIONS
54 TA_SECURITY       NONE
55 TA_SYSTEM_ACCESS  FASTPATH
56 TA_USIGNAL        SIGUSR2
57 TA_PREFERENCES
58 TA_COMPONENTS     TRANSACTIONS, QUEUE, TDOMAINS, TxRPC,
59 EVENTS, WEBGUI, WSCOMPRESSION, TDOMCOMPRESSION
60 TA_SIGNATURE_REQUIRED
61 TA_ENCRYPTION_REQUIRED
62 TA_SEC_PRINCIPAL_NAME
63 TA_SEC_PRINCIPAL_LOCATION
64 TA_SEC_PRINCIPAL_PASSVAR
```

How to Change an Application Password

1. Enter `tmconfig`.
2. Select the `RESOURCES` section, which is the default, by pressing the Enter key following the list of sections. (Refer to lines 2-4 in the following sample listing.)
3. Request the `CLEAR BUFFER` operation by entering `6` after the menu of operations. (Refer to line 6.)
4. Select the `RESOURCES` section again, by pressing the Enter key after the menu of sections. (Refer to lines 8-10.)
5. Select the `UPDATE` operation by entering `5` after the menu of operations. (Refer to lines 11-12.)
6. Enter the text editor by entering `y` at the prompt. (Refer to line 13.)
7. Enter (in the buffer):

```
TA_PASSWORD    new_password
```
8. Write (that is, save) your input and quit the editor. (Refer to lines 18-20).

The following sample listing shows a `tmconfig` session in which an application password is changed to `neptune`.

Listing 3-7 Changing an Application Password

```
1 $ tmconfig
2 Section: 1) RESOURCES, 2) MACHINES, 3) GROUPS 4) SERVERS
3 5)SERVICES 6) NETWORK 7) ROUTING q) QUIT 9) WSL
4 10) NETGROUPS 11) NETMAPS 12) INTERFACES [1]:
5 Operation: 1) FIRST 2) NEXT 3) RETRIEVE 4) ADD 5) UPDATE
6 6) CLEAR BUFFER 7) QUIT [4]: 6
7 Buffer cleared
8 Section: 1) RESOURCES, 2) MACHINES, 3) GROUPS 4) SERVERS
9 5)SERVICES 6) NETWORK 7) ROUTING q) QUIT 9) WSL
10 10) NETGROUPS 11) NETMAPS 12) INTERFACES [1]:
11 Operation: 1) FIRST 2) NEXT 3) RETRIEVE 4) ADD 5) UPDATE
12 6) CLEAR BUFFER 7) QUIT [6]: 5
13 Enter editor to add/modify fields [n]? y
14 1
```

3 Dynamically Modifying an Application

```
15 c
16 TA_PASSWORD          neptune
17 .
18 w
19 49
20 q
21 Perform operation [y]?
22 Return value TAUPDATED
23 Buffer contents:
24 TA_OPERATION         1
25 TA_SECTION           0
26 TA_STATUS            Operation completed successfully
27 TA_OCCURS            1
28 TA_PERM              432
29 TA_BBLQUERY          30
30 TA_BLOCKTIME         6
31 TA_DBBLWAIT         2
32 TA_GID               10
33 TA_IPCKEY            80997
34 TA_LICMAXUSERS       1000000
35 TA_MAXACCESSERS      100
36 TA_MAXBUFSTYPE       32
37 TA_MAXBUFTYPE        16
38 TA_MAXCONV           10
39 TA_MAXDRT            0
40 TA_MAXGROUPS         100
41 TA_MAXGTT            25
42 TA_MAXMACHINES       256
43 TA_MAXQUEUES         36
44 TA_MAXRFT            0
45 TA_MAXRTDATA         8
46 TA_MAXSERVERS        36
47 TA_MAXSERVICES       100
48 TA_MIBMASK           0
49 TA_SANITYSCAN        12
50 TA_SCANUNIT          10
51 TA_UID               5469
52 TA_MAXACLGROUPS      16384
53 TA_MAXNETGROUPS      8
54 TA_MAXINTERFACES     150
55 TA_MAXOBJECTS        1000
56 TA_PASSWORD          neptune
57 TA_STATE              ACTIVE
58 TA_AUTHSVC
59 TA_CMTRET            COMPLETE
60 TA_DOMAINID
61 TA_LDBAL             Y
62 TA_LICEXPIRE         1998-09-15
63 TA_LICSERIAL         1234567890
```

64	TA_MASTER	SITE1
65	TA_MODEL	SHM
66	TA_NOTIFY	DIPIN
67	TA_OPTIONS	
68	TA_SECURITY	NONE
69	TA_SYSTEM_ACCESS	FASTPATH
70	TA_USIGNAL	SIGUSR2
71	TA_PREFERENCES	
72	TA_COMPONENTS	TRANSACTIONS , QUEUE , TDOMAINS , TxRPC , EVENTS , WEBGUI ,
73		WSCOMPRESSION , TDOMCOMPRESSION

Limitations on Dynamic Modification Using `tmconfig`

Keep in mind the following restrictions when modifying your application dynamically using `tmconfig`. Be careful about setting parameters that cannot be changed easily.

- Associated with each section is a set of key fields that are used to identify the record upon which operations are performed. (For details, see `tmconfig`, `wtmconfig(1)` in *BEA Tuxedo Command Reference*.) Key field values cannot be changed while an application is running. Normally, it is sufficient to add a new entry, with a new key field, and use it instead of the old entry. When this is done, only the new entry is used; the old entry in the configuration is not booted by the administrator.
- Generally speaking, you cannot update a parameter while the configuration component with which it is associated is booted. For example, you cannot change an entry in the `MACHINES` section while the machine associated with that entry is booted. Specifically:
 - If any server in a group is booted, you cannot change the entry for that group.
 - If a server is booted, you cannot change its name, type (conversational or not), or parameters related to its message queue. (You can change other server parameters at any time but your changes do not take effect until the next time the server is booted.)

3 *Dynamically Modifying an Application*

- You can change a `SERVICES` entry at any time, but your changes do not take effect until the next time the service is advertised.
- Updates to the `RESOURCES` section are restricted by the following conditions: the `UID`, `GID`, `PERM`, `MAXACCESSERS`, `MAXGTT`, and `MAXCONV` parameters cannot be updated in the `RESOURCES` section but can be updated on a per-machine basis; and the `IPCKEY`, `MASTER`, `MODEL`, `OPTIONS`, `USIGNAL`, `MAXSERVERS`, `MAXSERVICES`, `MAXBUFTYPE`, and `MAXBUFSTYPE` parameters cannot be changed dynamically.
- Carefully track the section of the configuration file in which you are working; `tmconfig` does not warn against performing an operation in the incorrect section. For example, if you try to update the `ENVFILE` parameter (in the `MACHINES` section) while you are working in the `RESOURCES` section, the operation appears to succeed (that is, `tmconfig` returns `TAOK`), but the change does not appear in your unloaded `UBBCONFIG` file. You can be sure an update is done only when the `TAUPDATED` status message is displayed.

In a multiple-machine configuration, always perform the following tasks:

- Specify a backup for the `MASTER` machine, along with the `MIGRATE` option (even if a need for application server migration is not anticipated).
- For `MAXSERVERS`, `MAXSERVICES`, and other parameters that define maximum limits, assign settings that are high enough to allow for sufficient growth. If your application is initially deployed on only one machine, but is expected to grow to a multiple-machine configuration, use the `MP` model, specifying the `LAN` option and a network entry for the initial machine.
- Set the parameters in the `MACHINES` section carefully because updating them requires shutting down the machine (and switching the `MASTER` to the backup in the case of the `MASTER` machine).

Tasks that Cannot Be Performed on a Running System

Most elements of the BEA Tuxedo system can be changed dynamically, through either manual intervention or automatic processes. For example, new servers can be spawned, new machines can be added, timeout parameters can be changed, and so on. A few parameters, however, cannot be changed while a system is operational:

- Any parameter that affects the size of the bulletin board is not dynamic. Most of these parameters begin with the string `MAX`, such as `MAXGTT`, which defines the maximum number of in-flight transactions allowed within the BEA Tuxedo system at any time.
- The name of a machine being used in a running application is not dynamic. New machines (that is, machines with new names) can be added, but an existing machine name cannot be changed.
- Once server executables are assigned to run on both master and backup machines, the assignment of the master and backup cannot be changed.

Note: You can configure new copies of a server executable to run on additional machines, but you cannot change existing servers with unique identifiers.

Making Temporary Modifications to Your Configuration with tadmin

When you use the `tmconfig` command to update the `TUXCONFIG` file and any bulletin board entries associated with it, the changes you make are permanent; they persist after the system is shut down and rebooted.

In some situations, however you may want to make temporary changes to a running application. For example, you may want to:

- Suspend services or servers
- Resume services or servers
- Advertise services or servers
- Unadvertise services or servers
- Change service parameters
- Change the timeout value

You can perform these tasks with the `tadmin` command, as specified in the procedures provided in this section.

How to Set Environment Variables for `tmadmin`

Before you can start a `tmadmin` session, you must set your environment variables and any required permissions. For your convenience, you may also want to select a text editor other than the default editor.

Complete the following procedure to set up your working environment properly before running `tmadmin`.

1. Log in as the BEA Tuxedo application administrator if you want to add entries to `TUXCONFIG`, or to modify existing entries. This step is not required if you only want to view existing configuration file entries without changing or adding to them.
2. Assign values to two mandatory environment variables: `TUXCONFIG` and `TUXDIR`.
 - The value of `TUXCONFIG` must be the full path name of the binary configuration file on the machine on which `tmconfig` is being run.
 - The value of `TUXDIR` must be the root directory for the BEA Tuxedo system binary files. (`tmconfig` must be able to extract field names and identifiers from `$TUXDIR/udataobj/tpadmin`.)

How to Suspend Services or Servers

To suspend a server or a service, enter the `tmadmin` and `susp` (short for `suspend`) commands, as follows.

```
$ tmadmin
> susp
```

The `suspend` command marks one of the following as inactive:

- One service
- All services of a particular queue
- All services of a particular group ID or server ID combination

After you suspend a service or a server, any requests for it that remain on the queue are handled, but no new service requests are routed to the suspended server. If a group ID or server ID combination is specified and it is part of an MSSQ set, all servers in that MSSQ set become inactive for the services specified.

How to Resume Services or Servers

To have a server or a service resume, enter the `tmadmin` and `resume` (or `res`) commands, as follows.

```
$ tmadmin
> res
```

The `resume` command undoes the effect of the `suspend` command; it marks as active for the queue one of the following:

- One service
- All services of a particular queue
- All services of a particular group ID/server ID combination

If, in this state, the group ID or the server ID is part of an MSSQ set, all servers in that MSSQ set become active for the services specified.

How to Advertise Services or Servers

To advertise a service or server, enter the following commands.

```
$ tmadmin
> adv [{"-q queue_name"} | [{"-g grp_id"} [{"-i srvid"}]] service
```

Although a service must be suspended before it may be unadvertised, you do not need to *unsuspend* a service before readvertising it. If you simply advertise a service that was unadvertised earlier, and is currently suspended, the service is unsuspending.

How to Unadvertise Services or Servers

To unadvertise a service or server, you must suspend it by entering the following commands.

```
$ tadmin  
> unadv [{"-q queue_name" | ["-g grpId" ["-i srvid"}]] service
```

Unadvertising a service has more drastic results than suspending it. When you unadvertise a service, the service table entry for it is deallocated and the cleared space in the service table becomes available to other services.

How to Change Service Parameters

The `tadmin` command allows you to change, dynamically, the values of service parameters for a specific group ID/server ID combination or for a specific queue.

The following table lists the `tadmin` commands available for changing service parameters defined in this way.

To Change	Enter the Following Commands
Load value (LOAD)	<code>\$tadmin >chl -s service_name</code>
Dequeueing priority (PRIO)	<code>\$tadmin >chp -s service_name</code>
Transaction timeout value	<code>\$tadmin >cht -s service_name</code>

The `-s` option must be specified, either on the `tadmin default` command line or on the `tadmin chl`, `chp`, or `cht` command line. Because it is possible to set the `-s` option on the `default` command line, the `-s` option is considered optional on the `chl`, `chp`, and `cht` command lines.

How to Change the Timeout Value

To change the transaction timeout (`TRANTIME`) for an interface or service with the `AUTOTRAN` flag set, run the `changetrantime (chtt)` command, as follows.

```
$ tadmin
> chtt [-m machine] {-q qaddress [-g groupname] [-i srvid]
|-g groupname -i srvid} -s service newtlim
```

You cannot change transaction timeouts begun by application clients using `tpbegin()` or `tx_set_transaction_timeout()`.

3 *Dynamically Modifying an Application*

4 Managing the Network in a Distributed Application

- Running a Network for a Distributed Application
- Compressing Data Over a Network
- Balancing Network Request Loads
- How to Use Data-Dependent Routing
- How to Change Your Network Configuration

Running a Network for a Distributed Application

Most of the work associated with running the network for a distributed application is done in the configuration or setup phase. Once you have defined the network and booted the application, the software automatically runs the network for you.

This topic describes how the BEA Tuxedo system moves data through a network, and explains how to set the configuration file parameters that control network operations.

Compressing Data Over a Network

The BEA Tuxedo system allows you to compress data being sent from one application process to another. Data compression is useful in most applications and is vital in supporting large configurations. You can use data compression when the sender and receiver of a message are on the same machine (local data compression), or when the sender and receiver of a message are on different machines (remote data compression). Both forms of compression provide advantages:

- Because messages are sent over interprocess communication (IPC) queues, the advantage of *local data compression* is that it results in lower utilization of IPC resources.
- Because messages are sent over a network, the advantage of *remote data compression* is that it results in lower utilization of network bandwidth.

How to Set the Compression Level

If you decide to use data compression, you must set the `CMPLIMIT` parameter in the `MACHINES` section of the configuration file, as follows.

```
CMPLIMIT=string_value1[,string_value2]
```

The strings that make up the value of this parameter specify the threshold message size for messages bound to remote processes (*string_value1*) and local processes (*string_value2*). Only the first string is required. The default for both strings is the value of the `MAXLONG` parameter.

In addition, you have the option of setting the `TMCMPPRFM` parameter to establish an appropriate balance between compression and CPU performance. Higher and slower compression results in more efficient network bandwidth; lower but faster compression yields less CPU utilization.

To specify the desired level of compression, complete the following procedure.

1. Set the compression threshold using the `CMPLIMIT` parameter in the `UBBCONFIG` configuration file.
2. (optional step) Set the `TMCMPPRFM` environment variable. The value of `TMCMPPRFM` must be a single digit between 1 and 9; the default is 1.

A value of 1 specifies the lowest level of compression with the fastest performance; 9 represents the highest level of compression with the slowest performance. The lower the number, the more quickly the compression routine is executed.

For more information on setting the `TMCMPPRFM` variable, refer to `tuxenv(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*.

Selecting Data Compression Thresholds

You can designate a *compression threshold* for messages: any messages larger than the threshold you specify are compressed. To designate a compression threshold, set the `CMPLIMIT` parameter. For instructions, see “How to Set the Compression Level” on page 4-2.

When choosing data compression thresholds, keep in mind the following criteria:

- Consider using *remote data compression* if your sites are running BEA Tuxedo Release 4.2.1 or later. Your setting depends on the speed of your network. You may want to assign different settings, for example, to an Ethernet network (which is a high-speed network) and an X.25 network (which is a low-speed network).
 - For a *high-speed network*, consider setting remote data compression to the lowest limit for file transfers generated by the BEA Tuxedo system. (See the note about file transfers provided later in this list.) In other words, compress only messages that are large enough to be candidates for file transfer on either the sending site or the receiving site. Note that each machine in an application may have a different limit. If this is the case, choose the lowest limit possible for each machine.
 - For a *low-speed network*, consider setting remote data compression to zero on all machines; that is, compress all application and system messages.

- Consider using *local data compression* for sites running BEA Tuxedo Release 4.2.1 or later, even if they are interoperating with pre-Release 4.2.1 sites. This results in lower utilization of IPC resources. This setting also enables you to avoid file transfers in many situations that might otherwise require a transfer and, when file transfers cannot be avoided, this setting greatly reduces the size of the files used. For more information, refer to “Message Queues and Messages” on page 1-14 in *Installing the BEA Tuxedo System*.

For local data compression, you can assign a different threshold to each machine in an application. If this is the case, always choose the lowest limit possible for each machine.

Note: For high-traffic applications that involve a large volume of timeouts and discarding of messages due to IPC queue blocking, you may want to lower the demand of the application on the IPC queuing subsystem by having local compression done at all times.

Because compression depends on the type of data being transmitted, we strongly recommend that you try different settings in your environment to determine which one yields the best results.

See Also

- `DMCONFIG(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*
- `tuxenv(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*
- “What Is Data Compression” on page 2-31 in *Introducing the BEA Tuxedo System*

Balancing Network Request Loads

If load balancing is turned on (that is, if `LDBAL` is set to `Y` in the `RESOURCES` section of the application configuration file), the BEA Tuxedo system attempts to balance requests across the network. Because load information is not updated globally, each site has a unique view of the load at remote sites.

Use the `NETLOAD` parameter in the `MACHINES` section of the configuration file (or the `TMNETLOAD` environment variable) to force more requests to be sent to local queues. The value of this parameter is a number that is added to the load for remote queues, so the remote queues appear to have more work than they do. As a result, even if load balancing is turned on, local requests are sent to local queues more often than to remote queues.

As an example, assume servers A and B offer a service with load factor 50. Server A is running on the same machine as the calling client (local), and server B is running on a different machine (remote). If `NETLOAD` is set to 100, approximately three requests will be sent to A for every one sent to B.

Another mechanism that affects load balancing is local idle server preference. Requests are always sent to a server on the same machine as the client, assuming that the server offers the desired service and is idle. This decision overrides any load balancing considerations, because the local server is known to be available immediately.

See Also

- “What Is Load Balancing” on page 2-39 in *Introducing the BEA Tuxedo System*

How to Use Data-Dependent Routing

Data-dependent routing is useful when clients issue service requests to:

- Horizontally-partitioned databases
- Rule-based servers

A horizontally-partitioned database is an information repository that is divided into segments, each of which is used to store a different category of information. This arrangement is similar to a library in which each shelf of a bookcase holds books for a different category (for example, biography, fiction, and so on).

A rule-based server is a server that determines whether service requests meet certain, application-specific criteria before forwarding them to service routines. Rule-based servers are useful when you want to handle requests that are almost identical by taking slightly different actions for business reasons.

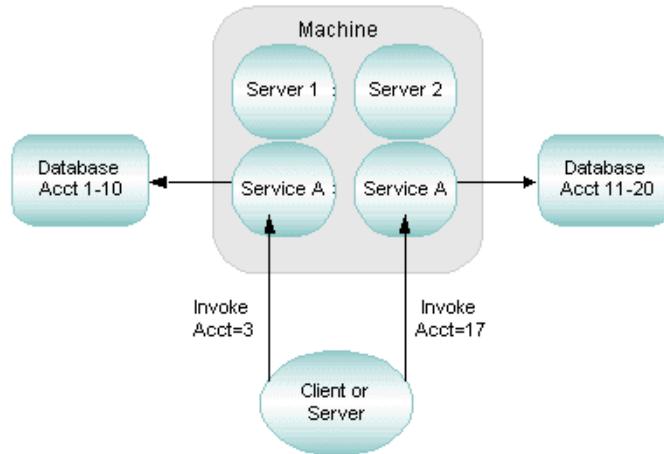
Example of Data-dependent Routing with a Horizontally-partitioned Database

Suppose two clients in a banking application issue requests for the current balance in two accounts: Account 3 and Account 17. If data-dependent routing is being used in the application, then the BEA Tuxedo system performs the following actions:

1. Gets the account numbers for the two service requests (3 and 17).
2. Checks the routing tables on the BEA Tuxedo bulletin board that show which servers handle which range of data. (In this example, server 1 handles all requests for accounts 1 through 10, and server 2 handles all requests for accounts 11 through 20.)
3. Sends each request to the appropriate server. Specifically, the system forwards the request about Account 3 to server 1, and the request about account 17 to server 2.

The following figure illustrates this process.

Figure 4-1 Data-dependent Routing with a Horizontally-partitioned Database



Example of Data-dependent Routing with Rule-based Servers

A banking application includes the following rules:

- Customers can withdraw up to \$500 without entering a special password.
- Customers must enter a special password to withdraw more than \$500.

Two clients issue withdrawal requests: one for \$100 and one for \$800. If data-dependent routing is enabled to support the withdrawal rules, then the BEA Tuxedo system performs the following actions:

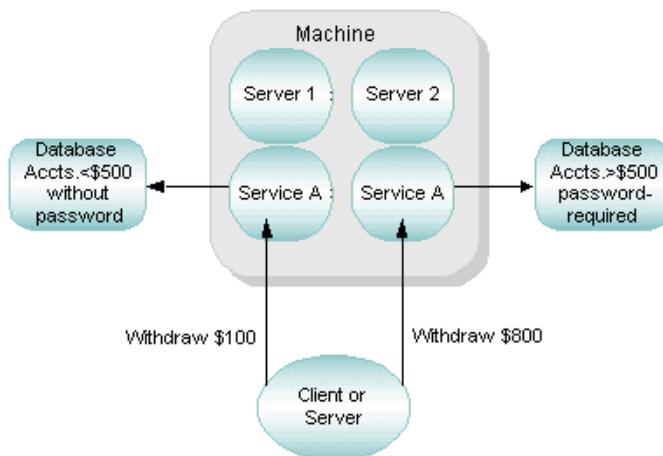
1. Gets the amount specified for withdrawal in the two service requests (\$100 and \$800).
2. Checks the routing tables on the BEA Tuxedo bulletin board that show which servers handle requests for the amount being requested. (In this example, server 1 handles all requests to withdraw amounts up to \$500; server 2 handles all requests to withdraw amount over \$500.)

4 Managing the Network in a Distributed Application

3. Sends each request to the appropriate server. Specifically, the system forwards the request for \$100 to server 1 and the request for \$800 to server 2.

The following figure illustrates this process.

Figure 4-2 Data-dependent Routing with Rule-Based Servers



See Also

- “What Is Data-dependent Routing” on page 2-32 in *Introducing the BEA Tuxedo System*

How to Change Your Network Configuration

To change configuration parameters while your application is running, run the `tmconfig(1)` command. This command is a shell-level interface to the BEA Tuxedo System Management Information Base (MIB).

Using `tmconfig`, you can browse and modify the `TUXCONFIG` file without bringing down your system. For example, you can add new components, such as machines and servers, while your application is running.

See Also

- “Using `tmconfig` to Modify Your Configuration” on page 3-21 in *Introducing the BEA Tuxedo System*
- `tmconfig`, `wtmconfig(1)` in *BEA Tuxedo Command Reference*
- `MIB(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*
- `TM_MIB(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*
- “Administering Link-Level Encryption” on page 2-35 in *Using BEA Tuxedo Security*
- “Administering Public Key Security” on page 2-41 in *Using BEA Tuxedo Security*

4 *Managing the Network in a Distributed Application*

5 About the EventBroker

- What Is an Event
- Differences Between Application-defined and System-defined Events
- What Is the EventBroker
- How the EventBroker Works
- What Are the Benefits of Brokered Events

What Is an Event

An event is a state change or other occurrence in a running application (such as a network connection being dropped) that may require intervention by an operator, an administrator, or the software. The BEA Tuxedo system reports two types of events:

- *System-defined events*, which are situations (primarily failures) defined by the BEA Tuxedo system, such as the exceeding of certain system capacity limits, server terminations, security violations, and network failures.
- *Application-defined events*, which are situations defined by a customer application, such as the ones listed in the following table.

In an application for this type of business . . .	An occurrence of this situation may be defined as an “event” . . .
Stock brokerage	A stock is traded at or above a specified price.

In an application for this type of business . . .	An occurrence of this situation may be defined as an “event” . . .
Banking	A withdrawal or deposit above a specified amount is made. The cash available in an ATM machine drops below a specified amount.
Manufacturing	An item is out of stock.

Application events are occurrences of application-defined events, and *system events* are occurrences of system-defined events. Both application and system events are received and distributed by the BEA Tuxedo EventBroker component.

Differences Between Application-defined and System-defined Events

Application-defined events are defined by application designers and are therefore application specific. Any of the events defined for an application may be tracked by the client and server processes running in the application.

System-defined events are defined by the BEA Tuxedo system code and are generally associated with objects defined in `TM_MIB(5)`. A complete list of system-defined events is published on the `EVENTS(5)` reference page. Any of these events may be tracked by users of the BEA Tuxedo system.

The BEA Tuxedo EventBroker posts both application-defined and system-defined events, and an application can subscribe to events of both types. The two types of events can be distinguished by their names: the names of system-defined events begin with a dot (.); the names of application-specific events cannot begin with a dot (.).

What Is the EventBroker

The BEA Tuxedo EventBroker is a tool that provides asynchronous routing of application events among the processes running in a BEA Tuxedo application. It also distributes system events to whichever application processes want to receive them.

The EventBroker performs the following tasks:

- Monitors events and notifies subscribers when events are posted via `tpost(3c)`.
- Keeps an administrator informed of changes in an application.
- Provides a system-wide summary of events.
- Provides a tool through which an event can trigger a variety of notification activities.
- Provides a filtering capability, providing additional conditions to the posted event's buffer.

Note: For a sample application that you can copy and run as a demo, see “Tutorial for bankapp, a Full C Application” on page 3-1 in *Tutorials for Developing a BEA Tuxedo Application*.

The EventBroker recognizes over 100 meaningful state transitions to a MIB object as system events. A posting for a system event includes the current MIB representation of the object on which the event occurred and some event-specific fields that identify the event that occurred. For example, if a machine is partitioned, an event is posted with the following:

- The name of the affected machine, as specified in the `T_MACHINE` class, with all the attributes of that machine
- Some event attributes that identify the event as *machine partitioned*

You can use the EventBroker simply by subscribing to system events. Then, instead of having to query for MIB records, you can be informed automatically when events occur in the MIB by receiving `FML` data buffers representing MIB objects.

How the EventBroker Works

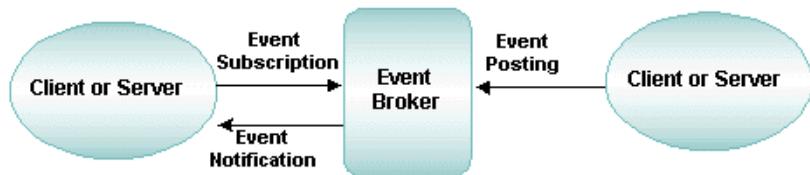
The BEA Tuxedo EventBroker is a tool through which an arbitrary number of *suppliers* of event notifications can post messages for an arbitrary number of *subscribers*. The suppliers of such notifications may be application or system processes operating as clients or servers. The subscribers of such notifications may be administrators or application processes operating as clients or servers.

Client and server processes using the EventBroker communicate with one another based on a set of *subscriptions*. Each process sends one or more subscription requests to the EventBroker, identifying the event types that the process wants to receive. The EventBroker, in turn, acts like a newspaper delivery person who delivers newspapers only to customers who have paid for a subscription. For these reasons, the paradigm on which the EventBroker is based is described as *publish-and-subscribe* communication.

Event suppliers (either clients or servers) notify the EventBroker of events as they occur. We refer to this type of notification as *posting* an event. Once an event supplier posts an event, the EventBroker matches the posted event with the subscribers that have subscribed for that event type. Subscribers may be administrators or application processes. When the EventBroker finds a match, it takes the action specified for each subscription; subscribers are notified and any other actions specified by subscribers are initiated.

The following diagram shows how the EventBroker handles event subscriptions and postings.

Figure 5-1 Posting and Subscribing to an Event

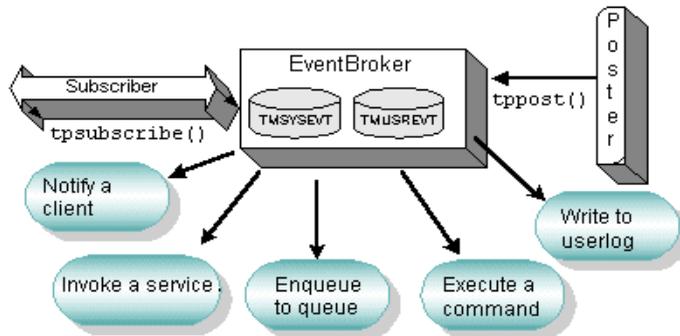


As the administrator for your BEA Tuxedo application, you can enter subscription requests on behalf of client and server processes through calls to the `T_EVENT_COMMAND` class of the `EVENT_MIB(5)`. You can also invoke the `tpssubscribe(3c)` function to subscribe, programmatically, to an event by using the EventBroker.

Event Notification Methods

The EventBroker subscription specifies one of the notification methods shown in the following diagram.

Figure 5-2 Supported Notification Methods



- *Notify a client*—The EventBroker keeps track of a client’s interest in particular events and notifies the client, without being prompted, when such an event occurs. For this reason, this method is called *unsolicited notification*.
- *Invoke a service*—If a subscriber wants event notifications to be passed to service calls, the subscriber process should invoke the `tpssubscribe()` function to provide the name of the service to be called.
- *Enqueue message to stable-storage queues*—For subscriptions with requests to send event notifications to stable-storage queues, the EventBroker will obtain a queue space, queue name, and correlation identifier. A subscriber specifies a queue name when subscribing to an event. The correlation identifier can be used to differentiate among multiple subscriptions for the same event expression and filter rule, that are destined for the same queue.
- *Execute a command*—When an event is posted, the buffer associated with it is transformed into a system command that is then executed. For example, the buffer may be changed to a system command that sends an email message. This process must be executed through the MIB.

- *Write messages to the userlog*—When events are detected and matched by the EventBroker, the specified messages are written to the userlog, or ULOG. This process must be executed through the MIB.

Severity Levels of System Events

The EventBroker assigns one of three levels of severity to system events such as server terminations or network failure.

The level of severity is . . .	When the EventBroker is informed of . . .
ERROR	An abnormal occurrence, such as a server being terminated or a network connection being dropped.
INFO (short for “Information”)	A state change resulting from a process or a change in the configuration.
WARN (short for “Warning”)	The fact that a client has not been allowed to join the application because it failed authentication. A configuration change that threatens the performance of the application has occurred.

What Are the Benefits of Brokered Events

- *Anonymous communication*—The Event Broker enables BEA Tuxedo programs to subscribe to events in which they are interested and it keeps track of all subscriptions. Therefore, a subscriber to one event does not need to know which programs subscribe to the same event, and a poster of an event does not need to know which other programs subscribe to that event. This anonymity allows subscribers to come and go without synchronizing with posters.
- *Decoupling of exception conditions*—A publish-and-subscribe communication model allows the software detecting an exception condition to be decoupled from the software handling the exception condition.

- *Tight integration with the BEA Tuxedo system*—The Event broker retains functionality such as message buffers, messaging paradigms, distributed transactions, and ACL permission checks for event postings.
- *Variety of notification methods*—When a client or server subscribes to a system event (such as the termination of a server) or an application event (such as an ATM machine running out of money), it specifies an action that the EventBroker should take when it is notified that the target event has occurred.

If the subscriber is a BEA Tuxedo client, it can do one of the following at the time it subscribes:

- Request unsolicited notification
- Name a service routine that should be invoked
- Name an application queue in which the EventBroker should store the data for later processing

If the subscriber is a BEA Tuxedo server, it can do one of the following at the time it subscribes:

- Specify a service request
- Name an application queue in which the EventBroker should store the data

See Also

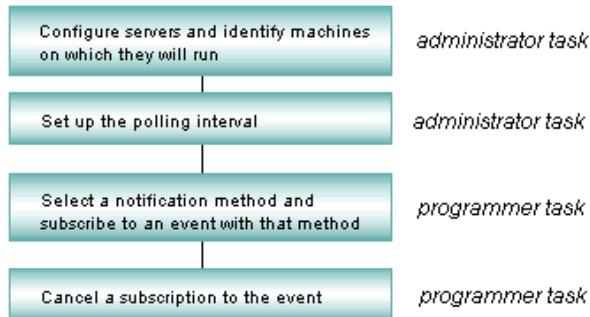
- “Subscribing to Events” on page 6-1
- “Subscribing to an Event” on page 3-16 in *Introducing the BEA Tuxedo System*
- `EVENT_MIB(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*
- `tpsubscribe(3c)` in *BEA Tuxedo C Function Reference*
- `tpunsubscribe(3c)` in *BEA Tuxedo C Function Reference*

6 Subscribing to Events

- Process of Using the EventBroker
- How to Configure EventBroker Servers
- How to Set the Polling Interval
- Subscribing, Posting, and Unsubscribing to Events with the ATMI and the EVENT_MIB
- How to Select a Notification Method
- How to Cancel a Subscription to an Event
- How to Use the EventBroker with Transactions

Process of Using the EventBroker

Use of the EventBroker requires the completion of several preparatory steps. The following flowchart lists these steps and indicates whether each step should be performed by an application administrator or programmer.



For instructions on any of these tasks, click on the appropriate box in the flowchart.

Note: A good way to learn how the EventBroker works is by running `bankapp`, the sample application delivered with the BEA Tuxedo system. To find out how to copy `bankapp` and run it as a demo, see “Tutorial for `bankapp`, a Full C Application” on page 3-1 in *Tutorials for Developing a BEA Tuxedo Application*.

How to Configure EventBroker Servers

A client accesses the EventBroker through either of two servers provided by the BEA Tuxedo system: `TMSUSREVT(5)`, which handles application events, and `TMSYSEVT(5)`, which handles system events. Both servers process events and trigger the sending of notification to subscribers.

To set up the BEA Tuxedo EventBroker on your system, you must configure either or both of these servers in the `SERVERS` section of the `UBBCONFIG` file, as shown in the following 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"

TMSUSREVT 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"
```

We recommend that you assign the principal server to the `MASTER` site, even though either server can reside anywhere on your network.

Note: You can reduce the network traffic caused by event postings and notifications by assigning secondary servers to other machines in your network.

How to Set the Polling Interval

Periodically, the secondary server polls the primary server to obtain the current subscription list, which includes filtering and notification rules. By default, polling is done every 30 seconds. If necessary, however, you can specify a different interval.

You can configure the polling interval (represented in seconds) with the `-p` command-line option in `TMUSREVT(5)` or `TMSYSEVT(5)` entries in the configuration file.

```
-p poll_seconds
```

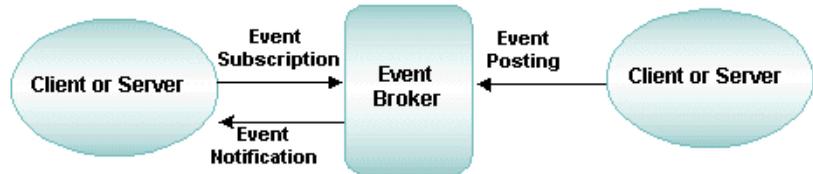
It may appear that event messages are lost while subscriptions are being added and secondary servers are being updated.

Subscribing, Posting, and Unsubscribing to Events with the ATMI and the EVENT_MIB

As the administrator for your BEA Tuxedo application, you can enter subscription requests on behalf of a client or server process through calls to the `T_EVENT_COMMAND` class of the `EVENT_MIB(5)`. You can also use invoke the `tpsubscribe(3c)` function to subscribe, programmatically, to an event.

The following figure shows how clients and servers use the EventBroker to subscribe to events, to post events, and to unsubscribe to events.

Figure 6-1 Subscribing to an Event



Identifying Event Categories Using `eventexpr` and `filter`

Clients or servers can subscribe to events by calling `tpsubscribe(3c)`. The `tpsubscribe()` function takes one required argument: `eventexpr`. The value of `eventexpr` can be a wildcard string that identifies the set of event names about which the user wants to be notified. Wildcard strings are described on the `recomp`, `rematch(3c)` reference page in *BEA Tuxedo C Function Reference*.

As an example, a user on a UNIX system platform who wants to be notified of all events related to the category of networking can specify the following value of `eventexpr`.

```
\.SysNetwork.*
```

The backslash preceding the period (.) indicates that the period is literal. (Without the preceding backslash, the period (.) would match any character except the end-of-line character.) The combination `.*` at the end of `\.SysNetwork.*` matches zero or more occurrences of any character except the end-of-line character.

In addition, clients or servers can filter event data by specifying the optional `filter` argument when calling `tpsubscribe()`. The value of `filter` is a string containing a boolean filter rule that must be evaluated successfully before the EventBroker posts the event.

As an example, a user who wants to be notified *only* about system events having a severity level of `ERROR` can specify the following value of `filter`.

```
"TA_EVENT_SEVERITY='ERROR' "
```

When an event name is posted that evaluates successfully against *eventexpr*, the EventBroker tests the posted data against the filter rule associated with *eventexpr*. If the data passes the filter rule or if there is no filter rule for the event, the subscriber receives a notification along with any data posted with the event.

Accessing the EventBroker

Your application can access the EventBroker through either the ATMI or the `EVENT_MIB(5)`. The following table describes both methods.

Method	Function	Purpose
ATMI	<code>tpost(3c)</code>	Notifies the EventBroker, or posts an event and any accompanying data. The event is named by the <i>eventname</i> argument and the <i>data</i> argument, if not NULL, points to the data. The posted event and data are dispatched by the BEA Tuxedo EventBroker to all subscribers with subscriptions that successfully evaluate against <i>eventname</i> and optional filter rules that successfully evaluate against <i>data</i> .
	<code>tpsubscribe(3c)</code>	Subscribes to an event or a set of events named by <i>eventexpr</i> . Subscriptions are maintained by the BEA Tuxedo EventBroker, and are used to notify subscribers when events are posted via <code>tpost()</code> . Each subscription specifies one of the following notification methods: client notification, service calls, message enqueueing to stable-storage queues, executing of commands, and writing to the userlog. Notification methods are determined by the subscriber's process type (that is, whether the process is a client or a server) and the arguments passed to <code>tpsubscribe()</code> .
	<code>tpunsubscribe(3c)</code>	Removes an event subscription or a set of event subscriptions from the BEA Tuxedo EventBroker's list of active subscriptions. <i>subscription</i> is an event subscription handle returned by <code>tpsubscribe()</code> . Setting <i>subscription</i> to the wildcard value, -1, directs <code>tpunsubscribe</code> to unsubscribe to all nonpersistent subscriptions previously made by the calling process. Non-persistent subscriptions are those made without the <code>TPEVPERSIST</code> bit setting in the <code>ctl->flags</code> parameter of <code>tpsubscribe()</code> . Persistent subscriptions can be deleted only by using the handle returned by <code>tpsubscribe()</code> .

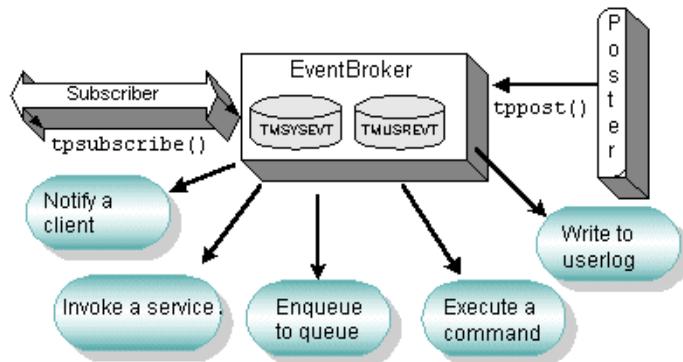
Method	Function	Purpose
EVENT_MIB(5)	N/A	<p>The EVENT_MIB is a management information base (MIB) that stores subscription information and filtering rules. In your own application, you cannot define new events for the BEA Tuxedo EventBroker using EVENT_MIB, but you can customize the EventBroker to track events and notify subscribers of occurrences of special interest to the application.</p> <p>You can use the EVENT_MIB to subscribe to an event, or to modify or cancel a subscription.</p>

Note: `tpost(3c)`, `tpsubscribe(3c)`, and `tpunsubscribe(3c)` are C functions. Equivalent routines (`TPPOST(3cb1)`, `TPSUBSCRIBE(3cb1)`, and `TPUNSUBSCRIBE(3cb1)`) are provided for COBOL programmers. See *BEA Tuxedo C Function Reference* and *BEA Tuxedo COBOL Function Reference* for details.

How to Select a Notification Method

The EventBroker supports a variety of methods for notifying subscribers of events, as shown in the following diagram.

Figure 6-2 Notification Methods Supported by the EventBroker



Whichever notification method you choose, the procedure for implementing it is the same: in your call to `tpssubscribe()`, specify an argument that refers to a structure of type `TPEVCTL`.

If the value of the argument is `NULL`, the EventBroker sends an unsolicited message to the subscriber. Two of these methods, having the notification sent to a service and having it sent to a queue in stable storage, cannot be requested directly by a client. Instead, a client must invoke a service routine to subscribe on its behalf.

For each subscription, you can select any of the following notification methods. The EventBroker can:

- *Notify the client*—The EventBroker keeps track of events in which the client is interested and sends unsolicited notifications to the client when they occur. Some events are anonymously posted. A client can join an application, regardless of whether any other clients have subscribed, and post events to the EventBroker. The EventBroker matches these events against its database of subscriptions and sends an unsolicited notification to the appropriate clients. (See the definition of the `T_EVENT_CLIENT` class in the `EVENT_MIB(5)` entry in *BEA Tuxedo File Formats and Data Descriptions Reference*.)
- *Invoke a service*—If a subscriber wants event notifications to be sent to service calls, then the `ctl` parameter must point to a valid `TPEVCTL` structure. (See the definition of the `T_EVENT_SERVICE` class in the `EVENT_MIB(5)` entry in *BEA Tuxedo File Formats and Data Descriptions Reference*.)
- *Enqueue messages to stable-storage queues*—For subscriptions to stable-storage queues, a queue space, queue name, and correlation identifier are specified, in addition to values for `eventexpr` and `filter`, so that matching can be performed. The correlation identifier can be used to differentiate among several subscriptions characterized by the same event expression and filter rule, and destined for the same queue. (See the definition of the `T_EVENT_QUEUE` class in the `EVENT_MIB(5)` entry in *BEA Tuxedo File Formats and Data Descriptions Reference*.)
- *Execute commands*—Using the `T_EVENT_COMMAND` class of the `EVENT_MIB`, subscribers can invoke an executable process. When a match is found, the data is used as the name of the executable process and any required options. (See the definition of the `T_EVENT_COMMAND` class in the `EVENT_MIB(5)` entry in *BEA Tuxedo File Formats and Data Descriptions Reference*.)
- *Write messages to the userlog (ULOG)*—Using the `T_EVENT_USERLOG` class of the `EVENT_MIB`, subscribers can write system `USERLOG` messages. When

events are detected and matched, they are written to the USERLOG. (See the definition of the T_EVENT_USERLOG class in the EVENT_MIB(5) entry in *BEA Tuxedo File Formats and Data Descriptions Reference*.)

How to Cancel a Subscription to an Event

When a client leaves an application by calling `tpterm(3c)`, all of its subscriptions are *canceled* unless the subscription is specified as persistent. (If persistent, the subscription continues to receive postings even after a client performs a `tpterm()`.) If the client later rejoins the application and wants to renew those subscriptions, it must subscribe again.

A well-behaved client unsubscribes before calling `tpterm()`. This is accomplished by issuing a `tpunsubscribe(3c)` call before leaving an application.

How to Use the EventBroker with Transactions

Special handling is needed to use the EventBroker with transactions.

- Before you can use the EventBroker with transactions, you must configure the `NULL_TMS` parameter with the `TMUSREVT(5)` server for the server groups in which the EventBroker is running.
- The advantage of posting an event in a transaction is that all of the work, including work not related to the posting, is guaranteed to be complete if the transaction is successful. If any work performed within the transaction fails, it is guaranteed that all the work done within the transaction will be rolled back. The disadvantage is that the poster takes a risk that something may cause the transaction to be aborted, and the posting will be lost.

- To specify that a subscription is part of a transaction, use the `TPEVTRAN` flag with `tpssubscribe(3c)`. If the subscription is made transactionally, the action taken in response to an event will be part of the caller's transaction.
Note: This method can be used only for subscriptions that cause a BEA Tuxedo service to be invoked, or that cause a record to be enqueued on a permanent queue.

How Transactions Work with the EventBroker

If both a poster and a subscriber agree to link their transactions, they create a form of voting. The poster makes an assertion that something is true and infects the message with this transaction. (In other words, the message that leaves the originating process is marked as being associated with the transaction.) The transaction goes to the EventBroker.

The EventBroker's actions, such as calling the service or putting a message in the queue for the subscriber, are also part of the same transaction. If a service routine that is running encounters an error, it can fail the transaction, rolling back everything, including all other transactional subscriptions and the poster's original transaction, which might have invoked other services and performed other database work. The poster makes an assertion ("I'm about to do this"), provides data, and links the data to its transaction.

A number of anonymous subscribers, that is, subscribers about which the poster knows nothing, are invoked transactionally. If any subscriber fails to link its work with the poster's work, the whole transaction is rolled back. All transactional subscribers must agree to link their work with the poster's work, or all the work is rolled back. If a poster has not allowed the posting to participate in its transaction, the EventBroker starts a separate transaction, and gathers all the transactional subscriptions into that transaction. If any of these transactions fail, all the work done on behalf of the transactional subscriptions is rolled back, but the poster's transaction is not rolled back. This process is controlled by the `TPEVTRAN` flag.

Example of Using the EventBroker with Transactions

A stock trade is about to be completed by a brokerage application. A number of database records have been updated by various services during the trade transaction. A posting states that the trade is about to happen.

An application responsible for maintaining an audit trail of such trades has subscribed to this event. Specifically, the application has requested the placement of a record in a specified queue whenever an event of this type is posted. A service routine responsible for determining whether trades can be performed, also subscribes to this type of event; it, too, is notified whenever such a trade is proposed.

If all goes well, the trade is completed and an audit trail is made.

If an error occurs in the queue and no audit trail can be made, the entire stock trade is rolled back. Similarly, if the service routine fails, the transaction is rolled back. If all is successful, the trade is made and the transaction is committed.

See Also

- “What Is the EventBroker” on page 5-3
- “How the EventBroker Works” on page 2-10 in *Introducing the BEA Tuxedo System*
- “What Is an Event” on page 3-15 in *Introducing the BEA Tuxedo System*
- “Using Event-based Communication” on page 1-14 in *Tutorials for Developing a BEA Tuxedo Application*
- `recomp`, `rematch(3c)`, `tpost(3c)`, `tpssubscribe(3c)`, and `tpunsubscribe(3c)` in *BEA Tuxedo C Function Reference*
- `TPPOST(3cb1)`, `TPSUBSCRIBE(3cb1)`, and `TPUNSUBSCRIBE(3cb1)` in *BEA Tuxedo COBOL Function Reference*
- `EVENT_MIB(5)`, `EVENTS(5)`, `TMSYSEVT(5)`, and `TMUSREVT(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*

7 Migrating Your Application

- What Is Migration
- Migration Options
- How to Switch the Master and Backup Machines
- How to Migrate Server Groups
- How to Migrate Server Groups from One Machine to Another
- How to Cancel a Migration
- How to Migrate Transaction Logs to a Backup Machine

What Is Migration

Under normal circumstances, an administrator performs daily administrative tasks on the configured `MASTER` machine. The `DBBL` on the `MASTER` machine monitors other machines in a configuration, handles configuration updates, and broadcasts dynamic changes to the `TMI`. If the `MASTER` machine fails, for example, due to a machine crash, database corruptions, BEA Tuxedo system problems, network partitioning, or application faults, the application does not stop running. Clients can still join the application, servers can still service requests, and naming is still available on each local machine. However, until the `MASTER` machine is restored, servers cannot be activated or deactivated, and an administrator cannot dynamically reconfigure the system.

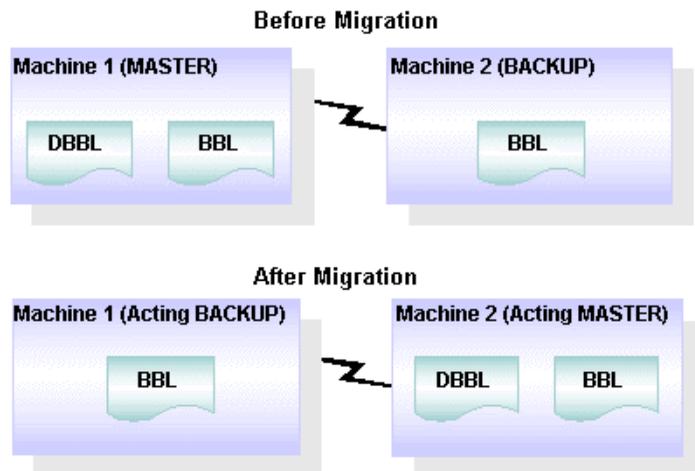
Similarly, application servers are configured to run on specific machines to service client requests. However, if a machine fails or must be brought down to be serviced, the servers on that machine become unavailable. In each case, you can migrate the servers to a configured `BACKUP` or alternate machine.

An administrator who performs a migration in preparation for shutting down a machine for service or upgrading, does not face the problems inherent in a machine failure. Therefore an administrator in this situation has a relatively high degree of control over migration activities.

Performing a Master Migration

A master migration is the process of moving the `DBBL` from the configured `MASTER` machine to the configured `BACKUP` machine so that servers can continue to be serviced while the configured `MASTER` machine is down. To start a migration, an administrator requests that the configured `BACKUP` assume the role of acting `MASTER`, and the configured `MASTER`, the role of acting `BACKUP`. The acting `MASTER` then performs all administrative functions: it begins monitoring other machines in the configuration and accepts any dynamic reconfiguration changes.

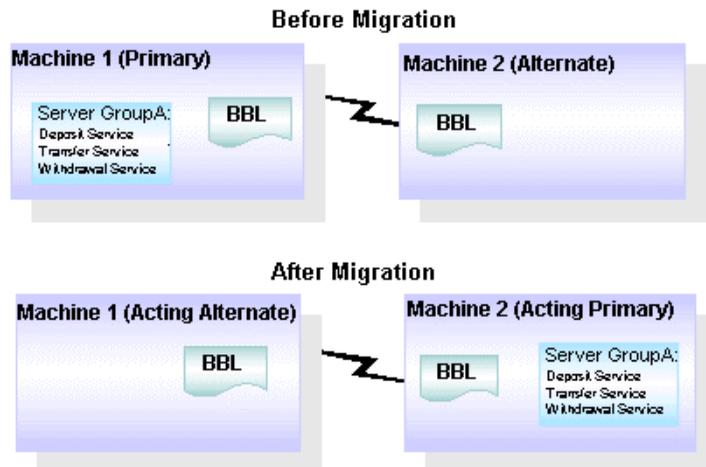
In the following illustration, Machine 2, the configured `BACKUP` machine, assumes the role of `MASTER`, while Machine 1, the configured `MASTER`, assumes the role of acting `BACKUP`. When the configured `MASTER` is available again, it can be reactivated from the acting `MASTER` (that is, the configured `BACKUP`). The configured `MASTER` then regains control as acting `MASTER`.



Migrating a Server Group

For each group of servers, an administrator specifies a primary machine and an alternate machine. The process of migrating a server group involves activating the server group on the alternate machine.

In the following illustration, GroupA is assigned to Machine 1 (that is, Machine 1 is configured as the primary machine); Machine 2 is configured as the alternate machine for GroupA. After migration, GroupA is activated on Machine 2, which means that all servers in this group and the services associated with them, are available on Machine 2 (the acting primary).



Migrating Machines

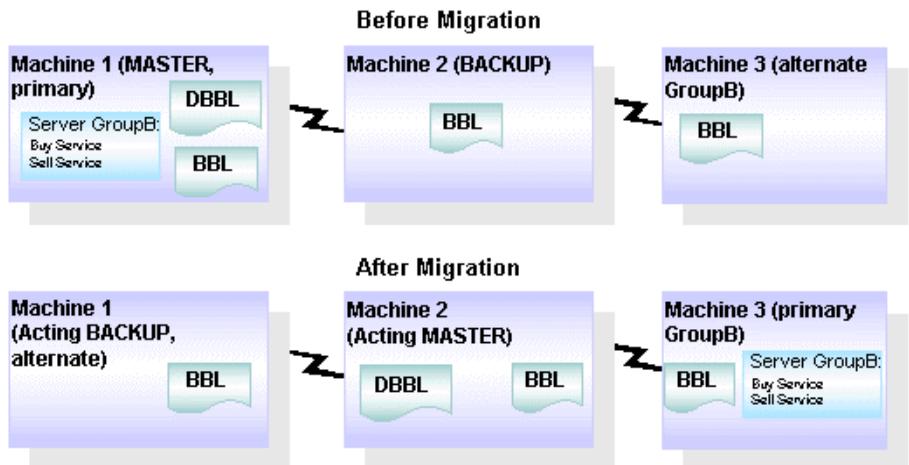
While it is sometimes useful to migrate only a single server group, it is more often necessary to migrate an entire machine. This type of migration may be necessary, for example, when a computer fails. Migrating a machine involves migrating each of the server groups running on the machine. An alternate machine must be configured for each server group.

Performing a Scheduled Migration

In a controlled situation, such as when a computer needs to be off-line for a while, or needs to be upgraded, an administrator can preserve information about the current configuration for servers and services, and use that information when activating servers on alternate machines. Such use of configuration information is possible because server entries are retained on a primary machine, even after the servers are deactivated and become unavailable in response to a request for a migration.

You can migrate an entire server group or an entire machine. Migration of an entire machine is possible when the same machine is configured as the alternate for all the server groups on a primary machine. When that is not the case (that is, when different alternate machines are configured for different server groups on a primary machine), then the servers must be migrated by group, rather than by machine.

In the following illustration, Machine 1 is the configured `MASTER` and the primary machine for GroupB; Machine 2 is the configured `BACKUP`. Server GroupB is configured with Machine 1 as its primary machine and Machine 3 as its alternate. If Machine 1 is taken down, Machine 2 becomes the acting `MASTER`, and Server GroupB is deactivated, migrated to its alternate (Machine 3), and reactivated.



After deactivating all the servers in a group, you can migrate the group from the acting primary to the acting alternate. You do not need to specify which servers are running, which services are currently advertised, or which, if any, dynamic configuration

changes are being made. The configured alternate machine obtains this information from the configuration information for the servers that is available on the configured primary machine, when the servers are deactivated. If data-dependant routing is being used and will continue to be used on the alternate machine, services are routed on the basis of the target group name, instead of the target machine name.

Whether you need to migrate an entire application or only portions of it, be sure to make the necessary changes with minimal service disruption. The integrity of all machines, networks, databases, and other components of your application must remain intact. The BEA Tuxedo system provides a way to migrate an application while preserving the integrity of all its components.

Migration Options

The BEA Tuxedo system allows you to migrate:

- [A MASTER machine to a BACKUP machine, and vice-versa](#)
- [A server group from its primary machine to its alternate machine](#)
- [All server groups on a primary machine to an alternate machine](#)
- [A transaction log](#)

You can also cancel a migration.

By migrating a combination of the application components listed here, and using the system utilities for recovering a partitioned network, you can migrate entire machines.

How to Switch the Master and Backup Machines

When a MASTER machine must be shut down for maintenance, or is no longer accessible due to an unanticipated problem (such as a partitioned network), then you must transfer the work of the MASTER to a configured BACKUP machine.

Note: Before you can migrate the MASTER, both the MASTER and BACKUP machines must be running the same release of the BEA Tuxedo system software.

This type of switching is done by migrating the DBBL from the MASTER to the BACKUP. To migrate the DBBL, enter the following command.

```
tmadmin master
```

In most cases, you need to migrate application servers to alternate sites, or restore the MASTER machine. For more detail about the `tmadmin` command, see the `tmadmin(1)` reference page in *BEA Tuxedo File Formats and Data Descriptions Reference*.

Examples of Switching MASTER and BACKUP Machines

The following two sample `tmadmin` sessions show how to switch MASTER and BACKUP machines regardless of whether the MASTER machine is accessible from the BACKUP machine. In the first example, the MASTER machine is accessible, so the DBBL process is migrated from the MASTER to the BACKUP.

Listing 7-1 Switching MASTER and BACKUP When MASTER Is Accessible from BACKUP

```
$ tmadmin
tmadmin - Copyright © 1987-1990 AT&T; 1991-1993 USL. All rights reserved.
> master
are you sure? [y,n] y
Migrating active DBBL from SITE1 to SITE2, please wait...
DBBL has been migrated from SITE1 to SITE2
> q
```

In the second example, because the MASTER machine is not accessible from the BACKUP machine, the DBBL process is created on the BACKUP machine.

Listing 7-2 Switching MASTER and BACKUP When MASTER Is Not Accessible from BACKUP

```
$ tmadmin
tmadmin - Copyright © 1987-1990 AT&T; 1991-1993 USL. All rights reserved.
TMADMIN_CATT:199: Cannot become administrator. Limited set of commands available.
> master
are you sure? [y,n] y
Creating new DBBL on SITE2, please wait... New DBBL created on SITE2
> q
```

How to Migrate Server Groups

1. Configure an alternate location in the LMID parameter (for the server group being migrated) in the GROUPS section of the UBBCONFIG file. Servers in the group must specify RESTART=Y and the MIGRATE option must be specified in the RESOURCES section of the UBBCONFIG file.

2. If you are planning to migrate a group of servers, shut down each server in the group by issuing the following command.

```
tmshutdown -R -g groupname
```
3. Start a `tmadmin` session by entering the following command.

```
tmadmin
```
4. At the `tmadmin` prompt, enter one of the following commands:
 - To migrate all the servers in a single group, enter

```
migrategroup(migg)
```

This command takes the name of a single server group as an argument.
 - To migrate all the server groups on a machine (as specified by an LMID), enter

```
migratemach(migm)
```
5. If transactions are being logged for a server being migrated as part of a group, you may need to move the `TLOG` to the `BACKUP` machine, load it, and “warm start” it.

How to Migrate a Server Group When the Alternate Machine Is Accessible from the Primary Machine

To migrate a server group when the alternate machine is accessible from the primary machine, complete the following procedure.

1. Shut down the `MASTER` machine by entering the following command.

```
tmshutdown -R -g groupname
```
2. On the primary machine, start a `tmadmin` session by entering the following command.

```
tmadmin
```
3. Migrate the appropriate group by entering the following command.

```
migrategroup groupname
```

4. If necessary, migrate the transaction log.
5. If necessary, migrate the application data.

How to Migrate a Server Group When the Alternate Machine Is Not Accessible from the Primary Machine

To migrate a server group when the alternate machine is not accessible from the primary machine, switch the `MASTER` and `BACKUP` machines, if necessary.

1. On the alternate machine, start a `tmadmin` session by entering the following command.

```
tmadmin
```

2. Request cleanup and restart of any servers on the primary machine that require these operations by entering the following command.

```
pclean primary_machine
```

3. Transfer the appropriate server group to a configured alternate machine by entering the following command:

```
migrate groupname
```

4. Boot the newly migrated server group by entering the following command.

```
boot -g groupname
```

Examples of Migrating a Server Group

The following two sample sessions show how you can migrate a server group, regardless of whether the alternate machine is accessible from the primary machine. In the first example, the alternate machine is accessible from the primary machine.

Listing 7-3 Migrating a Group When the Alternate Machine Is Accessible from the Primary Machine

```
$ tmsshutdown -R -g GROUP1
Shutting down server processes...
Server ID = 1 Group ID = GROUP1 machine = SITE1: shutdown succeeded
1 process stopped.
$ tadmin
tadmin - Copyright © 1987-1990 AT&T; 1991-1993 USL.
> migg GROUP1
migg successfully completed
> q
```

In the second example, the alternate machine is not accessible from the primary machine.

Listing 7-4 Migrating a Group When the Alternate Machine Is Not Accessible from the Primary Machine

```
$ tadmin
tadmin - Copyright © 1987-1990 AT&T; 1991-1993 USL.
> pclean SITE1
Cleaning the DBBL.
Pausing 10 seconds waiting for system to stabilize.
3 SITE1 servers removed from bulletin board
> migg GROUP1
migg successfully completed.
> boot -g GROUP1
Booting server processes ...
exec simpserve -A :
on SITE2 -> process id=22699 ... Started.
1 process started.
> q
```

How to Migrate Server Groups from One Machine to Another

1. Use the `LMID` parameter to name the processor on which the server group(s) have been running. The alternate location must be the same for all server groups on the `LMID`.
2. In the `RESOURCES` section of the `UBBCONFIG` file, set the following parameters:
 - Set `RESTART=Y` for each server on the machine indicated by the `LMID`.
 - Specify the `MIGRATE` options.
3. Shut down all server groups and mark the servers in the groups as restartable by entering the following command.

```
tmshutdown -R
```
4. Use the `tmadmin(1) migratemach (migm)` command to migrate all server groups from one machine to another when the primary machine must be shut down for maintenance or when the primary machine is no longer accessible. (The command takes one logical machine identifier as an argument.)

How to Migrate Machines When the Alternate Machine Is Accessible from the Primary Machine

To migrate a machine when the alternate machine is accessible from the primary machine, complete the following procedure.

1. Shut down the `MASTER` machine by entering the following command on that machine.

```
tmshutdown -R -l primary_machine
```
2. On the `MASTER` machine, start a `tmadmin` session by entering the following command.

```
tmadmin
```

3. At the `tmadmin` prompt, migrate the appropriate machine by entering the following command.

```
migratemach primary_machine
```

4. If necessary, migrate the transaction log.
5. If necessary, migrate the application data.

How to Migrate Machines When the Alternate Machine Is Not Accessible from the Primary Machine

To migrate a machine when the alternate machine is not accessible from the primary machine, switch the `MASTER` and `BACKUP` machines, if necessary.

1. On the alternate machine, start a `tmadmin` session by entering the following command.

```
tmadmin
```

2. Request cleanup and restart of the primary machine that require these operations by entering the following command.

```
pclean primary_machine
```

3. Transfer the appropriate server group to a configured alternate machine by entering the following command.

```
migratemach primary_machine
```

4. Boot the newly migrated server group by entering the following command.

```
boot -l alternate_machine
```

Examples of Migrating a Machine

The following sample session shows how to migrate machines. In the first example, the alternate machine is accessible from the primary machine.

Listing 7-5 Migrating a Machine When the Alternate Machine Is Accessible from the Primary Machine

```
$ tmsshutdown -R -l SITE1
Shutting down server processes...
Server ID = 1 Group ID = GROUP1 machine = SITE1: shutdown
succeeded 1 process stopped.
$ tadmin
tadmin - Copyright © 1987-1990 AT&T; 1991-1993 USL.
> migm SITE1
migm successfully completed
> q
```

In the second example, the alternate machine is not accessible from the primary machine.

Listing 7-6 Migrating a Machine When the Alternate Machine Is Not Accessible from the Primary Machine

```
$ tadmin
tadmin - Copyright © 1987-1990 AT&T; 1991-1993 USL.
> pclean SITE1
Cleaning the DBBL.
Pausing 10 seconds waiting for system to stabilize.
3 SITE1 servers removed from bulletin board
> migm SITE1
migm successfully completed.
> boot -l SITE1
Booting server processes ...
exec simpserv -A :
on SITE2 -- process id=22782 ... Started.
1 process started.
>q
```

How to Cancel a Migration

If you decide, after deactivating a server group or machine, that you do not want to continue, you can cancel the migration before reactivating the server group or machine. All the information in the name server for the deactivated servers and services is deleted.

To cancel a migration after a shutdown but before issuing the `migrate` command, enter one of the following commands.

To cancel . . .	Enter this command . . .	As a result . . .
Server migration	<code>tmadmin migrategroup -cancel</code> or <code>tmadmin migg -cancel</code>	Server entries are deleted from the bulletin board. You must reboot the servers once the migration procedure is canceled.
Machine migration	<code>tmadmin migratemach -cancel</code> or <code>tmadmin migm -cancel</code>	The migration is stopped.

Example of a Migration Cancellation

The following sample `tmadmin` session shows how a server group and a machine can be migrated between their respective primary and alternate machines.

Listing 7-7 Canceling a Server Group Migration for GROUP1

```
$tmadmin
tmadmin - Copyright © 1987-1990 AT&T; 1991-1993 USL.
> psr -g GROUP1

a.out Name   Queue Name   Grp Name   ID RqDone Ld Done Current Service
-----
simpserv    00001.00001  GROUP1    1   -   -           (DEAD MIGRATING)
> psr -g GROUP1
TMADMIN_CAT:121: No such server
migg -cancel GROUP1
>boot -g GROUP1
Booting server processes...
exec simpserv -A:
on SITE1 ->process id_27636 ... Started. 1 process started.
> psr -g GROUP1

a.out Name   Queue Name   Grp Name   ID RqDone Ld Done Current Service
-----
simpserv    00001.00001  GROUP1    1   -   -           ( - )
> q
```

How to Migrate Transaction Logs to a Backup Machine

To migrate a transaction log to a BACKUP machine, complete the following procedure.

1. Start a tmadmin session by entering the following command.

```
tmadmin
```
2. Shut down the servers in all the groups that write to the log, to prevent them from writing further entries.

3. Dump the TLOG into a text file by running the following command.

```
dumpnolog [-z config] [-o offset] [-n filename] [-g groupname]
```

Note: The TLOG is specified by the *config* and *offset* arguments. The value of *offset* defaults to 0; *name* defaults to TLOG. If the *-g* option is chosen, only those records coordinated by the TMS from *groupname* are dumped.

4. Copy *filename* to the BACKUP machine.
5. Read the file into the existing TLOG for the specified machine by entering the following command.

```
loadnolog -m machine filename
```

6. Force a warm start of the TLOG by entering the following command.

```
logstart machine
```

The system reads the information in the TLOG and uses it to create an entry in the transaction table in shared memory.

7. Migrate the servers to the BACKUP machine.

8 Tuning Your Application

- Maximizing Your Application Resources
- When to Use MSSQ Sets
- How to Enable Load Balancing
- How to Measure Service Performance Time
- How to Assign Priorities to Interfaces or Services
- Bundling Services into Servers
- Enhancing Efficiency with Application Parameters
- Determining Your System IPC Requirements
- Measuring System Traffic

Maximizing Your Application Resources

Making correct decisions in response to critical issues is the first step in ensuring the smooth performance of your BEA Tuxedo application. When making decisions about critical issues, consider including the following mechanisms as part of your strategy to improve the functioning of your system:

- Using MSSQ sets
- Enabling load balancing
- Measuring service performance time
- Assigning priorities to interfaces and/or services
- Bundling services into servers
- Setting application parameters
- Tuning operating system IPC parameters
- Eliminating bottlenecks

The rest of this section explains how to use these mechanisms and approaches to improve your system.

When to Use MSSQ Sets

The Multiple Servers, Single Queue, or MSSQ scheme offers additional load balancing. One queue is accommodated by several servers offering identical services at all times. If the server queue to which a request is sent is part of an MSSQ set, the message is dequeued to the first available server. Thus load balancing is provided at the individual queue level.

When a server is part of an MSSQ set, it must be configured with its own reply queue. When the server makes requests to other servers, the replies must be returned to the original requesting server; they must not be dequeued by other servers in the MSSQ set.

You can configure MSSQ sets to be dynamic so they automatically spawn and eliminate servers based upon a queue load.

The following table specifies when it is beneficial to use MSSQ sets.

You Should Use MSSQ Sets If . . .	You Should Not Use MSSQ Sets If . . .
You have between 2 and 12 servers.	There are many servers. (A compromise is to use many MSSQ sets.)
Buffer sizes are not too large, that is, large enough to exhaust a queue.	Buffer sizes are large enough to exhaust one queue.
All servers offer identical sets of services.	Each server offers different services.
Messages are relatively small.	Large messages are being passed to the services, causing the queue to be exhausted. When a queue is exhausted, either nonblocking sends fail or blocking sends block.
Optimization and consistency of service turnaround time are paramount.	

The following two analogies illustrate when it is beneficial to use MSSQ sets.

- A situation analogous to the appropriate use of MSSQ sets can be found in a bank at which several tellers performing identical services handle a single line of customers. The next available teller always takes the next person in line. In this scenario, each teller must be able to perform all customer services. In a BEA Tuxedo environment, all servers set up to share a single queue must offer an identical set of services at all times. The advantage of MSSQ sets is that they offer a second form of load balancing at the individual queue level.
- A super market at which different cashiers accept different forms of payment (some accept credit cards, while others accept only cash) is similar to a BEA Tuxedo application in which MSSQ sets should not be used.

How to Enable Load Balancing

To alleviate the performance degradation resulting from heavy system traffic, you may want to implement a load balancing algorithm on your entire application. With load balancing, a load factor is applied to each service within the system, and you can track the total load on every server. Every service request is sent to the qualified server that is least loaded.

To implement system-wide load balancing, complete the following procedure.

1. Run your application for an extended period of time.
2. Note the average amount of time it takes for each service to be performed.
3. In the `RESOURCES` section of the configuration file:
 - Set `LDBAL` to `Y`.
 - Assign a `LOAD` value of 50 (`LOAD=50`) to any service that takes approximately the average amount of time.
 - For any service taking longer than the average amount of time, set `LOAD>50`; for any service taking less than the average amount of time, set `LOAD<50`.

Note: This algorithm, although effective, is expensive and should be used only when necessary, that is, only when a service is offered by servers that use more than one queue. Services offered by only one server, or by multiple servers, all of which belong to the same MSSQ (multiple server, single queue) set, do not need load balancing.

How to Measure Service Performance Time

You can measure service performance time in either of two ways:

- *Administratively*—In the configuration file, you can arrange to have a log of services that are performed to be written to standard error. In the `SERVICES` section, specify

```
servopts -r
```

To analyze the information in the log, run the `txrpt(1)` command.

For details about `servopts(5)` and `txrpt(1)`, see *BEA Tuxedo File Formats and Data Descriptions Reference* and *BEA Tuxedo Command Reference*, respectively.

- *Programmatically*—Insert a call to `time()` at the beginning and end of a service routine. Services that take the longest time receive the highest load; those that take the shortest time receive the lowest load. (For details about `time()`, see the documentation for your C language libraries.)

How to Assign Priorities to Interfaces or Services

Assigning priorities enables you to exert significant control over the flow of data in an application, provide faster service to the most important requests, and provide slower service to the less important requests. You can also give priority to specific users—at all times or in specific circumstances.

You can assign priorities to BEA Tuxedo services in either of two ways:

- *Administratively*—In the `SERVICES` section of the configuration file, specify the `PRIO` parameter for each service named.
- *Programmatically*—Add calls to the `tpsrio()` function to the appropriate client and server applications, to allow designated clients and servers to change a

priority dynamically. Only preferred clients should be able to increase the service priority. In a system on which servers perform service requests, the server can call `tpsprio()` to increase the priority of its interface or service calls so the user does not wait in line for every interface or service request that is required.

Example of Using Priorities

Server 1 offers Interfaces A, B, and C. Interfaces A and B have a priority of 50; Interface C, a priority of 70. An interface requested for C is always dequeued before a request for A or B. Requests for A and B are dequeued equally with respect to one another. The system dequeues every tenth request in first-in, first-out (FIFO) order to prevent a message from waiting indefinitely on the queue.

Using the **PRIO** Parameter to Enhance Performance

The `PRIO` parameter determines the priority of an interface or a service on a server's queue. It should be used cautiously. Once priorities are assigned, it may take longer for some messages to be dequeued. Depending on the order of messages on the queue (for example, A, B, and C), some (such as A and B) are dequeued only one in ten times when there are more than 10 requests for C. This means reduced performance and potential slow turnaround time for some services.

When you are deciding whether to use the `PRIO` parameter, keep the following implications in mind:

- Because higher priorities get first preference, a higher priority should usually be assigned only to an interface or service that is not called frequently.
- A message with a lower priority does not remain enqueued indefinitely; every tenth message is retrieved on a FIFO basis. Before you assign a low priority to an interface or service you should be sure that response time for that interface or service is not important.

Bundling Services into Servers

The easiest way to package services into servers is to avoid packaging them at all. Unfortunately, if you do not package services, the number of servers, message queues, and semaphores rises beyond an acceptable level. Thus there is a trade-off between no bundling and too much bundling.

When to Bundle Services

We recommend that you bundle services if you have one of the situations or requirements described in the following list.

- *Functional similarity*—If multiple services play a similar role in the application, you can bundle them in the same server. The application can offer all or none of them at a given time. In the `bankapp` application, for example, the `WITHDRAW`, `DEPOSIT`, and `INQUIRY` services are all operations that can be grouped together in a “bank teller operations” server. Administration of services is simplified when functionally similar services are bundled.
- *Similar libraries*—Less disk space is required if you bundle services that use the same libraries. For example, if you have three services that use the same 100K library and three services that use different 100K libraries, bundling the first three services saves 200K. Functionally equivalent services often use similar libraries.
- *Filling the queue*—Bundle only as many services into a server as the queue can handle. Each service added to an unfilled MSSQ set may add relatively little to the size of an executable, and nothing to the number of queues in the system. Once the queue is filled, however, system performance is degraded and you must create more executables to compensate.

Do not put two or more services that call each other, that is, *call-dependent services*, in the same server. If you do so, the server issues a call to itself, causing a deadlock.

Enhancing Efficiency with Application Parameters

The following application parameters enable you to enhance the efficiency of your system:

- `MAXACCESSERS`, `MAXSERVERS`, `MAXINTERFACES`, and `MAXSERVICES`
- `MAXGTT`, `MAXBUFTYPE`, and `MAXBUFSTYPE`
- `SANITYSCAN`, `BLOCKTIME`, and individual transaction time-outs
- `BBLQUERY` and `DBBLWAIT`

Setting the `MAXACCESSERS`, `MAXSERVERS`, `MAXINTERFACES`, and `MAXSERVICES` Parameters

The `MAXACCESSERS`, `MAXSERVERS`, `MAXINTERFACES`, and `MAXSERVICES` parameters increase semaphore and shared memory costs, so you should carefully weigh these costs against the expected benefits before using these parameters, and choose the values that best satisfy the needs of your system. You should take into account any increased resources your system may require for a potential migration. You should also allow for variation in the number of clients accessing the system simultaneously. Defaults may be appropriate for a generous allocation of IPC resources; however, it is prudent to set these parameters to the lowest appropriate values for the application.

Setting the **MAXGTT**, **MAXBUFTYPE**, and **MAXBUFSTYPE** Parameters

To determine whether the default is adequate for your application, multiply the number of clients in the system times the percentage of time they are committing a transaction. If the product of this multiplication is close to 100, you should increase the value of the **MAXGTT** parameter. As a result of increasing **MAXGTT**:

- Your system may require a greater number of clients, depending on the speed of commits.
- You should also increase **TLOGSIZE** accordingly for every machine.
- You should set **MAXGTT** to 0 for applications in which distributed transactions are not used.

To limit the number of buffer types and subtypes allowed in the application, set the **MAXBUFTYPE** and **MAXBUFSTYPE** parameters, respectively. The current default for **MAXBUFTYPE** is 16. If you plan to create eight or more user-defined buffer types, you should set **MAXBUFTYPE** to a higher value. Otherwise, you do not need to specify this parameter; the default value is used.

The current default for **MAXBUFSTYPE** is 32. You may want to set this parameter to a higher value if you intend to use many different **VIEW** subtypes.

Tuning with the **SANITYSCAN**, **BLOCKTIME**, **BBLQUERY**, and **DBBLWAIT** Parameters

If a system is running on slow processors (for example, due to heavy usage), you can increase the timing parameters: **SANITYSCAN**, **BLOCKTIME**, and individual transaction time-outs.

If networking is slow, you can increase the value of the **BLOCKTIME**, **BBLQUERY**, and **DBBLWAIT** parameters.

Recommended Values for Tuning-related Parameters

In the following table we recommend values for the parameters available for tuning an application.

Use These Parameters . . .	To . . .
MAXACCESSERS, MAXSERVERS, MAXINTERFACES, and MAXSERVICES	Set the smallest satisfactory value because of IPC cost. (Allow for extra clients.)
MAXGTT, MAXBUFTYPE, and MAXBUFSTYPE	Increase MAXGTT for many clients; set MAXGTT to 0 for nontransactional applications. Use MAXBUFTYPE only if you create eight or more user-defined buffer types. Increase the value of MAXBUFSTYPE if you use many different VIEW subtypes.
BLOCKTIME, TRANTIME, and SANITYSCAN	Increase the values if the system is slow.
BLOCKTIME, TRANTIME, BBLQUERY, and DBBLWAIT	Increase the values if networking is slow.

Determining Your System IPC Requirements

The IPC requirements for your system are determined by the values of several system parameters:

- MAXACCESSERS
- REPLYQ
- RQADDR
- MAXSERVERS

- MAXSERVICES
- MAXGTT

You can use the `tmboot -c` command to display the minimum IPC requirements of your configuration.

The following table describes these system parameters.

Table 8-1 Parameters for Tuning IPC Resources

Parameter(s)	Description
MAXACCESSERS	Equals the number of semaphores. Number of message queues is almost equal to MAXACCESSERS + number of servers with reply queues (number of servers in MSSQ set * number of MSSQ sets).
MAXSERVERS, MAXSERVICES, and MAXGTT	While MAXSERVERS, MAXSERVICES, MAXGTT, and the overall size of the ROUTING, GROUP, and NETWORK sections affect the size of shared memory, an attempt to devise formulas that correlate these parameters can become complex. Instead, simply run <code>tmboot -c</code> or <code>tmloadcf -c</code> to calculate the minimum IPC resource requirements for your application.

Table 8-1 Parameters for Tuning IPC Resources

Parameter(s)	Description
Queue-related kernel parameters	<p>Need to be tuned to manage the flow of buffer traffic between clients and servers. The maximum total size (in bytes) of a queue must be large enough to handle the largest message in the application. A typical queue is not more than 75 to 85 percent full. Using a smaller percentage of a queue is wasteful; using a larger percentage causes message sends to block too frequently.</p> <p>Set the maximum size for a message to handle the largest buffer that the application sends.</p> <p>The maximum queue length (the largest number of messages that are allowed to sit on a queue at once) must be adequate for the application's operations.</p> <p>Simulate or run the application to measure the average fullness of a queue or its average length. This process may require a lot of trial and error: you may need to estimate values for your tunables before running the application, and then adjust them after running under performance analysis.</p> <p>For a large system, analyze the effects of parameter settings on the size of the operating system kernel. If they are unacceptable, reduce the number of application processes or distribute the application across more machines to reduce <code>MAXACCESSERS</code>.</p>

Measuring System Traffic

As on any road that supports a lot of traffic, bottlenecks can occur in your system. On a highway, cars can be counted with a cable strung across the road, that causes a counter to be incremented each time a car drives over it.

You can use a similar method to measure service traffic. For example, when a server is started (that is, when `tpsvrinit()` is invoked), you can initialize a global counter and record a starting time. Subsequently, each time a particular service is called, the counter is incremented. When the server is shut down (through the `tpsvrdone()` function), the final count and the ending time are recorded. This mechanism allows you to determine how busy a particular service is over a specified period of time.

In the BEA Tuxedo system, bottlenecks can originate from problematic data flow patterns. The quickest way to detect bottlenecks is to measure the amount of time required by relevant services from the client's point of view.

Example of Detecting a System Bottleneck

Client 1 requires 4 seconds to display the results. Calls to `time()` determine that the `tpcall` to service A is the culprit with a 3.7-second delay. Service A is monitored at the top and bottom and takes 0.5 seconds. This finding implies that a queue may be clogged, a situation that can be verified by running the `pq` command in `tadmin`.

On the other hand, suppose service A takes 3.2 seconds. The individual parts of service A can be bracketed and measured. Perhaps service A issues a `tpcall` to service B, which requires 2.8 seconds. Knowing this, you should then be able to isolate queue time or message send blocking time. Once the relevant amount of time has been identified, the application can be retuned to handle the traffic.

Using `time()`, you can measure the duration of the following:

- An entire client program
- A single client service request
- An entire service function
- A service function making a service request (if any)

Detecting Bottlenecks on UNIX Platforms

The UNIX system `sar(1)` command provides valuable performance information that can be used to find system bottlenecks. You can run `sar(1)` to do the following:

- Sample cumulative activity counters in the operating system at predetermined intervals
- Extract data from a system file

The following table describes the `sar(1)` command options.

Use This Option	To
<code>-u</code>	Gather CPU utilization numbers, including percentages of time during which the system: runs in user mode, runs in system mode, remains idle with some process waiting for block I/O, and otherwise remains idle.
<code>-b</code>	Report buffer activity, including number of data transfers, per second, between system buffers and disk (or other block devices).
<code>-c</code>	Report activity of system calls of all types, as well as specific system calls, such as <code>fork(2)</code> and <code>exec(2)</code> .
<code>-w</code>	Monitor system swapping activity, including the number of transfers for swapins and swapouts.
<code>-q</code>	Report average queue lengths while queues are occupied, and the percentage of time they are occupied.
<code>-m</code>	Report message and system semaphore activities, including the number of primitives per second.
<code>-p</code>	Report paging activity, including the number of address translation page faults, page faults and protection errors, and valid pages reclaimed for free lists.
<code>-r</code>	Report the number of unused memory pages and disk blocks, including the average number of pages available to user processes and disk blocks available for process swapping.

Note: Some flavors of the UNIX system do not support the `sar(1)` command, but offer equivalent commands, instead. BSD, for example, offers the `iostat(1)` command; Sun offers `perfmeter(1)`.

Detecting Bottlenecks on Windows NT Platforms

On Windows NT platforms, use the Performance Monitor to collect system information and detect bottlenecks. Select the following options from the Start menu.

Start —> Programs —> Administration Tools —> NT Performance Monitor

8 *Tuning Your Application*

9 Troubleshooting a BEA Tuxedo Application

- Determining Types of Failures
- How to Broadcast an Unsolicited Message
- Maintaining Your System Files
- Repairing Partitioned Networks
- Restoring Failed Machines
- How to Replace System Components
- How to Replace Application Components
- Cleaning Up and Restarting Servers Manually
- Aborting or Committing Transactions
- How to Recover from Failures When Transactions Are Used
- How to Use the IPC Tool When an Application Fails to Shut Down Properly
- Troubleshooting Multithreaded/ Multicontexted Applications

Determining Types of Failures

The first step in troubleshooting is determining problem areas. In most applications you must consider six possible sources of trouble:

- Application
- BEA Tuxedo system
- Database management software
- Network
- Operating system
- Hardware

Once you have determined the problem area, you must then work with the appropriate administrator to resolve the problem. If, for example, you determine that the trouble is caused by a networking problem, you must work with the network administrator.

How to Determine the Cause of an Application Failure

The following steps will help you detect the source of an application failure.

1. Check any BEA Tuxedo system warnings and error messages in the user log (ULOG).
2. Select the messages you think most likely reflect the current problem. Note the catalog name and the number of each of message, so you can look up the message in *BEA Tuxedo System Messages*. The manual entry provides:
 - Details about the error condition indicated by the message
 - Recommendations for recovery actions
3. Check any application warnings and error messages in the ULOG.

4. Check any warnings and errors generated by application servers and clients. Such messages are usually sent to the standard output and standard error files (named, by default `stdout` and `stderr`, respectively).
 - The `stdout` and `stderr` files are located in the directory defined by the `APPDIR` variable.
 - The `stdout` and `stderr` files for your clients and servers may have been renamed. (You can rename the `stdout` and `stderr` files by specifying `-e` and `-o` in the appropriate client and server definitions in your configuration file. For details, see `servopts(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*.)
5. Look for any core dumps in the directory defined by the `APPDIR` variable. Use a debugger such as `dbx` to get a stack trace. If you find core dumps, notify your application developer.
6. Check your system activity reports (for example, by running the `sar(1)` command) to determine why your system is not functioning properly. Consider the following reasons:
 - The system may be running out of memory.
 - The kernel might not be tuned correctly.

How to Determine the Cause of a BEA Tuxedo System Failure

The following steps will help you detect the source of a system failure.

1. Check any BEA Tuxedo system warnings and error messages in the user log (`ULOG`):
 - `TPEOS` messages indicate errors in the operating system.
 - `TPESYSTEM` messages indicate errors in the BEA Tuxedo system.
2. Select the messages you think most likely reflect the current problem. Note the catalog name and number of each of message, so you can look up the message in *BEA Tuxedo System Messages*. The manual entry provides:

- Details about the error condition flagged by the message
 - Recommendations for recovery actions
3. Prepare for debugging in the following ways:
 - Shut down the `suspend` service.
 - Use `tmboot -n -s(server) -d1`. (This will not boot the server, but prints the command line used to boot the server by the BEA Tuxedo system.) Use that command line with a debugger such as `dbx`.

How to Broadcast an Unsolicited Message

The EventBroker enhances troubleshooting by providing a system-wide summary of events and a mechanism whereby an event triggers notification. The EventBroker provides details about BEA Tuxedo system events, such as servers dying and networks failing, or application events, such as an ATM machine running out of money. A BEA Tuxedo client that receives unsolicited notification of an event, can name a service routine to be invoked, or name an application queue in which data should be stored for later processing. A BEA Tuxedo server that receives unsolicited notification can specify a service request or name an application queue to store data.

1. To send an unsolicited message, enter the following command.

```
broadcast (bcst) [-m machine] [-u username] [-c cltname] [text]
```

Note: By default, the message is sent to all clients.

2. You can limit distribution to one of the following recipients:
 - One machine (`-m machine`)
 - One client group (`-c client_group`)
 - One user (`-u user`)

The text may not include more than 80 characters. The system sends the message in a `STRING` type buffer, which means the client's unsolicited message handling function (specified by `tpsetunsol(0)`) must be able to handle this type of message. The `tpypes()` function may be useful in this case.

See Also

- “What Is Unsolicited Communication” on page 2-17 in *Introducing the BEA Tuxedo System*
- “Managing System Events Using EventBroker” on page 3-14 in *Introducing the BEA Tuxedo System*

Maintaining Your System Files

Periodically, you may need to perform the following tasks to maintain your file system:

- Print the Universal Device List
- Print VTOC information
- Reinitialize a device
- Create a device list
- Destroy a device list

Note: This file format is used for TUXCONFIG, TLOG, and /Q.

How to Print the Universal Device List (UDL)

To print a UDL, complete the following procedure.

1. Run `tmadmin -c`.
2. Enter the following command.

```
lidl
```

3. To specify the device from which you want to obtain the UDL, you have a choice of two methods:
 - Specify the device on the `lidl` command line.
`-z device_name [devindx]`
 - Set the environment variable `FSCONFIG` to the name of the desired device.

How to Print VTOC Information

To print VTOC information, complete the following procedure.

1. Run `tmadmin -c`.
2. To get information about all VTOC table entries, enter the following command.
`livtoc`
3. To specify the device from which you want to obtain the VTOC, you have a choice of two methods:
 - Specify the following on the `lidl` command line.
`-z device name [devindx]`
 - Set the environment variable `FSCONFIG` to the name of the desired device.

How to Reinitialize a Device

To reinitialize a device that is included on a device list, complete the following procedure.

1. Run `tmadmin -c`.
2. Enter the following command.
`initdl [-z devicename] [-yes] devindx`

Note: The value of `devindx` is the index to the file to be destroyed.

3. You can specify the device by:
 - Entering its name after the `-z` option (as shown here), or
 - Setting the environment variable `FSCONFIG` to the device name
4. If you include the `-yes` option on the command line, you are not prompted to confirm your intention to destroy the file before the file is actually destroyed.

How to Create a Device List

To create a device list, complete the following procedure.

1. Run `tmadmin -c`.
2. Enter the following command.


```
crdl [-z devicename] [-b blocks]
```

 - The value of `devicename` [`devindx`] is the desired device name. (Another way to assign a name to a new device is by setting the `FSCONFIG` environment variable to the desired device name.)
 - The value of `blocks` is the number of blocks needed. The default is 1000 blocks.

Note: Because 35 blocks are needed for the administrative overhead associated with a TLOG, be sure to assign a value higher than 35 when you create a TLOG.

How to Destroy a Device List

To destroy a device list with index `devindx`, complete the following procedure.

1. Run `tmadmin -c`.
2. Enter the following command.


```
dsdl [-z devicename] [yes] [devindx]
```

Note: The value of `devindx` is the index to the file to be destroyed.

3. You can specify the device by:
 - Entering its name after the `-z` option (as shown here), or
 - Setting the environment variable `FSCONFIG` to the device name
4. If you include the `yes` option on the command line, you are not prompted to confirm your intention to destroy the file before the file is actually destroyed.

Repairing Partitioned Networks

This topic provides instructions for troubleshooting a partition, identifying its cause, and taking action to recover from it. A network partition exists if one or more machines cannot access the `MASTER` machine. As the application administrator, you are responsible for detecting partitions and recovering from them.

A network partition may be caused by any the following failures:

- A network failure—either a transient failure, which corrects itself in minutes, or a severe failure, which requires you to take the partitioned machine out of the network
- A machine failure on either the `MASTER` machine or the nonmaster machine
- A `BRIDGE` failure

The procedure you follow to recover from a partitioned network depends on the cause of the partition.

Detecting a Partitioned Network

You can detect a network partition in one of the following ways:

- [Check the user log \(ULOG\) for messages that may shed light on the origin of the problem.](#)
- [Gather information about the network, server, and service, by running the `tadmin` commands provided for this purpose.](#)

How to Check the ULOG

When problems occur with the network, BEA Tuxedo system administrative servers start sending messages to the ULOG. If the ULOG is set up over a remote file system, all messages are written to the same log. In this scenario, you can run the `tail(1)` command on one file and check the failure messages displayed on the screen.

If, however, the remote file system is using the network in which the problem has occurred, the remote file system may no longer be available.

Listing 9-1 Example of a ULOG Error Message

```
151804.gumby!DBBL.28446: ... : ERROR: BBL partitioned, machine=SITE2
```

How to Gather Information About the Network, Server, and Service

The following is an example of a `tmadmin` session in which information is being collected about a partitioned network, a server, and a service on that network. Three `tmadmin` commands are run:

- `pnw` (the `printnetwork` command)
- `psr` (the `printserver` command)
- `psc` (the `printservice` command)

Listing 9-2 Example tadmin Session

```
$ tadmin
> pnw SITE2
Could not retrieve status from SITE2

> psr -m SITE1
a.out Name      Queue Name      Grp Name      ID   Rq Done      Load Done      Current Service
BBL             30002.00000     SITE1         0    -           -               ( - )
DBBL            123456          SITE1         0    121         6050            MASTERBB
simpserv        00001.00001     GROUP1        1    -           -               ( - )
BRIDGE          16900672        SITE1         0    -           -               ( DEAD )
>psc -m SITE1
Service Name    Routine Name     a.out      Grp Name ID Machine # Done Status
-----
ADJUNCTADMIN    ADJUNCTADMIN    BBL        SITE1   0   SITE1   - PART
ADJUNCTBB       ADJUNCTBB       BBL        SITE1   0   SITE1   - PART
TOUPPER         TOUPPER         simpserv   GROUP1  1   SITE1   - PART
BRIDGESVCNM     BRIDGESVCNM     BRIDGE     SITE1   1   SITE1   - PART
```

Restoring a Network Connection

This topic provides instructions for recovering from transient and severe network failures.

How to Recover from Transient Network Failures

Because the BRIDGE tries, automatically, to recover from any transient network failures and reconnect, transient network failures are usually not noticed. If, however, you need to perform a manual recovery from a transient network failure, complete the following procedure.

1. On the MASTER machine, start a `tadmin(1)` session.
2. Run the `reconnect` command (`rco`), specifying the names of nonpartitioned and partitioned machines.

```
rco non-partioned_node1 partioned_node2
```

How to Recover from Severe Network Failures

To recover from severe network failure, complete the following procedure.

1. On the MASTER machine, start a `tmadmin` session.
2. Run the `pclean` command, specifying the name of the partitioned machine.

```
pc1 partitioned_machine
```
3. Migrate the application servers or, once the problem has been corrected, reboot the machine.

Restoring Failed Machines

The procedure you follow to restore a failed machine depends on whether that machine was the MASTER machine.

How to Restore a Failed MASTER Machine

To restore a failed MASTER machine, complete the following procedure.

1. Make sure that all IPC resources for the BEA Tuxedo processes that are removed.
2. Start a `tmadmin` session on the ACTING MASTER (SITE2).

```
tmadmin
```
3. Boot the BBL on the MASTER (SITE1) by entering the following command.

```
boot -B SITE1
```

(The BBL does not boot if you have not executed `pclean` on SITE1.)
4. Still in `tmadmin`, start a DBBL running again on the MASTER site (SITE1) by entering the following.

```
MASTER
```
5. If you have migrated application servers and data off the failed machine, boot them or migrate them back.

How to Restore a Failed Nonmaster Machine

To restore a failed nonmaster machine, complete the following procedure.

1. On the MASTER machine, start a `tmadmin` session.
2. Run `pclean`, specifying the partitioned machine on the command line.
3. Fix the machine problem.
4. Restore the failed machine by booting the Bulletin Board Liaison (BBL) for the machine from the MASTER machine.
5. If you have migrated application servers and data from the failed machine, boot them or migrate them back.

In the following list, `SITE2`, a nonmaster machine, is restored.

Listing 9-3 Example of Restoring a Failed Nonmaster Machine

```
$ tmadmin
tmadmin - Copyright © 1987-1990 AT&T; 1991-1993 USL. All rights reserved

> pclean SITE2
Cleaning the DBBL.

Pausing 10 seconds waiting for system to stabilize.
3 SITE2 servers removed from bulletin board

> boot -B SITE2
Booting admin processes ...

Exec BBL -A :

on SITE2 -> process id=22923 ... Started.
1 process started.
> q
```

How to Replace System Components

To replace BEA Tuxedo system components, complete the following procedure.

1. Install the BEA Tuxedo system software that is being replaced.
2. Shut down those parts of the application that will be affected by the changes:
 - The BEA Tuxedo system servers may need to be shut down if libraries are being updated.
 - Application clients and servers must be shut down and rebuilt if relevant BEA Tuxedo system header files or static libraries are being replaced. (Application clients and servers do not need to be rebuilt if the BEA Tuxedo system message catalogs, system commands, administrative servers, or shared objects are being replaced.)
3. If relevant BEA Tuxedo system header files and static libraries have been replaced, rebuild your application clients and servers.
4. Reboot the parts of the application that you shut down.

How to Replace Application Components

To replace components of your application, complete the following procedure.

1. Install the application software. This software may consist of application clients, application servers, and various administrative files, such as the FML field tables.
2. Shut down the application servers being replaced.
3. If necessary, build the new application servers.
4. Boot the new application servers.

Cleaning Up and Restarting Servers Manually

By default, the BEA Tuxedo system cleans up resources associated with dead processes (such as queues) and restarts restartable dead servers from the Bulletin Board (BB) at regular intervals during BBL scans. You may, however, request cleaning at other times.

How to Clean Up Resources Associated with Dead Processes

To request an immediate cleanup of resources associated with dead processes, complete the following procedure.

1. Start a `tmadmin` session.
2. Enter `bbclean machine`.

The `bbclean` command takes one optional argument: the name of the machine to be cleaned.

If You Specify	Then
No machine	The resources on the default machine are cleaned.
A machine	The resources on the specified machine are cleaned.
DBBL	The resources on the Distinguished Bulletin Board Liaison (DBBL) and the Bulletin Boards at all sites are cleaned.

How to Clean Up Other Resources

To clean up other resources, complete the following procedure.

1. Start a `tmadmin` session.
2. Enter `pclean machine`.

Note: You must specify a value for *machine*; it is a required argument.

If the Specified Machine Is	Then
Not partitioned	<code>pclean</code> will invoke <code>bbclean</code> .
Partitioned	<code>pclean</code> will remove all entries for servers and services from all nonpartitioned Bulletin Boards.

This command is useful for restoring order to a system after partitioning has occurred unexpectedly.

Aborting or Committing Transactions

This topic provides instructions for aborting and committing transactions.

How to Abort a Transaction

To abort a transaction, complete the following procedure.

1. Enter the following command.


```
aborttrans (abort) [-yes] [-g groupname] tranindex
```
2. To determine the value of *tranindex*, run the `printtrans` command (a `tmadmin` command).

3. If *groupname* is specified, a message is sent to the TMS of that group to mark as “aborted” the transaction for that group. If a group is not specified, a message is sent, instead, to the coordinating TMS, requesting an abort of the transaction.
You must send abort messages to all groups in the transaction to control the abort.

This command is useful when the coordinating site is partitioned or when the client terminates before calling a commit or an abort. If the time-out is large, the transaction remains in the transaction table unless it is aborted.

How to Commit a Transaction

To commit a transaction, complete the following procedure.

1. Enter the following command.

```
committrans (commit) [-yes] [-g groupname] tranindex
```

Note: Both *groupname* and *tranindex* are required arguments.

The operation fails if the transaction is not precommitted or has been marked aborted. This message should be sent to all groups to fully commit the transaction.

Cautions About Using the `committrans` Command

Be careful about using the `committrans` command. The only time you need to run it is when both of the following conditions apply:

- The coordinating TMS has gone down before all groups got the commit message.
- The coordinating TMS will not be able to recover the transaction for some time.

Also, a client may be blocked on `tpcommit()`, which will be timed out. If you are going to perform an administrative commit, be sure to inform this client.

How to Recover from Failures When Transactions Are Used

When the application you are administering includes database transactions, you may need to apply an after-image journal (AIJ) to a restored database following a disk corruption failure. Or you may need to coordinate the timing of this recovery activity with your site's database administrator (DBA). Typically, the database management software automatically performs transaction rollback when an error occurs. When the disk containing database files has become corrupted permanently, however, you or the DBA may need to step in and perform the rollforward operation.

Assume that a disk containing portions of a database is corrupted at 3:00 P.M. on a Wednesday. For this example, assume that a shadow volume (that is, you have disk mirroring) does not exist.

1. Shut down the BEA Tuxedo application. (For instructions, see "Starting Up and Shutting Down an Application" on page 1-1 in *Setting Up a BEA Tuxedo Application*.)
2. Obtain the last full backup of the database and restore the file. For example, restore the full backup version of the database from last Sunday at 12:01 A.M.
3. Apply the incremental backup files, such as the incrementals from Monday and Tuesday. For example, assume that this step restores the database up until 11:00 P.M. on Tuesday.
4. Apply the AIJ, or transaction journal file, that contains the transactions from 11:15 P.M. on Tuesday up to 2:50 P.M. on Wednesday.
5. Open the database again.
6. Restart the BEA Tuxedo application.

Refer to the documentation for the resource manager (database product) for specific instructions on the database rollforward process.

How to Use the IPC Tool When an Application Fails to Shut Down Properly

Inter-process communication (IPC) resources are operating system resources, such as message queues, shared memory, and semaphores. When a BEA Tuxedo application shuts down properly with the `tmshutdown` command, all IPC resources are removed from the system. In some cases, however, an application may fail to shut down properly and stray IPC resources may remain on the system. When this happens, it may not be possible to reboot the application.

One way to address this problem is to remove IPC resources with a script that invokes the system `IPCS` command and scan for all IPC resources owned by a particular user account. However, with this method, it is difficult to distinguish among different sets of IPC resources; some may belong to the BEA Tuxedo system; some to a particular BEA Tuxedo application; and others to applications unrelated to the BEA Tuxedo system. It is important to be able to distinguish among these sets of resources; unintentional removal of IPC resources can severely damage an application.

The BEA Tuxedo IPC tool (that is, the `tmipcrm` command) enables you to remove IPC resources allocated by the BEA Tuxedo system (that is, for core BEA Tuxedo and Workstation components only) in an active application.

The command to remove IPC resources, `tmipcrm`, resides in `TUXDIR/bin`. This command reads the binary configuration file (`TUXCONFIG`), and attaches to the bulletin board using the information in this file. `tmipcrm` works only on the local server machine; it does not clean up IPC resources on remote machines in a BEA Tuxedo configuration.

To run this command, enter it as follows on the command line.

```
tmipcrm [-y] [-n] [tuxconfig_file]
```

The IPC tool lists all IPC resources used by the BEA Tuxedo system and gives you the option of removing them.

Note: This command will not work unless you have set the `TUXCONFIG` environment variable correctly or specified the appropriate `TUXCONFIG` file on the command line.

Troubleshooting Multithreaded/ Multicontexted Applications

Debugging Multithreaded/Multicontexted Applications

Multithreaded applications can be much more difficult to debug than single-threaded applications. As the administrator, you may want to establish a policy governing whether such multithreaded applications should be created.

Limitations of Protected Mode in a Multithreaded Application

When running in protected mode, an application attaches to shared memory only when an ATMI call is being executed. Protected mode is used to guard against problems that arise when BEA Tuxedo shared memory is accidentally overwritten by stray application pointers.

If your multithreaded application is running in protected mode, some threads may be executing application code while others are attached to the BEA Tuxedo Bulletin Board's shared memory within a BEA Tuxedo function call. Therefore, as long as at least one thread is attached to the bulletin board in an ATMI call, the use of protected mode cannot guard against stray application pointers in threads executing application code, which may overwrite the BEA Tuxedo shared memory. As a result, the usefulness of protected mode is relatively limited in multithreaded applications.

There is no solution to this limitation. We simply want to warn you that when running a multithreaded application you cannot rely on protected mode as much as you do when running a single-threaded application.

9 *Troubleshooting a BEA Tuxedo Application*
