



BEA Tuxedo

Setting Up a BEA Tuxedo Application

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.

Setting Up a BEA Tuxedo Application

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

Contents

1. Administrative Tasks and Tools

Tasks an Administrator Performs	1-1
Setup Tasks	1-1
Run-time Tasks	1-2
Planning the Design of Your Application	1-4
Tools to Help You Administer Your Application	1-6

2. About the Configuration File

What Is the Configuration File	2-1
Text and Binary Versions of the Configuration File	2-1
Contents of the Configuration File	2-2

3. Creating the Configuration File

How to Create a Configuration File	3-2
How to Create the Configuration File for a Single-machine Application	3-2
How to Create the Configuration File for a Multiple-machine (Distributed) Application	3-3
How to Create the Configuration File for a Multiple-domain Application	3-4
How to Create the RESOURCES Section of the Configuration File	3-7
Sample RESOURCES Section	3-9
Defining the Application Type	3-9
Characteristics of the MODEL and OPTIONS Parameters	3-10
Example Settings	3-10
Controlling the Number of Buffer Types and Subtypes	3-11
Characteristics of the MAXBUFTYPE and MAXBUFSTYPES Parameters	3-11
Example Settings	3-11

Controlling the Number of Conversations.....	3-12
Characteristics of the MAXCONV Parameter	3-12
Example Setting.....	3-12
Defining IPC Limits	3-12
Characteristics of MAXACCESSERS, MAXSERVERS, and MAXSERVICES Parameters	3-13
Example Settings	3-14
Enabling Load Balancing	3-15
Characteristics of the LDBAL Parameter.....	3-15
Example Settings	3-16
Identifying the Master Machine	3-16
Characteristics of the MASTER Parameter.....	3-16
Example Settings	3-17
Specifying the Maximum Number of Network Groups	3-17
Specifying the Number of Sanity Checks and Blocking Timeouts	3-17
Characteristics of the SCANUNIT, SANITYSCAN, and BLOCKTIME Parameters	3-18
Timeouts for Blocking ATMI Operations	3-18
Example Settings	3-19
Establishing Operating System-level Security	3-19
Characteristics of the UID, GID, and PERM Parameters	3-20
Specifying the Security Level.....	3-21
Characteristics of the SECURITY and AUTHSVC Parameters	3-21
Defining the Security Attributes of a Server	3-22
Protecting Shared Memory	3-23
Characteristics of the PROTECTED, FASTPATH, and NO_OVERRIDE Parameters	3-24
Example Settings	3-24
Setting the Address of the System Resources for an Application	3-24
Characteristics of the IPCKEY Parameter	3-25
Example Settings	3-25
Specifying How Clients Receive Unsolicited Notification	3-25
Characteristics of the NOTIFY and USIGNAL Parameters	3-26
How to Create the MACHINES Section of the Configuration File	3-27
Sample MACHINES Section	3-30

Specifying the Maximum Number of ACL Entries in the Cache	3-32
Defining an Additional Service Request Load	3-33
Reserving the Physical Address and Machine ID	3-33
Characteristics of the Address and the LMID Parameter	3-34
Setting the Number of Lock Spins	3-34
Characteristics of the SPINCOUNT Parameter	3-34
Specifying Machines as Types	3-35
Characteristics of the TYPE Parameter	3-35
Identifying the Location of the Configuration File	3-36
Characteristics of the TUXCONFIG Parameter	3-36
Indicating the Size of the DTP Transaction Log	3-36
Defining the DTP Transaction Log Name	3-37
Specifying Environment Variable Settings	3-37
Characteristics of the ENVFILE Parameter	3-38
Defining the BEA Tuxedo File System Containing the TLOG	3-38
Specifying a Machine's Maximum Number of Simultaneous Global Transactions	3-39
Defining the Number of Accesser Entries on a Workstation Client	3-39
Defining Space Limits for Messages Transmitted by the BRIDGE	3-40
Indicating the Offset for the DTP Transaction Log	3-40
Defining the Offset for TUXCONFIG	3-41
Characteristics of the TUXOFFSET Parameter	3-41
Identifying the Locations of the System Software and Application Server Software	3-41
Characteristics of the APPDIR and TUXDIR Parameters	3-42
Indicating a Threshold Message Size for Compression	3-42
Example	3-43
Specifying the Path Name for the ULOG	3-43
Characteristics of the ULOGPFX Parameter	3-43
How to Create the GROUPS Section of the Configuration File	3-44
Sample GROUPS Section	3-45
Specifying a Group Name, Number, and LMID	3-46
Characteristics of the Group Name, Group Number, and LMID	3-46
Indicating a Transaction Manager Server Name and Numbers per Group	3-47
Identifying the Environment File Location for Servers in a Group	3-48

Defining Information Needed When Opening and Closing the Resource Manager	3-48
How to Create the NETWORK Section of the Configuration File	3-51
Sample NETWORK Section	3-52
Specifying a Device Name for the BRIDGE Process	3-52
Assigning a BRIDGE Network Address	3-53
Assigning Encryption Levels	3-54
Example	3-54
Assigning a tlisten Network Address	3-54
How to Create the NETGROUPS Section of the Configuration File	3-56
Sample Network Groups Configuration	3-57
Configuring a Sample UBBCONFIG File with Netgroups	3-58
Assigning a Name to a Network Group	3-59
Assigning a Network Group Number	3-60
Assigning a Priority to the Network Group	3-60
How to Create the SERVERS Section of the Configuration File	3-60
Sample SERVERS Section	3-64
Specifying a Server as Conversational	3-66
Characteristics of the CONV Parameter	3-66
Setting the Order in Which Servers Are Booted	3-66
Characteristics of the SEQUENCE, MIN, and MAX Parameters	3-67
Specifying Server Command-line Options	3-68
Characteristics of the CLOPT Parameter	3-69
Identifying the Location of the Server Environment File	3-69
Characteristics of the Server Environment File	3-70
Defining Server Name, Group, and ID	3-70
Characteristics of the Server Name, SRVGRP, and SRVID Parameters	3-70
Identifying Server Queue Information	3-71
MSSQ Example	3-71
Characteristics of the RQADDR, RQPERM, REPLYQ, and RPPER Parameters	3-72
Defining Server Restart Information	3-73
Characteristics of the RESTART, RCMD, MAXGEN, and GRACE Parameters	3-73
Defining Server Access to Shared Memory	3-74

Characteristics of the SYSTEM_ACCESS Parameter.....	3-74
Defining the Server Dispatch Threads	3-75
How to Create the SERVICES Section of the Configuration File	3-75
Sample SERVICES Section	3-77
Specifying Automatic Starts and Timeout Intervals for Transactions	3-77
Specifying a List of Allowable Buffer Types for a Service	3-78
Examples of the BUFTYPE Parameter	3-78
Designating How Much Time to Process a Request	3-79
What Happens When a Timeout Occurs	3-79
How a Service Timeout Is Reported	3-80
Enabling Load Balancing	3-81
Characteristics of the LDBAL Parameter	3-81
Defining the Name of the Routing Criteria	3-82
Specifying Service Parameters for Different Server Groups	3-82
Controlling the Flow of Data by Service Priority	3-82
Characteristics of the PRIO Parameter.....	3-83
Sample SERVICES Section Using Different Priorities	3-83
Indicating Service Processing Time	3-83
How to Create the ROUTING Section of the Configuration File	3-84
ROUTING Section Example.....	3-85
Defining the Routing Buffer Field and Field Type	3-85
Specifying Range Criteria	3-86
Defining Buffer Types.....	3-87
How to Configure the BEA Tuxedo System to Take Advantage of Threads .	3-87
How to Compile a Configuration File	3-90

4. About Transactions

What Is a Transaction.....	4-1
What Are the ACID Properties	4-2
How a Transaction Succeeds or Fails	4-3
Benefits of Using Transactions	4-3
Example of a Global Transaction	4-4
What Is the BEA Tuxedo Transaction Manager (TM).....	4-5
How the System Tracks Distributed Transaction Processing.....	4-6
How the System Uses Global Transaction Identifiers (GTRIDs) for	

Tracking	4-7
How the System Uses a Transaction Log (TLOG) for Tracking	4-8
How the System Uses a 2-Phase Commit to Commit Transactions	4-8
How the System Handles Transaction Infection	4-9
How the ATMI Protects a Transaction's Integrity Before a 2-Phase Commit	4-10
See Also	4-12

5. Configuring Your Application to Use Transactions

Modifying the UBBCONFIG File to Accommodate Transactions	5-1
Specifying Global Transaction Parameters in the RESOURCES Section	5-2
Creating a Transaction Log (TLOG) in the MACHINES Section	5-3
Creating the UDL	5-4
Defining Transaction-related Parameters in the MACHINES Section	5-4
Creating the Domains Transaction Log	5-5
See Also	5-6
Defining Resource Managers and the Transaction Manager Server in the GROUPS Section	5-6
Sample of the GROUPS Section	5-7
Enabling a Service to Begin a Transaction in the SERVICES Section	5-8
Characteristics of the AUTOTRAN, TRANTIME, and ROUTING Parameters	5-9
Modifying the Domains Configuration File to Support Transactions	5-10
Characteristics of the DMTLOGDEV, DMTLOGNAME, DMTLOGSIZE, MAXRDTRAN, and MAXTRAN Parameters	5-10
Characteristics of the AUTOTRAN and TRANTIME Parameters	5-11
Example: A Distributed Application with Transactions	5-12
Sample RESOURCES Section	5-13
Sample MACHINES Section	5-14
Sample GROUPS and NETWORK Sections	5-15
Sample SERVERS, SERVICES, and ROUTING Sections	5-16
See Also	5-17

6. Distributing Applications Across a Network

What Is a Distributed Application	6-1
Example of a Distributed Application	6-3

Implementing a Distributed Application.....	6-4
Why Distribute an Application Across a Network.....	6-5
Features of a Distributed Application	6-6

7. Creating the Configuration File for a Distributed Application

Configuration File Requirements for a Distributed Application.....	7-2
Creating the RESOURCES Section	7-3
Creating the MACHINES Section.....	7-5
Creating the GROUPS Section.....	7-7
Creating the SERVICES Section.....	7-8
Creating the ROUTING Section	7-10
Example Configuration File for a Distributed Application	7-12
Modifying the Domain Gateway Configuration File to Support Routing.....	7-13
Description of ROUTING Section Parameters in DMCONFIG.....	7-13

8. Setting Up the Network for a Distributed Application

Configuring the Network for a Distributed Application	8-1
How Data Moves Over a Network	8-5
How Data Moves Over Parallel Networks	8-6
Example of a Network Configuration for a Simple Distributed Application....	8-8
How Failover and Failback Work in Scheduling Network Data.....	8-8
Example Configuration of Multiple Netgroups.....	8-9
Configuration File for the Sample Network.....	8-11
Assigning Priorities for Each Network Group	8-11

9. About Workstation Clients

What Is the Workstation Component	9-1
Sample Application with Four Workstation Clients	9-2
How the Workstation Client Connects to an Application	9-4

10. Setting Up Workstation Clients

Defining Workstation Clients.....	10-1
Specifying the Maximum Number of Workstation Clients.....	10-3
Defining a Workstation Listener (WSL) as a Server.....	10-4
Passing Information to a WSL Process	10-4
Using Command-line Options Set with CLOPT.....	10-5

Detecting Network Failures	10-7
Using the Keep-alive Option	10-8
Limitations When Using the Keep-alive Option	10-9
Using the Network Timeout Option	10-10
How Network Timeout Works	10-10
Limitations When Using the Network Timeout Option	10-11
Setting the Network Timeout Option	10-11
Sample Configuration File that Supports Workstation Clients	10-11
Modifying the MACHINES and SERVERS Sections	10-12

1 Administrative Tasks and Tools

- Tasks an Administrator Performs
- Planning the Design of Your Application
- Tools to Help You Administer Your Application

Tasks an Administrator Performs

An administrator's job can be viewed as two broadly defined tasks:

- **Setup tasks**—all the tasks required to prepare your system before booting your application
- **Run-time administration**—any tasks performed on an application that has been booted

Setup Tasks

During the setup phase, an administrator is responsible for the planning, design, installation, security, and configuration of the BEA Tuxedo system. The following table describes the required and optional tasks during the setup phase.

1 Administrative Tasks and Tools

Setup Task	Required	Optional
Collect information from designers, programmers, and business users of the application	X	
Set up the hardware and software, and install the BEA Tuxedo system and the application (installation)	X	
Set up the BEA Tuxedo system parameters that govern how the application uses components (configuration)	X	
Configure transactions for domains, machines, groups, interfaces, services, and other required components (configuration)	X	
Select and implement security methods for protecting the application and data	X	
Set up distributed applications with routing tools		X
Set up networked applications		X
Configure local and remote domains		X
Set up Workstation clients: add environment tables and a Workstation Listener, and modify the machine configuration		X
Create an application queue space and modify the configuration to support queued messages		X

Run-time Tasks

With your BEA Tuxedo system installed and your `TUXCONFIG` file loaded, you are ready to boot your application. When your application is launched, you must start monitoring its activities for problems—both actual and potential. The following table describes the required and optional tasks during the run-time phase.

Run-Time Task	Required	Optional
Start up and shut down an application	X	

Run-Time Task	Required	Optional
Manage buffers	X	
Administer the security of your application	X	
Monitor the activities, problems, and performance of your application	X	
Manage transactions		X
Manage networked applications		X
Manage remote Workstation clients		X
Subscribe to events		X
Use queued messaging		X
Identify and resolve problems as they occur (troubleshoot)		X
Reassign primary responsibility for your application from the MASTER machine to an alternate (BACKUP) machine (migration) when problems occur on the MASTER (migration)		X
Change system parameters and the selection of services to meet evolving needs (dynamic modification)		X
Refine your application to reflect additional components, such as new machines or servers (dynamic reconfiguration)		X

During run time, you may need to respond quickly to potential problems or evolving requirements of an application. To help you perform these functions, you have a choice of three tools: the BEA Administration Console, the command-line interface, and the AdminAPI. The following chart describes some of the circumstances in which your intervention may be needed.

To	You May Want To
Maximize performance	Add load balancing or set priorities for interfaces and services
Fix problems that may develop on the MASTER machine	Replace it with a designated BACKUP machine

To	You May Want To
Change processing and resource usage requirements	Add machines, servers, clients, interfaces, services, and so on

See Also

- “Planning the Design of Your Application” on page 1-4
- “Tools to Help You Administer Your Application” on page 1-6

Planning the Design of Your Application

An administrator needs to know a customer’s business requirements and how the software will be used. Once these needs are understood, administrators can work with their system designers and application developers to make sure that the application’s configuration can support its requirements.

Answers to the following preliminary questions may help in planning the design of your application.

1. How many machines will be used? _____
2. Will client applications reside on machines that are remote from the server applications? _____
3. Which services will your BEA Tuxedo application offer?

4. What resource managers (database) will the application use and where will they be located?

5. What “open” strings will the resource managers need?

6. What setup information will be needed for an RDBMS?

7. Will transactions be distributed? _____

8. Will the application use global transactions? _____

9. What buffer types will be used?

10. Will data be distributed across machines?

11. To which external domains will the application export services? From which external domains will the application import services?

12. Will data-dependent routing be used? _____

13. In what order of priority should services be available?

14. What are the reliability requirements? Will redundant listener and handler ports be needed? Will replicated server applications be needed?

15. Are there any conversational services?

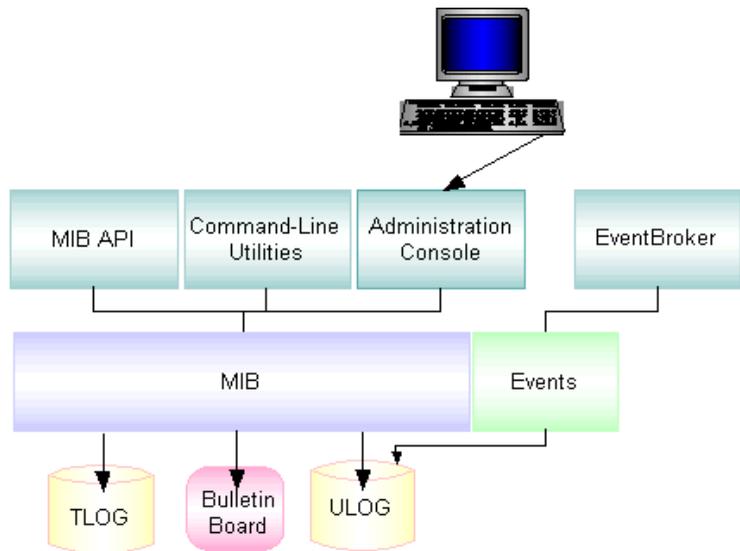
See Also

- “Tools to Help You Administer Your Application” on page 1-6

Tools to Help You Administer Your Application

The BEA Tuxedo system gives you a choice of several methods for performing the same set of administrative tasks. Whether you are more comfortable using a graphical user interface or entering commands at a shell prompt, you will be able to find a comfortable method of doing your job as the administrator of a BEA Tuxedo application. The following figure illustrates the tools you can use to write the configuration file and administer your BEA Tuxedo application during runtime.

Figure 1-1 Administration Tools



- *BEA Administration Console*—a Web-based tool used to monitor an application, and to dynamically configure its operation.
- *BEA Tuxedo MIB Application Programming Interface*—an interface to a set of procedures for accessing and modifying information in the MIBs.
- *Command-line utilities*—a set of commands used to manage, activate, configure, and deactivate the application (that is, `tmadmin(1)`, `tmboot(1)`, `tmconfig`, `wtmconfig(1)`, `tmshutdown(1)`, respectively). (See the *BEA Tuxedo Command Reference*.)

If You Use this Tool	You Must
BEA Administration Console	Use a graphical user interface (GUI) to create and edit the TUXCONFIG file. Full descriptions of the GUI are available by accessing Help directly from the GUI.
BEA Tuxedo MIB Application Programming Interface	Write a program that modifies the TUXCONFIG file for you.
Command-line interface	<ol style="list-style-type: none">1. Create and edit the UBBCONFIG file (a text version of TUXCONFIG) with a text editor.2. Run <code>tmloadcf</code> to convert the UBBCONFIG file into a TUXCONFIG (binary) file. <p>(For specific details about the <code>tmloadcf</code> command options, see <code>tmloadcf(1)</code> in <i>BEA Tuxedo Command Reference</i>.)</p>

See Also

- “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*
- “Using Command-Line Utilities” on page 3-12 in *Introducing the BEA Tuxedo System*
- “Tasks an Administrator Performs” on page 1-1
- “BEA Tuxedo System Architecture” on page 2-1 in *Introducing the BEA Tuxedo System*
- `ACL_MIB(5)`, `APPQ_MIB(5)`, `EVENT_MIB(5)`, `MIB(5)`, `TM_MIB(5)`, `WS_MIB(5)`, and `UBBCONFIG(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*
- `tmshutdown(1)` in *BEA Tuxedo Command Reference*

2 About the Configuration File

- What Is the Configuration File
- Contents of the Configuration File

What Is the Configuration File

Configuring each BEA Tuxedo application is a central task of the administrator. By configuring a file, you are describing your application using a set of parameters that the software interprets to create a viable application. The configuration file is a repository that contains all the information necessary to boot and run an application, such as specifications for application resources, machines, machine groups, servers, available services, and so on.

Text and Binary Versions of the Configuration File

The configuration file exists in two versions:

- The `UBBCONFIG` file is a text version of the configuration file, created and edited with any text editor. Except for sample configuration files distributed with BEA Tuxedo sample applications, no `UBBCONFIG` file is provided. You must create a `UBBCONFIG` file for each new application. The syntax used for entries in the file

is described in the `UBBCONFIG(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*.

Note: The BEA Tuxedo software provides three sample `UBBCONFIG` files—`ubbshm`, `ubbmp`, and `ubbsimple`—as part of the `bankapp` and `simpapp` applications. (See *Tutorials for Developing a BEA Tuxedo Application*.)

- The `TUXCONFIG` file is a binary version of the configuration file, created from the text version by the `tmloadcf(1)` command. Before `tmloadcf` is executed, the environment variable `TUXCONFIG` must be set to the full path name of the device or system file where `TUXCONFIG` is to be loaded. If necessary, many parameters in `TUXCONFIG` can be changed while the application is running by using `tmconfig`, `wtmconfig(1)` or the MIB.

Contents of the Configuration File

The following table lists the eight sections of the configuration file and describes the purpose of each section.

This Section	Required or Optional	Purpose
RESOURCES	Required	Defines all system parameters.
MACHINES	Required	Specifies all the machines in your application.
GROUPS	Required	Defines all groups, group names, and group IDs for your application.
SERVERS	Optional	Specifies the initial conditions for servers started in the system.
SERVICES	Optional	Provides information on services used by the application.
NETWORK	Optional	Describes the network configuration for a LAN environment.
NETGROUPS	Optional	Describes the network groups available to the application in the LAN environment.

This Section	Required or Optional	Purpose
ROUTING	Optional	Provides information for data-dependent routing of service requests using FML buffers and views.

The file must also contain a minimum of nine parameters. There are 80 different parameters, and all sections but the first, may contain multiple entries, each with its own selection of parameters. In all sections other than `RESOURCES`, you can use a default to specify parameters that are included in multiple entries.

You can use the command-line interface or BEA Administration Console to create the binary version of the configuration file (`TUXCONFIG`). First you need to determine the type of configuration you are defining in the file.

- A single-machine application—One or more local or remote clients communicate with one or more servers residing on the same machine.
- A multiple-machine (distributed) application—One or more local or remote clients communicate with one or more servers residing on several machines.
- A multiple-domain application—Two or more applications communicate with each other through the use of the BEA Tuxedo Domains extension. Each application included in such a configuration is called a domain.

See Also

- “How to Create a Configuration File” on page 3-2
- “What Is a Single-machine Configuration” on page 3-43 in *Introducing the BEA Tuxedo System*
- “What Is a Multiple-machine (Distributed) Configuration” on page 3-45 in *Introducing the BEA Tuxedo System*
- “What Is a Multiple-domain Configuration” on page 3-49 in *Introducing the BEA Tuxedo System*
- “How to Create the TUXCONFIG File” on page 1-4 in *Administering a BEA Tuxedo Application at Run Time*

2 *About the Configuration File*

3 Creating the Configuration File

- How to Create a Configuration File
- How to Create the Configuration File for a Single-machine Application
- How to Create the Configuration File for a Multiple-machine (Distributed) Application
- How to Create the Configuration File for a Multiple-domain Application
- How to Create the RESOURCES Section of the Configuration File
- How to Create the MACHINES Section of the Configuration File
- How to Create the GROUPS Section of the Configuration File
- How to Create the NETWORK Section of the Configuration File
- How to Create the NETGROUPS Section of the Configuration File
- How to Create the SERVERS Section of the Configuration File
- How to Create the SERVICES Section of the Configuration File
- How to Create the ROUTING Section of the Configuration File
- How to Configure the BEA Tuxedo System to Take Advantage of Threads
- How to Compile a Configuration File

How to Create a Configuration File

Configuration file requirements are determined by the needs of your application. Following are instructions for several types of configurations:

- How to Create the Configuration File for a Single-machine Application
- How to Create the Configuration File for a Multiple-machine (Distributed) Application
- How to Create the Configuration File for a Multiple-domain Application
- How to Configure the BEA Tuxedo System to Take Advantage of Threads

See Also

- “About the Configuration File” on page 2-1
- `UBBCONFIG(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*

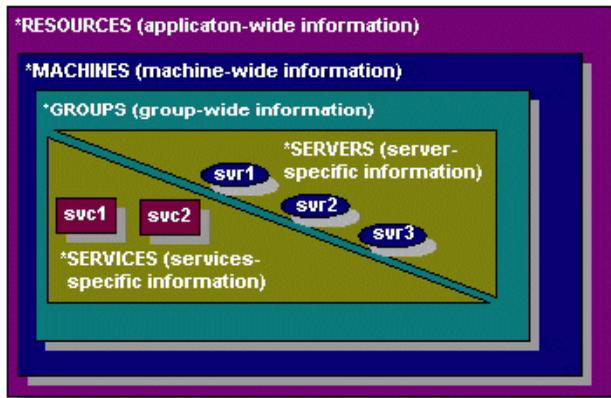
How to Create the Configuration File for a Single-machine Application

For a single-machine configuration, you need to create the following sections of the configuration file. Click on each task for instructions on completing that task.

1. [Create the `RESOURCES` section of the configuration file.](#)
2. [Create the `MACHINES` section of the configuration file.](#)
3. [Create the `GROUPS` section of the configuration file.](#)
4. [Create the `SERVERS` section of the configuration file.](#)

5. Create the `SERVICES` section of the configuration file.
6. Create the `ROUTING` section of the configuration file.

You can also click on any area of the following diagram to learn how to create the section named in that area.



How to Create the Configuration File for a Multiple-machine (Distributed) Application

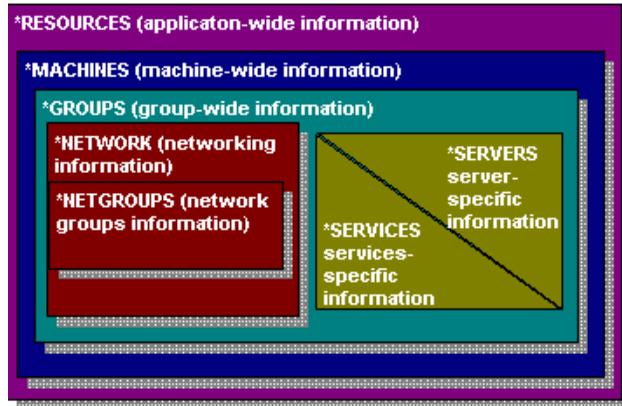
For a distributed application, you need to create the following sections of the configuration file. Click on each of the following tasks for instructions on completing that task.

1. Create the `RESOURCES` section of the configuration file.
2. Create the `MACHINES` section of the configuration file.
3. Create the `GROUPS` section of the configuration file.
4. Create the `NETWORK` section of the configuration file.
5. Create the `NETGROUPS` section of the configuration file.

3 *Creating the Configuration File*

6. Create the `SERVERS` section of the configuration file.
7. Create the `SERVICES` section of the configuration file.
8. Create the `ROUTING` section of the configuration file. (optional)

You can also click on any area of the following diagram to learn how to create the section named in that area.



How to Create the Configuration File for a Multiple-domain Application

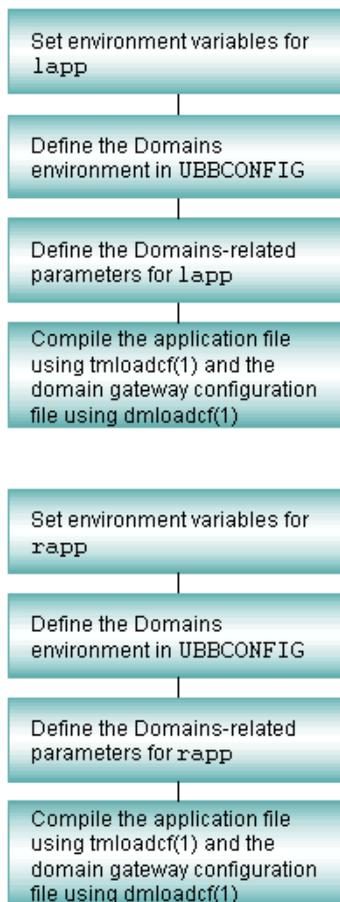
For a multiple-domain configuration, you need to create two configuration files for each participating domain:

- `UBBCONFIG`—the application configuration file
- `DMCONFIG`—the domains configuration file

For an application that consists of two domains (for example, `lapp` and `rapp` for local and remote domains, respectively), the following tasks are required.

Click on each task for instructions on completing that task.

Figure 3-1 Configuration Tasks for a Sample Multiple-domain Application

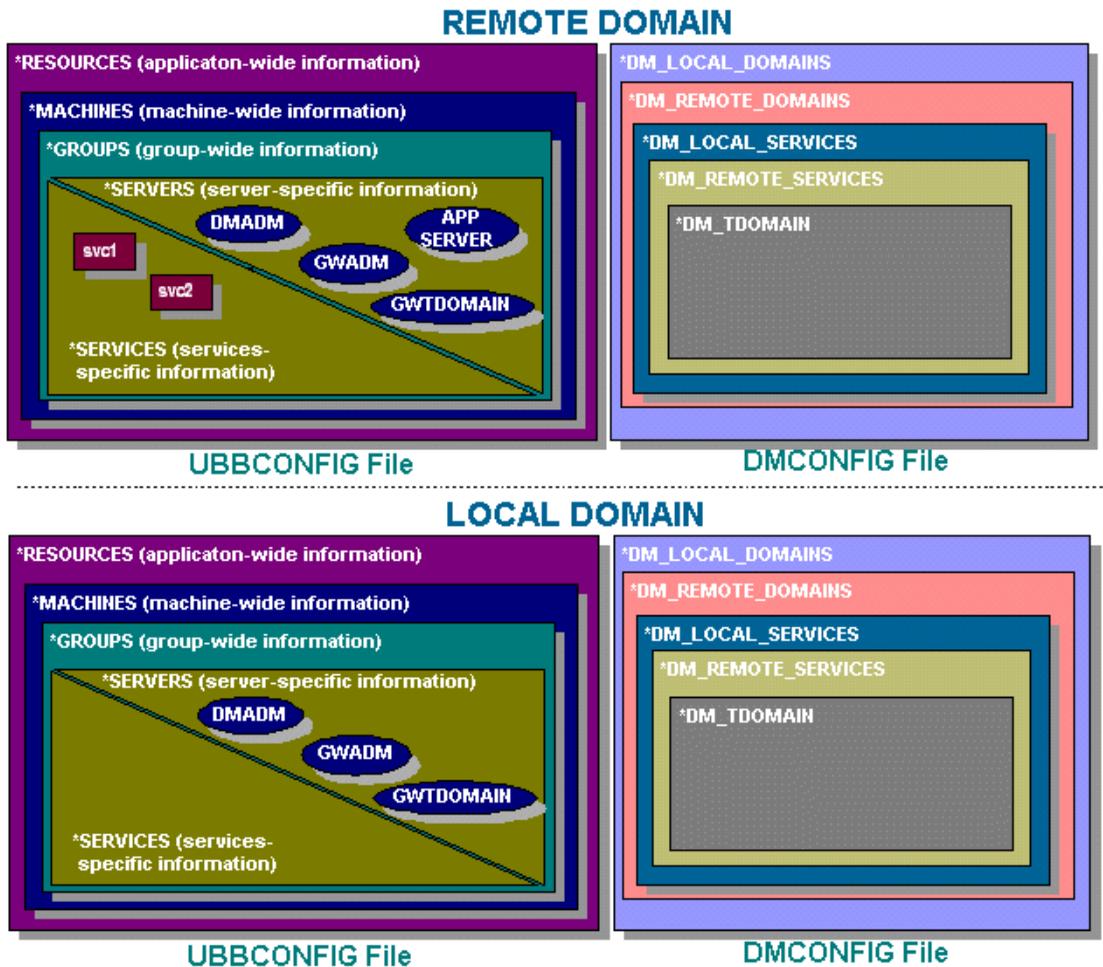


The following diagram shows which sections of the `UBBCONFIG` and `DMCONFIG` files you need to configure for a 2-domain application. One domain represents the local domain; the other, the remote domain.

3 Creating the Configuration File

Click on any area of the following diagram for instructions on creating that section of the configuration file.

Figure 3-2 Configuring a Multiple-domain Application



See Also

- “About Domains” on page 1-1 in *Using the BEA Tuxedo Domains Component*
- “Configuring a Domains Environment” on page 2-18 in *Using the BEA Tuxedo Domains Component*
- `DMCONFIG(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*

How to Create the RESOURCES Section of the Configuration File

The first section of every configuration file must be the `RESOURCES` section. The parameters defined in this section control the application as a whole and serve as system-wide defaults. The values of `RESOURCES` parameters can be overridden, however, on a per-machine basis by assigning other values in the `MACHINES` section.

For each parameter in the `RESOURCES` section, the following table provides a description and links to reference pages and additional information.

To Specify This Information in the RESOURCES Section . . .	Set This Parameter (Required/Optional)	For More Information, Click the Following
Unique address of interprocess communication (IPC) resources	<code>IPCKEY</code> (required)	shared memory address
Security access	<code>UID</code> , <code>GID</code> , and <code>PERM</code> (optional)	security access
Maximum number of processes that can be simultaneously connected to a bulletin board	<code>MAXACCESSERS</code> (optional)	IPC limits
Maximum number of server table entries in a bulletin board	<code>MAXSERVERS</code> (optional)	IPC limits
Maximum number of service table entries in a bulletin board	<code>MAXSERVICES</code> (optional)	IPC limits

3 *Creating the Configuration File*

To Specify This Information in the RESOURCES Section . . .	Set This Parameter (Required/Optional)	For More Information, Click the Following
Distinguished Bulletin Board Liaison (DBBL) location at which booting, shutdown, and other administrative tasks are performed	MASTER (required)	master processor
Bulletin board architecture	MODEL, SHM or MP, and LAN or MIGRATE options (required)	application type
Security level	SECURITY, AUTHSVC (optional)	security levels
Principal name of the process used for identification, location of private key of principal user, and the environment variable containing the password	SEC_PRINCIPAL_NAME, SEC_PRINCIPAL_LOCATION, and SEC_PRINCIPAL_PASSVAR	security attributes
Default method for clients to detect unsolicited messages	NOTIFY, USIGNAL (optional)	unsolicited notification
Protecting shared memory	SYSTEM_ACCESS (optional)	shared memory protection
Whether server load balancing is enabled	LDBAL (optional)	load balancing
Maximum number of buffer types and subtypes	MAXBUFTYPE, MAXBUFSTYPES (optional)	buffer types/subtypes
Maximum number of conversations allowed on a machine	MAXCONV (optional)	conversation limits
Maximum number of network groups	MAXNETGROUPS (optional)	network groups
Sanity check frequency and amount of time allowed for blocking calls	SCANUNIT, SANITYSCAN, BLOCKTIME (optional)	sanity check frequency and blocking timeouts

Sample RESOURCES Section

```
*RESOURCES
IPCKEY      39211
UID         0
GID         1
PERM        0660
MAXACCESSERS 75
MAXSERVERS  40
MAXSERVICES 55
MASTER     SITE1, SITE2
MODEL       MP
OPTIONS     LAN, MIGRATE
SECURITY    APP_PW
AUTHSVC     "AUTHSVC"
NOTIFY      DIPIN
SYSTEM_ACCESS PROTECTED, NO_OVERRIDE
LDBAL       Y
```

See Also

- `UBBCONFIG(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*
- “How to Create the MACHINES Section of the Configuration File” on page 3-27

Defining the Application Type

Among the architectural decisions needed for a BEA Tuxedo application are the following:

- Should this application run on a single processor or multiprocessor with global shared memory?
- Will the application be networked?
- Will server migration be supported?

3 *Creating the Configuration File*

Use the `MODEL` and `OPTIONS` parameters to define the application type.

The `MODEL` parameter specifies whether an application runs on a single processor. It is set to `SHM` for uniprocessors and also for multiprocessors with global shared memory. A `MODEL` value of `MP` is used for multiprocessors that do not have global shared memory, as well as for networked applications. This is a required parameter.

The `OPTIONS` parameter is a comma-separated list of application configuration options. Two available options are `LAN` (indicating a networked configuration) and `MIGRATE` (indicating that application server migration is allowed).

Characteristics of the `MODEL` and `OPTIONS` Parameters

Parameter	Characteristics
<code>MODEL</code>	It is a required parameter. A value of <code>SHM</code> indicates a single machine with global shared memory. A value of <code>MP</code> indicates either multiple machines without global shared memory, or a networked application.
<code>OPTIONS</code>	It is a comma-separated list of application configuration options. A value of <code>LAN</code> indicates a local area network. A value of <code>MIGRATE</code> enables server migration. In the sample <code>RESOURCES</code> section, <code>MODEL</code> is set to <code>MP</code> ; <code>OPTIONS</code> is set to <code>LAN</code> and <code>MIGRATE</code> .

Example Settings

```
*RESOURCES
  MODEL      MP
  OPTIONS    LAN, MIGRATE
```

Controlling the Number of Buffer Types and Subtypes

You can control the number of buffer types and subtypes allowed in the application with the `MAXBUFTYPE` and `MAXBUFSTYPE` parameters, respectively. Unless you are creating many user-defined buffer types, you can omit `MAXBUFTYPE`. If you intend to use many different `VIEW` types, you may want to set `MAXBUFSTYPE` to a value higher than its current default.

Characteristics of the `MAXBUFTYPE` and `MAXBUFSTYPES` Parameters

Parameter	Characteristics
<code>MAXBUFTYPE</code>	Maximum number of buffer types allowed in the system. Use only if you create 8 or more user-defined buffer types. The value of <code>MAXBUFTYPE</code> must be greater than 0 and less than 32,768. If not specified, the default is 16. Example: <code>MAXBUFTYPE 20</code>
<code>MAXBUFSTYPE</code>	Maximum number of buffer subtypes allowed in the system. The value of <code>MAXBUFSTYPE</code> must be greater than 0 and less than 32,768. If not specified, the default is 32. Example: <code>MAXBUFSTYPE 40</code>

Example Settings

```
*RESOURCES
    MAXBUFTYPE 20
    MAXBUFSTYPE 40
```

In this example, the maximum number of buffer types is 20; the maximum number of subtypes is 40.

Controlling the Number of Conversations

You can specify the maximum number of simultaneous conversations on a machine with the `MAXCONV` parameter. The value of `MAXCONV` must be greater than 0 and less than 32,768.

Characteristics of the `MAXCONV` Parameter

The `MAXCONV` parameter has the following characteristics:

- It defines the maximum number of simultaneous conversations allowed on each machine.
- The default for an application that has conversational servers listed in the `SERVERS` section is 10; otherwise, the default is 1.
- You can overwrite this parameter for any machine by specifying a different value in the `MACHINES` section.

Example Setting

```
*RESOURCES
      MAXCONV    15
```

In this example, the maximum number of simultaneous conversations allowed on each machine is 15.

Defining IPC Limits

Because most interprocess communication (IPC) and shared memory bulletin board tables are statically allocated for speedy processing, it is important to tune them correctly. If they are sized too generously, memory and IPC resources are wasted; if

too small, processes fail when the limits are exceeded. You can use the `tmloadcf -c` command to find out the maximum IPC resources required by a specific application. (See `tmloadcf(1)` in *BEA Tuxedo Command Reference*.)

`MAXACCESSERS`, `MAXSERVERS`, and `MAXSERVICES` are the tunable parameters that control IPC sizing. The amount of shared memory allocated in an application is controlled by the `MAXGTT` and `MAXCONV` parameters.

Characteristics of `MAXACCESSERS`, `MAXSERVERS`, and `MAXSERVICES` Parameters

Parameter	Characteristics
<code>MAXACCESSERS</code>	<p>Maximum number of overall processes that can be simultaneously connected to the bulletin board at any particular site in the BEA Tuxedo application. This number includes all clients and system-supplied and application servers, but does not include administrative processes such as the Bulletin Board Liaison (BBL) and <code>tmadmin()</code>, which have reserved access slots to the bulletin board.</p> <p>The value of <code>MAXACCESSERS</code> must be greater than 0 and less than 32,768. If not specified, the default is 50. You can overwrite <code>MAXACCESSERS</code>, on a per-machine basis, in the <code>MACHINES</code> section.</p>
<code>MAXSERVERS</code>	<p>Maximum number of server processes available to the application. This number includes all system-supplied and application servers.</p> <p>The value of <code>MAXSERVERS</code> must be greater than 0 and less than 8,192. If not specified, the default is 50.</p>
<code>MAXSERVICES</code>	<p>Maximum number of different BEA Tuxedo services that can be advertised in the application. The value of <code>MAXSERVICES</code> must be greater than 0 and less than 32,768. If not specified, the default is 100.</p>

Note: Examples of system-supplied servers are `AUTHSVR`, `TMQUEUE`, `TMQFORWARD`, `TMUSREVT`, `TMSYSEVT`, `TMS`, `TMS_QM`, `GWTDOMAIN`, and `WSL`.

3 Creating the Configuration File

The cost incurred by increasing `MAXACCESSERS` is one additional *semaphore* per site per client or server process (accesser—see note that follows). There is a small fixed semaphore overhead for system processes in addition to that added by the `MAXACCESSERS` value. The cost of increasing `MAXSERVERS` and `MAXSERVICES` is a small amount of shared memory that is kept for each server, service, and client entry, respectively. The general idea for these parameters is to allow for future growth of the application. It is more important to scrutinize `MAXACCESSERS`.

Note: The system allocates one semaphore for each access slot to the bulletin board. A semaphore is a latch circuit that prevents more than one process from accessing the same shared memory in the bulletin board at the same time.

For BEA Tuxedo pre-Release 7.1 (6.5 or earlier), both the `MAXACCESSERS` and `MAXSERVERS` parameters for an application play a part in the user license checking scheme. Specifically, a machine is not allowed to boot if the number of `MAXACCESSERS` for that machine + the number of `MAXACCESSERS` for the machine (or machines) already running in the application is greater than the number of `MAXSERVERS` + user licenses for the application. Thus, the total number of `MAXACCESSERS` for an application must be less than or equal to the number of `MAXSERVERS` + user licenses for the application.

The user license checking scheme in BEA Tuxedo Release 7.1 or later considers only the following two factors when performing its checks: the number of user licenses for an application and the number of licenses currently in use for the application. When all user licenses are in use, no new clients are allowed to join the application.

Example Settings

```
*RESOURCES
    MAXACCESSER  75
    MAXSERVERS   40
    MAXSERVICES  55
```

In this example, at most 75 processes (clients and servers) can access the system at any one time. There is room for 40 servers advertising 55 services in the bulletin board.

Enabling Load Balancing

You can control whether a load balancing algorithm is used on the BEA Tuxedo application as a whole. When load balancing is used, a load factor is applied to each service within the system, allowing you to track the total load on every server. Every service request is sent to the qualified server that is least loaded.

To specify whether load balancing should be used, set the `LDBAL` parameter to `Y` (Yes) or `N` (No). By default, it is set to `N`.

You should use load balancing only if necessary; that is, whenever a service is offered by servers that use more than one queue. Load balancing is not appropriate for services offered by only one server, or by servers in an `MSSQ` (multiple-server, single-queue) set. If you have only these types of services in your configuration, set the `LDBAL` parameter to `N`. If `LDBAL` is set to `N` and multiple queues offer the same service, the first available queue is selected.

Characteristics of the `LDBAL` Parameter

The `LDBAL` parameter has the following characteristics:

- If `LDBAL` is set to `Y`, then load balancing is used.
- If `LDBAL` is set to `Y` and the application is networked, you can use `TMNETLOAD` for local preference.
- If `LDBAL` is set to `N`, the server assigned is the first available server.
- The default is `N`.
- Because `LDBAL` incurs overhead, use it only when necessary.
- Do not use load balancing if every BEA Tuxedo service is offered by only one server.
- Do not use load balancing if every BEA Tuxedo service is offered by one `MSSQ` server set.

Example Settings

```
*RESOURCES
    LDBAL    Y
```

See Also

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

Identifying the Master Machine

The MASTER machine controls the booting and administration of the entire application. You must specify a MASTER machine for every application by setting the MASTER parameter. The value of MASTER is the Logical Machine Identifier (LMID) for the appropriate computer. The LMID, in turn, is defined as an alphanumeric string, chosen by the administrator, that is assigned to the LMID parameter in the MACHINES section. Therefore, for example, if the value of the LMID parameter is SITE1, then the value of MASTER must also be SITE1.

If you want to be able to bring down the MASTER machine without shutting down the application, you must be able to migrate the MASTER. To enable migration, you must specify two values for LMID: the primary MASTER and the backup MASTER.

Characteristics of the MASTER Parameter

The MASTER parameter has the following characteristics:

- It is required and it controls booting and administration.
- Two LMIDS are required for migration to back up the master machine.
- In the sample RESOURCES section, the master site is SITE1; the backup site is SITE2.

Example Settings

```
*RESOURCES
    MASTER SITE1, SITE2
```

Site1 is the MASTER machine; SITE2 is the backup machine.

Specifying the Maximum Number of Network Groups

To specify the maximum number of configured network groups, set the `MAXNETGROUPS` parameter. The value must be greater than or equal to 1 and less than 8192. The default is 8. This parameter is optional.

Specifying the Number of Sanity Checks and Blocking Timeouts

Periodically (every 120 seconds, by default) the bulletin board liaison (BBL) checks the sanity of the servers on its machine. You can change the frequency of these checks, however, by setting the `SCANUNIT` and `SANITYSCAN` parameters. In addition, you can specify the number of timeout periods for blocking messages, transactions, and other system activities by setting the `BLOCKTIME` parameter. The value you assign must be a positive multiple of 5.

Use the `SANITYSCAN` parameter to specify how many `SCANUNITs` elapse between sanity checks of the servers. Its current default is set so that `SANITYSCAN * SCANUNIT` is approximately 120 seconds.

Characteristics of the SCANUNIT, SANITYSCAN, and BLOCKTIME Parameters

Parameter	Characteristics
SCANUNIT	Controls the granularity of check intervals and timeouts. SCANUNIT must be a multiple of 5 between 0 and 60 seconds. Example: SCANUNIT 10 The default is 10.
SANITYSCAN	Specifies how many scan units elapse between sanity checks of the servers. SANITYSCAN may be any number up to 32767. The default is such that SCANUNIT * SANITYSCAN is approximately 120 seconds.
BLOCKTIME	Controls how long a message can block before it times out. SCANUNIT * BLOCKTIME must not exceed 32767. The default is such that SCANUNIT * BLOCKTIME is approximately 60 seconds.

Timeouts for Blocking ATMI Operations

The term *timeout* is used to refer, collectively, to the amount of time that elapses while a client:

- Waits to send a message into the request queue
- Waits to receive a message from the reply queue
- Is processed by the server
- Travels on the network

The term *blocking timeout* refers to the amount of time spent by a client request waiting for a blocking condition to clear up. Block timeouts for asynchronous service requests and conversations apply to individual send and receive operations. When a process

sends a message using `tpacall (3c)`, `tpconnect (3c)`, or `tpsend (3c)`, the timeout applies only to the period during which the request waits to get on the queue if the queue is full. When a client process issues a `tpgetreply (3c)` or `tprecv(3c)` call to receive a message, the timeout specifies how long the client may wait for the incoming message if its queue is empty.

Example Settings

```
*RESOURCES
  SCANUNIT      10
  SANITYSCAN    3
  BLOCKTIME     1
```

In this example, sanity scans are performed every 30 seconds and requests block for no more than 10 seconds. A `SCANUNIT` of 10 and a `SANITYSCAN` of 3 allow 3 blocks of 10 seconds or 30 seconds to elapse before the BBL scans.

Establishing Operating System-level Security

You can restrict access to BEA Tuxedo administrative functions to authorized administrators only, by setting three parameters: `UID`, `GID`, and `PERM`.

The defaults of `UID` and `GID` are the user ID and group ID, respectively, of the person who runs the `tmloadcf(1)` command on the configuration, unless overriding values have been specified in the `MACHINES` section.

Characteristics of the UID, GID, and PERM Parameters

Parameter	Characteristics
UID	<p>The user ID of the administrator. The value is a numeric string corresponding to the UNIX system user ID of the person who boots and shuts down the system.</p> <p>The default is the user ID of the person who runs <code>tmloadcf(1)</code>.</p> <p>Example: <code>UID=3002</code></p> <p>Note: On Windows NT, this value must be set to 0.</p>
GID	<p>The numeric group ID of the administrator.</p> <p>The default is the group ID of the person who runs <code>tmloadcf(1)</code>.</p> <p>Example: <code>GID=100</code></p> <p>Note: On Windows NT, this value must be set to 0.</p>
PERM	<p>The value is an octal number that specifies permissions for the IPC resources created when the application is booted. This parameter provides the first level of defense of the BEA Tuxedo system IPC structures against unauthorized access. These values should be specified for production applications.</p> <p>The default is 0666, which gives read/write access to all.</p> <p>Example: <code>PERM=0660</code></p>

Note: You can overwrite the values assigned to these parameters for remote machines. The user and group IDs on a remote machine are not required to be the same as the user and group IDs on the `MASTER` machine. You can override the defaults by specifying different user and group IDs in the `MACHINES` section of the configuration file. If not specified, values specified in the `RESOURCES` section are used.

Specifying the Security Level

You can set the following three levels of security:

- **PERM** parameter—provides minimal security by restricting, through permissions, the ability to write to the application queues.
- **SECURITY** parameter—provides greater security. When this parameter is set, a client must supply a password when joining the application. This password is checked against the password supplied by the administrator when the `TUXCONFIG` file is generated from the `UBBCONFIG` file.
- **AUTHSVC** parameter—sets the maximum level of security. When this parameter is set, any client request to join the application is sent to an authentication service. The authentication service may be the default service supplied by the BEA Tuxedo system or a third-party vendor service, such as a Kerberos service. This level of security cannot be used unless the **SECURITY** parameter is set.

Characteristics of the SECURITY and AUTHSVC Parameters

Parameter	Characteristics
<code>SECURITY</code>	<p>Security level that requires a password to join an application. Accepted values are: <code>NONE</code> (default), <code>APP_PW</code>, <code>USER_AUTH</code>, <code>ACL</code>, and <code>MANDATORY_ACL</code>.</p> <p>Default is <code>NONE</code>.</p> <p>Example: <code>SECURITY APP_PW</code></p>
<code>AUTHSVC</code>	<p>The name of the authentication service.</p> <p><code>SECURITY APP_PW</code> or higher must be specified.</p> <p>Default is no authentication service.</p> <p>Client authentication with Kerberos is possible.</p> <p>Example: <code>AUTHSVC "AUTHSVC"</code></p>

See Also

- “Introducing Security” on page 1-1 in *Using BEA Tuxedo Security*

Defining the Security Attributes of a Server

You can use the `SEC_PRINCIPAL_NAME`, `SEC_PRINCIPAL_LOCATION`, and `SEC_PRINCIPAL_PASSVAR` parameters to identify the security attributes of any servers used for authentication.

- `SEC_PRINCIPAL_NAME`—defines the principal name used by the server for various security operations.
- `SEC_PRINCIPAL_LOCATION`—specifies the location of the private key of the principal user.
- `SEC_PRINCIPAL_PASSVAR`—specifies the environment variable that contains the password used to open the private key of the principal user.

If Specified in This Section	This Parameter Defines	And Overrides Parameter Settings in This Section
RESOURCES	All system servers booted in the domain.	N/A
MACHINES	All system servers booted on a machine.	RESOURCES
GROUPS	All system and interoperating application servers booted within a group.	MACHINES
SERVERS	All system and interoperating application services booted within a server.	GROUPS

Note: These policies apply to the Workstation Handler, Domains gateway processes, and interoperating application servers.

See Also

- “Introducing Security” on page 1-1 in *Using BEA Tuxedo Security*
- “Administering Security” on page 2-1 in *Using BEA Tuxedo Security*

Protecting Shared Memory

You can shield system tables kept in shared memory from application clients and/or servers using the `SYSTEM_ACCESS` parameter. This parameter is useful when applications are being developed because faulty application code can inadvertently corrupt shared memory with a bad pointer. Once an application is fully debugged and tested, the value of this parameter can be changed to allow for faster responses. Following are valid values for this parameter:

- `PROTECTED`—BEA Tuxedo libraries compiled with application code do not attach to shared memory while executing system code.
- `FASTPATH`—BEA Tuxedo libraries attach to shared memory at all times.

Once you select a value, you can specify `NO_OVERRIDE`, which means that the selected option cannot be changed either by the client, in the `TPINIT` structure of the `tpinit()` call, or by the administrator, in the `SERVERS` section for servers.

Characteristics of the PROTECTED, FASTPATH, and NO_OVERRIDE Parameters

Parameter	Characteristics
PROTECTED	Internal structures in shared memory are not corrupted inadvertently by application processes.
FASTPATH (default)	Application processes join the application with access to shared memory at all times.
NO_OVERRIDE	The specified option (either PROTECTED or FASTPATH) cannot be changed.

Example Settings

```
SYSTEM_ACCESS PROTECTED, NO_OVERRIDE
```

Setting the Address of the System Resources for an Application

To set the address of shared memory, set the `IPCKEY` parameter. This parameter is used by the BEA Tuxedo system to allocate application IPC resources such that they may be located easily by new processes joining the application. This key and its variations are used internally to allocate the bulletin board, message queues, and semaphores that must be available to new application processes. In single processor mode, this key names the bulletin board; in multiprocessor mode, this key names the message queue of the DBBL.

Characteristics of the `IPCKEY` Parameter

The `IPCKEY` parameter has the following characteristics:

- It is required.
- It is used to access the bulletin board and other IPC resources.
- Its value must be an integer in the range 32,769 to 262,144.
- No other application on the system may use this specific value for its `IPCKEY`. Its value must be unique among all applications.

Example Settings

```
*RESOURCES
  IPCKEY 39211
```

Specifying How Clients Receive Unsolicited Notification

You can select the default method by which clients receive unsolicited messages by setting the `NOTIFY` parameter. The client, however, can override this choice when calling `tpinit()`.

Following are four possible methods:

- `IGNORE`—Clients ignore unsolicited messages.
- `DIPIN`—Clients receive unsolicited messages only when they call `tpchkunsol()` or when they make an ATMI call.
- `SIGNAL`—Clients receive unsolicited messages by having the system generate a signal that has the signal handler call the function, that is, set with `tpsetunsol()`.

3 Creating the Configuration File

Note: This method is not allowed for multithreaded or multicontexted applications.

- **THREAD**—Unsolicited messages are handled by a separate thread managed by the BEA Tuxedo system for this purpose.

The **USIGNAL** parameter specifies the signal to be used if **SIGNAL**-based notification is used. Two types of signals can be generated: **SIGUSR1** and **SIGUSR2**. The default is **SIGUSR2**. This method has the advantage of immediate notification, but is limited when you are running a native client. In that case, you must have the same user ID as the sending process. Workstation clients do not have this limitation.

Note: This method is not available on all platforms.

Characteristics of the NOTIFY and USIGNAL Parameters

Parameter	Characteristics
NOTIFY	<p>Value of IGNORE means clients should ignore unsolicited messages.</p> <p>Value of DIPIN means clients should receive unsolicited messages only when they call <code>tpchkunsol()</code> or when they make an ATMI call.</p> <p>Value of SIGNAL means clients should receive unsolicited messages by signals.</p> <p>Default is DIPIN.</p> <p>Example: NOTIFY SIGNAL</p>
USIGNAL	<p>Value of SIGUSR1 and SIGUSR2 means notify clients with this type of signal.</p> <p>Default is SIGUSR2.</p> <p>Example: USIGNAL SIGUSR1</p>

How to Create the MACHINES Section of the Configuration File

The second section of every configuration file must be the `MACHINES` section. The `MACHINES` section defines parameters for each machine in an application. These parameters provide the following information:

- The mapping of the machine *address* to a logical identifier (`LMID`)
- The location of the configuration file (`TUXCONFIG`)
- The location of the installed BEA Tuxedo software (`TUXDIR`)
- The location of the application servers (`APPDIR`)
- The location of the application log file (`ULOGPFX`)
- The location of the environment file (`ENVFILE`)

Note: For a particular machine, you can override the following system-wide parameters: `UID`, `GID`, `PERM`, `MAXACCESSERS`, `MAXCONV`, and `MAXGTT`. Each parameter, except `MAXGTT`, is described in the `RESOURCES` section.

For each parameter in the `MACHINES` section, the following table provides a description and links to reference pages and additional information.

To Specify This Information in the <code>MACHINES</code> Section . . .	Set This Parameter (Required/Optional)	For More Information, Click the Following
The number of entries in the cache used for ACL entries when <code>SECURITY</code> is set to <code>ACL</code> or <code>MANDATORY_ACL</code> .	<code>MAXACLCACHE</code> (optional)	ACL entries in the cache
The additional load to be added when computing the cost of sending a service request from this machine to another machine.	<code>NETLOAD</code> (optional)	additional loads

3 Creating the Configuration File

To Specify This Information in the MACHINES Section . . .	Set This Parameter (Required/Optional)	For More Information, Click the Following
The address is the name of the physical processor, which all other entries describe. The LMID parameter specifies the logical name of the computer.	LMID (required)	Address and machine ID
The number of attempts that should be made at user level to lock the bulletin board before blocking processes on a UNIX semaphore.	SPINCOUNT (optional)	bulletin board locking limit
A value used for grouping machines into classes.	TYPE (optional)	class grouping value
The absolute path name of the file or device where the binary TUXCONFIG file is found on this machine. Note: The path name specified for this parameter must match exactly (including case) the path name specified for the TUXCONFIG environment variable. Otherwise, <code>tmloadcf(1)</code> cannot be run successfully.	TUXCONFIG (required)	configuration file location
The maximum number of simultaneous conversations in which processes on a particular machine can be involved.	MAXCONV (optional)	conversation limits
The numeric size, in pages, of the DTP transaction log for this machine.	TLOGSIZE (optional)	DTP TLOG size
The name of the DTP transaction log for this machine.	TLOGNAME (optional)	DTP transaction log name
A value that specifies that all clients and servers on the machine are to be executed with the environment specified in the named file.	ENVFILE (optional)	environment variable settings
The BEA Tuxedo file system that contains the DTP transaction log (TLOG) for this machine.	TLOGDEVICE (optional)	file system containing the TLOG
The maximum number of processes that can have access to the bulletin board on this processor at any one time.	MAXACCESSERS (optional)	IPC limits

How to Create the MACHINES Section of the Configuration File

To Specify This Information in the MACHINES Section . . .	Set This Parameter (Required/Optional)	For More Information, Click the Following
The maximum number of simultaneous global transactions in which a particular machine can be involved.	MAXGTT (optional)	limit of simultaneous global transactions
The number of accessor entries on this processor to be reserved for Workstation clients. The parameter is only used when the BEA Tuxedo system Workstation component is used.	MAXWSCLIENTS (optional)	limit of workstation accessor entries
A limit for the amount of space that can be allocated for messages waiting to be transmitted by the bridge process.	MAXPENDINGBYTES (optional)	message space limits
The numeric offset in pages (from the beginning of the device) to the start of the BEA Tuxedo file system that contains the DTP transaction log for this machine.	TLOGOFFSET (optional)	numeric offset containing the DTP TLOG
The numeric offset in pages (from the beginning of the device) to the start of the BEA Tuxedo file system that contains the TUXCONFIG file for this machine.	TUXOFFSET (optional)	numeric offset containing the TUXCONFIG
The numeric group ID to be associated with the IPC structures created for the bulletin board. The valid range is 0-2147483647. If not specified, the default is the value specified in the RESOURCES section.	GROUPID (optional)	security access
The numeric permissions associated with the IPC structures that implement the bulletin board. This parameter is used to specify the read/write permissions for processes in the usual UNIX system fashion (that is, with an octal number such as 0600). The value can be between 0001 and 0777, inclusive. If not specified, the default is the value specified in the RESOURCES section.	PERM (optional)	security access
The numeric user ID to be associated with the IPC structures created for the bulletin board. The valid range is 0-2147483647. If not specified, the default is the value specified in the RESOURCES section.	UID (optional)	security access

3 Creating the Configuration File

To Specify This Information in the MACHINES Section . . .	Set This Parameter (Required/Optional)	For More Information, Click the Following
Principal name of the process used for identification, location of private key of principal user, and the environment variable containing the password	SEC_PRINCIPAL_NAME, SEC_PRINCIPAL_LOCA TION, SEC_PRINCIPAL_PASS VAR	security attributes
The absolute path name of the application directory (APPDIR), which is the current directory for all application and administrative servers booted on this machine; and the absolute path name of the directory where the BEA Tuxedo system software is found on this machine.	TUXDIR (required)	system and application software locations
The threshold message size for messages—bound to remote processes (<i>string_value1</i>) and local processes (<i>string_value2</i>), respectively—on which automatic data compression will be performed.	CMPLIMIT (optional)	threshold message size
The full path name to be used as the prefix of the name of the <code>userlog(3c)</code> message file on this machine.	ULOGPFX (optional)	ULOG path name

Sample MACHINES Section

Following is a sample MACHINES section of a configuration file.

```
*MACHINES
gumby          LMID=SITE1
               TUXDIR="/tuxdir"
               APPDIR="/home/apps/mortgage"
               TUXCONFIG="/home/apps/mortgage/tuxconfig"
               ENVFILE="/home/apps/mortgage/ENVFILE"
               ULOGPFX="/home/apps/mortgage/logs/ULOG"
               MAXACCESSERS=100
               MAXCONV=15
```

Sample MACHINES Parameters

In the preceding sample MACHINES section, the following parameters and values are specified.

Parameter	Meaning
gumby	The machine name obtained with the command <code>uname -n</code> on UNIX systems. On a Windows NT system, the value can be set using the Computer Name value in the Network Control Panel and must be specified in upper case.
LMID=SITE1	The logical machine identifier of the machine <i>gumby</i> .
TUXDIR	The full path to the installed BEA Tuxedo software (shown in double quotation marks).
APPDIR	The full path to the application directory (shown in double quotation marks).
TUXCONFIG	The full path name of the configuration file (shown in double quotation marks). Note: The path name specified for this parameter must match exactly (including case) the path name specified for the <code>TUXCONFIG</code> environment variable. Otherwise, <code>tmloadcf(1)</code> cannot be run successfully.
ENVFILE	The full path name of a file containing environment information (shown in double quotation marks).
ULOGPFX	The full path name to be used as the prefix of the name of the log file (shown in double quotation marks).
MAXACCESSERS	For this machine, override the system-wide value (defined in the RESOURCES section) with 100.
MAXCONV	For this machine, override the system-wide value (defined in the RESOURCES section) with 15.

How to Customize the Sample MACHINES Section

You can customize the MACHINES section by indicating the following:

- Your machine name for *gumby*

Note: On a Windows NT system, the machine name must be specified in upper case.

- The full path of your BEA Tuxedo software directory as the value of TUXDIR
- The full path of your application directory as the value of APPDIR
- The full path names for ENVFILE, TUXCONFIG, and ULOGPFX on your system

See Also

- UBBCONFIG(5) in *BEA Tuxedo File Formats and Data Descriptions Reference*
- “How to Create the GROUPS Section of the Configuration File” on page 3-44

Specifying the Maximum Number of ACL Entries in the Cache

You can use the `MAXACLCACHE` parameter to specify the number of ACL entries in the cache when `SECURITY` is set to `ACL` or `MANDATORY_ACL`. By setting of this parameter to an appropriate value, you can:

- Help conserve shared memory resources
- Reduce the number of disk accesses performed in order to do ACL checking

The value must be a number greater than or equal to 10, and less than or equal to 30,000. The default is 100.

Defining an Additional Service Request Load

You can use the `NETLOAD` parameter to specify a load to be added when computing the cost of sending a service request from one machine to another. The value must be a number greater than or equal to 0, and less than 32,768. The default is 0.

See Also

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

Reserving the Physical Address and Machine ID

You initially define the address of your `MASTER` machine in the address portion, which is the basis for a `MACHINES` section entry. All other parameters in the entry describe the machine specified by this address. You must set the address to the value printed by calling `uname -n` on UNIX systems. On Windows NT systems, see the Computer Name value in the Network Control Panel.

The `LMID` parameter is mandatory. It specifies a logical name used to designate the computer for which an address has just been provided. It may be any alphanumeric value, but it must be unique among other machines in the application.

Characteristics of the Address and the LMID Parameter

The address and machine ID have the following characteristics:

- The address and machine ID are specified as follows.

```
address LMID=logical_machine_name
```

The address identifies the physical processor name.

- The LMID is specified as follows.

```
LMID=logical_machine_name
```

The LMID is the logical machine name for a physical processor. It may be any alphanumeric string, but it must be unique within the `MACHINES` section.

Setting the Number of Lock Spins

For some BEA Tuxedo system operations (such as service name lookups and transactions), the bulletin board must be locked for exclusive access: that is, it must be accessible by only one process. If a process or thread finds that the bulletin board is locked by another process or thread, it retries, or *spins on* the lock for `SPINCOUNT` number of times before giving up and going to sleep on a waiting queue. Because sleeping is a costly operation, it is efficient to do some amount of spinning before sleeping.

Characteristics of the SPINCOUNT Parameter

Though the value of the `SPINCOUNT` parameter is application- and system-dependent, it may be helpful to keep the following basic guidelines in mind:

- A process on a uniprocessor system should not spin. If the bulletin board is locked when a uniprocessor process tries to access it, then the process with the lock should be allowed to run as quickly as possible. This is possible only if the *newcomer* process gives up immediately.

- A `SPINCOUNT` value of 1 is appropriate for uniprocessors.
- On multiprocessors, a good starting value is 5000, but some customers have benefited from a `SPINCOUNT` value as high as 100000.
- Set the `SPINCOUNT` value and observe your application throughput. Because you can tune the `SPINCOUNT` value using the `TMIB`, you can adjust it while the system is running.

Specifying Machines as Types

You can use the `TYPE` parameter to group machines into classes. You can set `TYPE` to any string that contains 15 or fewer characters.

Characteristics of the `TYPE` Parameter

- If two machines have the same `TYPE` value, data encoding/decoding is not performed when data is sent between the machines.
- `TYPE` can be given any string value. It is used simply for comparisons.
- The `TYPE` parameter should be used when the application involves a heterogeneous network of machines or when different compilers are used on the machines in the network.
- If a value not specified, the default is the null string, which matches any other entry for which a value has not been specified.

Identifying the Location of the Configuration File

To identify the configuration file location and filename for an entry that identifies a machine, set `TUXCONFIG`, a required parameter. The value of the `TUXCONFIG` parameter is enclosed in double quotes and represents a full path name, which may contain up to 64 characters.

Note: The path name specified for this parameter must match exactly (including case) the path name specified for the `TUXCONFIG` environment variable. Otherwise, `tmloadcf(1)` cannot be run successfully.

Characteristics of the TUXCONFIG Parameter

The `TUXCONFIG` parameter has the following characteristics:

- The syntax of the `TUXCONFIG` parameter is
`TUXCONFIG="full_path_of_tuxconfig"`.
- This parameter identifies the location and name of the configuration file.
- The value of `TUXCONFIG` can include up to 64 characters.
- The value of `TUXCONFIG` must match the value of the `TUXCONFIG` environment variable.

Indicating the Size of the DTP Transaction Log

Use the `TLOGSIZE` parameter to indicate the size, in pages, of the DTP transaction log for this machine. The value must be a number greater than 0, and less than or equal to 2048, subject to the amount of space available on the operating system file system. The default is 100 pages.

Defining the DTP Transaction Log Name

Use the `TLOGNAME` parameter to define the name of the DTP transaction log for this machine. The default is `TLOG`. If more than one `TLOG` exists on the same `TLOGDEVICE`, each must have a unique name. The value of `TLOGNAME` must be different from the name of any other table in the `VTOC` (Volume Table of Contents) on the `TLOGDEVICE` where the `TLOG` table is created. The value of `TLOGNAME` must be an alphanumeric string containing 30 or fewer characters.

Specifying Environment Variable Settings

With the `ENVFILE` parameter, you can specify a file that contains environment variable settings for all processes to be booted by the BEA Tuxedo system. The system sets `TUXDIR` and `APPDIR` for each process, so these parameters should not be specified in this file.

You can, however, specify settings for the following parameters because they affect an application's operation:

- `FIELDTBLS, FLDTBLDIR`
- `VIEWFILES, VIEWDIR`
- `TMCMLIMIT`
- `TMNETLOAD`

Characteristics of the ENVFILE Parameter

ENVFILE is an optional parameter with the following characteristics:

- The syntax of the value of the ENVFILE parameter is a string enclosed in double quotes: `ENVFILE="envfile"`.
- ENVFILE is the file containing environment variable settings for all processes booted by the BEA Tuxedo system. (The UBBCONFIG file issues warnings in a similar way, that is, using fully qualified path names.)
- Set FIELDTBLS, FLDTBLDIR, and so on, but do not set TUXDIR and APPDIR.
- All settings must be hard coded. No evaluations such as `FLDTBLDIR=$APPDIR` are allowed.
- The format for entries in the file is `VARIABLE=string`.

Defining the BEA Tuxedo File System Containing the TLOG

Use the TLOGDEVICE parameter to specify the BEA Tuxedo file system that contains the DTP transaction log (TLOG) for this machine. The TLOG is stored as a BEA Tuxedo system VTOC table on the specified device. The value of TLOGDEVICE must be a string containing a maximum of 64 characters.

If this parameter is not specified, then it is assumed that the machine does not have a TLOG.

Specifying a Machine's Maximum Number of Simultaneous Global Transactions

Use the `MAXGTT` parameter to indicate the maximum number of simultaneous global transactions in which a particular machine can be involved. The value must be a number greater than or equal to 0, and less than 32,768. You can override the value specified in the `RESOURCES` section with a value specified in the `MACHINES` section for an individual machine.

Defining the Number of Accesser Entries on a Workstation Client

Use the `MAXWSCLIENTS` parameter to define the number of entries on a machine to be reserved for Workstation clients. Set the number of accesser slots reserved for `MAXWSCLIENTS` cautiously, since this number takes a portion of the total accesser slots specified with `MAXACCESSERS` for this machine; the accesser slots reserved for `MAXWSCLIENTS` are unavailable for use by other clients and servers on this machine. By setting this parameter to an appropriate value, you can help conserve IPC resources because Workstation client access to the system is multiplexed through a BEA Tuxedo system-supplied surrogate, the Workstation Handler (WSH).

The value of `MAXWSCLIENTS` must be greater than or equal to 0 and less than 32,768. If not specified, the default is 0. It is an error to set this parameter to a number greater than `MAXACCESSERS`.

Note: The value of `MAXWSCLIENTS` is constrained by the number of your licensed users.

Defining Space Limits for Messages Transmitted by the BRIDGE

Use the `MAXPENDINGBYTES` parameter to define a limit for the amount of space that can be allocated for messages waiting to be transmitted by the `BRIDGE` process. This number must be between 100,000 and `MAXLONG`.

There are two situations when `MAXPENDINGBYTES` is significant:

- When the `BRIDGE` requests an asynchronous connection
- When all circuits are busy

You can configure larger computers that have more memory and disk space, with larger `MAXPENDINGBYTES`, and smaller computers with smaller `MAXPENDINGBYTES`.

Indicating the Offset for the DTP Transaction Log

Every BEA Tuxedo file system has a Volume Table of Contents (VTOC): a list of the files on the devices named in the Universal Device List (UDL). The UDL specifies the location of the physical storage space for BEA Tuxedo system tables. In a BEA Tuxedo system application, all system files might be stored together on the same raw disk slice or operating system file-system file.

Use the `TLOGOFFSET` parameter to indicate the offset in pages (from the beginning of the device) to the start of the BEA Tuxedo file system that contains the DTP transaction log for this machine. The offset must be a number greater than or equal to 0, and less than the number of pages on the device. The default is 0.

Defining the Offset for TUXCONFIG

Every BEA Tuxedo file system has a Volume Table of Contents (VTOC): a list of the files on the devices named in the Universal Device List (UDL). The UDL specifies the location of the physical storage space for BEA Tuxedo system tables. In a BEA Tuxedo system application, all system files might be stored together on the same raw disk slice or operating system file-system file.

Use the `TUXOFFSET` parameter to define the offset in pages (from the beginning of the device) to the start of the BEA Tuxedo file system that contains the `TUXCONFIG` for this machine. (For information on how this value is used in the environment, see the `ENVFILE` parameter in the `MACHINES` section.)

Characteristics of the TUXOFFSET Parameter

- The offset must be a number greater than or equal to 0, and less than the number of pages on the device.
- The default offset is 0.
- The value of `TUXOFFSET`, if non-zero, is placed in the environment of all servers booted on a machine.

Identifying the Locations of the System Software and Application Server Software

Each machine in an application that supports servers must have a copy of the BEA Tuxedo system software and application software. You identify the location of system software with the `TUXDIR` parameter. You identify the location of the application software with the `APPDIR` parameter. Both parameters are mandatory. The `APPDIR` parameter becomes the current working directory of all server processes. The BEA Tuxedo software looks in `TUXDIR/bin` and `APPDIR` for executables.

Characteristics of the APPDIR and TUXDIR Parameters

Parameter	Characteristics
APPDIR	<p>The syntax requires a full path name enclosed in double quotes: APPDIR="APPDIR".</p> <p>APPDIR identifies the location of application software.</p> <p>APPDIR is a required parameter.</p> <p>APPDIR becomes the current working directory of server processes.</p>
TUXDIR	<p>The syntax requires a full path name enclosed in double quotes: TUXDIR="TUXDIR".</p> <p>TUXDIR identifies the location of the BEA Tuxedo software.</p> <p>TUXDIR is a required parameter.</p>

Indicating a Threshold Message Size for Compression

Use the `CMPLIMIT` parameter to define the threshold message sizes at which automatic data compression is performed for messages bound to remote processes (*string_value1*) and local processes (*string_value2*), respectively.

Both values must be either a non-negative numeric value or the string `MAXLONG`. If not specified, the default is `MAXLONG,MAXLONG`.

Note: Set the `CMPLIMIT` value and observe your application throughput. Because you can tune the `CMPLIMIT` value using the `TMIB`, you can adjust it while the system is running.

Example

```
CMPLIMIT=string_value1,string_value2
```

Specifying the Path Name for the ULOG

Set the `ULOGPFX` parameter to specify the full path name to be used as the prefix of the name of the `userlog(3c)` message file on this machine. The value of `ULOGPFX` for a given machine is used to create the `userlog(3c)` message file for all servers, clients, and administrative processes executed on that machine. If this parameter is not specified, the path specified by the `APPDIR` environment variable is used. *mmddyy* (month, day, year) is appended to the prefix to form the full name of the log file.

Characteristics of the ULOGPFX Parameter

The `ULOGPFX` parameter has the following characteristics:

- The syntax of the value of the `ULOGPFX` parameter is a string enclosed in double quotes: `ULOGPFX="ULOGPFX"`.
- The application log contains all messages for `TPESYSTEM` and `TPEOS` errors.
- You can use the user log to log application errors.
- The `ULOGPFX` defaults to `APPDIR/ULOG`.
- For the sample filename `BANKLOG.022667`, the prefix of the name of the `userlog` is specified as follows.
`ULOGPFX="/mnt/usr/appdir/logs/BANKLOG"`

See Also

- “How to Create the GROUPS Section of the Configuration File” on page 3-44

How to Create the GROUPS Section of the Configuration File

Use the `GROUPS` section to designate logically grouped sets of servers, which can later be used to access resource managers, and facilitate server group migration. The `GROUPS` section of the configuration file contains definitions of server groups. You must define at least one server group for a machine to have application servers running on it. If no group is defined for a machine, the group can still be part of the application and you can run the administrative command `tadmin(1)` from that site.

For nontransactional, nondistributed systems, groups are relatively simple. You only need to map the group name to the number and logical machine ID for each group. Additional flexibility is available to support distributed transactional systems.

For each parameter in the `GROUPS` section, the following table provides a description and links to reference pages and additional information.

To Specify This Information in the GROUPS Section . . .	Set This Parameter (Required/Optional)	For More Information, Click the Following
The logical name of the group	<code>GROUPNAME</code> (required)	group name
The group number associated with this server group. This number must be greater than 0 and less than 30000, and must be unique among all entries in the <code>GROUPS</code> section.	<code>GRPNO</code> (required)	group number
The resource manager dependent information needed when closing the resource manager	<code>CLOSEINFO</code> (optional)	information for closing the resource manager
The resource manager dependent information needed when opening the resource manager	<code>OPENINFO</code> (optional)	information for opening the resource manager

How to Create the GROUPS Section of the Configuration File

To Specify This Information in the GROUPS Section . . .	Set This Parameter (Required/Optional)	For More Information, Click the Following
The number of transaction manager servers to start for the associated group, if TMSNAME is specified	TMSCOUNT (optional)	number of TMS servers in the group
Principal name of the process used for identification, location of private key of principal user, and the environment variable containing the password	SEC_PRINCIPAL_NAME, SEC_PRINCIPAL_LOCATION, SEC_PRINCIPAL_PASSWORD	security attributes
A value that specifies that all servers in the group are to be executed with the environment specified in the named file	ENVFILE (optional)	server group environment
A value that specifies that this group of servers resides on the machine symbolically named by <i>string_value1</i> in the MACHINES section (or the default in SHM mode)	LMID (required)	server group location
The name of the transaction manager server process associated with this group	TMSNAME (optional)	transaction manager server for group

Sample GROUPS Section

```
##EVBGRP1 LMID=SITE1          GRPNO=104

DEFAULT:TMSNAME=TMS_SQL TMSCOUNT=2 LMID=SITE1
BANKB1GRPNO=1 OPENINFO="TUXEDO/SQL:APPDIR1/bankd11:bankdb:readwrite"
BANKB2GRPNO=2 OPENINFO="TUXEDO/SQL:APPDIR1/bankd12:bankdb:readwrite"
BANKB3GRPNO=3 OPENINFO="TUXEDO/SQL:APPDIR1/bankd13:bankdb:readwrite"
```

See Also

- “How to Create the SERVERS Section of the Configuration File” on page 3-60

Specifying a Group Name, Number, and LMID

The group name, which is the basis for a `GROUPS` section entry, is an alphanumeric name by which the group is identified; it specifies the logical name (*string_value*) of the group. It is given a mandatory, unique group number (`GRPNO`). Each group must reside wholly on one logical machine (`LMID`).

The `LMID` specifies that this group of servers resides on the machine symbolically named by *string_value1* in the `MACHINES` section.

Characteristics of the Group Name, Group Number, and LMID

Parameter	Characteristics
<i>Group_name required_ parameters [optional_ parameters]</i>	It is required. It is an alphanumeric name by which the group is identified. It is unique and specifies the logical name of the group.
<code>GRPNO (Group Number)</code>	It is required and is unique.
<code>LMID=string_value1 [, string_value2]</code>	It is required. Each <code>LMID</code> value must be an alphanumeric string containing 30 or fewer characters. Up to two logical machine names can be specified. If a second logical name is given and server group migration is enabled, the machine with which the server group is associated can be migrated.

See Also

- `UBBCONFIG(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*
- “How to Create the NETWORK Section of the Configuration File” on page 3-51

Indicating a Transaction Manager Server Name and Numbers per Group

The name of the transaction manager server, TMS, must be specified in the entry for any group with servers that will participate in distributed transactions (transactions across multiple resource managers—and possibly machines). To specify a TMS, set the `TMSNAME` parameter. This parameter specifies the file (*string_value*) to be executed by `tmbboot(1)` when booting the server group.

The value `TMS` is reserved to indicate use of the null XA interface. This interface can be used for server groups that do not have resource managers. If you do not have a resource manager, you may not need a TMS. This server group may be infected with transactional messages. If a non-empty value other than `TMS` is specified, then a `TLOGDEVICE` must be specified for the machine(s) associated with the `LMID` value(s) for this entry. A unique server identifier is selected automatically for each TM server. Servers are restartable an unlimited number of times.

If `TMSNAME` is specified, `TMSCOUNT=number` must also be specified to indicate the number of transaction manager servers to start for the associated group. The default for `TMSCOUNT` is 3. If specified and the value is non-zero, the minimum value is 2 and the maximum value is 256. The servers are set up in an `MSSQ` set automatically.

Identifying the Environment File Location for Servers in a Group

If the value of the `ENVFILE` environment variable (`ENVFILE=string_value`) is an invalid filename, no values are added to the environment. Lines must be of the form `ident=value` where `ident` contains only underscores or alphanumeric characters.

Within `value`, strings of the form `${env}` are expanded when the file is processed using variables already defined for the environment. (Forward referencing is not supported. If a value is not set, the variable is replaced with an empty string.) You can use a back slash (`\`) to escape dollar signs and other back slashes. All other shell quoting and escape mechanisms are ignored and the expanded value is placed in the environment.

Environment files are provided in at least two sections of the configuration file. The BEA Tuxedo system reads them in the following order.

1. `MACHINES` section `ENVFILE`
2. `GROUPS` section `ENVFILE`
3. `SERVERS` section `ENVFILE` (optional)

Values in the `SERVERS` section override values in the `GROUPS` section. Values in the `GROUPS` section override values in the `MACHINES` section.

Defining Information Needed When Opening and Closing the Resource Manager

The values of both the `OPENINFO` and `CLOSEINFO` parameters must be alphanumeric strings that contain a maximum of 256 characters, and are enclosed in double quotation marks. These settings specify the resource manager dependent information needed when opening and closing the resource manager for this group (that is, for this group name).

This value is ignored if the `TMSNAME` parameter for this group is *not* set or is set to `TMS`. If the `TMSNAME` parameter is set to a value other than `TMS` but the `OPENINFO` string is set to the null string (" ") or is not specified, a resource manager exists for the group but does not require any information for executing an `open` operation. If the `TMSNAME` parameter is set to a value other than `TMS` but the `CLOSEINFO` string is set to the null string (" ") or is not specified, a resource manager exists for the group but does not require any information for executing a `close` operation.

The format of the `OPENINFO` string is dependent on the requirements of the vendor providing the underlying resource manager. The information required by the vendor must be prefixed with the published name of the vendor's transaction (XA) interface, followed immediately by a colon (:).

For BEA Tuxedo /Q databases, the format of `OPENINFO` is as follows.

- # On UNIX #
`OPENINFO = "TUXEDO/QM:qmconfig:qspace"`
- # On Windows NT #
`OPENINFO = "TUXEDO/QM:qmconfig;qspace"`
- # In AS/400 environment #
`OPENINFO = "TUXEDO/QM:qmconfig;qspace"`
- # In OpenVMS environment #
`OPENINFO = "TUXEDO/QM,[a.b.c]qmconfig,qspace"`

In all these settings, `TUXEDO/QM` is the published name of the BEA Tuxedo /Q XA interface, `qmconfig` is replaced with the name of the `QMCONFIG` (see `qmadmin(1)` in *BEA Tuxedo Command Reference*) on which the queue space resides, and `qspace` is replaced with the name of the queue space. For NT and AS/400, the separator after `qmconfig` must be a semicolon (;). For OpenVMS, the separator after `TUXEDO/QM` and after `qmconfig` must be a comma (,).

Note: The `CLOSEINFO` string is not used for BEA Tuxedo /Q databases.

For other vendors' databases, the format of the `OPENINFO` string is specific to the particular vendor providing the underlying resource manager. As an example, the following `OPENINFO` string demonstrates the type of information needed when opening the Oracle resource manager.

```
OPENINFO="Oracle_XA:  
Oracle_XA+Acc=P/Scott/*****+SesTm=30+LogDit=/tmp"
```

3 Creating the Configuration File

Oracle_XA is the published name of the Oracle XA interface. The series of five asterisks (*) in the OPENINFO string pertains to the encrypting of a password, which is described in the paragraphs that follow.

Passwords passed to a resource manager in the OPENINFO string can be stored in either clear text or encrypted form. To encrypt a password, first enter a series of five or more continuous asterisks in the OPENINFO string at the place where you want the password to go. Then load the UBBCONFIG file by running `tmloadcf(1)`. When `tmloadcf()` encounters the string of asterisks, it prompts you to create a password. For example:

```
tmloadcf -y /usr5/apps/bankapp/myubbcconfig
Password for OPENINFO (SRVGRP=BANKB3):
password
```

`tmloadcf()` stores the password in the TUXCONFIG file in encrypted form. If you then regenerate the UBBCONFIG file from the TUXCONFIG file using `tmunloadcf(1)`, the password is printed in the regenerated UBBCONFIG file in encrypted form with @@ as delimiters. For example:

```
OPENINFO="Oracle_XA:
Oracle_XA+Acc=P/Scott/@@A0986F7733D4@@+SesTm=30+LogDit=/tmp"
```

When `tmloadcf()` encounters an encrypted password in a UBBCONFIG file generated by `tmunloadcf()`, it does not prompt the user to create a password.

How to Create the NETWORK Section of the Configuration File

If you have more than one machine in your distributed application, you need to create a NETWORK section in your configuration file. This section sets up communications among your machines. You can configure network groups in both the NETGROUPS and NETWORK sections of an application's UBBCONFIG file.

For each parameter in the NETWORK section, the following table provides a description and links to reference pages and additional information.

To Specify This Information in the NETWORK Section . . .	Set This Parameter (Required/Optional)	For More Information, Click the Following
The device name to be used by the BRIDGE process placed on that LMID to access the network.	BRIDGE (optional)	BRIDGE device name
The complete network address to be used by the BRIDGE process; that is, the listening address on the LMID.	NADDR (required)	BRIDGE network address
The minimum level of encryption required when a network link to this machine is being established.	MINENCRYPTBITS (optional)	encryption levels
The maximum level of encryption allowed when a network link is being established.	MAXENCRYPTBITS (optional)	encryption levels
The network group associated with this network entry. If unspecified, then the default, DEFAULTNET, is assumed. (If not set to DEFAULTNET, this parameter must be defined as a group name in the NETGROUPS section.)	NETGROUP (optional)	network group
The network address used by the tlisten(1) process servicing the network on the node identified by the LMID.	NLSADDR (optional)	tlisten network address

Sample NETWORK Section

The following configuration file excerpt shows a NETWORK section for a 2-site configuration.

```
*NETWORK
  SITE1  NADDR= "//mach1:80952"
         NLSADDR= "//mach1:serve"
#   SITE2  NADDR= "//mach386:80952"
         NLSADDR= "//mach386:serve"
```

See Also

- UBBCONFIG(5) in *BEA Tuxedo File Formats and Data Descriptions Reference*
- “How to Create the NETGROUPS Section of the Configuration File” on page 3-56

Specifying a Device Name for the BRIDGE Process

To specify the device name to be used by the BRIDGE process placed on the LMID to access the network, set the BRIDGE parameter as follows.

```
BRIDGE=string_value
```

If you are using TCP/IP, you do not need to specify the device name for the BRIDGE.

The path name for the network transport endpoint file has the following form.

```
/dev/provider_name
```

Assigning a BRIDGE Network Address

To specify the complete network address to be used by the BRIDGE process placed on the LMID as its listening address, set the NADDR parameter as follows.

```
NADDR = string_value
```

The listening address for a BRIDGE is the location at which it is contacted by other BRIDGE processes participating in the application.

The listening address for a BRIDGE may also be specified in one of the following three forms:

- *//host.name:port_number*
- *//#. #. #. #:port_number*
- *0xhex-digits* or *\\xhex-digits*

In the first of these formats, *host.name* is resolved to the address of the TCP/IP host address at the time the address is bound. This format is based on locally configured name resolution facilities accessed via an operating system command. The value of *port_number* can be a symbolic name or a decimal number.

In the second format, the string *#. #. #. #* represents four decimal numbers (each of which is between 0 and 255), separated by periods. The value of *port_number* is a decimal number in the range 0 to 65535 (the hexadecimal representations of the string specified). The value of *port_number* can be a symbolic name or a decimal number.

In the third format, the string *0xhex-digits* or *\\xhex-digits* must contain an even number of valid hex digits. A string in either of these forms is translated internally into a character array containing TCP/IP addresses.

Note: On some platforms lower numbers may be reserved for the system.

Assigning Encryption Levels

To set up the minimum level of encryption required when establishing a network link to the machine, set the `MINENCRYPTBITS` parameter. Valid values are 0, 56, and 128. 0 means no encryption, while 56, and 128 specify the encryption key length (in bits). If this minimum level of encryption cannot be met, link establishment fails. The default is 0.

To set up a maximum level of encryption when establishing a network link, set the `MAXENCRYPTBITS` parameter. Valid values are 0, 56, and 128. 0 means no encryption, while 56, and 128 specify the encryption key length (in bits). The default is 128.

Example

```
MAXENCRYPTBITS=128
MINENCRYPTBITS=0
```

See Also

- “Link-Level Encryption” on page 1-23 in *Using BEA Tuxedo Security*

Assigning a `tlisten` Network Address

To specify the network address used by the `tlisten(1)` process servicing the network on the machine identified by the `LMID`, set the `NLSADDR` parameter as follows.

```
NLSADDR=string_value
```

The value of *string* is a network address in the same format as that specified for the `NADDR` parameter.

The `tlisten` address for `NLSADDR` may be specified in one of the following three forms:

- `//host.name:port_number`
- `//#. #. #. #:port_number`
- `0xhex-digits` or `\\xhex-digits`

In the first of these formats, `host.name` is resolved to the address of the TCP/IP host address at the time the address is bound. This format is based on locally configured name resolution facilities accessed via an operating system command. The value of `port_number` can be a symbolic name or a decimal number.

In the second format, the string `#. #. #. #` represents four decimal numbers (each of which is between 0 and 255), separated by periods. The value of `port_number` is a decimal number in the range 0 to 65535 (the hexadecimal representations of the string specified). The value of `port_number` can be a symbolic name or a decimal number.

In the third format, the string `0xhex-digits` or `\\xhex-digits` must contain an even number of valid hex digits. A string in either of these forms is translated internally into a character array containing TCP/IP addresses.

`tmloadcf(1)` prints an error if `NLSADDR` is missing from an entry for any machine besides the `MASTER LMID`, for which it prints a warning. If `NLSADDR` is missing from the `MASTER LMID`, `tmadmin(1)` cannot run in administrator mode on remote machines; it is limited to read-only operations. In addition, the backup site cannot reboot the `MASTER` site after failure.

How to Create the NETGROUPS Section of the Configuration File

The `NETGROUPS` section of the `UBBCONFIG` file describes the network groups available to an application in a LAN environment. There is no limit to the number of network groups to which you can assign a pair of machines. The method of communication to be used by members of different networks in a network group is determined by the priority mechanism (`NETPRIO`).

Every `LMID` must be a member of the default network group (`DEFAULTNET`). The network group number for this group (that is, the value of `NETGRPNO`) must be zero. However, you can modify the default priority of `DEFAULTNET`. Networks defined in the BEA Tuxedo system prior to Release 6.4 are assigned to the `DEFAULTNET` network group.

For each parameter in the `NETGROUPS` section, the following table provides a description and links to reference pages and additional information.

To Specify This Information in the NETGROUPS Section (Optional)	Set This Parameter (Required/Optional)	For More Information, Click the Following
Allow more netgroups to be defined than the default (8). This value is specified in the <code>RESOURCES</code> section.	<code>MAXNETGROUPS</code> (optional)	maximum netgroups
The maximum size of data waiting for the network to become available. This value is specified in the <code>MACHINES</code> section.	<code>MAXPENDINGBYTES</code> (optional)	message space limits
The network group associated with this network entry.	<code>NETGROUP</code> (required)	network group name
A unique network group number that you must assign to use in failover and failback situations.	<code>NETGRPNO</code> (required)	network group number
The priority of this network group.	<code>NETPRIO</code> (optional)	network group priority

Sample Network Groups Configuration

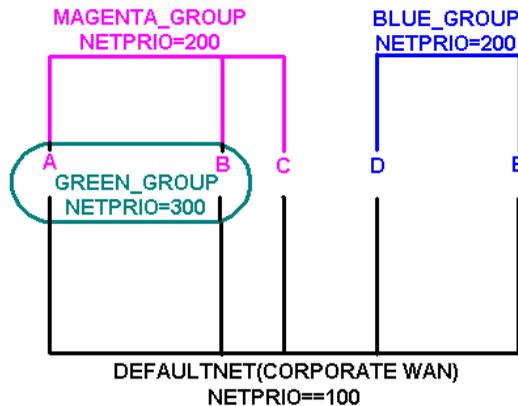
You can associate network addresses with a network group. The following example illustrates how this capability may be useful.

First State Bank has a network of five machines (A-E). Each machine belongs to two or three of four netgroups that you have defined in the following way:

- DEFAULTNET (the default network, which is the corporate WAN)
- MAGENTA_GROUP (a LAN)
- BLUE_GROUP (a LAN)
- GREEN_GROUP (a private LAN that provides high-speed, fiber, point-to-point links between member machines)

Every machine belongs to DEFAULTNET (the corporate WAN). In addition, each machine is associated with either the MAGENTA_GROUP or the BLUE_GROUP. Finally, some machines in the MAGENTA_GROUP LAN also belong to the private GREEN_GROUP. The following illustration shows machines A through E in the networks for which they have addresses.

Figure 3-3 Example of a Network Grouping



3 Creating the Configuration File

The following table shows which machines have addresses for which groups.

This Machine	Has Addresses for These Groups
A and B	DEFAULTNET (the corporate WAN) MAGENTA_GROUP (LAN) GREEN_GROUP (LAN)
C	DEFAULTNET (the corporate WAN) MAGENTA_GROUP (LAN)
D and E	DEFAULTNET (the corporate WAN) BLUE_GROUP (LAN)

Note: Because the local area networks are not routed among locations, machine D (in the BLUE_GROUP LAN) may contact machine A (in the GREEN_GROUP LAN) only by using the single address they have in common: the corporate WAN network address.

Configuring a Sample UBBCONFIG File with Netgroups

To set up the configuration just described, the First State Bank system administrator defines each group in the NETGROUPS section of the UBBCONFIG file, as shown in the following configuration file sample.

Listing 3-1 Sample NETGROUPS and NETWORK Sections

```
*NETGROUPS

DEFAULTNET    NETGRPNO = 0           NETPRIO = 100 #default
BLUE_GROUP    NETGRPNO = 9           NETPRIO = 200
MAGENTA_GROUP NETGRPNO = 125       NETPRIO = 200
GREEN_GROUP   NETGRPNO = 13        NETPRIO = 300

*NETWORK

A    NETGROUP=DEFAULTNET    NADDR="//A_CORPORATE:5723"
A    NETGROUP=MAGENTA_GROUP NADDR="//A_MAGENTA:5724"
A    NETGROUP=GREEN_GROUP   NADDR="//A_GREEN:5725"
```

```
B      NETGROUP=DEFAULTNET      NADDR="//B_CORPORATE:5723"
B      NETGROUP=MAGENTA_GROUP   NADDR="//B_MAGENTA:5724"
B      NETGROUP=GREEN_GROUP     NADDR="//B_GREEN:5725"

C      NETGROUP=DEFAULTNET      NADDR="//C_CORPORATE:5723"
C      NETGROUP=MAGENTA_GROUP   NADDR="//C_MAGENTA:5724"

D      NETGROUP=DEFAULTNET      NADDR="//D_CORPORATE:5723"
D      NETGROUP=BLUE_GROUP      NADDR="//D_BLUE:5726"
E      NETGROUP=DEFAULTNET      NADDR="//E_CORPORATE:5723"
E      NETGROUP=BLUE_GROUP      NADDR="//E_BLUE:5726"
```

See Also

- `UBBCONFIG(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*
- “How to Create the SERVERS Section of the Configuration File” on page 3-60
- “Setting Up the Network for a Distributed Application” on page 8-1

Assigning a Name to a Network Group

To assign a name to a network group, set the `NETGROUP` parameter as follows.

```
NETGROUP required_parameters [optional_parameters]
```

If you set `NETGROUP` to `DEFAULTNET`, then the entry describes the default network group. All network entries with a `NETGROUP` parameter of `DEFAULTNET` are represented in the `T_MACHINE` class of the `TM_MIB`, while `NETWORK` entries associated with any other `NETGROUP` are represented in the `T_NETMAP` class of the `TM_MIB`, so they can interoperate with previous releases.

Assigning a Network Group Number

To accommodate circumstances in which you may need to use failover and failback, you must set the `NETGRPNO` parameter as follows.

```
NETGRPNO=numeric_value
```

If this entry describes `DEFAULTNET`, the value of `NETGRPNO` must be zero.

Assigning a Priority to the Network Group

A pair of machines in multiple network groups of the same priority can communicate simultaneously over the circuits with the highest priority. To assign network group priorities, use the `NETPRIO` parameter. If all network circuits of a certain priority are torn down by an administrator or by network conditions, the next lower priority circuit is used. Retries of the higher priority circuits are attempted. The value of the `NETPRIO` parameter must be a number greater than zero and less than 8,192. The default is 100.

How to Create the `SERVERS` Section of the Configuration File

The `SERVERS` section of the configuration file contains information specific to a server process. While this section is not required, an application without this section has no application servers and little functionality. Each entry in this section represents a server process to be booted in the application and includes the following information:

- The name, group, and numeric identifier for a server (`SRVGRP`, `SRVID`)
- Server command-line options defined by `servopts` (`CLOPT`)

How to Create the SERVERS Section of the Configuration File

- Parameters to determine the booting order and number of servers to boot (SEQUENCE, MIN, MAX)
- A server-specific environment file (ENVFILE)
- Server queue-related information (RQADDR, RQPERM, REPLYQ, RPPERM)
- Restart information (RESTART, RCMD, MAXGEN, GRACE)
- Designation as a conversational server (CONV)
- Overriding of system-wide shared memory access (SYSTEM_ACCESS)

Note: Command-line options supported by the BEA Tuxedo system are described on `servopts(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*.

For each parameter in the `SERVERS` section, the following table provides a description and links to reference pages and additional information.

To Specify This Information in the <code>SERVERS</code> Section (Optional)	Set This Parameter (Required/Optional)	For More Information, Click the Following
Whether the server is a conversational server. Connections can be made only to conversational servers, and <code>rpc</code> requests (via <code>tpacall(3c)</code> or <code>tpcall(3c)</code>) can be made only to non-conversational servers.	<code>CONV</code> (optional run-time parameter)	conversational server
Principal name of the process used for identification, location of the principal user's private key, and the environment variable containing the password	<code>SEC_PRINCIPAL_NAME</code> , <code>SEC_PRINCIPAL_LOCATION</code> , <code>SEC_PRINCIPAL_PASSVAR</code>	security attributes
When this server should be booted or shut down relative to other servers.	<code>SEQUENCE</code> (optional boot parameter)	server boot order
The minimum number of occurrences of the server to be booted by <code>tmboot</code> .	<code>MIN</code> (optional boot parameter)	server boot order
The maximum number of occurrences of the server that can be booted.	<code>MAX</code> (optional boot parameter)	server boot order

3 Creating the Configuration File

To Specify This Information in the SERVERS Section (Optional)	Set This Parameter (Required/Optional)	For More Information, Click the Following
A list of <code>servopts(5)</code> options to be passed to a server process at boot time. If none are specified, the default is <code>-A</code> . <i>string_value</i> may contain up to 256 characters.	<code>CLOPT</code> (optional boot parameter)	server command-line options
A request for the addition of the values in this file to the environment of the server during its initialization. If a server is associated with a server group that can be migrated to a second machine, the <code>ENVFILE</code> must be in the same location on both machines.	<code>ENVFILE</code> (optional run-time parameter)	server environment file
The name of the group in which the server is to run. <i>string_value</i> must be the logical name associated with a server group in the <code>GROUPS</code> section.	<code>SRVGRP</code> (required)	server group
An integer that uniquely identifies a server within a group. Identifiers must be between 1 and 30,000 inclusive.	<code>SRVID</code> (required)	server ID
The symbolic name of the request queue for the process.	<code>RQADDR</code> (optional run-time parameter)	server queue information
The numeric permissions on the request queue.	<code>RQPERM</code> (optional run-time parameter)	server queue information
Whether a reply queue should be established for the process.	<code>REPLYQ</code> (optional run-time parameter)	server queue information
The numeric permissions on the reply queue.	<code>RPPERM</code> (optional run-time parameter)	server queue information
The command that should be executed when the process abnormally terminates, if the process is restartable.	<code>RCMD</code> (optional run-time parameter)	server restart information
The maximum number minus one time that the process can be restarted within the period specified by <code>GRACE</code> , if the process is restartable.	<code>MAXGEN</code> (optional run-time parameter)	server restart information

How to Create the SERVERS Section of the Configuration File

To Specify This Information in the SERVERS Section (Optional)	Set This Parameter (Required/Optional)	For More Information, Click the Following
A parameter that specifies that the process can have up to MAXGEN lives within the specified number of seconds, if the process is restartable.	GRACE (optional run-time parameter)	server restart information
Whether the process is restartable. Default is N. If server migration is specified, RESTART must be set to Y. (A server terminated with a SIGTERM signal must be rebooted.)	RESTART (optional run-time parameter)	server restart information
The default mode used by BEA Tuxedo system libraries within application processes to gain access to BEA Tuxedo system internal tables.	SYSTEM_ACCESS (optional run-time parameter)	system access to servers
The minimum number of server dispatch threads started on initial server boot. The separate dispatched thread that is used when MAXDISPATCHTHREADS>1 is not counted as part of the MAXDISPATCHTHREADS value. It is required that MINDISPATCHTHREADS<=MAXDISPATCHTHREADS. The default for this parameter is 0.	MINDISPATCHTHREADS	threads
The maximum number of concurrently dispatched threads that each server process may spawn. If MAXDISPATCHTHREADS>1, then a separate dispatcher thread is used and does not count against this limit. It is required that MINDISPATCHTHREADS<=MAXDISPATCHTHREADS. The default for this parameter is 1.	MAXDISPATCHTHREADS	threads
The stack size in bytes for each server thread after the initial thread. If not specified or specified as 0, the operating system default is used. This option has an affect on the server only when a value greater than 1 is specified for MAXDISPATCHTHREADS.	THREADSTACKSIZE	threads

Sample SERVERS Section

Following is a sample `SERVERS` section of a configuration file.

```
*SERVERS
DEFAULT:      RESTART=Y MAXGEN=5 GRACE=3600
               REPLYQ=N CLOPT="-A"
               ENVFILE="/usr/home/envfile"
               SYSTEM_ACCESS=PROTECTED

RINGUP1      SRVGRP=GROUP1 SRVID=1 MIN=3
               RQADDR="ring1"

RINGUP2      SRVGRP=GROUP1 SRVID=4 MIN =3
               RQADDR="ring2"
```

Note: Omitted from this sample are `SEQUENCE` (the order of booting is 1 to 6), `REPLYQ` and `RPPERM` (the server does not receive replies), `RCMD` (no special commands are desired on restart), and `CONV` (servers are not conversational). Defaults are applied to all servers unless a different setting is specified for a specific server.

Sample SERVERS Section Parameters

In the preceding sample `SERVERS` section, the following parameters and values are specified.

Parameter	Meaning
<code>RESTART=Y</code> (default)	Restart the servers.
<code>MAXGEN=5</code> (default)	The <code>MAXGEN</code> parameter specifies a number greater than 0 and less than 256 that controls the number of times a server can be started within the period specified by the <code>GRACE</code> parameter. The default is 1. If the server is to be restartable, <code>MAXGEN</code> must be ≥ 2 . The number of restarts is at most <i>number</i> - 1 times. <code>RESTART</code> must be Y or <code>MAXGEN</code> is ignored.

How to Create the SERVERS Section of the Configuration File

Parameter	Meaning
GRACE=3600 (default)	If RESTART is Y, the GRACE parameter specifies the time period (in seconds) during which this server can be restarted as MAXGEN - 1 times. The number assigned must be equal to or greater than 0. The maximum is 2,147,483,648 seconds (or a little more than 68 years). If GRACE is not specified, the default is 86,400 seconds (24 hours). As soon as one GRACE period is over, the next grace period begins. Setting the grace period to 0 removes all limitations; the server can be restarted an unlimited number of times.
REPLYQ=N (default)	There is no reply queue.
CLOPT="-A" (default)	Specify -A on the command line of each server.
ENVFILE="/usr/home/envfile" (default)	Read environment settings from the file ENVFILE.
SYSTEM_ACCESS=PROTECTED (default)	Deny access to internal tables outside system code.
RINGUP1	Sample name of the first server to be booted.
SRVGRP=GROUP1 SRVID=1 MIN=3 RQADDR="ring1"	Three instances of the sample server will be booted in group GROUP1 with server IDs of 1, 2, and 3, respectively. The three servers will form an MSSQ set and will read requests from queue <i>ring1</i> . Note: RQADDR assigns a symbolic name to the request queue of this server. MSSQ sets are established by using the same symbolic queue name for more than one server, as well as same executable name for all the servers (and by specifying a value greater than 1 for MIN).
RINGUP2	Name of the second sample server to be booted.

See Also

- UBBCONFIG(5) in *BEA Tuxedo File Formats and Data Descriptions Reference*
- “How to Create the SERVICES Section of the Configuration File” on page 3-75

Specifying a Server as Conversational

If a server is conversational (that is, if it establishes a 2-way connection between a client and a dedicated server), the `CONV` parameter is required and must be set to `Y`. The default is `N`, indicating that the server will not be part of a conversation.

Characteristics of the CONV Parameter

The `CONV` parameter has the following characteristics:

- A `Y` value indicates a server is conversational; an `N` value indicates a server is not conversational.
- A `Y` value is required if the server is to receive conversational requests.
- The default is `N`.

Setting the Order in Which Servers Are Booted

To specify the sequence of servers to be booted, set the `SEQUENCE` parameter for each server. The value of `SEQUENCE` can be any number between 1 and 10,000. A server with a smaller `SEQUENCE` value is booted before a server with a larger value. If the `SEQUENCE` parameter is not set for any servers, the servers are booted in the order in which they are listed in the `SERVERS` section. If some, but not all servers are sequenced, the sequenced servers are booted first. The order in which servers are shut down is the reverse of the order in which they were booted.

The `SEQUENCE` parameter is optional. It may be helpful in a large application in which control over boot order is important.

To boot multiple servers, set the `MIN` parameter, which provides a shortcut to booting. All servers share the same options. If you specify `RQADDR`, the servers form an `MSSQ` set. The default for `MIN` is 1.

To specify the maximum number of servers that can be booted, set the `MAX` parameter. The `tmboot(1)` command boots `MIN` servers at run time. Additional servers can be booted up to `MAX`. The default is `MIN`.

The `MIN` and `MAX` parameters are helpful in keeping the size of the configuration files for large applications manageable. Allowances for `MAX` values must be made in the `IPC` resources. The `MIN` and `MAX` parameters are also used for conversational services and automatic server spawning.

Characteristics of the SEQUENCE, MIN, and MAX Parameters

Parameter	Characteristics
SEQUENCE	<p>It is an optional parameter with a numeric range of 1 - 10,000.</p> <p>Smaller values are booted before larger values.</p> <p>Servers for which this parameter is not set are booted in the order in which they are listed in the <code>SERVERS</code> section.</p> <p>All sequenced servers are booted before any unsequenced servers.</p>
MIN	<p>It represents the minimum number of servers to boot during run time.</p> <p>If <code>RQADDR</code> is specified and <code>MIN</code>>1, an <code>MSSQ</code> set is created.</p> <p>All instances have the same server options.</p> <p>The range of values is 0 to 1000 .</p> <p>The default is 1.</p>
MAX	<p>It represents the maximum number of servers to boot.</p> <p>The range of values for <code>MAX</code> is 0 to 1000. If <code>MAX</code> is not specified, the default is the value of <code>MIN</code>.</p>

Specifying Server Command-line Options

The BEA Tuxedo system allows you to specify options that are used when a server processes a request. These options are defined in `servopts`, which lists the run-time options for server processes. The server may need to obtain information from the command line. The `CLOPT` parameter allows you to specify command-line options that can change some defaults in the server, or pass user-defined options to the `tpsvrinit()` function.

The standard `main()` of a server parses one set of options ending with the argument `--`, and passes the remaining options to `tpsvrinit()`. The default for `CLOPT` is `-A`, which tells the server to advertise all the services built into it with `buildserver(1)`. The following table provides a partial list of the available options.

Use This Option	To
<code>-o filename</code>	Redirect standard output to file <i>filename</i> .
<code>-e filename</code>	Redirect standard error to file <i>filename</i> .
<code>-s services</code>	Advertise services. For example, <code>-s x,y,z</code> to advertise services <i>x</i> , <i>y</i> , and <i>z</i> .
<code>-s x,y,z:funcname</code>	Advertise services <i>x</i> , <i>y</i> , and <i>z</i> , but process requests for those services with function <i>funcname</i> . This is called <i>aliasing</i> a function name.
<code>-r</code>	Specify that the server should log the services performed.
<code>-v</code>	Print out the list of the service name/function name to standard output. This option cannot be used in the <code>CLOPT</code> in the <code>UBBCONFIG</code> . It must be used when manually invoking the server.

Note: You can find other standard `main()` options listed on `servopts(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*.

Characteristics of the CLOPT Parameter

- The syntax is `CLOPT="servopts -- application_opts"`.
- This is an optional parameter with a default of `-A`.
- Both `main()` and `tpsvrinit()` use server command-line options.
- `servopts(5)` options are passed to `main()`.
- Application options are passed to `tpsvrinit()`.

In the `BANKAPP` sample application, command-line options are specified as follows.

```
CLOPT="-A -- -T 10"
```

The server is given the option of advertising all services (`-A`) and teller ID of 10 so it can update a specific teller record with each operation. The use of this option, especially the options passed to `tpsvrinit()`, require communication between the system administrator and the application programmer.

See Also

- `servopts(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*

Identifying the Location of the Server Environment File

Use the `ENVFILE` parameter in the `MACHINES` section to specify environment settings. You can also specify the same parameter for a specific server process; the semantics are the same. If both the `MACHINES` section `ENVFILE` and the `SERVERS` section `ENVFILE` are specified, both go into effect. For any overlapping variable defined in both the `MACHINES` and `SERVERS` sections, the setting in the `SERVERS` section prevails.

Characteristics of the Server Environment File

`ENVFILE`, the parameter that defines the server environment file, has the following characteristics:

- It is an optional parameter that contains the same semantics as the `ENVFILE` parameter in the `MACHINES` section, but defines only one server.
- For overlapping variables, the setting in the `SERVERS` section `ENVFILE` overrides the setting in the `MACHINES` and `GROUPS` sections `ENVFILE`.

Defining Server Name, Group, and ID

You initially assign a name to a server in the `SERVERS` section. The name you specify must be the name of an executable file built with `buildserver(1)`. You must also specify a group identifier (`SRVGRP`) for each server. The value of `SRVGRP` must be the name specified in the beginning of a `GROUPS` section entry. Finally, you must also provide each server process in a given group with a unique numeric identifier (`SRVID`). Every server entry must include the `SRVGRP` and `SRVID` parameters. Because the entries describe machines to be booted and not just applications, it is possible that in some cases the same server name will be displayed in many entries.

Characteristics of the Server Name, `SRVGRP`, and `SRVID` Parameters

Parameter	Characteristics
<code>Server_name</code>	It identifies the executable to be booted. It is built with <code>buildserver(1)</code> . It is required, but may not be unique within a server group.

Parameter	Characteristics
SRVGRP (Server Group)	It identifies the group affiliation. The group name begins with a GROUPS section entry. It is required.
SRVID (Server ID)	It is numeric. It is required and unique within a server group.

Identifying Server Queue Information

Server queue information controls the creation and access of server message queues. On a BEA Tuxedo system, you can create multiple-server, single-queue (MSSQ) sets by using the RQADDR parameter. For any given server, you can set this parameter to an alphanumeric value. By specifying the same value for RQADDR on all servers that offer the same services, you can consolidate those services under one message queue, thus creating an MSSQ set and establishing load balancing.

MSSQ Example

An MSSQ set is similar to a bank staff. Four tellers may be available to handle the business requests of many customers who wait in a single line. All customers are assured of an equitable wait in line. Understandably, a loan officer is not included in the group of tellers handling requests from customers in that line. The loan officer cannot handle requests for deposits and withdrawals (as the tellers can), and not all customers want loans. Similarly, a server cannot join an MSSQ set if the services it offers are not the same as the services offered by the servers in an MSSQ set.

The RQPERM parameter allows you to specify the permissions for server request queues, along the lines of the UNIX system convention (for example, 0666). This setting allows services to control access to the request queue.

If the service routines within an MSSQ server perform service requests, they must receive replies to their requests on a reply queue. You can set up such a reply queue by specifying `REPLYQ=Y`. By default, `REPLYQ` is set to `N`. If `REPLYQ` is set to `Y`, you can also assign permissions to it with the `RPPERM` parameter.

Characteristics of the **RQADDR**, **RQPERM**, **REPLYQ**, and **RPPERM** Parameters

Parameter	Characteristics
<code>RQADDR</code>	It is an alphanumeric value that allows MSSQ sets to be created. The value is the same for all members of an MSSQ set. All members of an MSSQ set must offer the same set of services and the servers in an MSSQ set should have the same executable name. In order to boot multiple servers, set the value greater than 1 for <code>Min</code> parameter.
<code>RQPERM</code>	Represents the permissions on a request queue. If no parameter is specified, the permissions of the bulletin board, as specified by <code>PERM</code> in the <code>RESOURCES</code> section, are used. If no value is specified there, the default of 0666 is used. When the default is used, your application is available to anyone with a login on the system.
<code>REPLYQ</code>	Specifies whether a reply queue, separate from the request queue, is to be set up for this server. If only one server is using the request queue, replies can be picked up from the request queue without causing problems. On a BEA Tuxedo system, if the server is a member of an MSSQ set and contains services programmed to receive reply messages, <code>REPLYQ</code> should be set to <code>Y</code> so that an individual reply queue is created for this server. If not, the reply is sent to the request queue shared by all servers of the MSSQ set, and there is no way of assuring that it will be picked up by the server that is waiting for it. Multithreaded servers automatically create <code>REPLYQs</code> even if this parameter is not set.
<code>RPPERM</code>	Assigns permissions to the reply queue. This parameter is useful only when <code>REPLYQ=Y</code> . If requests and replies are read from the same queue, only <code>RQPERM</code> is needed; <code>RPPERM</code> is ignored.

Defining Server Restart Information

A properly debugged server should not terminate on its own. By default, servers that do terminate while the application is running are not restarted by the BEA Tuxedo system. You can set the `RESTART` parameter to `Y` if you want the server to restart. The `RCMD`, `MAXGEN`, and `GRACE` parameters are relevant to a server if `RESTART=Y`.

The `RCMD` parameter lets you specify a command to be performed in parallel with restarting a server. For example, you may want to have mail sent to the developer of the server or to someone who is auditing such activity.

The `MAXGEN` parameter represents the total number of *lives* to which a server is entitled within the period specified by `GRACE`. The server can then be restarted `MAXGEN-1` times during `GRACE` seconds. If `GRACE` is set to zero, there is no limit on server restarts. `MAXGEN` defaults to 1 and may not exceed 256. `GRACE` must be greater than or equal to zero and must not exceed 2,147,483,647 ($2^{31} - 1$).

Note: A fully debugged server should not need to be restarted. `RESTART` and associated parameters should have two settings: one for the testing phase, and another for production.

Characteristics of the `RESTART`, `RCMD`, `MAXGEN`, and `GRACE` Parameters

Parameter	Characteristics
<code>RESTART</code>	A setting of <code>Y</code> enables a server to restart. The default is <code>N</code> .
<code>RCMD</code>	Specifies an executable file to be run at restart time. Allows you to take an action when a server is restarted.
<code>MAXGEN</code>	Represents the maximum number of server lives in a specific interval. The default is 1; the maximum is 256.

Parameter	Characteristics
GRACE	Represents the interval used by MAXGEN. Zero represents unlimited restart. It must be between 0 and 2147,483,647 ($2^{31} - 1$). The default is 24 hours.

Defining Server Access to Shared Memory

The `SYSTEM_ACCESS` parameter determines whether a server process may attach to shared memory and thus have access to internal tables outside system code. During application development, we recommend that such access be denied (`PROTECTED`). When the application is fully tested, you can change the value of `SYSTEM_ACCESS` to `FASTPATH` to yield better performance.

This parameter setting overrides the value specified in the `RESOURCES` section unless the `NO_OVERRIDE` value has been specified. In this case, the parameter is ignored. The `NO_OVERRIDE` value may not be used in this section.

Characteristics of the `SYSTEM_ACCESS` Parameter

The `SYSTEM_ACCESS` parameter has the following characteristics:

- A value of `PROTECTED` indicates that the server may not attach to shared memory outside of system code.
- A value of `FASTPATH` indicates that the server will attach to shared memory at all times.
- If `NO_OVERRIDE` is specified in the `RESOURCES` section, this parameter is ignored.
- The default is the value of the `SYSTEM_ACCESS` parameter in the `RESOURCES` section.
- The BEA Tuxedo system runs more slowly when a value of `PROTECTED` is set.

Defining the Server Dispatch Threads

`MAXDISPATCHTHREADS` is the maximum number of concurrently dispatched threads that each server process may spawn. If `MAXDISPATCHTHREADS > 1`, then a separate dispatcher thread is used and does not count against this limit. It is required that `MINDISPATCHTHREADS <= MAXDISPATCHTHREADS`. If not specified, the default for this parameter is 1.

`MINDISPATCHTHREADS` is the minimum number of server dispatch threads started on initial server boot. The separate dispatched thread that is used when `MAXDISPATCHTHREADS > 1` is not counted as part of the `MAXDISPATCHTHREADS` value. It is required that `MINDISPATCHTHREADS <= MAXDISPATCHTHREADS`. The default for this parameter is 0.

You must specify the stack size in bytes for each server thread after the initial thread. If not specified or specified as 0, the operating system default is used. This option has an affect on the server only when a value greater than 1 is specified for `MAXDISPATCHTHREADS`.

How to Create the SERVICES Section of the Configuration File

Detailed information about the services in your application can be entered in the `SERVICES` section of the configuration file. For nontransactional, nondistributed applications, such information is relatively simple. The `SERVICES` section includes the following types of information:

- Load balancing information (`SRVGRP`)
- Assignment of priorities to services
- Different service parameters for different server groups
- Buffer type checking information (`BUFTYPE`)

There are no required parameters for services. You need to list services only if you are setting optional parameters.

3 *Creating the Configuration File*

For each parameter in the `SERVICES` section, the following table provides a description and links to reference pages and additional information.

To Specify This Information in the SERVICES Section . . .	Set This Parameter (Required/Optional)	For More Information, Click the Following
Whether a transaction should be started automatically when a request message is received that is not already in transaction mode.	<code>AUTOTRAN</code> (for DTP applications only)	automatic starts for transactions
A list of types and subtypes of data buffers accepted by this service. This parameter may contain up to 256 characters with a maximum of 32 type/subtype combinations.	<code>BUFTYPE</code> (optional)	buffer types
A load factor to be imposed on the system by <code>SVCNAM</code> .	<code>LOAD</code> (optional)	load balancing
The name of the routing criteria used for this service when data- dependent routing is used.	<code>ROUTING</code> (optional)	routing criteria name
The name of the sever group from which <code>SVCNAM</code> gets all group parameter settings.	<code>SRVGRP</code> (optional)	server group parameters
The dequeuing priority of <code>SVCNM</code> .	<code>PRIO</code> (optional)	service priorities
The amount of time, in seconds, that is allowed for processing of the indicated service.	<code>SVCTIMEOUT</code> (optional)	service processing time
The default timeout interval, in seconds, for a transaction automatically started for the associated service.	<code>TRANTIME</code> (for DTP applications only)	timeout values for transactions

Sample SERVICES Section

Following is a sample of the SERVICES section of a configuration file.

```
*SERVICES
#
DEFAULT:  LOAD=50  PRIO=50
RINGUP   BUFTYPE="VIEW:ringup"
```

In this example, the default load and priority of a service are 50; the one service declared is a RINGUP service that accepts a RINGUP VIEW as its required buffer type.

See Also

- UBBCONFIG(5) in *BEA Tuxedo File Formats and Data Descriptions Reference*
- “How to Create the ROUTING Section of the Configuration File” on page 3-84

Specifying Automatic Starts and Timeout Intervals for Transactions

You can determine whether a transaction should be started automatically if a request message is already in transaction mode by coding the `AUTOTRAN = {Y|N}` parameter. The default is N.

You can specify a timeout interval between the time at which a transaction for a service begins and the time at which it is rolled back if not completed. To specify a timeout interval that will be used automatically, set the `TRANTIME` parameter as follows.

```
TRANTIME=number
```

The default is 30 seconds. A value of 0, the maximum timeout value for the computer, means a transaction will never time out.

Specifying a List of Allowable Buffer Types for a Service

With the `BUFTYPE` parameter, you can tune a service to check buffer types independently of the service code. Set this parameter with a list of allowable buffer types for a service in the following format.

```
type[ : subtype[ , subtype] ]
```

To allow all subtypes, set the value of *subtype* to `*`.

If the value of the `BUFTYPE` parameter for a service is `ALL`, this service accepts all buffer types. The default is `ALL`.

Examples of the `BUFTYPE` Parameter

BUFTYPE Example	Meaning
<code>BUFTYPE="FML;VIEW:aud,aud2"</code>	FML and VIEW buffer types with subtypes aud and aud2 are allowed.
<code>BUFTYPE="FML;VIEW:*"</code>	All FML and VIEW buffer types are allowed.
<code>BUFTYPE=ALL</code>	All buffer types are allowed (the default).

Designating How Much Time to Process a Request

Sometimes an unexpected system error occurs, freezing a service or causing it to run out of control while it is processing a request. Obviously, it is a good idea to remove these processes, but it is difficult to detect them or determine how they developed errors. The BEA Tuxedo system provides a mechanism for terminating such processes even when you cannot identify them. To use this mechanism, set the `SVCTIMEOUT` parameter.

The `SVCTIMEOUT` parameter allows you to designate an amount of time (in seconds) in which a service should be able to process a request. If the interval defined by this parameter elapses and a service has not finished processing a request, the process for that request is killed. In essence, the service timeout mechanism acts like a scavenger for frozen or out of control application servers. By default, the BEA Tuxedo system does not terminate any service process; you must set the `SVCTIMEOUT` parameter to activate this feature.

You can assign a value to the `SVCTIMEOUT` parameter in the `UBBCONFIG` file or by dynamically changing the `TA_SVCTIMEOUT` attribute in `TM_MIB`. We recommend that you set the value of `SVCTIMEOUT` or `TA_SVCTIMEOUT` to at least two to three times the number of seconds it takes for your longest running service to process a request. Setting the service timeout in this way guarantees that the BEA Tuxedo system removes only frozen processes.

This section describes the causes and results of service timeout errors, and explains how the BEA Tuxedo system reports such errors. Advice about how to handle errors is also provided.

What Happens When a Timeout Occurs

When a timeout occurs, the BEA Tuxedo system terminates the server process running the frozen service (but not its child processes, if any). It then returns a `TPESVCERR` error, indicating that an unknown problem occurred during processing. In a conversational service, the conversation event `TPEV_SVCERR` is returned.

How a Service Timeout Is Reported

The BEA Tuxedo system reports a service timeout through the following three mechanisms:

- `TPED_SVCTIMEOUT`—timeout error detail that provides more information than `tpstrerror(3c)`
- `.SysServiceTimeout`—a system event
- `ULOG` information about `.SysServiceTimeout`

Because the `SVCTIMEOUT` value is configurable, it is important for clients to be able to easily distinguish between a `TPESVCERR` caused by exceeding the value set for `SVCTIMEOUT`, and a `TPESVCERR` caused by other situations. Although the `ULOG` contains this information, it is difficult for client programs to extract it. To differentiate a service timeout `TPESVCERR` from others, a program can include a call to the `tperrordetail(3c)` routine (after a `TPESVCERR` has been detected), which yields `TPED_SVCTIMEOUT` when a service timeout occurs.

In addition, a system event, `.SysServiceTimeout`, is generated when a service timeout occurs. When a `.SysServiceTimeout` event occurs, it is reflected in the `ULOG` in the following way.

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

How to Control a Service Timeout

- Application administrators may control the service timeout by changing the `SVCTIMEOUT` parameter in the `SERVICES` section of the `UBBCONFIG` file, or by modifying the `TA_SVCTIMEOUT` attribute of the `T_SERVER` or `T_SERVICE` class of the `TM_MIB`. They may also monitor the `ULOG` file for service timeout activity.
- In addition to monitoring the `ULOG` file for service timeout activity, application operators can subscribe to the `.SysServiceTimeout` event, which alerts them when a service timeout occurs.
- Application programmers can use the `tperrordetail(3c)` and `tpstrerrordetail(3c)` functions, and the `TPED_SVCTIMEOUT` error detail code. They may want to add one or more subscriptions to the `.SysServiceTimeout` system event, which is generated when a service timeout occurs.

Enabling Load Balancing

To activate load balancing, set the `RESOURCES` section parameter `LDBAL` to `Y`. A load factor is assigned to each service performed (via the `LOAD` parameter) and the BEA Tuxedo system keeps track of the total load of services that each server has performed. Each service request is routed to the server with the smallest total load. The routing of that request causes the server's total to be increased by the `LOAD` factor of the service requested.

Load information is stored only on the site originating the service request. It would be inefficient for the BEA Tuxedo system to make continuous attempts to propagate load information to all sites in a distributed application. When performing load balancing in such an environment, each site knows only about the load it originated and performs load balancing accordingly. This means that each site has different load statistics for a given server (or queue). The server perceived as being the least busy differs from site to site.

When load balancing is not activated, and multiple servers offer the same service, the first available queue receives the request.

Characteristics of the `LDBAL` Parameter

The `LDBAL` parameter has the following characteristics:

- Load balancing is used if the `RESOURCES LDBAL` parameter is set to `Y`.
- The load factor is added to a server's total load.
- The load is relative to other services.

Defining the Name of the Routing Criteria

When using data-dependent routing, you need to specify the routing criteria to be used for a service. To specify such criteria, set the `ROUTING` parameter as follows.

```
ROUTING=string_value
```

If this parameter is not set, the service does not perform data-dependent routing.

The maximum value of *string* is 15 characters. No more than one value may be assigned to the `ROUTING` parameter for a given service. Even if you have multiple entries for one service and those entries contain different `SRVGRP` parameters, the value of `ROUTING` must be the same in all entries.

Specifying Service Parameters for Different Server Groups

You can assign the same service to multiple groups and assign different values to the various service-specific parameters you set for the service entries for the different groups. To do this, create a separate entry for the service for each group, specifying a group-specific value for the `SRVGRP` parameter.

Controlling the Flow of Data by Service Priority

You can exert significant control over the flow of data in an application by assigning service priorities using the `PRIO` parameter. The value of `PRIO` must be a number between 0 and 100. The higher the number, the higher the priority of the service to which it is assigned. Higher priority services are dequeued before lower priority services, but the system dequeues every tenth request in FIFO order to prevent a message from waiting indefinitely on the queue.

For instance, Server 1 offers Services A, B, and C. Services A and B have a priority of 50 and Service C has a priority of 70. A service requested for C will always be dequeued before a request for A or B. Requests for A and B are dequeued equally with respect to one another.

Note: A priority can also be changed dynamically with the `tpsrio()` call.

Characteristics of the PRIO Parameter

The `PRIO` parameter has the following characteristics:

- It determines the priority of a service on the server's queue.
- The highest assigned priority gets first preference.
- Every tenth request is dequeued FIFO.

Sample SERVICES Section Using Different Priorities

The following sample from the `SERVICES` section of a configuration file shows how priorities are assigned to services.

```
*SERVICES
A  SRVGRP=GRP1  PRIO=50  LOAD=60
A  SRVGRP=GRP2  PRIO=70  LOAD=30
```

In this example, different service-specific parameters are assigned to two server groups. Service A is assigned a priority of 50 and a load of 60 in server group `GRP1`, and a priority of 70 and a load of 30 in server group `GRP2`.

Indicating Service Processing Time

To indicate the maximum amount of time, in seconds, allowed for processing a service, set the `SVCTIMEOUT` parameter as follows.

```
SVCTIMEOUT=number
```

The value must be greater than or equal to 0. A value of 0 indicates that the service will be timed out: the server processing the server request will be terminated with a `SIGKILL` signal. The default for this parameter is 0.

How to Create the ROUTING Section of the Configuration File

The `ROUTING` section of `UBBCONFIG` allows you to provide a full definition of the routing criteria named in the `SERVICES` section (for BEA Tuxedo data-dependent routing).

For each parameter in the `ROUTING` section, the following table provides a description and links to reference pages and additional information.

To Specify This Information in the <code>ROUTING</code> Section (Optional)	Set This Parameter (Required/Optional)	For More Information, Click the Following
Ranges and associated server groups for the routing field.	<code>RANGES</code> (required)	range criteria
Routing criteria name specified as the value of the <code>ROUTING</code> parameter in the <code>SERVICES</code> section for data-dependent routing. The value must be a string with a maximum length of 15 characters.	<code>criterion_name</code> (required)	
Name of the routing field, which is assumed to be an FML buffer, XML element or element attribute, view field name identified in an FML field table (using <code>FLDTBLDIR</code> and <code>FIELDTBLS</code> environment variables), or an FML view table (using the <code>VIEWDIR</code> and <code>VIEWFILES</code> environment variables), respectively. This information is used to obtain the associated field value for data-dependent routing when sending a message.	<code>FIELD</code> (required)	routing buffer field and type
A list of types and subtypes of data buffers for which this routing entry is valid. This parameter may contain up to 256 characters with a maximum of 32 type/subtype combinations.	<code>BUFTYPE</code> (required)	buffer types and subtypes

ROUTING Section Example

```
BRNCH FIELD=B_FLD
RANGES="0-2:DBG1,3-5:DBG2,6-9:DBG3"
BUFTYPE="FML"
```

Defining the Routing Buffer Field and Field Type

The following table describes the routing buffer field and field type.

Parameter	Characteristics
FIELD	<p>The name of the buffer field on which the routing is performed. It may contain up to 30 characters.</p> <p>In BEA Tuxedo data-dependent routing, the value of this parameter is one of the following: the name of an FML field (for FML buffers); an XML element or attribute; a VIEW field name identified in an FML field table (using the FLDTBLDIR and FIELDTBLS environment variables); or an FML view table (using the VIEWDIR and VIEWFILES environment variables). This information is used to obtain the associated field value for data-dependent routing during message processing. If a field in an FML32 buffer is used for routing, it must have a field number less than or equal to 8191.</p> <p>In routing XML documents, the FIELD syntax contains either a <i>routing element type</i> (or name) or a <i>routing element attribute name</i>. You must define the FIELD parameter with the following syntax.</p> <pre>root_element[/child_element][/child_element][/. . .][/@attribute_name]</pre> <p>The element is assumed to be an <i>element type</i> (or name) or an <i>element attribute name</i> of an XML document or datagram. This information is used to obtain the associated element content or element attribute value for data-dependent routing when a document or datagram is being sent. Because indexing is not supported, the BEA Tuxedo system recognizes only the first occurrence of a given element type when processing an XML buffer for data-dependent routing.</p>

Parameter	Characteristics
-----------	-----------------

<code>FIELDTYPE</code>	This parameter is used only for routing XML buffers. It indicates the type of the routing field specified in <code>FIELD</code> . The syntax is as follows.
------------------------	---

`FIELDTYPE=type`

where *type* is one of the following: `string`, `char`, `short`, `long`, `float`, or `double`.

The default type of the routing field is `string`.

Specifying Range Criteria

The `RANGES` parameter allows you to map field values to a group name as follows.

```
RANGES=" [ val1[-val2]:group1] [ , val3[-val4]:group2] ... [ , *:groupn] "
```

where *val1*, *val2*, and so on, are values of a field and *groupn* may be either a group name or the wildcard character (*) denoting that any group may be selected. The * character occupying the place of *val* at the end is a *catch-all* choice, that is, it specifies if the data does not fall into any range that has been specified then it goes to the default group on the other hand if the data fall into the range but there is no viable server in the group associated with the range entry, then the service request is forwarded to the default group specified on the wildcard "*" range entry. The value of *val1* may be:

- A number (when it is used in a numeric field)
- A `STRING` or `CARRAY` buffer (enclosed in single quotation marks)
- `MIN` or `MAX`, to show a machine minimum or maximum data value

There is no limit to the number of ranges that may be specified, but routing information incurs a cost because it is stored in shared memory.

Note: Overlapping ranges are allowed, but values that belong to both ranges map to the first group. For example, if `RANGES` is specified as

```
RANGES="0-5:Group1 , 3-5:Group2",
```

 then a range value of 4 routes to Group1.

Defining Buffer Types

For BEA Tuxedo data-dependent routing, the `BUFTYPE` parameter determines the buffer type allowed. This parameter is similar to its `SERVICES` section counterpart in that it restricts the routing criteria to a specific set of buffer types and subtypes. Only `FML`, `XML` and `VIEW` types can be used for routing. The syntax is the same as the syntax in the `SERVICES` section, a semicolon-separated list of `type:subtype[, subtype]`. You can specify only one type for routing criteria. This restriction limits the number of buffer types allowed in routing services.

How to Configure the BEA Tuxedo System to Take Advantage of Threads

To configure a multicontexted application, edit your `UBBCONFIG` file as usual and add those parameters, listed in the following table, that are needed for your application. Use a text editor or the BEA Administration Console.

3 Creating the Configuration File

Table 3-1 Setting Parameters in the Configuration File to Use Threads

In This Section	Set These Parameters	With These Considerations.
RESOURCES	MAXACCESSERS	<i>Optional parameter</i> but you must assign a value to it you want more than 50 accessers (the default number). Each context of a multicontexted client is counted separately for licensing purposes.
	NOTIFY	<i>Optional parameter</i> that defines the default method to be used for unsolicited notification. Valid values for multicontexted applications are: <ul style="list-style-type: none">■ DIPIN■ THREAD■ IGNORE
MACHINES	MAXACCESSERS	<i>Optional parameter</i> but you must assign a value to it you want more than 50 accessers (the default number). Each context of a multicontexted client is counted separately for licensing purposes.
	MAXWSCLIENTS	<i>Optional parameter</i> Each context of a multicontexted Workstation client is counted separately for licensing purposes. Because the default is 0, this parameter must be set if any Workstation clients are to access the system via the machine being defined.

Table 3-1 Setting Parameters in the Configuration File to Use Threads

In This Section	Set These Parameters	With These Considerations.
SERVERS	MINDISPATCHTHREADS	<i>Optional parameter</i>
	MAXDISPATCHTHREADS	<i>Required parameter</i> in multithreaded servers. When making an existing server multithreaded, an experienced programmer must verify that the source code for the server has been written in a thread-safe manner. In other words, it is not possible to convert a single-threaded server, written with static variables, to a multithreaded server simply by increasing the value of MAXDISPATCHTHREADS in the configuration file. This server must also be built for multithreading.
	THREADSTACKSIZE	<i>Optional parameter</i> ; you may need to set it if your server dispatch threads require an especially large stack. The default, 0, should be sufficient for most applications. (Keep in mind that when 0 is passed to the operating system, the operating system invokes its own default.)

How to Compile a Configuration File

Compiling a configuration file means generating a binary version of the file (TUXCONFIG) from the text version (UBBCONFIG). To compile a configuration file, run the `tmloadcf` command. `tmloadcf` parses a UBBCONFIG file and loads the binary file.

`tmloadcf` reads a file (or standard input written in UBBCONFIG syntax), checks the syntax, and optionally loads a binary configuration file called TUXCONFIG. The TUXCONFIG and (optionally) TUXOFFSET environment variables point to the TUXCONFIG file and (optional) offset where the information should be stored. You can run `tmloadcf` only on the machine designated as MASTER in the RESOURCES section of the UBBCONFIG file, unless the `-c` or `-n` option is specified.

Notes: The user identifier (UID) of the person running `tmloadcf` must match the UID, if specified, in the RESOURCES section of the UBBCONFIG file.

The path name specified for the TUXCONFIG environment variable must match exactly (including case) the path name specified for TUXCONFIG parameter within the MACHINES section of the UBBCONFIG file. Otherwise, `tmloadcf(1)` cannot be run successfully.

4 About Transactions

- What Is a Transaction
- Benefits of Using Transactions
- Example of a Global Transaction
- What Is the BEA Tuxedo Transaction Manager (TM)
- How the System Tracks Distributed Transaction Processing
- How the System Uses a 2-Phase Commit to Commit Transactions

What Is a Transaction

A *transaction* is a set of related actions. A *global transaction* is a set of related actions that span multiple programs and resource managers. In this topic, whenever we use the term transaction, we are referring to a global transaction.

A simple example of a transaction is a withdrawal from a bank account, which can be described as a set of actions that changes the state of an account balance (by reducing it). For this transaction, the system must execute a procedure that consists of three operations.

Procedure for Any Transaction	Procedure for Bank Withdrawal Example
1. Verify the activity to be performed	1. Verify that a withdrawal will be made
2. Perform the work of the transaction	2. Withdraw a specified amount from the account
3. Create a permanent record of the completed work	3. Update the record of the balance of the account

These steps are performed by a discrete software module created expressly for the purpose of executing this transaction. The module must also include or use code that launches and ends the transaction. If the code sections that launch and end the transaction are not part of the main transaction software module, then they are usually packaged together in a separate module.

A *transaction coordinator* is a software module that executes the logic to manage a transaction among all participating resources.

What Are the ACID Properties

When a transaction such as a bank withdrawal is performed, it is imperative that all its constituent operations either succeed or fail together. Consider the problems that can occur if one operation in a transaction succeeds while another operation in the same transaction fails: a bank that allows a customer to withdraw money without recording the reduced balance in an updated account record will not stay in business for long!

A transaction that adheres to the rule that all constituent operations either succeed or fail is characterized by *atomicity*. The BEA Tuxedo system requires all transactions to be characterized by atomicity and three related attributes: *consistency*, *isolation*, and *durability*. These four attributes are known collectively as the *ACID properties* of transactions performed within the BEA Tuxedo system.

Table 4-1 ACID Properties of BEA Tuxedo Transactions

This Property . . .	Means that . . .
Atomicity	A transaction is a discrete unit of work: all constituent operations must either succeed or fail. These operations may include queuing messages, updating databases, and displaying the results of a transaction on a screen.
Consistency	A transaction must either (a) leave the system in a correct state or (b) abort. If a transaction cannot achieve a stable state, it must return to its initial state.
Isolation	The behavior of a transaction is not affected by other transactions being executed simultaneously. A transaction must serialize all access to shared resources and guarantee that concurrent programs do not corrupt each other's operations.

This Property . . .	Means that . . .
Durability	The effects of a committed transaction are permanent. Even if the system fails, the changes resulting from a transaction are permanent and durable.

How a Transaction Succeeds or Fails

Whether a transaction succeeds or fails depends on the requirements of atomicity.

If . . .	Then . . .
Any operation within the transaction fails for any reason	<ul style="list-style-type: none">■ The transaction <i>aborts</i>, that is, it terminates abruptly■ The transaction <i>rolls back</i>, that is, it undoes its own work and restores the state of the enterprise to its pre-transaction state. For example, after an attempt to withdraw money from a bank account fails and is rolled back, the bank account contains the same amount of money it contained before the transaction, and the record of the account balance shows the same amount that it showed before the transaction.
All operations within the transaction succeed	The client <i>commits</i> the transaction. In other words, it formally signals that it is ready to terminate and the effects of the transaction should be preserved: the order database is updated permanently and the order sent to the shipping department is kept as a permanent record in that department's queue.

Benefits of Using Transactions

The BEA Tuxedo system, including its communication APIs and protocols, is designed to support the use of transactions. The BEA Tuxedo communication calls, which make it easy to create transactions, are indispensable tools for writing distributed applications.

By using transactions you can:

- Create distributed applications easily
- Commit the effects of your communications as a single unit
- Quickly manage potential problems that may occur in a distributed environment, such as machine, program, or network failures
- Undo work, when errors occur, in a simple, programmatic way

Example of a Global Transaction

An e-retailer uses a service called `CUST_ORDER`. When a customer places an order through the company's Web site, the `CUST_ORDER` service performs two operations:

- It updates the company's database of orders.
- It sends the new order to the shipping department, where it is put on a queue, awaiting fulfillment.

The company wants to be sure that the `CUST_ORDER` service adheres to the principle of atomicity: whenever `CUST_ORDER` is executed, both the database update and the enqueueing of the customer request on the shipping department queue must be completed successfully. To make sure that the `CUST_ORDER` service always handles customer orders with atomicity, the client that invokes `CUST_ORDER` associates its request with a *global transaction*.

To associate a service with a global transaction, a client:

1. Calls `tpbegin()` to begin the transaction
2. Issues a service request
3. Calls `tpcommit()` to end the transaction

As part of a global transaction, the operation is performed as a single unit of work. When the `CUST_ORDER` service is invoked, the server is propagated with the client's transaction. The two resulting operations, accessing the order database and enqueueing the order to the shipping queue, become part of the client's transaction.

If either operation fails for any reason, whether due to a system error or an application error, the work of the transaction is undone or *rolled back*. In other words, the transaction is returned to its initial state.

If both operations succeed, however, the client *commits* the transaction. In other words, it formally signals that the effects of the transaction should be made permanent: the order database is updated permanently and the order sent to the shipping department is kept in that department's queue.

What Is the BEA Tuxedo Transaction Manager (TM)

A *Resource Manager (RM)* is a data repository, such as a database management system or the Application Queuing Manager, with tools for accessing the data. The BEA Tuxedo system uses one or more RMs to maintain the state of an application. For example, bank records in which account balances are maintained are kept in an RM. When the state of the application changes through a service that allows a customer to withdraw money from an account, the new balance in the account is recorded in the appropriate RM.

The BEA Tuxedo system helps you manage transactions involving resource managers that support the XA interface. To coordinate all the operations performed and all the modules affected by a transaction, the BEA Tuxedo system plays the role of the Transaction Manager (TM).

The TM coordinates global transactions involving system-wide resources. Local resource managers (RMs) are responsible for individual resources. The Transaction Manager Server (TMS) begins, commits, and aborts transactions involving multiple resources. The application code uses the normal embedded SQL interface to the RM to perform reads and updates. The TMS uses the XA interface to the RM to perform the work of a global transaction.

The following table summarizes the actions taken by the Transaction Manager on behalf of each transaction.

Table 4-2 Actions Performed by the Transaction Manager

When . . .	The Transaction Manager . . .
The application launches a transaction	Assigns a global transaction identifier (GTRID) to the transaction
Other processes communicate with the process that launched the transaction	Tracks those <i>communication partners</i>
The RM is accessed as part of the work of the transaction	Passes the appropriate GTRID to the RM so the RM can monitor which database records are being accessed for the transaction.
The application signals that a transaction is to be committed	Performs a 2-phase commit protocol. Specifically, it (a) contacts communication partners during Phase 1, (b) logs the successful outcome of Phase 1, and (c) contacts partners in Phase 2.
The application indicates that the transaction is to be aborted	Executes a rollback procedure
A failure occurs	Executes a recovery procedure

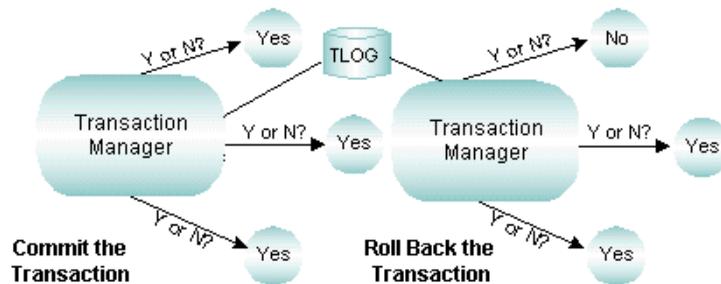
How the System Tracks Distributed Transaction Processing

BEA Tuxedo transactions can be used in a distributed architecture: a local machine involved in a transaction can communicate with a remote machine which may, in turn, communicate with another remote machine. The work of transactions executed in this type of arrangement is referred to as *distributed transaction processing*.

Because the system must constantly maintain enough information about a transaction to be able to roll it back (that is, to restore it to its initial state) at any moment, tracking distributed transaction processing (DTP) can be a complex task. To perform this task successfully, the BEA Tuxedo system stores tracking information about all the participants in a transaction in a dedicated file called a *transaction log*, or TLOG.

The following diagram shows an application in which two Transaction Managers (TMs) are being used. Both TMs record tracking data in the same TLOG.

Figure 4-1 Transaction Management



Before committing a transaction, the TM must repeatedly answer the question of whether to proceed. If necessary, the TM makes the decision to roll back.

How the System Uses Global Transaction Identifiers (GTRIDs) for Tracking

The BEA Tuxedo system tracks the flow of all transactions being executed within a distributed system, including those being executed concurrently. When it is time to commit a transaction, the coordinator must know which RMs have participated in the transaction and, therefore, needs to be able to distinguish among transactions. For this reason the BEA Tuxedo system assigns a *global transaction identifier*, or GTRID to each transaction.

The BEA Tuxedo system communicates with any RM accessed by an application through the XA interface. The RMs track transactions by assigning local transaction identifiers, and map global identifiers to local identifiers.

How the System Uses a Transaction Log (TLOG) for Tracking

A global transaction is recorded in the transaction log (TLOG) only when it is in the process of being committed. At the end of the first phase of a 2-phase-commit protocol, the TLOG records the reply from the global transaction participants.

The existence of a TLOG record indicates that a global transaction should be committed; no TLOG records are written for transactions that are to be rolled back.

In the first “pre-commit” phase, each Resource Manager must commit to performing the transaction request. If all parties commit, transaction management performs the second phase: it commits and completes the transaction. If either task fails because of an application or system failure, both tasks fail and the work performed is undone or “rolled back” to its initial state.

The TMS that coordinates global transactions uses the TLOG file. Each machine should have its own TLOG.

If you are using the Domains component in your application, keep in mind that the Domains gateway performs the functions of the TMS in Domains groups. However, Domains uses its own transaction log containing information similar to that recorded in the TLOG, in addition to Domains-specific information.

How the System Uses a 2-Phase Commit to Commit Transactions

A *2-phase commit* is an algorithm used to ensure the integrity of a committing transaction.

To understand how this algorithm works, consider the following sample scenario. A group of six friends wants to rent a house for a one-week vacation. No member of the group can afford to pay more than one sixth of the rent; if any of the six cannot participate, then the house cannot be rented.

1. In Phase 1 of this project, the organizer of the vacation contacts each person to verify availability and collect a sixth of the rent. If the organizer learns that even one person cannot participate, she contacts every member of the group, individually, to notify him or her that the house cannot be rented. If, however, each member of the group confirms availability and pays one sixth of the rent, the Phase 1 concludes successfully.
2. In Phase 2 of the project, the organizer notifies each member of the group that the vacation will take place as planned.

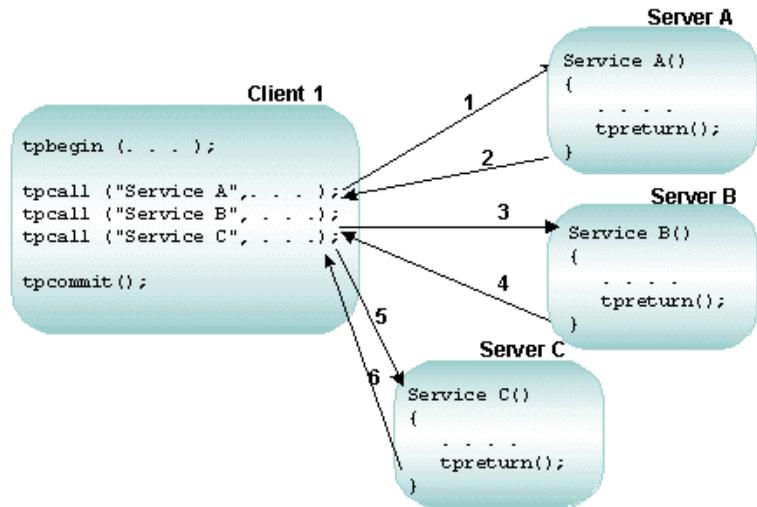
A 2-phase transaction commit works in much the same way as the vacation planning project.

1. In Phase 1, the transaction coordinator contacts potential participants in the transaction. The participants all agree to make the results of the transaction permanent, but do not do so immediately. The participants log information to disk to ensure they can complete Phase 2. If all the participants agree to commit, the coordinator logs that agreement and the outcome is decided. The recording of this agreement in the log ends Phase 1.
2. In Phase 2, the coordinator informs each participant of the decision, and they permanently update their resources.

How the System Handles Transaction Infection

Any application module called by another module to participate in a transaction is said to be *transactionally infected*. Once an application module is infected, the BEA Tuxedo system tracks all participants to determine which of them should be involved in the 2-phase commit. The following figure shows how the system tracks participants.

Figure 4-2 Transactional Infection



In the preceding figure, Client 1 begins the transaction and calls three services: A, B, and C. Because they have been called into the transaction, Services A, B, and C are transactionally infected. All work performed by servers A, B, and C is part of the transaction begun by Client 1. All work is performed as one unit; either it is performed together and is successful, or it fails and is rolled back by calling `tpabort`. If the transaction fails, it returns to its initial state and its effects of the transaction on resource managers are undone. (Resource managers that are not transactionally aware and those that are accessed from outside the transaction cannot be rolled back.)

How the ATMI Protects a Transaction's Integrity Before a 2-Phase Commit

All work performed by each resource involved in a transaction must be completed before a 2-phase commit is begun. The ATMI ensures that all the work of the transaction is stopped when it is time for the 2-phase commit protocol to begin.

The following step-by-step description of a transaction shows how the ATMI stops a transaction process before a 2-phase commit.

1. Client_1 initiates (with `tpbegin()`) a transaction.

2. Client_1 invokes (with `tpcall()`) Service_A, which:
 - a. Is *infected* with the transaction
 - b. Executes its operations
 - c. Calls `tpreturn()`
 - d. Completes its work for the transaction
3. Client_1 invokes (with `tpcall()`) Service_B, which:
 - a. Is *infected* with the transaction
 - b. Executes its operations
 - c. Calls `tpreturn()`
 - d. Completes its work for the transaction
4. Client_1 invokes (with `tpcall()`) Service_C, which:
 - a. Is *infected* with the transaction
 - b. Executes its operations
 - c. Calls `tpreturn()`
 - d. Completes its work for the transaction
5. Client_1 initiates (with `tpcommit()`) the commitment process.

If, during the transaction, an invoked service is performing another service, or is involved in an open conversation, the ATMI tracks that activity and prevents the application from proceeding to the commitment process until the activity is complete.

The ATMI guarantees that the transaction is committed only if all invoked services have performed their transaction work successfully. When all work has been performed successfully, the Transaction Manager informs the resource managers that all updates made during the transaction are permanent.

See Also

- “Modifying the UBBCONFIG File to Accommodate Transactions” on page 5-1
- “Modifying the Domains Configuration File to Support Transactions” on page 5-10
- “Example: A Distributed Application with Transactions” on page 5-12
- “Writing Global Transactions” on page 9-1 in *Programming a BEA Tuxedo Application Using C*
- “What You Can Do Using the ATMI” on page 2-4 in *Introducing the BEA Tuxedo System*

5 Configuring Your Application to Use Transactions

- Modifying the UBBCONFIG File to Accommodate Transactions
- Specifying Global Transaction Parameters in the RESOURCES Section
- Creating a Transaction Log (TLOG) in the MACHINES Section
- Defining Resource Managers and the Transaction Manager Server in the GROUPS Section
- Enabling a Service to Begin a Transaction in the SERVICES Section
- Modifying the Domains Configuration File to Support Transactions
- Example: A Distributed Application with Transactions

Modifying the UBBCONFIG File to Accommodate Transactions

To accommodate transactions, you must modify the `RESOURCES`, `MACHINES`, `GROUPS`, and `SERVICES` sections of the application's `UBBCONFIG` file in the following ways.

In This Section . . .	Specify . . .
RESOURCES	The number of transactions allowed in the application, and the value of the commit control flag
MACHINES	The TLOG information for each machine
GROUPS	Information about each resource manager, and about the transaction manager server
SERVICES	Enabling of the automatic transaction option

Specifying Global Transaction Parameters in the RESOURCES Section

The following table describes the transaction-related parameters in the RESOURCES section.

Set This Parameter . . .	To . . .
MAXGTT	Limit the total number of global transaction identifiers (GTRIDS) allowed on one machine at one time. The maximum value allowed is 2048; the minimum, 0; and the default, 100. You can override the value of MAXGTT on a per-machine basis in the MACHINES section. Entries remain in the table only while a global transaction is active, so this parameter has the effect of setting a limit on the number of simultaneous transactions.

Set This Parameter . . .	To . . .
CMTRET	<p>Indicate the initial setting of the TP_COMMIT_CONTROL characteristic as one of the following:</p> <ul style="list-style-type: none">■ LOGGED—The TP_COMMIT_CONTROL characteristic is set to TP_CMT_LOGGED, which means that <code>tpcommit()</code> returns when all the participants have successfully pre-committed.■ COMPLETE—The TP_COMMIT_CONTROL characteristic is set to TP_CMT_COMPLETE, which means that <code>tpcommit()</code> does not return until all the participants have successfully committed. <p>The default is COMPLETE.</p> <p>To determine the appropriate setting, consult your Resource Manager (RM) vendors. If any RM in the application uses the <i>late commit</i> implementation of the XA standard, the setting should be COMPLETE. If all RMs use the <i>early commit</i> implementation, the setting should be LOGGED for performance reasons. (You can override this setting with <code>tpscmt()</code>.)</p>

Creating a Transaction Log (TLOG) in the MACHINES Section

To create a TLOG, complete the following tasks:

- Create a Universal Device List (UDL).
- Define transaction-related parameters in the MACHINES section.
- Create a Domains transaction log.

Creating the UDL

The Universal Device List (UDL) is a map of the BEA Tuxedo file system. The UDL gets loaded into shared memory when an application is booted. The TLOG refers to a log in which information about transactions is kept until the transaction is completed. To create an entry in the UDL for the TLOG device, create a UDL on each machine using global transactions. (If the TLOGDEVICE is mirrored between two machines, it is unnecessary to do this on the paired machine.) The Bulletin Board Liaison (BBL) then initializes and opens the TLOG during the boot process.

To create a UDL, enter the following command before the application is booted.

```
tadmin -c crdl -z config -b blocks
```

Note: The command fails if the device already exists.

The value of *config* must be the full path name of the device on which you create the UDL. It should match the value of the TLOGDEVICE parameter in the MACHINES section of the configuration file. The value of *blocks* must be the number of blocks to be allocated on the device.

Note: If the value of *blocks* is less than the value of TLOGSIZE, you risk a performance degradation. Therefore, you should specify a value for *blocks* that is greater than that of TLOGSIZE. For example, if TLOGSIZE is specified as 200 blocks, specifying *-b 500* does not cause a degradation.

For more information about storing the TLOG, see *Installing the BEA Tuxedo System*.

Defining Transaction-related Parameters in the MACHINES Section

To define a global transaction log (TLOG), you must set several parameters in the MACHINES section of the UBBCONFIG file.

For one of these parameters, TLOGDEVICE, you must manually create a device list entry for the TLOGDEVICE on each machine where a TLOG is needed. You can do this either before or after TUXCONFIG has been loaded, but you must complete this step before the system is booted.

Creating a Transaction Log (TLOG) in the MACHINES Section

The following table describes the transaction-related parameters in the `MACHINES` section.

Set This Parameter . . .	To Specify . . .
<code>TLOGNAME</code>	The name of the DTP transaction log for the machine.
<code>TLOGDEVICE</code>	The BEA Tuxedo file system that contains the DTP transaction log (TLOG) for the machine. If this parameter is not specified, it is assumed that there is no TLOG on the machine. The value may contain a maximum of 64 characters.
<code>TLOGSIZEE</code>	The size, in physical pages, of the TLOG file. The value must be between 1 and 2048; the default, 100. Assign a value that is large enough to hold the number of outstanding transactions on the machine at a given time. One transaction is logged per page. The default should be enough for most applications.
<code>TLOGOFFSET</code>	<p>The offset, in pages, from the beginning of the <code>TLOGDEVICE</code> to the start of the VTOC that contains the transaction log for the machine. The value must be greater than or equal to 0, and less than the number of pages on the device. The default is 0.</p> <p><code>TLOGOFFSET</code> is rarely necessary. However, if two VTOCs share the same device, or if a VTOC is stored on a device (such as a file system) that is shared with another application, you can use <code>TLOGOFFSET</code> to indicate a starting address relative to the address of the device.</p>

Creating the Domains Transaction Log

Before starting a Domains gateway group, you must create a Domains transaction log. Specifically, you must create a Domains transaction log for the named local domain on the current machine (that is, the machine on which `DMADM` is running). To create a log, enter the following command.

```
dmadmin crdmlog crdlog -d local_domain_name
```

The command uses the parameters specified in the `DMCONFIG` file. This command fails if the named local domain is active on the current machine or if a log already exists. If a transaction log has not been created, the Domains gateway group creates one when that group starts.

See Also

- “What Is the Transaction Log (TLOG)” on page 2-17 in *Administering a BEA Tuxedo Application at Run Time*

Defining Resource Managers and the Transaction Manager Server in the GROUPS Section

The parameters available for GROUPS section entries allow you to define the attributes of transaction manager servers and resource managers (RMs) for a particular group.

- For a transaction manager server, a server that performs most of the work that controls global transactions, you can define the following parameters:
 - TMSNAME contains the name of the executable for the transaction manager server associated with the group defined in the entry. The BEA Tuxedo system provides a null Transaction Manager Server called TMS, which is used by groups that participate in transactions, but do not use an RM. This TMS server does not communicate with any resource manager; it simply manages transactions without communicating with an RM.
 - TMSCOUNT contains the number of transaction manager servers to be booted (minimum of 2, maximum of 10, default of 3)
- For each resource manager you can define the OPENINFO and CLOSEINFO parameters. The value of each is a string that contains information needed to open or close a resource manager, respectively. Appropriate values for these parameters are supplied by RM vendors. For example, if you are using an Oracle database as your RM, you might supply the value shown in the following entry.

```
OPENINFO="ORACLE_XA:  
Oracle_XA+Acc=P/Scott/*****+SesTm=30+LogDir=/tmp"
```

Sample of the GROUPS Section

The following sample entry is from the GROUPS section in bankapp, the sample banking application you received with the BEA Tuxedo system.

```
BANKB1 GRPNO=1 TMSNAME=TMS_SQL TMSCOUNT=2
OPENINFO="TUXEDO/SQL:APPDIR/bankd11:bankdb:readwrite"
```

Description of Transaction Values in the Sample GROUPS Section

The following table describes the transaction values shown in the sample GROUPS entry.

Transaction Value	Purpose
BANKB1 GRPNO=1 TMSNAME=TMS_SQL TMSCOUNT=2	Contains the name of the transaction manager server (TMS_SQL), and the number (2) of these servers to be booted in the group BANKB1
TUXEDO/SQL	Published name of the resource manager
APPDIR/bankd11	Device name
bankdb	Database name
readwrite	Access mode

Characteristics of the TMSNAME, TMSCOUNT, OPENINFO, and CLOSEINFO Parameters

The following table lists the characteristics of the TMSNAME, TMSCOUNT, OPENINFO, and CLOSEINFO parameters.

Set This Parameter . . .	To Specify the . . .
TMSNAME	Name of the transaction manager server executable. Required parameter for applications with transactions. TMS is a null transactional manager server.

Set This Parameter . . .	To Specify the . . .
TMSCOUNT	Number of transaction manager servers (must be between 2 and 10). Default is 3. This parameter is optional.
OPENINFO, CLOSEINFO	Information needed to open or close a resource manager. Content depends on the resource manager. Value starts with the name of the resource manager. Omission means the RM needs no information to open or close.

Enabling a Service to Begin a Transaction in the SERVICES Section

In certain situations, you may want to set three transaction-related parameters—`AUTOTRAN`, `TRANTIME`, and `ROUTING`—in the `SERVICES` section.

- If you want a transaction to be started by a service instead of a client, you must set the `AUTOTRAN` flag to `Y`. This setting is useful if a service is not needed as part of any larger transaction, and if the application wants to relieve the client of making transaction decisions. If the service is called when a transaction already exists, this call becomes part of it. (The default is `N`.)

Note: Generally, clients are the best initiators of transactions because a service can participate in a larger transaction.

- If `AUTOTRAN` is set to `Y`, you must set the `TRANTIME` parameter, which is the length of the time-out for transactions to be created. The value must be greater than or equal to 0, and must not exceed 2,147,483,647 (that is, $2^{31} - 1$, or about 70 years). A value of zero implies there is no time-out for the transaction. (The default is 30 seconds.)
- You must define the `ROUTING` parameter for transactions that use data-dependent routing.

Characteristics of the AUTOTRAN, TRANTIME, and ROUTING Parameters

The following table lists the characteristics of the AUTOTRAN, TRANTIME, and ROUTING parameters.

Set This Parameter . . .	To . . .
AUTOTRAN	<p>Make a service the initiator of a transaction.</p> <p>To work properly, may be dependent on personal communication between the application designer and the application administrator. If the administrator sets this value to Y without prior knowledge of the ICF parameters set by the developer, the wrong application behavior, or failure of the application might be observed.</p> <p>If a transaction already exists, a new one is not started.</p> <p>Default is N.</p>
TRANTIME	<p>Specify the length of the time-out for the AUTOTRAN transactions.</p> <p>Valid values are between 0 and $2^{31} - 1$, inclusive.</p> <p>0 represents no time-out.</p> <p>Default is 30 seconds.</p>
ROUTING	<p>Point to an entry in the ROUTING section where data-dependent routing is specified for transactions that request this service.</p>

Modifying the Domains Configuration File to Support Transactions

To enable transactions across domains, you need to set parameters in both the `DM_LOCAL_DOMAINS` and the `DM_REMOTE_SERVICES` sections of the Domains configuration file (`DMCONFIG`). Entries in the `DM_LOCAL_DOMAINS` section define local domain characteristics. Entries in the `DM_REMOTE_SERVICES` section define services that are *imported*, or available from remote domains.

Characteristics of the `DMTLOGDEV`, `DMTLOGNAME`, `DMTLOGSIZE`, `MAXRDTRAN`, and `MAXTRAN` Parameters

The `DM_LOCAL_DOMAINS` section of the Domains configuration file identifies local domains and the gateway groups associated with them. For each gateway group (Local Domain), you must create an entry that specifies the parameters required for the Domains gateway processes running in that group.

The following table describes the five transaction-related parameters in this section: `DMTLOGDEV`, `DMTLOGNAME`, `DMTLOGSIZE`, `MAXRDTRAN`, and `MAXTRAN`.

Set This Parameter . . .	To Specify . . .
<code>DMTLOGDEV</code>	The BEA Tuxedo file system that contains the Domains transaction log (<code>DMTLOG</code>) for this machine. The <code>DMTLOG</code> is stored as a BEA Tuxedo VTOC table on the <code>TLOGDEVICE</code> (a BEA Tuxedo file system). If this parameter is not specified, the Domains gateway group is not allowed to process requests in transaction mode. Local domains running on the same machine can share the same <code>DMTLOGDEV</code> file system, but a separate log (a table in the <code>DMTLOGDEV</code>) must be created for each local domain. The name of each log is determined by the <code>DMTLOGNAME</code> parameter.

Set This Parameter . . .	To Specify . . .
DMTLOGNAME	The name of the Domains transaction log for this domain. If this domain resides on the same file system as other local domains (as reflected by a common value for DMTLOGDEV), then the value of DMTLOGNAME must be unique for each log. The value may contain a maximum of 30 characters. The default is DMTLOG.
DMTLOGSIZE	The size, in pages, of the Domains transaction log for this machine. The value must be greater than zero and less than the amount of available space on the BEA Tuxedo file system. The default is 100 pages. Note: The number of domains in a transaction determines the number of pages you must specify in the DMTLOGSIZE parameter. There is no one-to-one mapping between transactions and log pages.
MAXRDTRAN	The maximum number of domains that can be involved in a transaction. It must be greater than zero and less than 32,768. The default is 16.
MAXTRAN	The maximum number of simultaneous global transactions allowed in this local domain. It must be greater than or equal to zero, and less than or equal to the MAXGTT parameter (which is defined in the configuration file). The default is the value of MAXGTT.

Characteristics of the AUTOTRAN and TRANTIME Parameters

The `DM_REMOTE_SERVICES` section of the Domains configuration file provides information about services that are *imported* and thus available from remote domains. Each remote service is associated with a particular remote domain.

You have the option of setting two parameters in the `DM_REMOTE_SERVICES` section that support transactions: `AUTOTRAN` and `TRANTIME`. The following table describes these parameters.

This Parameter . . .	Is Used . . .
AUTOTRAN	By gateways to automatically start and terminate transactions for remote services. This capability is required if you want to enforce reliable network communication with remote services. To request this capability, set the AUTOTRAN parameter to Y in the entry for the appropriate remote service.
TRANTIME	To specify the default time-out, in seconds, for a transaction automatically started for the service being defined. The value must be greater than or equal to zero, and less than 2147483648. A value of zero implies the maximum time-out value for the machine. The default is 30 seconds.

Example: A Distributed Application with Transactions

This section provides sample entries from a configuration file that defines bankapp as an application that supports transactions and is distributed over three sites. The application is characterized by the following:

- Data-dependent routing on ACCOUNT_ID
- Data distributed over three databases
- BRIDGE processes communicating with the system via the ATMI interface
- Application administration from one site

The file includes seven sections: RESOURCES, MACHINES, GROUPS, NETWORK, SERVERS, SERVICES, and ROUTING.

Sample RESOURCES Section

The following listing shows a sample RESOURCES section.

Listing 5-1 Sample RESOURCES Section

```
*RESOURCES
#
IPCKEY      99999
UID         1
GID         0
PERM       0660
MAXACCESSERS 25
MAXSERVERS 25
MAXSERVICES 40
MAXGTT      20
MASTER     SITE3, SITE1
SCANUNIT    10
SANITYSCAN  12
BBLQUERY    180
BLOCKTIME   30
DBBLWAIT    6
OPTIONS     LAN, MIGRATE
MODEL       MP
LDBAL       Y
```

In the preceding listing, note the following:

- MAXSERVERS, MAXSERVICES, and MAXGTT are set to values that are smaller than the defaults, which reduces the size of the bulletin board.
- The MASTER is SITE3 and the backup master is SITE1.
- It is possible to use a networked configuration with migration because MODEL is set to MP and OPTIONS is set to LAN, MIGRATE.
- Because BBLQUERY is set to 180 and SCANUNIT is set to 10, the DBBL will check the remote BBLs every 1800 seconds (that is, every half hour).

Sample MACHINES Section

The following listing shows a sample MACHINES section.

Listing 5-2 Sample MACHINES Section

```
*MACHINES
giselle      LMID=SITE1
             TUXDIR="/usr/tuxedo"
             APPDIR="/usr/home"
             ENVFILE="/usr/home/ENVFILE"
             TLOGDEVICE="/usr/home/TLOG"
             TLOGNAME=TLOG
             TUXCONFIG="/usr/home/tuxconfig"
             TYPE="3B600"

romeo       LMID=SITE2
             TUXDIR="/usr/tuxedo"
             APPDIR="/usr/home"
             ENVFILE="/usr/home/ENVFILE"
             TLOGDEVICE="/usr/home/TLOG"
             TLOGNAME=TLOG
             TUXCONFIG="/usr/home/tuxconfig"
             TYPE="SEQUENT"

juliet      LMID=SITE3
             TUXDIR="/usr/tuxedo"
             APPDIR="/usr/home"
             ENVFILE="/usr/home/ENVFILE"
             TLOGDEVICE="/usr/home/TLOG"
             TLOGNAME=TLOG
             TUXCONFIG="/usr/home/tuxconfig"
             TYPE="AMDAHL"
```

In the preceding listing, note the following:

- TLOGDEVICE and TLOGNAME are specified, which implies that transactions will be done.
- The TYPE parameters are all different, which indicates that all messages sent between machines will be encoded and decoded.

Sample GROUPS and NETWORK Sections

The following listing shows sample GROUPS and NETWORK sections.

Listing 5-3 Sample GROUPS and NETWORK Sections

```
*GROUPS
DEFAULT:          TMSNAME=TMS_SQL          TMSCOUNT=2
BANKB1           LMID=SITE1              GRPNO=1
  OPENINFO="TUXEDO/SQL:/usr/home/bankd11:bankdb:readwrite"
BANKB2           LMID=SITE2              GRPNO=2
  OPENINFO="TUXEDO/SQL:/usr/home/bankd12:bankdb:readwrite"
BANKB3           LMID=SITE3              GRPNO=3
  OPENINFO="TUXEDO/SQL:/usr/home/bankd13:bankdb:readwrite"

*NETWORK
SITE1            NADDR="0X0002ab117B2D4359"
                BRIDGE="/dev/tcp"
                NLSADDR="0X0002ab127B2D4359"

SITE2            NADDR="0X0002ab117B2D4360"
                BRIDGE="/dev/tcp"
                NLSADDR="0X0002ab127B2D4360"

SITE3            NADDR="0X0002ab117B2D4361"
                BRIDGE="/dev/tcp"
                NLSADDR="0X0002ab127B2D4361"
```

In the preceding listing, note the following:

- The TMSCOUNT is set to 2, which means that only two TMS_SQL transaction manager servers will be booted per group.
- The OPENINFO string indicates that the application will perform database access.

Sample SERVERS, SERVICES, and ROUTING Sections

The following listing shows sample SERVERS, SERVICES, and ROUTING sections.

Listing 5-4 Sample SERVERS, SERVICES, and ROUTING Sections

```
*SERVERS
DEFAULT: RESTART=Y MAXGEN=5 REPLYQ=N CLOPT="-A"
TLR      SRVGRP=BANKB1      SRVID=1      CLOPT="-A -- -T 100"
TLR      SRVGRP=BANKB2      SRVID=3      CLOPT="-A -- -T 400"
TLR      SRVGRP=BANKB3      SRVID=4      CLOPT="-A -- -T 700"
XFER     SRVGRP=BANKB1      SRVID=5      REPLYQ=Y
XFER     SRVGRP=BANKB2      SRVID=6      REPLYQ=Y
XFER     SRVGRP=BANKB3      SRVID=7      REPLYQ=Y

*SERVICES
DEFAULT: AUTOTRAN=N
WITHDRAW      ROUTING=ACCOUNT_ID
DEPOSIT       ROUTING=ACCOUNT_ID
TRANSFER      ROUTING=ACCOUNT_ID
INQUIRY       ROUTING=ACCOUNT_ID

*ROUTING
ACCOUNT_ID    FIELD=ACCOUNT_ID      BUFTYPE="FML"
              RANGES="MON - 9999:*,
              10000 - 39999:BANKB1
              40000 - 69999:BANKB2
              70000 - 100000:BANKB3
              ""
```

In the preceding listing, note the following:

- Calls to the `tpsvrinit()` function by TLR servers will include a number (100, 400, or 700) specified with the `-T` option.
- All service requests are routed on the `ACCOUNT_ID` field.
- No services are performed in `AUTOTRAN` mode.

See Also

- “What Is a Transaction” on page 4-1
- “Writing Global Transactions” on page 9-1 in *Programming a BEA Tuxedo Application Using C*
- “What You Can Do Using the ATMI” on page 2-4 in *Introducing the BEA Tuxedo System*

6 Distributing Applications Across a Network

- What Is a Distributed Application
- Why Distribute an Application Across a Network

What Is a Distributed Application

A distributed application consists of one or more local or remote clients that communicate with one or more servers on several machines linked through a network. With this type of application, business operations can be conducted from any geographical location. For example, a corporation may distribute the following types of operations across a large region, or even across international boundaries:

- Forecasting sales
- Ordering supplies
- Manufacturing, shipping, and billing for goods
- Updating corporate databases

6 *Distributing Applications Across a Network*

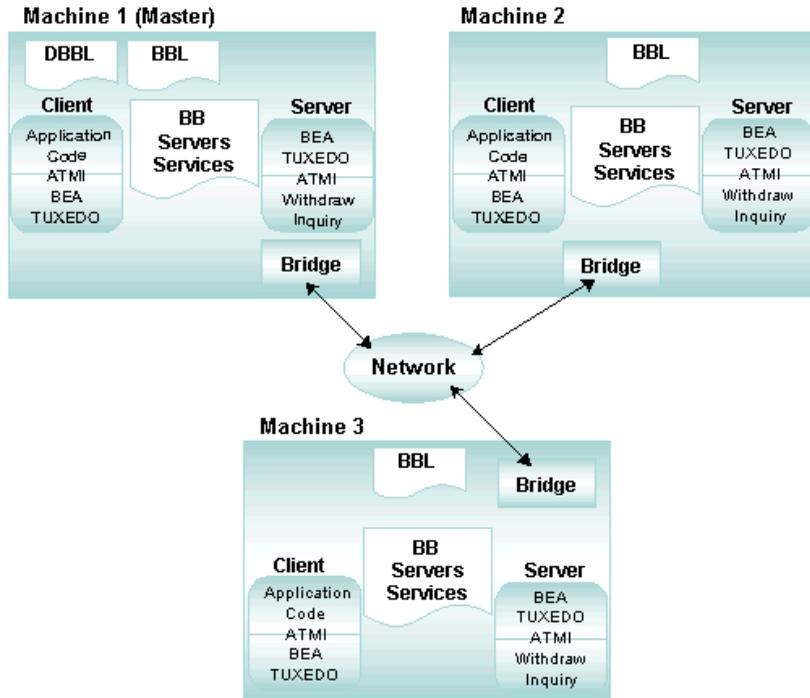
State of the art telecommunications and data networks are making distributed operations of this sort increasingly common. Applications developed to implement this type of strategy allow businesses to reduce costs and enhance their offerings of services to customers around the world.

The BEA Tuxedo system supports this type of architecture by simplifying the task of managing a distributed application. Whether an application comprises only one computer or thousands of computers working together over a network, all the elements of that application, including clients, servers, and the networks that connect them, are managed through a single BEA Tuxedo configuration file.

Example of a Distributed Application

The following diagram illustrates the basic parts of an application distributed across three machines.

Figure 6-1 Sample of a Distributed Application



Implementing a Distributed Application

A distributed application is implemented on a network defined in the `NETWORK` (and optionally `NETGROUPS`) section(s) of the configuration file. It frequently uses data-dependent routing, defined in the `ROUTING` section of the configuration file. A critical part of the design of a distributed application is the arrangement between server groups, processes, transaction manager servers, and resource managers.

To set up a distributed application over a network, the application administrator must work with the network administrator. In most instances, the application administrator writes the configuration file for a distributed application (defining parameters in the `RESOURCES`, `MACHINES`, `GROUPS`, `SERVICES`, and `ROUTING` sections), and the network administrator or MIS representative writes or contributes to the networking sections.

See Also

- “Creating the Configuration File for a Distributed Application” on page 7-1
- “Setting Up the Network for a Distributed Application” on page 8-1
- “Managing the Network in a Distributed Application” on page 4-1 in *Administering a BEA Tuxedo Application at Run Time*

Why Distribute an Application Across a Network

Distributed applications provide several important benefits. Early business applications were developed to run on one large mainframe computer. Because all computing was performed on a single machine, a failure could bring down an entire system. With the increasing popularity of distributed applications, this threat of system failure is declining.

Another advantage is that by distributing an application, you can group parts of an application logically and position these logical groups in the most effective locations. By creating groups of servers, for example, you can partition a large application into separate, business-specific components of manageable size and optimal location.

A distributed application allows you to do the following:

- Perform data-dependent partitioning
- Manage multiple resources
- Enlarge the client and/or server model
- Obtain transparent access to BEA Tuxedo system services
- Establish multiple server groups
- Use multiple computers simultaneously to do the work of one application, providing better throughput and response time
- Provide for replicated resources for increased availability

Features of a Distributed Application

- *Coordination of autonomous actions*—Autonomous actions are actions that involve multiple server groups and/or multiple resource manager interfaces. The BEA Tuxedo system enables you to coordinate autonomous actions among separate applications as a single logical *unit of work*.
- *Resilience*—When one of many machines fails, the remaining machines continue to operate. Similarly, when one server in a server group fails, the remaining servers continue the work.
- *Scalability*—Application load or capacity can be increased by:
 - Placing more servers in a group
 - Adding machines to an application and redistributing groups across machines
 - Replicating a server group that resides on one machine, on other machines, and using load balancing
 - Segmenting a database using data-dependent routing for groups that meet specific criteria

See Also

- “What Is a Multiple-machine (Distributed) Configuration” on page 3-45 in *Introducing the BEA Tuxedo System*
- “What Is Load Balancing” on page 2-39 in *Introducing the BEA Tuxedo System*
- “What Is Data-dependent Routing” on page 2-32 in *Introducing the BEA Tuxedo System*

7 Creating the Configuration File for a Distributed Application

- Configuration File Requirements for a Distributed Application
- Creating the RESOURCES Section
- Creating the MACHINES Section
- Creating the GROUPS Section
- Creating the SERVICES Section
- Creating the ROUTING Section
- Example Configuration File for a Distributed Application
- Modifying the Domain Gateway Configuration File to Support Routing

Configuration File Requirements for a Distributed Application

A distributed application consists of one or more local or remote clients that communicate with one or more servers residing on several machines linked through a network, all of which are administered as a single entity in one BEA Tuxedo configuration file. To set up a distributed configuration, you must create a configuration file that includes the following sections:

- [RESOURCES section](#)
- [MACHINES section](#)
- [GROUPS section](#)
- [NETGROUPS section \(optional\)](#)
- [NETWORK section](#)
- [SERVICES section](#)
- [ROUTING section \(if data-dependent routing is used\)](#)

If your configuration spans multiple domains and uses data-dependent routing, you must also modify the domain gateway configuration file (DMCONFIG) to support routing functionality.

Creating the RESOURCES Section

In the RESOURCES section you define governing parameters for system-wide resources, such as the maximum number of servers allowed in the application. All parameter settings in this section apply to the entire application.

Note: The parameters described in the tables in this topic are used only for distributed applications. For a description of the basic parameters that are available for any kind of BEA Tuxedo application, see `UBBCONFIG(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*.

Table 7-1 RESOURCES Section Parameters

Parameter	Description
BBLQUERY (optional)	<p>BBLQUERY sets a multiplier of the basic SCANUNIT between status checks by the DBBL of all BBLs. The DBBL checks to ensure that all BBLs have reported in within the BBLQUERY cycle. If a BBL has not been heard from, the DBBL sends a message to that BBL asking for status. If no reply is received, the BBL is partitioned.</p> <p>The value of BBLQUERY must be greater than 0. If this parameter is not specified, the default is set so that $(SCANUNIT * BBLQUERY)$ is approximately 300 seconds.</p>
BLOCKTIME (optional)	<p>BLOCKTIME sets a multiplier of the basic SCANUNIT after which a blocking call (for example, receiving a reply) times out. The value of BLOCKTIME must be greater than 0. If this parameter is not specified, the default is set so that $(SCANUNIT * BLOCKTIME)$ is approximately 60 seconds.</p>
DBBLWAIT (optional)	<p>DBBLWAIT sets a multiplier of the basic SCANUNIT for the maximum amount of wall time a DBBL should wait for replies from all its BBLs before timing out. Every time the DBBL forwards a request to its BBLs, it waits for all of them to reply with a positive acknowledgment before replying to the requester. This option can be used for detecting dead or insane BBLs in a timely manner.</p> <p>The value of DBBLWAIT must be greater than 0. If this parameter is not specified, the default is set so that $(SCANUNIT * DBBLWAIT)$ is the greater of SCANUNIT or 20 seconds.</p>

Table 7-1 RESOURCES Section Parameters

Parameter	Description
IPCKEY (required)	<p>IPCKEY specifies the numeric key for the bulletin board. In a single-processor environment, this key <i>names</i> the bulletin board. In a multiprocessor environment, this key <i>names</i> the message queue of the DBBL. This key is also used as a basis for deriving the names of resources other than this well-known address, such as the names for bulletin boards throughout a multiprocessor.</p> <p>The value of IPCKEY must be greater than 32,768 and less than 262,143.</p>
MASTER (required)	<p>MASTER (<i>string_value1</i> [, <i>string_value2</i>]) specifies the LMID of the machine on which the master copy of TUXCONFIG is located. Also, if the application is run in MP mode, MASTER indicates the machine on which the DBBL is run. <i>string_value2</i> names an alternate LMID location used during process relocation and booting. If the primary location is not available, the DBBL is booted at the alternate location and the alternate TUXCONFIG file found there is used.</p> <p>The value of both <i>string_value1</i> and <i>string_value2</i> must be LMIDs of machines defined in the MACHINES section. Each string may contain up to 30 characters.</p>
MAXGROUPS (optional)	<p>MAXGROUPS specifies the maximum number of configured server groups to be accommodated in the group table of the bulletin board.</p> <p>The value of MAXGROUPS must be greater than or equal to 100 and less than 32,768. The default is 100.</p>
MAXSERVERS (optional)	<p>MAXSERVERS specifies the maximum number of servers to be accommodated in the server table of the bulletin board.</p> <p>The value of MAXSERVERS must be greater than 0 and less than 8192. The default is 50.</p>
MAXSERVICES (optional)	<p>MAXSERVICES specifies the maximum number of services to be accommodated in the services table of the bulletin board.</p> <p>The value of MAXSERVICES must be greater than 0 and less than 32,768. The default is 100.</p>

Table 7-1 RESOURCES Section Parameters

Parameter	Description
SANITYSCAN (optional)	<p>SANITYSCAN sets a multiplier of the basic SCANUNIT between sanity checks of the system.</p> <p>The value of SCANUNIT must be greater than 0. The default is set so that (SCANUNIT * SANITYSCAN) is approximately 120 seconds.</p> <p>Sanity checks are performed on servers as well as on the bulletin board data structure itself.</p>
SCANUNIT (optional)	<p>SCANUNIT sets the time interval (in seconds) between scans by the bulletin board liaison for timed-out transactions and blocking calls within service requests. This value is used as the basic unit of scanning by the BBL. It affects the granularity with which transaction time-out values can be specified on <code>tpbegin(3c)</code> and the blocking time-out value specified with the BLOCKTIME parameter. The SANITYSCAN, BBLQUERY, DBBLWAIT, and BLOCKTIME parameters are multipliers of this unit for other timed operations within the system.</p> <p>The value of SCANUNIT must be a multiple of 5 greater than 0 and less than or equal to 60 seconds. The default is 10 seconds.</p>

Creating the MACHINES Section

In the MACHINES section you assign logical names to all the physical machines in your configuration (including all the processing elements in multiprocessor machines) and define other parameters for individual machines. The following table describes the parameters available for defining machine names and other machine-specific parameters for each machine that participates in a distributed application.

Table 7-2 MACHINES Section Parameters

Parameter	Description
ENVFILE (optional)	<p>ENVFILE specifies a file that defines the environment with which all clients and servers on the machine are to be executed.</p> <p>Lines must be in the form <i>ident=value</i> where <i>ident</i> contains only underscores and/or alphanumeric characters, and begins with an underscore or a letter of the alphabet.</p> <p>If the value of ENVFILE is an invalid filename, no values are added to the environment.</p>
MAXACCESSERS (optional)	<p>MAXACCESSERS specifies the maximum number of processes that can access the bulletin board on this processor at any one time. When calculating the appropriate number, you are not required to count system administration processes, such as the BBL and tmadmin, but you must count all application servers and clients, and TMS servers.</p> <p>The value of MAXACCESSERS must be greater than 0 and less than 32,768. The default is the value specified in the RESOURCES section.</p>
MAXCONV (optional)	<p>MAXCONV specifies the maximum number of simultaneous conversations allowed for processes on a particular machine.</p> <p>The value of MAXCONV must be greater than 0 and less than 32,768. The maximum number of simultaneous conversations per server is 64. The default is the value specified in the RESOURCES section.</p>
MAXWSCLIENTS (optional)	<p>MAXWSCLIENTS specifies the number of accesser entries on this processor to be reserved for Workstation clients only. This parameter is used only when the BEA Tuxedo System Workstation component is used. This number takes a portion of the total accesser slots specified with MAXACCESSERS. The appropriate setting of this parameter helps conserve IPC resources because Workstation client access to the system is multiplexed through a BEA Tuxedo system-supplied surrogate, the Workstation Handler.</p> <p>The value of MAXWSCLIENTS must be greater than or equal to 0, and less than 32,768; it may not be greater than the value of MAXACCESSERS. (Assigning a value to MAXWSCLIENTS that is higher than the value of MAXACCESSERS is an error.) The default is 0.</p>

Creating the GROUPS Section

In the `GROUPS` section you identify each server group in your application so that the BEA Tuxedo system can route requests to the member servers of specific groups.

The `GROUPS` section is populated with the number of server groups required for the application. Server groups can all reside on the same site (`SHM` mode) or, in a distributed application, they can reside on different sites (`MP` mode).

Parameters in the `GROUPS` section implement two important aspects of distributed transaction processing:

- They associate a group of servers with a particular `LMID` and a particular instance of a resource manager.
- By allowing a second `LMID` to be associated with the server group, they name an alternate machine to which a group of servers can be migrated if the `MIGRATE` option is specified.

The following table describes the parameters in the `GROUPS` section.

Table 7-3 GROUPS Section Parameters

Parameter	Description
<code>ENVFILE</code>	<p><code>ENVFILE</code> specifies a file that defines the environment with which all servers in the group are executed.</p> <p>Lines must be in the form <code>ident=value</code> where <code>ident</code> contains only underscores and/or alphanumeric characters.</p> <p>If the value of <code>ENVFILE</code> is an invalid filename, no values are added to the environment.</p>
<code>GRPNO</code> (required)	<p><code>GRPNO</code> associates a number with a particular server group.</p> <p>The number must be greater than 0 and less than 30000. It must be unique among entries in the <code>GROUPS</code> section.</p>

Table 7-3 GROUPS Section Parameters

Parameter	Description
LMID (required)	LMID identifies the machine on which the server group being defined runs. A second LMID value can be specified (separated from the first by a comma) for an alternate machine to which this server group can be migrated if the MIGRATE option has been specified. Servers in the group can be migrated if RESTART=Y to migrate is specified in the GROUPS section. The values of LMID must be the values assigned to the LMID parameter in the MACHINES section.

Creating the SERVICES Section

The SERVICES section contains parameters that determine how application services are handled. Every line of every entry in this section is associated with a service by its identifier name.

You must identify the service provided by each server group in the SERVICES section. Because the same service can be link edited with more than one server, the SRVGRP parameter is provided to tie the parameters for an instance of a service to a particular group of servers.

The following table describes the parameters in the SERVICES section that are available for defining distributed applications.

Table 7-4 SERVICES Section Parameters

Parameter	Description
LOAD (optional)	LOAD specifies the size of the load imposed by SVCNM on the system. The value of LOAD must be a number between 1 and 32767, inclusive. A higher number indicates a greater load. The default is 50.

Table 7-4 SERVICES Section Parameters

Parameter	Description
PRIO (optional)	<p>PRIO specifies the dequeuing priority of SVCNM.</p> <p>The value of PRIO must be greater than 0 and less than or equal to 100, with 100 being the highest priority. The default is 50.</p>
ROUTING (optional)	<p>ROUTING specifies the name of the routing criteria used for this service when data-dependent routing is being performed. If this parameter is not specified, data-dependent routing is not performed for this service.</p> <p>The value of ROUTING may contain up to 15 characters. If multiple entries exist for the same service name but with different SRVGRP parameters, the ROUTING parameter must be the same for all entries.</p>
SRVGRP (optional)	<p>SRVGRP specifies the host server group for the service that is specified by SVCNM and controlled by the parameters set in this section.</p> <p>By setting SRVGRP, you can assign different parameter settings to the same service when it is offered by different server groups. For example, suppose your application provides two server groups, GROUP1 and GROUP2, that offer a service called WITHDRAW. By setting SRVGRP you can assign different load factors to each copy of the service, as follows.</p> <pre>WITHDRAW ROUTING=123 LOAD=60 SRVGRP=GROUP1 WITHDRAW ROUTING=123 LOAD=60 SRVGRP=GROUP2</pre> <p>The value of SRVGRP may contain up to 30 characters.</p>
SVCTIMEOUT (optional)	<p>SVCTIMEOUT specifies the amount of time, in seconds, that is allowed for processing of the indicated service. A timed-out service causes the server processing the service request to be terminated with a SIGKILL signal.</p> <p>The value of SVCTIMEOUT must be greater than or equal to 0. A value of 0 indicates that the service will not be timed out. The default is 0.</p>

If your application includes transaction processing, you may also want to set three other parameters in the SERVICES section: AUTOTRAN, ROUTING, and TRANTIME. These parameters are described in “Configuring Your Application to Use Transactions” on page 5-1.

The following listing shows a sample of the `SERVICES` section.

```
*SERVICES  
  
WITHDRAW  ROUTING=ACCOUNT_ID  
DEPOSIT   ROUTING=ACCOUNT_ID  
OPEN_ACCT ROUTING=BRANCH_ID
```

Creating the ROUTING Section

In the `ROUTING` section you specify the criteria to be used when data-dependent routing is performed. If a service is listed in multiple entries, each with a different `SRVGRP` parameter, the `ROUTING` section must be set with the same value in all entries. Otherwise, routing cannot be done consistently for that service. Because a service can be routed on one field only, the value of that field must be the same in all entries for the same service.

You can add a `ROUTING` section to the configuration file to show mappings between data ranges and groups. The information in this section enables the system to send a request to a server in a specific group. Each `ROUTING` section item contains an identifier that is used in the `SERVICES` section.

Lines within the `ROUTING` section have the following form.

```
CRITERION_NAME required_parameters
```

where *CRITERION_NAME* is the name of the routing entry specified in the `SERVICES` section for data-dependent routing. The value of *CRITERION_NAME* must be a string with a maximum of 15 characters.

The following table describes the parameters in the ROUTING section.

Table 7-5 ROUTING Section Parameters

Parameter	Description
RANGES	Ranges and associated server groups for the routing field.
FIELD	Name of the routing field, which is assumed to be one of the following: an FML buffer, an XML element or element attribute, a view field name identified in an FML field table (using the FLDTBLDIR and FIELDTBLS environment variables), or an FML view table (using the VIEWDIR and VIEWFILES environment variables). This information is used to obtain the associated field value for data-dependent routing when sending a message.
BUFTYPE	A list of types and subtypes of data buffers for which this routing entry is valid. The value of this parameter may contain up to 256 characters with a maximum of 32 type/subtype combinations.

See Also

- “What Is a Multiple-machine (Distributed) Configuration” on page 3-45 in *Introducing the BEA Tuxedo System*
- “How to Create the Configuration File for a Multiple-machine (Distributed) Application” on page 3-3
- UBBCONFIG(5) in *BEA Tuxedo File Formats and Data Descriptions Reference*

Example Configuration File for a Distributed Application

The following excerpt from a sample UBBCONFIG file shows the GROUPS, SERVICES, and ROUTING sections, which support data-dependent routing in a BEA Tuxedo application.

```
*GROUPS
BANKB1          GRPNO=1
BANKB2          GRPNO=2
BANKB3          GRPNO=3
#
*SERVICES
WITHDRAW        ROUTING=BY_ACCOUNT_ID
DEPOSIT         ROUTING=BY_ACCOUNT_ID
INQUIRY         ROUTING=BY_ACCOUNT_ID
OPEN_ACCT       ROUTING=BY_BRANCH_ID
CLOSE_ACCT      ROUTING=BY_BRANCH_ID
#
*ROUTING
BY_ACCOUNT_ID  FIELD=ACCOUNT_ID BUFTYPE="FML"
                RANGES="MIN - 9999:*,
                10000-49999:BANKB1,
                50000-79999:BANKB2,
                80000-109999:BANKB3,
                *: *"
BY_BRANCH_ID    FIELD=BRANCH_ID BUFTYPE="FML"
                RANGES="MIN - 0:*,
                1-4:BANKB1,
                5-7:BANKB2,
                8-10:BANKB3,
                *: *"
```

Modifying the Domain Gateway Configuration File to Support Routing

All domain gateway configuration information is stored in a binary file called `BDMCONFIG`. This file is created by first writing a text configuration file called `DMCONFIG` and then compiling it into a binary version called `BDMCONFIG`. The compiled `BDMCONFIG` file can be updated while the system is running by using the `dmadmin(1)` command. Although the BEA Tuxedo documentation refers to these configuration files as `DMCONFIG` and `BDMCONFIG`, you can give these files any names.

You must have one `BDMCONFIG` file for each BEA Tuxedo application to which you want to add Domains functionality. System access to the `BDMCONFIG` file is provided through the Domains administrative server, `DMADM(5)`. When a gateway group is booted, the gateway administrative server, `GWADM(5)`, requests from the `DMADM` server a copy of the configuration required by that group. The `GWADM` server and the `DMADM` server also ensure that run-time changes to the configuration are reflected in the corresponding domain gateway groups.

Note: For more information about the `DMCONFIG` file, refer to `DMCONFIG(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*.

Description of ROUTING Section Parameters in DMCONFIG

The `DM_ROUTING` section provides information for data-dependent routing of service requests using `FML`, `XML`, `VIEW`, `X_C_TYPE`, and `X_COMMON` typed buffers. Lines within the `DM_ROUTING` section have the following form.

```
CRITERION_NAME required_parameters
```

where `CRITERION_NAME` is the name of the routing entry specified in the `SERVICES` section. The value of `CRITERION_NAME` must be a string with a maximum of 15 characters.

7 *Creating the Configuration File for a Distributed Application*

The following table describes the parameters in the `DM_ROUTING` section.

Parameter	Description
<code>FIELD</code> (optional)	<p>Specifies the name of the routing field, which is assumed to be one of the following: an FML buffer, an XML element or element attribute, a view field name identified in an FML field table (using the <code>FLDTBLDIR</code> and <code>FIELDTBLS</code> environment variables), or an FML view table (using the <code>VIEWDIR</code> and <code>VIEWFILES</code> environment variables). This information is used to obtain the associated field value for data-dependent routing when sending a message.</p> <p>If a field in an FML32 buffer is used for routing, it must have a field number less than or equal to 8191.</p>

Modifying the Domain Gateway Configuration File to Support Routing

Parameter	Description
RANGES (optional)	<p>Specifies the ranges and associated remote domain names (RDOM) for the routing field. The value of RANGES must be a string enclosed in double quotes. The enclosed string, in turn, must consist of a comma-separated ordered list of <i>range/RDOM</i> pairs.</p> <p>The value of <i>range</i> may be either a single value (a signed numeric value or a character string enclosed in single quotes), or a range of the form <i>lower - upper</i> (where <i>lower</i> and <i>upper</i> are both signed numeric values or character strings in single quotes).</p> <p>The value of <i>lower</i> must be less than or equal to <i>upper</i>. A single quote embedded in a character string value, as in "O'Brien," for example, must be preceded by two back slashes: "O\\'Brien".</p> <p>Use MIN to indicate the minimum value for the data type of the associated FIELD. For strings and arrays, it is the null string; for character fields, it is 0; for numeric values, it is the minimum numeric value that can be stored in the field.</p> <p>Use MAX to indicate the maximum value for the data type of the associated FIELD. For strings and arrays, it is effectively an unlimited string of octal-255 characters; for a character field, it is a single octal-255 character; for numeric values, it is the maximum numeric value that can be stored in the field. Thus, MIN - -5 is all numbers less than or equal to -5, and 6 - MAX is all numbers greater than or equal to 6.</p> <p>The metacharacter * (wildcard) in the position of a range indicates any values not covered by other ranges previously seen in the entry. Only one wildcard range is allowed per entry and it should be listed last (ranges following it are ignored).</p>

Parameter	Description
BUFTYPE (optional)	<p>BUFTYPE provides a list of types and subtypes of data buffers for which this routing entry is valid. Valid types are FML, VIEW, X_C_TYPE, and X_COMMON. No subtype can be specified for type FML, and subtypes are required for the other types (* is not allowed). Duplicate type/subtype pairs cannot be specified for the same routing criteria name; more than one routing entry can have the same criteria name as long as the type/subtype pairs are unique.</p> <p>If multiple buffer types are specified for a single routing entry, the data types of the routing field for each buffer type must be the same. If the field value is not set (for FML buffers), or does not match any specific range, and a wildcard range has not been specified, an error is returned to the application process that requested the execution of the remote service.</p>

Routing Field Description

The value in the routing field can be any data type supported in FML or VIEW; it may be a numeric range or a string range. The following rules apply to string range values for string, carray, and character field types:

- They must be enclosed by single quotation marks and cannot be preceded by a plus or minus sign.
- A short or long integer value must be a string of digits, optionally preceded by a plus or minus sign.
- Floating point numbers must be written in the form required by the C compiler or `atof()`: a plus or minus sign, followed by a string of digits (optionally containing a decimal point), then an optional `e` or `E` followed by an optional sign or space, followed by an integer.
- When a field value matches a range, the associated `RDOM` value specifies the remote domain to which the request should be routed. An `RDOM` value of `*` indicates that the request may be sent to any remote domain known by the gateway group. Within a `range/RDOM` pair, the range must be separated from the `RDOM` by a colon.

Example of a 5-Site Domain Configuration Using Routing

The following sample configuration file defines a two-domain application distributed across five sites. The five sites include a Central Bank Office and four bank branches. Three of the branches belong to a BEA Tuxedo domain. The fourth branch belongs to another TP domain, and OSI-TP is used to communicate with that domain.

The example shows the BEA Tuxedo system domain gateway configuration file from the Central Bank point of view. In the `DM_TDOMAIN` section, this example shows a mirrored gateway for `b01`.

Listing 7-1 Domains Configuration File for Five Sites

```
# TUXEDO DOMAIN CONFIGURATION FILE FOR THE CENTRAL BANK
#
#
*DM_LOCAL_DOMAINS
# local_domain_name Gateway_Group_name domain_type domain_ID log_device
#           [audit log] [blocktime]
#           [log name] [log offset] [log size]
#           [maxrdom] [maxrdtran] [maxtran]
#           [maxdatalen] [security]
#           [tuxconfig] [tuxoffset]

#
#
DEFAULT: SECURITY = NONE
c01      GWGRP = bankg1
        TYPE = TDOMAIN
        DOMAINID = "BA.CENTRAL01"
        DMTLOGDEV = "/usr/apps/bank/DMTLOG"
        DMTLOGNAME = "DMTLG_C01"
c02      GWGRP = bankg2
        TYPE = OSITP
        DOMAINID = "BA.CENTRAL01"
        DMTLOGDEV = "/usr/apps/bank/DMTLOG"
        DMTLOGNAME = "DMTLG_C02"
        NWDEVICE = "OSITP"
        URCH = "ABCD"

#
*DM_REMOTE_DOMAINS
#remote_domain_name domain_type domain_ID
#
b01      TYPE = TDOMAIN
        DOMAINID = "BA.BANK01"
```

7 Creating the Configuration File for a Distributed Application

```
b02     TYPE = TDOMAIN
        DOMAINID = "BA.BANK02"
b03     TYPE = TDOMAIN
        DOMAINID = "BA.BANK03"
b04     TYPE = OSITP
        DOMAINID = "BA.BANK04"
        URCH = "ABCD"

#
*DM_TDOMAIN
#
#     local_or_remote_domain_name network_address [nwdevice]
#
# Local network addresses
c01     NWADDR = "//newyork.acme.com:65432"      NWDEVICE = "/dev/tcp"
c02     NWADDR = "//192.76.7.47:65433"         NWDEVICE = "/dev/tcp"
# Remote network addresses: second b01 specifies a mirrored gateway
b01     NWADDR = "//192.11.109.5:1025"         NWDEVICE = "/dev/tcp"
b01     NWADDR = "//194.12.110.5:1025"         NWDEVICE = "/dev/tcp"
b02     NWADDR = "//dallas.acme.com:65432"     NWDEVICE = "/dev/tcp"
b03     NWADDR = "//192.11.109.156:4244"       NWDEVICE = "/dev/tcp"
#
*DM_OSITP
#
#local_or_remote_domain_name apt aeq
#
#                               [aet] [acn] [apid] [aeid]
#                               [profile]
#
c02     APT = "BA.CENTRAL01"
        AEQ = "TUXEDO.R.4.2.1"
        AET = "{1.3.15.0.3},{1}"
        ACN = "XATMI"
b04     APT = "BA.BANK04"
        AEQ = "TUXEDO.R.4.2.1"
        AET = "{1.3.15.0.4},{1}"
        ACN = "XATMI"

*DM_LOCAL_SERVICES
#service_name [Local_Domain_name] [access_control] [exported_svcname]
#
#           [inbuftype] [outbuftype]
#
open_act      ACL = branch
close_act     ACL = branch
credit
debit
balance
loan          LDOM = c02      ACL = loans
*DM_REMOTE_SERVICES
#service_name [Remote_domain_name] [local_domain_name]
#
#           [remote_svcname] [routing] [conv]
#
#           [trantime] [inbuftype] [outbuftype]
```

Modifying the Domain Gateway Configuration File to Support Routing

```
#
tlr_add  LDOM = c01  ROUTING = ACCOUNT
tlr_bal  LDOM = c01  ROUTING = ACCOUNT
tlr_add  RDOM = b04  LDOM = c02  RNAME = "TPSU002"
tlr_bal  RDOM = b04  LDOM = c02  RNAME = "TPSU003"
*DM_ROUTING
# routing_criteria      field typed_buffer ranges
#
ACCOUNT FIELD = branchid  BUFTYPE = "VIEW:account"
          RANGES = "MIN - 1000:b01, 1001-3000:b02, *:b03"
*DM_ACCESS_CONTROL
#acl_name  Remote_domain_list
#
branch  ACLIST = b01, b02, b03
loans   ACLIST = b04
```

See Also

- “What Is the Domains Configuration File” on page 1-19 in *Using the BEA Tuxedo Domains Component*
- “Configuring a Domains Environment” on page 2-18 in *Using the BEA Tuxedo Domains Component*

8 Setting Up the Network for a Distributed Application

- Configuring the Network for a Distributed Application
- How Data Moves Over a Network
- How Data Moves Over Parallel Networks
- Example of a Network Configuration for a Simple Distributed Application
- How Failover and Failback Work in Scheduling Network Data
- Example Configuration of Multiple Netgroups

Configuring the Network for a Distributed Application

A distributed application is an application that runs on multiple computers, each of which supports an installation of the BEA Tuxedo system. These computers are connected and can communicate with each other through a network that includes hardware, software, access methods, and communication protocols. The BEA Tuxedo system encodes, routes, and decodes messages, and uses the network to ship those messages between machines. The system performs these tasks automatically.

8 *Setting Up the Network for a Distributed Application*

To configure the networking functionality required to support a distributed application, include the following entries in the configuration file.

In This Section . . .	Set This Parameter . . .	To . . .
RESOURCES	MODEL (required)	MP. This parameter enables all other networking parameters. It is used only for networked machines. SHM is used for a single-machine configuration, even if the machine is a multiprocessor.
	OPTIONS (required)	LAN (Local Area Network) to indicate that communication will take place between separate machines, rather than between separate processes on the same machine.
	MAXNETGROUPS (optional)	Designate a limit on the number of NETGROUPS that can be defined. The default is 8; the upper limit, 8192.

In This Section ...	Set This Parameter ...	To ...
MACHINES	<code>TYPE=string</code> (optional)	<p>Determine whether encoding is required when messages are exchanged by two machines. The <code>TYPE</code> parameter specifies the data representation being used on each machine being defined. If a message is being sent from a machine on which one type of data representation is being used to a machine on which a different type of data representation is being used, the message to be sent must be encoded before transmission and decoded upon arrival.</p> <p>If the machines in question both use the same type of data representation, however, the system skips the encoding/decoding process.</p> <p>Example 1</p> <pre>LMID_1 TYPE = "abc" LMID_2 TYPE = "abc"</pre> <p>Encoding is not used in this case.</p> <p>Example 2</p> <pre>LMID_1 TYPE = "HP" LMID_2 TYPE = "SUN"</pre> <p>Encoding is used in this case.</p> <p>You do not need to set this parameter if the same type of data representation is used on all machines that will exchange messages. The parameter must be set only for a machine on which a different type is used. For example, if you have nine SPARC machines and one HP machine, you must specify <code>TYPE=string</code> only for the HP. For the SPARC machines, the default null string identifies them as the same type.</p>

8 Setting Up the Network for a Distributed Application

In This Section ...	Set This Parameter ...	To ...
	<code>CMPLIMIT=<i>remote</i> [, <i>local</i>]</code> (optional)	<p>Specify the compression threshold, that is, the minimum byte size for a message to be compressed before being sent to a remote and/or local destination. The value of both <i>remote</i> and <i>local</i> is a number between 0 and MAXLONG. If CMPLIMIT is set to only one value, it is assumed that the specified value is the <i>remote</i> argument and that messages sent to local destinations are never compressed.</p> <p>For example, if you set <code>CMPLIMIT=1024</code>, than any message greater than 1024 bytes bound for a remote location is compressed.</p> <p>Compression thresholds can also be specified with the variable <code>TMCPLIMIT</code>. See the discussion, in <code>tuxenv(5)</code>, about the variable <code>TMCMPPRFM</code>, which sets the degree of compression in a range of 1 to 9.</p>
	<code>NETLOAD=<i>number</i></code> (optional)	<p>Add an application-specific number to the value of <code>LOAD</code> for a remote service. The result is used by the system to evaluate whether a request should be processed locally or sent to a remote machine. A higher <code>NETLOAD</code> results in less traffic being sent to a remote machine.</p>
<code>NETGROUPS</code> (optional)	<code>NETGROUP</code> (required)	<p>Specify the name assigned by the application to a particular group of machines. The name may contain up to 30 characters. One group, consisting of all the machines on the network, must be named <code>DEFAULTNET</code>.</p>
	<code>NETGRPNO=<i>number</i></code> (required)	<p>Specify a number by which the system can identify a group of machines. The value can be any number between 1 and 8192. For <code>DEFAULTNET</code>, the value of <code>NETGRPNO</code> must be 0.</p>
	<code>NETPRIO=<i>number</i></code> (optional)	<p>Assign a priority to a <code>NETGROUP</code>. This parameter helps the system determine which network connection to use. The number must be between 0 and 8192. Assign a higher priority to your faster circuits; give your lowest priority to <code>DEFAULTNET</code>.</p>

In This Section ...	Set This Parameter ...	To ...
NETWORK (optional)	LMID (required)	Map the specified machine to one of the entries in the MACHINES section.
	NADDR= <i>string</i> (required)	Specify the listening address for the BRIDGE process on this LMID. There are four valid formats for specifying this network address. See the NETWORK section of UBBCONFIG(5) for details.
	NLSADDR= <i>string</i> (required)	Specify the network address for the tlisten process on this LMID. Valid formats are the same as the valid formats for NADDR.
	NETGROUP= <i>string</i> (optional)	Specify a NETWORK group name. The value of <i>string</i> must be a group name specified in the NETGROUPS section. The default is DEFAULTNET.

How Data Moves Over a Network

In a distributed application, data is sent across the network as follows:

- *At the sending end:* The BRIDGE sends a message to *destination_machine* by writing the message to a virtual circuit and delegating, to the operating system, responsibility for sending it. The operating system retains a copy of every pending message. If a network error occurs, however, pending messages are lost.
- *At the receiving end:* The BRIDGE process listens on a particular network address for incoming messages.

How Data Moves Over Parallel Networks

In a distributed application there are several advantages to using parallel data circuits for sending data across the network:

- By listening at more than one address, the `BRIDGE` achieves higher availability.
- By sending data simultaneously on parallel data circuits, the `BRIDGE` can achieve a higher throughput, if the network was the limiting factor before.
- When you configure parallel data circuits, the software does not necessarily fail to deliver a message if the original destination circuit is busy. The system attempts to schedule traffic over the circuit with the highest network group number (`NETGRPNO`). If this circuit is busy, the traffic is automatically scheduled over the circuit with the next (that is, the second highest) network group number. When all circuits are busy, data is queued until a circuit is available.

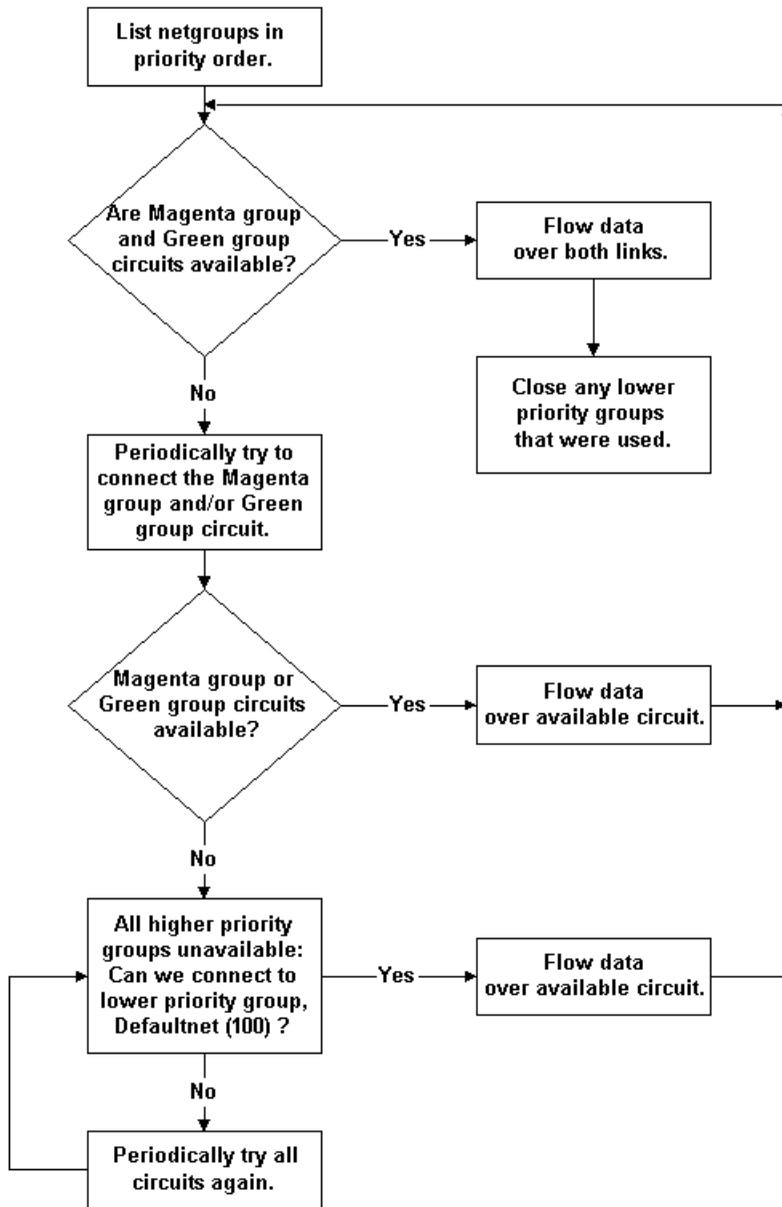
Before making a decision to use parallel data circuits, however, you should determine whether it will be important, in your application, for messages to be kept in sequence. The system guarantees that conversational messages are kept in the correct sequence by binding the conversation connection to one particular data circuit.

If your application will require all messages to be kept in sequence, you must program the application to keep track of the sequence for nonconversational messages. If you are using this approach, you may not want to configure parallel data circuits.

The following figure describes how data flows when one machine tries to contact another. The figure is based on a sample scenario involving two machines: machine A and machine B. First, the `BRIDGE` identifies the network groups that are common to both machines: the `MAGENTA_GROUP`, the `GREEN_GROUP`, and the `DEFAULTNET`.

Data flows in parallel on network groups with the same priority (that is, groups for which the same value is assigned to the `NETPRIO` parameter). Network groups with different priorities are used for failover.

Figure 8-1 Flow of Data over the BRIDGE



Example of a Network Configuration for a Simple Distributed Application

The following example shows how to configure a simple network.

```
# The following configuration file excerpt shows a NETWORK
# section for a 2-site configuration.

*NETWORK
  SITE1  NADDR="//mach1:80952"
         NLSADDR="//mach1:serve"
#
  SITE2  NADDR="//mach386:80952"
         NLSADDR="//mach386:serve"
```

How Failover and Failback Work in Scheduling Network Data

Data flows over the highest available priority circuit. If all network groups have the same priority, data travels over all networks simultaneously. If all circuits at the current priority fail, data is sent over the next lower priority circuit. This process is called *failover*. When failover occurs, the failed connections are retried periodically.

When higher priority network connections are reestablished, *failback* occurs and no further data is scheduled for the lower priority connection. The lower priority connection is disconnected in an orderly fashion.

If attempts to connect to all network addresses have been made and have failed, new attempts to connect are made the next time application or system data needs to be sent between machines.

Example Configuration of Multiple Netgroups

The hypothetical First State Bank has a network of five machines (A-E). These machines are configured in four network groups and each machine is used in two or three groups.

Note: The hardware and system software prerequisites for configuring multiple network groups (NETGROUPS) are beyond the scope of this document. For example, machines are frequently required to belong to more than one physical network. Each TCP/IP symbolic address must be identified in the `/etc/hosts` file or in the DNS (Domain Name Services).

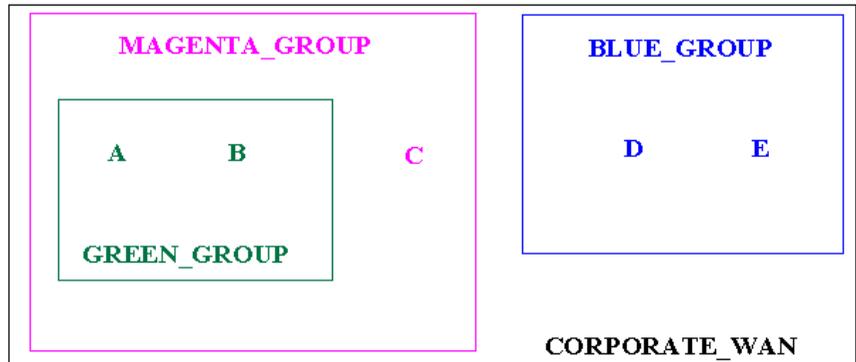
In the following example, it is assumed that in addresses written in the form `//A_CORPORATE:5345`, the string `A_CORPORATE` is specified in the `/etc/hosts` file or in DNS.

The four groups in the First State Bank network include:

- `DEFAULTNET` (the default network, which is the corporate WAN)
- `MAGENTA_GROUP` (a LAN)
- `BLUE_GROUP` (a LAN)
- `GREEN_GROUP` (a private LAN that provides high-speed, fiber, point-to-point links between member machines)

All machines belong to `DEFAULTNET` (the corporate WAN). In addition, each machine is associated with either the `MAGENTA_GROUP` or the `BLUE_GROUP`. Finally, some machines in the `MAGENTA_GROUP` also belong to the `GREEN_GROUP`. The following diagram illustrates group assignments for the network.

Figure 8-2 Example Network Groups



In this example, machines A and B have addresses for the following:

- DEFAULTNET (the corporate WAN)
- MAGENTA_GROUP (LAN)
- GREEN_GROUP (LAN)

Machine C has addresses for the following:

- DEFAULTNET (the corporate WAN)
- MAGENTA_GROUP (LAN)

Machines D and E have addresses for the following:

- DEFAULTNET (the corporate WAN)
- BLUE_GROUP (LAN)

Because the local area networks are not routed to all locations, machine D (in the BLUE_GROUP LAN) may contact machine A (in the GREEN_GROUP LAN) only by using the single address they have in common: the corporate WAN network address.

Configuration File for the Sample Network

To set up the configuration described in the preceding section, the First State Bank administrator defines each group in the `NETGROUPS` and `NETWORK` sections of the `UBBCONFIG` file as follows:

```
*NETGROUPS

DEFAULTNET      NETGRPNO = 0           NETPRIO = 100 #default
BLUE_GROUP      NETGRPNO = 9           NETPRIO = 200
MAGENTA_GROUP   NETGRPNO = 125        NETPRIO = 200
GREEN_GROUP     NETGRPNO = 13         NETPRIO = 300

*NETWORK

A      NETGROUP=DEFAULTNET      NADDR="//A_CORPORATE:5723"
A      NETGROUP=MAGENTA_GROUP   NADDR="//A_MAGENTA:5724"
A      NETGROUP=GREEN_GROUP     NADDR="//A_GREEN:5725"

B      NETGROUP=DEFAULTNET      NADDR="//B_CORPORATE:5723"
B      NETGROUP=MAGENTA_GROUP   NADDR="//B_MAGENTA:5724"
B      NETGROUP=GREEN_GROUP     NADDR="//B_GREEN:5725"

C      NETGROUP=DEFAULTNET      NADDR="//C_CORPORATE:5723"
C      NETGROUP=MAGENTA_GROUP   NADDR="//C_MAGENTA:5724"

D      NETGROUP=DEFAULTNET      NADDR="//D_CORPORATE:5723"
D      NETGROUP=BLUE_GROUP      NADDR="//D_BLUE:5726"

E      NETGROUP=DEFAULTNET      NADDR="//E_CORPORATE:5723"
E      NETGROUP=BLUE_GROUP      NADDR="//E_BLUE:5726"
```

Assigning Priorities for Each Network Group

Assigning priorities appropriately for each `NETGROUP` enables you to maximize the capability of network `BRIDGE` processes. When determining `NETGROUP` priorities, keep in mind the following considerations:

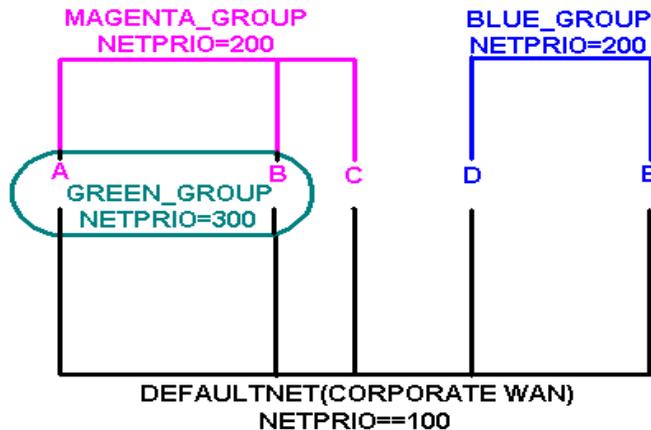
- Data flows over only the highest available priority circuit.
- If all network groups have the same priority, data travels over all circuits simultaneously.

- If *all* circuits at the current priority fail, data is sent over the next lower priority circuit.
- When a higher priority circuit becomes available, data flows over it.
- All unavailable higher priority circuits are retried periodically.
- After connections to all network addresses have been tried and have failed, connections are tried again the next time data needs to be sent between machines.
- The default value of `NETPRIO` is 100.

Example Assignment of Priorities to Network Groups

The following diagram shows how the First State Bank administrator assigns priorities to the available network groups.

Figure 8-3 Assigning Priorities to Network Groups



The following priorities are assigned:

- `BLUE_GROUP=200`
- `DEFAULTNET=100`
- `GREEN_GROUP=300`
- `MAGENTA_GROUP=200`

Example NETGROUP and NETWORK Sections

The lowest priority among network groups is reserved for the default network group, that is, the group that is not used unless all others are unavailable. Therefore, if you want to limit the use of a particular network, such as a satellite link for which per-minute fees are incurred, designate that network as the default network group.

You can assign a network priority to the default network group by setting the `NETPRIO` parameter for `DEFAULTNET` just as you do for any other group. If you do not specify a priority for `DEFAULTNET`, a default of 100 is used, as shown in the following example.

```
*NETGROUP
DEFAULTNET NETGRPNO = 0 NETPRIO = 100
```

For `DEFAULTNET`, the value of the network group number (`NETGRPNO`) must be zero; any other number is invalid. The value of `NETGRPNO` must be *unique* for each entry.

On the other hand, the *same* value of `NETPRIO` may be assigned to multiple network groups. For example, in the First State Bank configuration file, the same network priority (`NETPRIO=200`) is assigned to both the `MAGENTA_GROUP` and the `GREEN_GROUP`.

Each network address (`NETWORK`) is associated by default with the `DEFAULTNET` network group. This parameter may be specified explicitly for either of two reasons: to maintain uniformity among entries, or to associate the network address being defined with a second network group.

```
*NETWORK
D          NETGROUP=BLUE_GROUP NADDR="//D_BLUE:5726"
```


9 About Workstation Clients

- What Is the Workstation Component
- Sample Application with Four Workstation Clients
- How the Workstation Client Connects to an Application

What Is the Workstation Component

The Workstation component of the BEA Tuxedo system allows application clients to reside on a machine that does not have a full server-side installation, that is, a machine that does not support any administration or application servers. All communication between the client and the application servers takes place over the network.

A Workstation client process can run on a Windows 95, Windows 98, Windows NT, or UNIX platform. The client has access to the ATMI. The networking behind requests is transparent to the user. The Workstation client registers with the system through a Workstation Handler (WSH) and has access to the same capabilities as a native client.

All communication between a Workstation client and application server is done through a Workstation Handler (WSH) process.

Workstation clients can perform almost all the same functions that can be performed by network clients. They can, for example:

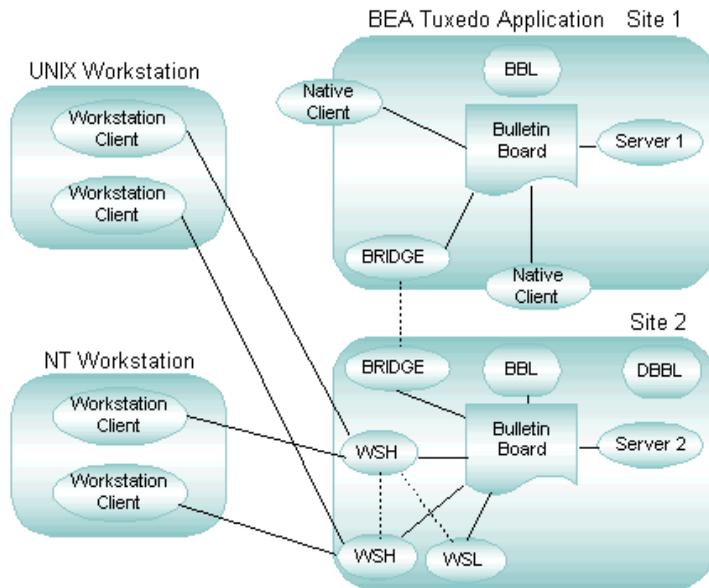
- Send and receive messages

- Begin, end, or commit transactions
- Send and receive unsolicited messages
- Take full advantage of any security mechanism offered to BEA Tuxedo clients

Sample Application with Four Workstation Clients

The following figure shows an example of an application with four Workstation clients.

Figure 9-1 Bank Application with Four Workstation Clients



Two workstation clients are running on a UNIX system; another two Workstation clients, on Windows NT. All workstation clients initially joined the application through the Workstation Listener (WLS), which delegates subsequent communication

to a Workstation Handler. This process differs from the process that occurs when native clients join an application: in the latter case, the native clients attach directly to the bulletin board upon joining.

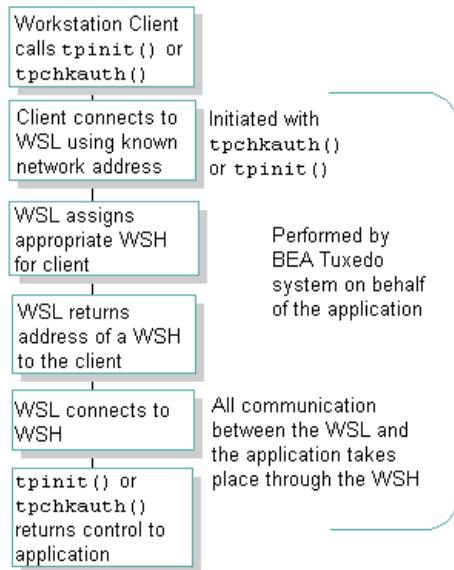
Administrative servers and application servers are located on `SITE1` and `SITE2`. Any service request by a Workstation client to the application is sent over the network to the WSH. This process forwards the request to the appropriate server, gets a reply from the server, and sends the reply to the Workstation client.

Note: The term *resource manager* refers to an implementation of the XA standard interfaces that provides transaction capabilities and permanence of actions for a BEA Tuxedo application. The most common example of a resource manager is a database. A resource manager is accessed and controlled within a global transaction.

Because the application is distributed across two machines in this example, it is running in `MP` mode. The Workstation client sends a request to one Workstation Handler, the Workstation Handler forwards the request to a `BRIDGE` process, and the `BRIDGE` process, in turn, forwards the request to the correct machine.

How the Workstation Client Connects to an Application

The following flowchart shows how a workstation client connects to an application.



The client connects to the WSL process using a known network address. The process for establishing this connection is initiated when the client calls `tpchkauth()` or `tpinit()`. The WSL returns the address of a WSH to the client, and then notifies the Workstation Handler process of the connection request. The WSC connects to the WSH. All further communication between the WSC and the application takes place through the WSH.

10 Setting Up Workstation Clients

- Defining Workstation Clients
- Specifying the Maximum Number of Workstation Clients
- Defining a Workstation Listener (WSL) as a Server
- Detecting Network Failures
- Sample Configuration File that Supports Workstation Clients

Defining Workstation Clients

Before a Workstation client can join a BEA Tuxedo application, the application environment must be prepared to accommodate it. The BEA Tuxedo system provides the variables described in the following table for setting up your environment. Two (`TUXDIR` and `WSNADDR`) are required; the rest are optional. Defaults are available for all parameters except `WSENVFILE`.

To specify . . .	Set This Environment Variable . . .
The application password. (Useful only for applications in which security is implemented through password usage.) Clients that run from scripts can get the application password from this variable.	<code>APP_PW</code> (optional)

10 Setting Up Workstation Clients

To specify . . .	Set This Environment Variable . . .
The maximum number of significant bits of the encryption key for link-level encryption. Value can be 0 (if no encryption is used), or 40, 56, or 128 (if the number specified is the number of significant bits in the encryption key).	TMMAXENCRYPTBITS (optional)
The minimum number of significant bits of the encryption key for link-level encryption. Value can be 0 (if no encryption is used), or 40, 56, or 128 (if the number specified is the number of significant bits in the encryption key).	TMMINENCRYPTBITS (optional)
The directory in which replies are stored when the WSRPLYMAX limit has been reached. The default is the working directory.	TMPDIR (optional)
The location of the BEA Tuxedo system software on this workstation. <i>The client cannot connect unless this environment variable is set.</i>	TUXDIR (required)
The network device to be used. The default is an empty string.	WSDEVICE (optional)
The name of the file in which all environment variables may be set. There is no default for this variable.	WSENVFILE (optional)
The network address used by the workstation client when connecting to the workstation listener or workstation handler. This variable, along with the WSFRANGE variable, determines the range of TCP/IP ports to which a workstation client attempts to bind before making an outbound connection. This address must be a TCP/IP address	WSFADDR (optional)
The range of TCP/IP ports to which a workstation client process attempts to bind before making an outbound connection. The WSFADDR parameter specifies the base address of the range.	WSFRANGE (optional)
A list of one or more network addresses of the WSL that the client wants to contact. This address must match the address of a WSL process in the application configuration file.	WSNADDR (required)
The amount of core memory to be used for buffering application replies. The default is 256,000 bytes.	WSRPLYMAX (optional)

To specify . . .	Set This Environment Variable . . .
The machine type. If the value of <code>WSTYPE</code> matches the value of <code>TYPE</code> in the configuration file for the WSL machine, no encoding/decoding is performed. The default is the empty string.	<code>WSTYPE</code> (optional)

Specifying the Maximum Number of Workstation Clients

To enable Workstation clients to join an application, you must specify the `MAXWSCLIENTS` parameter in the `MACHINES` section of the `UBBCONFIG` file.

`MAXWSCLIENTS` is the only parameter that has special significance for the Workstation feature. `MAXWSCLIENTS` tells the BEA Tuxedo system at boot time how many *accesser slots* to reserve exclusively for Workstation clients. For native clients, each accesser slot requires one semaphore. However, the Workstation handler process (executing on the native platform on behalf of Workstation clients) multiplexes Workstation client accesses through a single accesser slot and, therefore, requires only one semaphore. This capability is an additional benefit of the Workstation component. By putting more clients on workstations instead of on the native platform, an application reduces its IPC resource requirements.

`MAXWSCLIENTS` takes its specified number of accesser slots from the total set in `MAXACCESSERS`. This is important to remember when specifying `MAXWSCLIENTS`; enough slots must be left to accommodate native clients as well as servers. If you specify a value for `MAXWSCLIENTS` greater than that of `MAXACCESSERS`, native clients and servers fail at `tpinit()` time. The following table describes the `MAXWSCLIENTS` parameter.

Parameter	Description
MAXWSCLIENTS	<p>Specifies the maximum number of WSCs that may connect to a machine.</p> <p>The syntax is <code>MAXWSCLIENTS=number</code>. The default is 0.</p> <p>If <code>MAXWSCLIENTS</code> is not specified, WSCs may not connect to the machine being described.</p>

Defining a Workstation Listener (WSL) as a Server

Workstation clients access your application through a WSL process and one or more WSH processes. The WSL can support multiple workstation clients. It acts as the single point of contact for all the workstation clients connected to your application at the network address specified on the WSL command line. The listener schedules work for one or more workstation handler processes.

A WSH process acts as a surrogate within the administrative domain of your application for clients on remote workstations. The WSH uses a multiplexing scheme to support multiple Workstation clients concurrently.

To join Workstation clients to an application, you must specify the Workstation Listener (WSL) processes in the `SERVERS` section of the `UBBCONFIG` file. Use the same syntax you use to specify a server.

Passing Information to a WSL Process

To pass information to a WSL process, you can use the command-line option string, `CLOPT`. The format of the `CLOPT` parameter is as follows.

```
CLOPT="[ -A ] [servopts_options] -- -n netaddr [-d device]
        [-w WSHname][ -t timeout_factor ][ -T Client_timeout ]
        [-m minh][ -M maxh ][ -x mpx_factor ]
```

```
[ -p minwshport ][ -P maxwshport ]  
[ -I init_timeout ][ -c compression_threshold ]  
[ -k compression_threshold ]  
[ -z bits ][ -Z bits ][ -H external_netaddr ]  
[ -N network_timeout ][ -K { client | handler | both | none } ] "
```

The `-A` option requests that the WSL offer all its services when it is booted. This option is included by default, but it is shown here to emphasize the distinction between system-supplied servers and application servers. When application servers are booted, they sometimes offer only a subset of their available services.

The double-dash (`--`) marks the beginning of a list of parameters that is passed to the WSL after it has been booted.

Using Command-line Options Set with CLOPT

You can specify any of the following command-line options in the CLOPT string after the double-dash string (`--`).

Note: For a complete list of the CLOPT command-line options, see `servopts(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*.

Use This Command-line Option . . .	To specify . . .
<code>-n netaddr</code> (required)	The network address used by WSCs to contact the listener. The WSC must set the appropriate environment variable (WSNADDR) to the value specified after <code>-n</code> .
<code>[-d device]</code> (required for some transport interfaces)	Specify the network device name. This is an optional parameter because only some transport interfaces require it. Sockets, for example, does not require this parameter.

Use This Command-line Option . . .	To specify . . .
<code>[-t <i>timeout</i>]</code>	<p>The amount of time to allow for a client to connect to the WSH.</p> <p>To calculate the total amount of time to allow for this purpose, the system multiplies the value of <code>timeout</code> by the value of the <code>SCANUNIT</code> parameter.</p> <p>The default is 3 in a nonsecure application, and 6 in a secure application. In this context we refer to an application as secure if one of the following parameters is set:</p> <ul style="list-style-type: none">■ <code>USER_AUTH</code>■ <code>ACL</code>■ <code>MANDATORY_ACL</code>■ <code>APP_PW</code>
<code>[-w <i>name</i>]</code>	<p>The name of the WSH process that should be booted for this listener. The default is <code>WSH</code>, which is the name of the handler provided. If another handler process is built with the <code>buildwsh(1)</code> command, that name is specified here.</p>
<code>[-m <i>number</i>]</code>	<p>The minimum number of handlers that should be booted and always available. The default is 0.</p>
<code>[-M <i>number</i>]</code>	<p>The maximum number of handlers that can be booted. The default is the value of <code>MAXWSCLIENTS</code> for the machine being configured, divided by the multiplexing value (specified with <code>-x</code>).</p>
<code>[-x <i>number</i>]</code>	<p>The maximum number of clients that a WSH can multiplex at one time. The value must be greater than 0. The default is 10.</p>
<code>[-T <i>client_timeout</i>]</code>	<p>The amount of time (in minutes) that a client can remain idle without being disconnected. If a client does not make any requests within this time period, the WSH disconnects the client. If this argument is not given or is set to 0, the time-out is infinite.</p>
<code>[-p <i>minwshport</i>]</code> and <code>[-P <i>maxwshport</i>]</code>	<p>The range for port numbers available for use by WSHs associated with this listener server. Port numbers must fall in the range between 0 and 65535. The default is 2048 for <code>minwshport</code> and 65535 for <code>maxwshport</code>.</p>

Use This Command-line Option . . .	To specify . . .
<code>[-z]</code> and <code>[-Z]</code>	The range of bits that can be used, on the WSL side, for link-level encryption: use <code>-z</code> to specify the minimum number of bits, and <code>-Z</code> to specify the maximum number of bits.
<code>[-N network_timeout]</code>	The minimum amount of time (in seconds) that a workstation client is allowed to wait to receive a response from the WSL/WSH. A value of 0 indicates no network time-out.
<code>[-K {client handler both none}]</code>	The viability of a network connection between the workstation handler and a workstation client if no traffic has occurred over that connection within a specified period of time.

See Also

- `servopts(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*

Detecting Network Failures

The Workstation component provides two administrative options to WSL that enable you to avoid hanging indefinitely when a network connection is lost. Specifically, these options allow you to:

- Check client connections periodically (*keep-alive option*)
- Limit the amount of time that a client waits for a response from a WSH before dropping the connection to that WSH (*network timeout option*)

Using the Keep-alive Option

Keep-alive is a networking operation that periodically checks the viability of a network connection between the Workstation handler and a Workstation client if no traffic has occurred over that connection within a specified period of time.

You can request the keep-alive option by adding the `-K` option to the `WSL CLOPT` entry in the `SERVERS` section of the `UBBCONFIG` file. The `-K` option accepts the following arguments: `client`, `handler`, `both`, or `none`.

Use this option	To
<code>-K client</code>	Generate keep-alive messages from the client machines. If the keep-alive message is not acknowledged, the client machine considers the network down. Subsequent ATMI calls fail with a <code>tperrno</code> of <code>TPESYSTEM</code> .
<code>-K handler</code>	Generate keep-alive messages from the handler machine. If the keep-alive message is not acknowledged, the handler machine considers the network down. The handler then cleans up the entry associated with the client that does not respond. This reduces the possibility that the handler will exhaust the number of clients that a workstation can multiplex at one time (as specified by <code>-x</code>) with stale clients.
<code>-K both</code>	Generate keep-alive message from both the client and handler machines. The availability and timeout thresholds for this component are determined by tunable parameters in the operating system.
<code>-K none</code>	Turn off the keep-alive option. Using this setting has the same effect as not specifying <code>-K</code> at all.

Your entry in the `UBBCONFIG` file should look like the following.

```
WSL SRVGRP="WSLGRP" SRVID=1000 RESTART=Y GRACE=0  
CLOPT="-A -- -n //ws.beasys.com:5120 -d /dev/tcp -K both"
```

In the example, `-K` turns on keep-alive checking on both the Workstation client and the server.

For details about the format of a `WSL` entry in `UBBCONFIG`, see `WSL(5)` in *BEA Tuxedo File Formats and Data Descriptions Reference*.

Note: Any timeout period that you specify applies to the entire system. If you specify a timeout with one application in mind, and you later change the amount of time specified, all applications that use keep-alive are also affected.

Limitations When Using the Keep-alive Option

The keep-alive option is supported only on platforms for which the BEA Tuxedo system uses sockets:

- AIX
- Tru64 UNIX
- HP UX
- Windows

You cannot use this option on any other platform. The BEA Tuxedo system lets you specify the `-k` option for any server machine, but it will not execute it properly on any platform other than those previously listed. If you try to perform a keep-alive operation on any other platform, your attempt fails and a message is written to the `userlog` (once per process for the `WSH`). Processing continues normally.

Note: The keep-alive operation works only for TCP/IP communications.

Using the Network Timeout Option

Network timeout is an option that lets you decide how long you are willing to wait for an operation in a Workstation client before your request for that operation is canceled (timed out) on a network.

You can request the network timeout function through an administrative option to the `WSL: -N`. The `-N` option uses a network timeout to receive data in the Workstation client.

How Network Timeout Works

The network timeout option establishes a waiting period (in seconds) for any BEA Tuxedo operation in the Workstation client that receives data from the network. If the period is exceeded, the operation fails and the client is disconnected from the application. A value of 0 (the default) indicates no timeout.

Note: Setting this value too low may cause too many disconnects.

Each ATMI function returns an error whenever a timeout occurs. When a link times out, the application is notified. An existing error code is used. (Additional error detail on the specific error can be retrieved by a call to `tperrordetail(3c)`.) Once a network timeout occurs, the status of outstanding operations is in doubt: transactions cannot be completed; incoming replies can be lost, and so on. The only safe action is to terminate the connection to the application by doing the equivalent of a `tpterm(3c)` without communicating with the `WSH`.

By the time the operation returns, the client is no longer part of the BEA Tuxedo application. The client can rejoin the application in either of two ways:

- By calling `tpinit(3c)`
- By using an implicit connection (if security is not configured)

Limitations When Using the Network Timeout Option

- Network timeout does not handle network send operations.
- If the value of the network timeout is less than the value of the transaction timeout or the block time, then the client may be disconnected before the processing of the request is complete.
- Network timeout disconnects the Workstation client after timeout even though the connection may still be viable.

Setting the Network Timeout Option

To use the network timeout option in your BEA Tuxedo application, add the `-N` option to the `WSL CLOPT` argument.

Sample Configuration File that Supports Workstation Clients

The following excerpt from a sample configuration file shows how you can add the Workstation component to the `bankapp` application. It contains modifications to the `MACHINES` and `SERVERS` sections.

Listing 10-1 Sample UBBCONFIG File Supporting Workstation Clients

```
*MACHINES
SITE1
    ...
    MAXWSCLIENTS=150
    ...
SITE2
```

```
...
MAXWSCLIENTS=0
...

*SERVERS
...
WSL SRVGRP="BANKB1" SRVID=500 RESTART=Y
      CLOPT="-A -- -n //ws.beasys.com:5120 -m 5 -M 30 -x 5"
...

```

Modifying the MACHINES and SERVERS Sections

The following changes are shown in the MACHINES and SERVERS sections:

- In the MACHINES section, the default for MAXWSCLIENTS is overridden in the entries for two sites. For SITE1, the default is raised to 150, while it is lowered to 0 for SITE2, because no Workstation clients will be connected to that site.
- In the SERVERS section, a WSL process is specified for group BANKB1. The WSL has a server ID of 500 and it is marked as restartable.
- The command-line options show the following:
 - The WSL will advertise all of its services (-A).
 - The WSL will listen at network address //ws.beasys.com:5120 (-n).
 - A minimum of 5 WSHs will be booted (-m).
 - A maximum of 30 WSHs will be booted (-M).
 - Each handler will be allowed a maximum of 5 clients connected at any one time (-x).