BEA WebLogic Enterprise

Administration Guide

WebLogic Enterprise 5.1
Document Edition 5.1
May 2000

Administration Guide

Document Edition	Date	Software Version
5.1	May 2000	BEA WebLogic Enterprise 5.1

Contents

About This Document

This document describes how to administer BEA WebLogic Enterprise™ and BEA Tuxedo® systems.

This document covers the following topics:

- Introduces the administration tasks.
- Identifies the administration tools that are part of the BEA WebLogic Enterprise and Tuxedo systems.
- Details the application, machine, group, server, services, interfaces, routing, and network parameters in an application's `UBBCONFIG` configuration file.
- Explains how to start and shut down applications.
- Explains how to distribute applications.
- Explains how to build networked applications.
- Explains how to configure transactions.
- Explains how to manage Interface Repositories. This chapter is specific to the BEA WebLogic Enterprise system.
- Explains how to configure multiple domains. This chapter is specific to the BEA WebLogic Enterprise system.
- Explains how to manage multiple domains. This chapter is specific to the BEA Tuxedo system.
- Explains how to manage Workstation clients. This chapter is specific to the BEA Tuxedo system.
- Explains how to manage remote BEA WebLogic Enterprise client applications. This chapter is specific to the BEA WebLogic Enterprise system.

- Explains how to manage queued messages. This chapter is specific to the BEA Tuxedo system.
- Explains how to implement application security. The access control list (ACL) mechanism used in the BEA Tuxedo system is not present in the BEA WebLogic Enterprise system. Therefore, the ACL section of this chapter is specific to BEA Tuxedo systems. Other sections of this security chapter, however, are relevant to the BEA WebLogic Enterprise administrator.
- Explains how to monitor a running system.
- Explains how to monitor log files.
- Explains how to tune applications.
- Explains how to migrate applications.
- Explains how to modify systems dynamically.
- Explains how to reconfigure applications dynamically.
- Explains how to use Event Broker. This chapter is specific to the BEA Tuxedo system.
- Explains how to troubleshoot problems.

What You Need to Know

This document is intended for administrators who configure operational parameters that support mission-critical BEA WebLogic Enterprise and BEA Tuxedo systems.

e-docs Web Site

The BEA WebLogic Enterprise product documentation is available on the BEA Systems, Inc. corporate Web site. From the BEA Home page, click the Product Documentation button or go directly to the “e-docs” Product Documentation page at http://e-docs.bea.com.

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser.

A PDF version of this document is available on the WebLogic Enterprise documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Enterprise documentation Home page, click the PDF Files button, and select the document you want to print.

If you do not have Adobe Acrobat Reader installed, you can download it for free from the Adobe Web site at http://www.adobe.com/.

Related Information

For more information about CORBA, Java 2 Enterprise Edition (J2EE), BEA Tuxedo, distributed object computing, transaction processing, C++ programming, and Java programming, see the *BEA WebLogic Enterprise Bibliography* in the WebLogic Enterprise online documentation.

Contact Us!

Your feedback on the BEA WebLogic Enterprise documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA Systems, Inc. professionals who create and update the WebLogic Enterprise documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Enterprise 5.1 release.

If you have any questions about this version of BEA WebLogic Enterprise, or if you have problems installing and running BEA WebLogic Enterprise, contact BEA Customer Support through BEA WebSUPPORT at www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.

Convention	Item
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Examples*: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

Convention	Item
|	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

`...`	Indicates one of the following in a command line:
	■ That an argument can be repeated several times in a command line
	■ That the statement omits additional optional arguments
	■ That you can enter additional parameters, values, or other information
	The ellipsis itself should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

CHAPTER

1 Introduction to Administration

As the administrator of your organization’s computing applications, you are responsible for setting up and running a system that is critical to your corporate mission. You must plan how to maximize the performance and reliability of your BEA WebLogic Enterprise or BEA Tuxedo systems, and then make it happen.

This topic includes the following sections:

- The Administrator’s Job
- Roadmap for Your Responsibilities
- Planning Your Configuration

The Administrator’s Job

You are the person responsible for configuring and booting an application and then keeping it running smoothly. Your job can be viewed in two phases:

- During the “groundwork phase,” you establish the foundation of your application by planning, designing, installing, and configuring your application with the BEA WebLogic Enterprise or BEA Tuxedo system. You also select a security scheme for your application.

  Most of the work you do during this phase is necessary only once. The exception to this rule is the configuration work: the BEA WebLogic Enterprise or BEA

Tuxedo system allows you to reconfigure your application whenever necessary to maximize performance and reliability.

- During the “operational phase,” you run the application, monitor it and reconfigure it when necessary. You also diagnose and correct run-time problems.

The remainder of this chapter lists the specific tasks you need to do during each phase.

The Groundwork Phase

During the this phase, you must do the tasks listed in Table 1-1.

Table 1-1 Groundwork Phase Tasks

Plan		Collect information from the application designers, the programmers, and the business that will use the application. Use this information to configure your system.
Install		Set up your environment (including hardware and software), and install the BEA WebLogic Enterprise system and the application.
Configure	Your system	Set the parameters provided by the BEA WebLogic Enterprise system that govern how the components of your application will be used.
	Transactions	Add transactions functionality to your definitions of domains, machines, groups, interfaces, services, and any other required components of your application.
Implement	Security	Select and implement one or more methods provided by the BEA WebLogic Enterprise system for protecting your application and data.

Depending on your application, you may also need to set up the tasks listed in Table 1-2.

Table 1-2 Additional Groundwork Phase Tasks

Distributed applications	Create distributed applications with the routing tools: factory-based routing in BEA WebLogic Enterprise applications and data-dependent routing in BEA Tuxedo applications.
Networked applications	Set up any networked applications.
BEA WebLogic Enterprise remote client applications	To support BEA WebLogic Enterprise remote client applications, configure an Internet Inter-ORB Protocol (IIOP) Listener/Handler and modify the machine configuration.

Note: This guide provides instructions for all the tasks listed in Table 1-3, except installation. For installation instructions, see the BEA WebLogic Enterprise *Installation Guide.*

The Operational Phase

During the this phase, you must do the tasks listed in Table 1-4.

Table 1-3 Operational Phase Tasks

Start up	Boot your application.
Monitor	Log the activities, problems, and performance of your application and analyze the results regularly.
Troubleshoot	Identify and resolve problems as they occur.

Depending on your application, you may also have to do the following:

Table 1-4 Additional Operational Phase Tasks

Tune	Use techniques such as load balancing and prioritizing to maximize the performance of your application.
Migrate	Reassign primary responsibility for your application from your original MASTER machine to an alternate (BACKUP) machine when problems occur on the MASTER.

Table 1-4 Additional Operational Phase Tasks (Continued)

Dynamically modify	Change system parameters and the menu of services offered, when necessary, to meet the evolving needs of your customers.
Dynamically reconfigure	Redefine your application to reflect the addition of a component, such as a new machine or server.

Differences Between the BEA WebLogic Enterprise and BEA Tuxedo Systems

For the BEA WebLogic Enterprise system, the existing BEA Tuxedo administration facilities have been extended to support the administration of applications running within the context of the BEA WebLogic Enterprise Object Request Broker (ORB) and the BEA WebLogic Enterprise TP Framework.

The `UBBCONFIG` configuration file for BEA WebLogic Enterprise systems includes the following enhancements to support the configuration of client and server applications:

- The `RESOURCES` section is enhanced to provide application-wide defaults for the sizing of Bulletin Board tables.
- The `MACHINES` section is enhanced to allow the specification of processor-specific values for sizing of those tables.
- A new section, `INTERFACES`, is added to allow the specification of information about CORBA interfaces used by the application.
- The `ROUTING` section is enhanced to provide support for a different type of routing criteria used with BEA WebLogic Enterprise systems. Also, existing `ROUTING` sections that specify BEA Tuxedo data-dependent routing parameters continue to work without modification.
- In the BEA Tuxedo system, you configure workstation handlers and listeners for connections from client applications to server applications. From an administrative viewpoint, this task is similar in BEA WebLogic Enterprise systems.

  However, BEA WebLogic Enterprise systems use a different communications protocol to connect remote and foreign clients to BEA WebLogic Enterprise server applications. The protocol is the standard Internet Inter-ORB Protocol

(IIOP). Instead of the BEA Tuxedo Workstation Handler (WSH) process and Workstation Listener (WSL) process, the BEA WebLogic Enterprise system calls its gateway processes the IIOP Handler (ISH) and the IIOP Listener (ISL). This results in a slight syntax difference, `ISL` instead of `WSL`, in the `SERVERS` section of each application's `UBBCONFIG` configuration file.

Overall, the administration tasks for the BEA WebLogic Enterprise and BEA Tuxedo systems are similar. There are a few principal differences between the systems, however, as follows:

- In both systems you use a routing criteria to distribute processing to specific server groups. The routing mechanism in a BEA WebLogic Enterprise system is known as factory-based routing. It is fundamentally different than the BEA Tuxedo data-dependent routing mechanism.

  In the BEA Tuxedo system, you can examine any FML field used for a service invocation to determine the data-dependent routing criteria. In BEA WebLogic Enterprise systems, the system designer must personally communicate to you the routing criteria of CORBA interfaces. For BEA WebLogic Enterprise systems, there is no service request message data or associated buffer information available for routing. This occurs because BEA WebLogic Enterprise routing is performed at the factory, not on a method invocation on the target CORBA object.

- You cannot dynamically advertise CORBA interfaces at run time. However, you can suspend or reactivate CORBA interfaces.

- No direct ACL control is provided for CORBA interfaces. No control over servants is provided at the administrative level. In the `UBBCONFIG` configuration file, the `MANDATORY_ACL` parameter to the `SECURITY` parameter is ignored.

Details on these differences and exceptions are provided in subsequent chapters of this document.

Note: The Management Information Base (MIB) defines the set of classes through which the fundamental aspects of an application can be configured and managed. The MIB classes provide an administrative programming interface to the BEA WebLogic Enterprise or BEA Tuxedo system.

The *BEA Tuxedo Reference Manual* includes, in the `TM_MIB`(5) section, reference material about the `T_INTERFACE` MIB class, `T_IFQUEUE` MIB class, and `T_FACTORY` MIB class. Those MIB classes were added for BEA WebLogic Enterprise.

An online version of the *BEA Tuxedo Reference Manual* is available on the BEA WebLogic Enterprise online documentation. At the online documentation Home page, click on Tuxedo Documentation in the left navigation bar. The Tuxedo ATMI topics page is displayed. Then click Reference in the left navigation bar. The BEA Tuxedo Reference Manual page is displayed.

See also the descriptions of the `T_DOMAIN` MIB class, `T_MACHINE` MIB class, `T_SERVER` MIB class, `T_TRANSACTION` MIB class, and `T_ROUTING` MIB class. These MIB classes were enhanced for BEA WebLogic Enterprise.

Roadmap for Your Responsibilities

At the beginning of this chapter, we summarized your job responsibilities in two phases. For software descriptions and procedures that help you perform your work, refer to the appropriate documentation, as follows:

- During the groundwork phase, see the *Installation Guide* and Chapters 3 through 15 of this document.
- During the operational phase, see Chapters 16 through 23 of this document.

Planning Your Configuration

As an administrator, you need to work with your system designers and application designers to understand how the administrative configuration of your application can support the requirements for it. In addition, you need to know the requirements of your customer: the business unit using the new software.

Before you can start configuring your system, you need answers to questions about the design of your application and about the server applications developed from that design, as defined in the following section.

Questions About the Design

The following questions may help you start the planning process:

- How many machines will be used?
- Will client applications reside on machines that are remote from the server applications?
- Which CORBA interfaces will your BEA WebLogic Enterprise client or server application use?
- What resource managers will the application use and where will they be located?
- What “open” strings will the resource managers need?
- What setup information will be needed for a database resource manager?
- Will transactions be distributed?
- What buffer types will be used?
- Will data be distributed across machines?
- Will factory-based routing be used in your BEA WebLogic Enterprise application?
- Will data-dependent routing be used in your BEA Tuxedo application?
- In what order of priority should interfaces in BEA WebLogic Enterprise applications or BEA Tuxedo services be available?
- For BEA WebLogic Enterprise systems, will the domain need an Interface Repository (IR) database? If so, will the domain benefit from having IR replicas, and how many IR server applications should be defined?
- What are the reliability requirements? Will redundant listener and handler ports be needed? Will replicated server applications be needed?

Questions About Server Applications

The following questions may help you focus on the issues related to your server application that need to be resolved in your plan:

- What are the names of the BEA WebLogic Enterprise interfaces or BEA Tuxedo services?
- Are there any conversational services (BEA Tuxedo system)?
- What resource managers do they access?
- What buffer types do they use?

As you start putting together a configuration plan, you will discover more questions to which you need answers.

2 Administration Tools

Your BEA WebLogic Enterprise or BEA Tuxedo systems give 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 domain. This chapter describes the menu of administration tools.

All administrative tools can be used to administer BEA WebLogic Enterprise C++ servers and Java servers. C++ servers support CORBA applications. Java servers support both CORBA applications and Enterprise JavaBean (EJB) applications.

This topic includes the following sections:

- Configuration and Run-Time Administration
- BEA Administration Console
- Command-line Interface
- AdminAPI

Configuration and Run-Time Administration

At the highest level, the job of an administrator can be viewed as two broadly defined tasks:

- Configuration—the most important (and complicated) part of setting up your system before booting your application
- Run-time administration—the set of tasks that are performed on an application that has been booted

The BEA WebLogic Enterprise and BEA Tuxedo systems offer three tools for both of these tasks:

- BEA Administration Console
- Command-line Interface

  Note: You can enter administration commands either at a shell prompt on any supported UNIX platform, or from an MS-DOS command line on a Windows NT platform.

- AdminAPI

This chapter describes how these tools can be used to configure an application and to administer a running system.

Tools for Configuration

Because the BEA WebLogic Enterprise and BEA Tuxedo systems offer great flexibility and many options to application designers and programmers, no two applications are alike. An application, for example, may be small and simple (a single client and server running on one machine), or complex enough to handle transactions among thousands of clients and servers. For this reason, for every BEA WebLogic Enterprise application being managed, an administrator must provide a file that defines and governs the components of that application.

The components are as follows:

domain

The collection of servers, services, interfaces, machines, and associated resource managers defined by a single `UBBCONFIG` (ASCII) or `TUXCONFIG` (binary) configuration file; a collection of programs that perform a function. A domain represents an administrative set of functionality.

server

A software program (or the hardware on which it runs) in which BEA WebLogic Enterprise interfaces or BEA Tuxedo services offered to your users are stored.

client

A software program that requests services from servers (and sometimes resides on nonserver hardware).

queue
: A set of requests that are submitted to servers in a particular order (which may be determined by the administrator).

service
: A program that takes client requests as input and performs a particular function in response.

interface
: In a BEA WebLogic Enterprise system, a set of operations and attributes. An interface is defined by an application programmer using the Object Management Group Interface Definition Language (OMG IDL). The definition contains operations and attributes that can be used to manipulate a CORBA object.

server group
: A set of interfaces or a logical grouping of servers.

These components (and others, when appropriate) are defined, or configured, in an ASCII file that is referred to, in the BEA WebLogic Enterprise and BEA Tuxedo documentation, as `UBBCONFIG`. The `UBBCONFIG` file may, in fact, be given any filename. When compiled into a binary file, the file is referred to as `TUXCONFIG`. During the groundwork (or setup) phase of administration, the administrator's goal is to create a `TUXCONFIG` file. You have a choice of the following three tools.

If you select the . . .	**You must . . .**
BEA Administration Console	Use a graphical user interface (GUI) to create and edit the `TUXCONFIG` file. For details, see the BEA Administration Console online help.
Command-line interface	1. Edit the `UBBCONFIG` file (an ASCII version of `TUXCONFIG`) with a text editor. 2. Run `tmloadcf` to convert the `UBBCONFIG` file into a `TUXCONFIG` (binary) file. For details about using the command-line interface to perform administrative tasks, see the applicable chapters in this document. For information about the `tmloadcf` command, see Chapter 4, "Starting and Shutting Down Applications." For details about the `tmloadcf` command options, see `tmloadcf`(1) in the *BEA Tuxedo Reference Manual*.

If you select the . . .	You must . . .
AdminAPI	Write a program that modifies the `TUXCONFIG` file for you. For details, see Chapter 21, “Event Broker/Monitor (BEA Tuxedo Systems).”

Tools for Run-Time Administration

With your BEA WebLogic Enterprise or BEA Tuxedo system installed, your client or server application installed, and your `TUXCONFIG` file loaded, you are ready to boot your application. As soon as your application is launched, you must start monitoring its activities and watching for problems—both actual and potential.

When problems occur, you must identify and solve them. If performance is degraded, you may want to do load balancing or prioritize your interfaces or services. If trouble develops on a MASTER machine, you may want to replace it with a designated BACKUP machine.

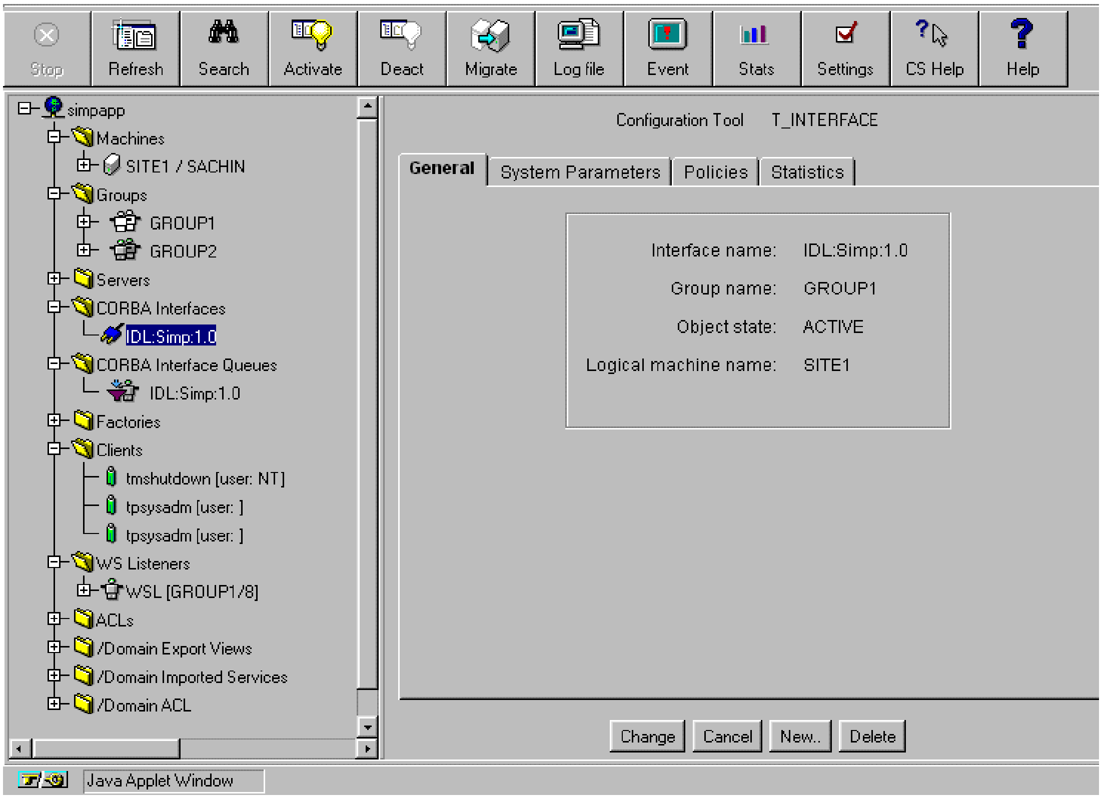
As the processing and resource usage requirements of your application evolve, you may need to add machines, servers, clients, interfaces, services, and so on, to your existing system.

The job of run-time administration encompasses many tasks, from starting and stopping the application, to monitoring activity, troubleshooting problems, and dynamically reconfiguring the application. Again, you have a choice of three tools for performing these tasks: the BEA Administration Console, the command-line interface, and the AdminAPI.

BEA Administration Console

The BEA Administration Console is a graphical user interface that enables you to perform most administration tasks for BEA WebLogic Enterprise and BEA Tuxedo applications. Figure 2-1 shows a sample Administration Console screen.

Figure 2-1 Sample BEA Administration Console Screen



Configuration Tool T_INTERFACE

Interface name: IDL:Simp:1.0
Group name: GROUP1
Object state: ACTIVE
Logical machine name: SITE1

The BEA Administration Console is implemented as a Java applet. You can run the applet on platforms that support a Java-enabled Web browser, such as Netscape 3.01 or higher, or Microsoft Internet Explorer 3.0 or higher.

For the BEA Administration Console startup procedure, see the *Installation Guide*.

For more information about how to use the BEA Administration Console, see the online help.

Command-line Interface

You can use the following commands to administer the BEA WebLogic Enterprise or BEA Tuxedo system. This document provides procedures for administrative tasks that are based on the command-line interface. For details about individual commands, see the *BEA Tuxedo Reference Manual*.

- `tmboot`—activates the application that is referenced in the specified configuration file. Depending on the options used, the entire application or parts of the application are started.
- `tmloadcf`—parses the `UBBCONFIG` file and loads the binary `TUXCONFIG` configuration file.
- `tmunloadcf`—unloads the `TUXCONFIG` configuration file.
- `tmconfig`—dynamically updates and retrieves information about the configuration for a running system.
- `dmadmin`—updates the compiled `DMCONFIG` (binary domain configuration file) while the system is running.
- `tmadmin`—produces information about configuration parameters. Once invoked, you can enter many administrative commands that duplicate the functions of other commands. For example, the `tmadmin shutdown` command is identical to the `tmshutdown` command.
- `tmshutdown`—shuts down a set of specified BEA WebLogic Enterprise or BEA Tuxedo servers, or removes a set of BEA WebLogic Enterprise interfaces or BEA Tuxedo services listed in a configuration file.

AdminAPI

The AdminAPI is an application programming interface (API) for directly accessing and manipulating system settings in the BEA Tuxedo Management Information Bases (MIBs). The advantage of the AdminAPI is that it can be used to automate administrative tasks, such as monitoring log files and dynamically reconfiguring an application, thus eliminating the need for human intervention. This advantage can be crucial in mission-critical, real-time applications.

For details about the MIBs, see `ACL_MIB`(5), `APPQ_MIB`(5), `EVENT_MIB`(5), `MIB`(5), `TM_MIB`(5), and `WS_MIB`(5) in the *BEA Tuxedo Reference Manual*.

Note: The *BEA Tuxedo Reference Manual* includes, in the `TM_MIB`(5) section, reference material about the `T_INTERFACE` MIB class, `T_IFQUEUE` MIB class, and `T_FACTORY` MIB class. These MIB classes were added for BEA WebLogic Enterprise.

An online version of the *BEA Tuxedo Reference Manual* is available on the Online Documentation CD. On the CD, click the Reference button from the main menu. Next, click the hyperlink “BEA Tuxedo Manuals.” On the BEA Tuxedo home page, click the hyperlink “Reference Manual: Section 5.”

See also the descriptions of the `T_DOMAIN` MIB class, `T_MACHINE` MIB class, `T_SERVER` MIB class, `T_TRANSACTION` MIB class, and `T_ROUTING` MIB class. These MIB classes were enhanced for BEA WebLogic Enterprise.

CHAPTER

3 Creating a Configuration File

This topic includes the following sections:

- About the Configuration File
- Administrative Requirements and Performance
- Configuring Resources
- Configuring Machines
- Configuring Groups
- Configuring Servers
- Configuring Modules
- Configuring Services (BEA Tuxedo System)
- Configuring Interfaces (BEA WebLogic Enterprise Servers)
- Configuring Routing
- Configuring Network Information

About the Configuration File

The configuration file is the primary way to define the configuration of BEA WebLogic Enterprise applications. It consists of parameters that the BEA WebLogic Enterprise software interprets to create an executable application.

This file is usually created by programmers who develop and build BEA WebLogic Enterprise applications. Administrators modify the configuration file as necessary to satisfy application and system requirements.

Build Environment

In addition to the configuration file, you need the following three basic components to build a BEA WebLogic Enterprise application:

- A server application that performs the operations requested by the client.
- A client application that issues the operation requests to the server application.
- The development commands that you use to build the client and server executables.

Forms of the Configuration File

The configuration file exists in two forms:

`UBBCONFIG`

The `UBBCONFIG` file is an ASCII version of the configuration file. You can create and edit this version with any editor. Sample configuration files are provided with each of the BEA WebLogic Enterprise sample applications, including the simple sample—Simpapp. You must create a `UBBCONFIG` file for each new application. You can use any of the sample `UBBCONFIG` files as a starting point and edit it to meet the requirements of your particular application. The syntax used for entries in the file is described in detail in the `ubbconfig`(5) reference pages in the *BEA Tuxedo Reference Manual*.

`TUXCONFIG`

The `TUXCONFIG` file is a binary version of the configuration file that you generate from the ASCII version using the `tmloadcf`(1) command. You cannot create this file directly; a `UBBCONFIG` file must be created first. You can, however, use the `tmconfig`(1) command to edit many of the parameters in this file while the application is running.

The `TUXCONFIG` file contains information used by `tmboot`(1) to start the servers and initialize the Bulletin Board of a BEA Tuxedo system Bulletin Board instantiation in an orderly sequence. The `tmadmin`(1) command line utility uses the configuration file (or a copy of it) to carry out monitoring activities. The `tmshutdown`(1) command references the configuration file for information needed to shut the application down.

Note: When `tmloadcf`(1) is executed, the `TUXCONFIG` environment variable must be set to the full pathname of the device or system file where `TUXCONFIG` is to be loaded.

Configuration File Content

The configuration file can contain up to ten specification sections and many different parameters. Lines beginning with an asterisk (*) indicate the beginning of a specification section and the name of the section immediately follows the asterisk.

Section Names and Functions

Supported section names and their functions are as follows:

- `RESOURCES`—specifies system-wide resources, such as the number of machines, servers, server groups, services, and network groups that can exist within a service area. (Required)
- `MACHINES`—specifies the logical names for physical machines for the configuration and parameters specific to a given machine. (Required)
- `GROUPS`—defines all application server groups by group name, logical machine, and group number. (Required)
- `SERVERS`—specifies server processes to be booted in the application. (Optional)

Note: While the SERVERS section is not required, an application without this section has no application servers and so little functionality that it is not practical to leave this section out. The following warning is issued if this section is not supplied: `Missing Servers Section`.

- SERVICES—defines parameters for BEA Tuxedo services used by the application. (Required)
- INTERFACES—defines application-wide, default parameters for CORBA interfaces used by the application. (Optional)
- ROUTING—defines the routing criteria named in the INTERFACES section for BEA WebLogic Enterprise factory-based routing, or in the SERVICES section for BEA Tuxedo data-dependant routing. (Optional)
- NETGROUPS—specifies the network groups available to an application in a LAN environment. (Optional)
- NETWORK—describes the network configuration for a LAN environment. (Optional)
- JDBCCONNPOOLS—describes the pooling of JDBC connections for Java servers. (Optional)

Each of these topics and the associated parameters are discussed in the following sections of this document. Also, the syntax used for entries in this file is described in detail in the `ubbconfig`(5) reference pages in the *BEA Tuxedo Reference Manual*.

Arrangement of Sections in the Configuration File

These sections must be arranged in the file as follows:

- The RESOURCES and MACHINES sections must appear as the first two sections, in that order.
- The GROUPS section must precede the SERVERS, SERVICES, INTERFACES, and ROUTING sections.
- The NETGROUPS section must precede the NETWORK section.
- The SERVERS section must be precede the JDBCCONNPOOLS section

For all sections except the RESOURCES section you can:

- Specify multiple entries, each with its own selection of parameters.
- Make use of a `DEFAULT` parameter to specify parameters that repeat from one entry to the next. For example, in the `SERVERS` section in Listing 3-1, the default specified for all servers is that if a server crashes, it will be restarted up to 5 times in 24 hours.

Sample UBBCONFIG File

Listing 3-1 shows a basic `UBBCONFIG` file. This is the `UBBCONFIG` file used for the University Basic sample application that is provided with the BEA WebLogic Enterprise software.

This file contains configuration information in four sections: `RESOURCES`, `MACHINES`, `GROUPS`, and `SERVERS`. Each of these sections and the associated parameters are discussed in the following sections of this document. This `UBBCONFIG` file also contains the required `SERVICES` section, but this section contains no information. For more information about the syntax used for entries in the file, see the `ubbconfig`(5) reference pages in the *BEA Tuxedo Reference Manual*.

Listing 3-1 University Basic Sample Application UBBCONFIG File (ubb_b.nt)

```
*RESOURCES
    IPCKEY    55432
    DOMAINID  university
    MASTER    SITE1
    MODEL     SHM
    LDBAL     N
#-----------------------------------------------------------------
*MACHINES

#   Specify the name of your server machine
#
    PCWIZ
        LMID = SITE1

#   Pathname of your copy of this sample application.
#   Must match "APPDIR" in "setenv.cmd"
#
    APPDIR = "C:\MY_APP_DIR\basic"

#   Pathname of the tuxconfig file.
#   Must match "TUXCONFIG" in "setenv.cmd"
#
```

```
    TUXCONFIG="C:\MY_APP_DIR\basic\tuxconfig"

#   Pathname of the BEA WebLogic Enterprise installation.
#   Must match "TUXDIR" in "setenv.cmd"
#

    TUXDIR="C:\wledir"

    MAXWSCLIENTS=10
#-----------------------------------------------------------------
*GROUPS

    SYS_GRP
        LMID    = SITE1
        GRPNO   = 1

    ORA_GRP
        LMID    = SITE1
        GRPNO   = 2
#-----------------------------------------------------------------
*SERVERS

# By default, restart a server if it crashes, up to 5 times in
# 24 hours.
    DEFAULT:
        RESTART = Y
        MAXGEN  = 5

# Start the Tuxedo System Event Broker. This event broker must
# be started before any servers providing the NameManager Service
#
    TMSYSEVT
        SRVGRP  = SYS_GRP
        SRVID   = 1

# Start the NameManager Service (-N option). This name manager
# is being started as a Master (-M option).
#
    TMFFNAME
        SRVGRP  = SYS_GRP
        SRVID   = 2
        CLOPT   = "-A -- -N -M"

# Start a slave NameManager Service
#
    TMFFNAME
        SRVGRP  = SYS_GRP
        SRVID   = 3
        CLOPT   = "-A -- -N"
```

```
# Start the FactoryFinder (-F) service
#
    TMFFNAME
        SRVGRP  = SYS_GRP
        SRVID   = 4
        CLOPT   = "-A -- -F"

# Start the interface repository server
#
    TMIFRSVR
        SRVGRP  = SYS_GRP
        SRVID   = 5

# Start the university server
#
    univb_server
        SRVGRP  = ORA_GRP
        SRVID   = 1
        RESTART = N

# Start the listener for IIOP clients
# Specify the host name of your server machine as
# well as the port. A typical port number is 2500
#
    ISL
        SRVGRP  = SYS_GRP
        SRVID   = 6
        CLOPT   = "-A -- -n //PCWIZ:2500"

#-----------------------------------------------------------------
*SERVICES
```

Administrative Requirements and Performance

This section provides information to assist you in administering your system.

Configuring NameManager

Adhering to the following requirements is fundamental to successful system administration.

- NameManagers should coordinate their activities with each other using the BEA Tuxedo Event Broker without administrative or operations intervention. The Event Broker must be started before any servers providing the NameManager service. If the Event Broker is not configured into the application and is not running when the NameManager service is booted, the NameManager aborts its startup and writes an error message to the user log.
- At least two servers must be configured to run the NameManager service as part of any application. This requirement is to ensure that a working copy of the "name-to-IOR" mapping is always available. If the servers are on different machines, and one machine crashes, when the machine and application are restarted, the new NameManager obtains the mapping from the other NameManager. If an application is solely contained on one machine and the machine crashes, the NameManagers are rebooted as part of the application startup because the application must be rebooted. If two NameManagers are not configured in the application when a NameManager service is booted, the NameManager aborts its startup and writes an error message to the user log.
- NameManagers can be designated as either Master or Slave, the default being Slave. If a Master NameManager server is not configured in the application and is not running when a slave NameManager server starts, the server terminates itself during boot and writes an error message to the user log.
- If a NameManager service is not configured in the application when a FactoryFinder service is booted, the FactoryFinder aborts its startup and writes an error message to the user log. It is not necessary for the NameManager service to start before a FactoryFinder service because the FactoryFinder only communicates with a NameManager when a `"find"` request is received from an application. NameManagers, on the other hand, attempt to communicate with each other when they boot. FactoryFinders do not communicate with each other except when a request is received to find a factory that is in a remote domain.
- BEA Tuxedo Event Broker, NameManager, and FactoryFinder services must be started before any of the application-specific servers. However, if more than one Event Broker is to be configured in the application, all secondary Event Brokers must be started after all application servers are started. There is no system

protocol to enforce this in an application server; therefore, you accomplish this by positioning all secondary Event Brokers after the application servers.

- The Master NameManager must be started and must be running before any application server can register a reference to a factory object. The existence of an executing Slave NameManager is not sufficient.

Reliability Requirements

This section contains information that will improve system reliability.

Managing Factory Entries

When application servers "die," they often fail to unregister their factories with the NameManager. In some cases, the FactoryFinder may give out object references for factories that are no longer active. This occurs because the servers containing those factories have become unavailable, have failed to unregister their factories with the NameManager, and there is no other server capable of servicing the interface for that factory.

In general, an application factory can restart shortly thereafter, and then offer the factories. However, to ensure that factory entries are not kept indefinitely, the NameManager is notified when application servers die. Upon receipt of this notification, the NameManager may remove those factory entries that are not supported in any currently active server.

Configuring Multiple NameManagers and FactoryFinders

At a minimum, two NameManagers, a Master and a Slave, must be configured in an application, preferably on different machines, to provide querying capabilities for a FactoryFinder. Multiple FactoryFinders should also be configured in an application.

Designating a Master NameManager

A Master NameManager must be designated in the `UBBCONFIG` file. All registration activities are sent to the Master NameManager. The Master NameManager then notifies the Slave NameManagers about the updates. If the Master NameManager is down, registration/unregistration of factories is disabled until the Master restarts.

Performance Hint

Implementing the following hint may improve system performance of the administrative servers:

- You can optimize FactoryFinder and NameManager performance by running these services on separate servers within the same machine rather than running these services on different machines. This provides a quicker response because it eliminates the need for machine-to-machine communication.

Configuring Resources

The following paragraphs explain how to set `RESOURCES` parameters that control the application as a whole.

`RESOURCES` is a required section and must appear as the first section in the configuration file. Some of the parameters in the `RESOURCES` section serve as system-wide defaults (`UID`, `GID`, `PERM`, `MAXACCESSERS`, `MAXCONV`, and `MAXOBJECTS`) and can be overridden on a per-machine basis.

Table 3-1 lists some of the parameters in the `RESOURCES` section and gives sample values for a BEA WebLogic Enterprise server application. For more detailed information about these parameters, see the `ubbconfig`(5) reference page in the *BEA Tuxedo Reference Manual*.

Table 3-1 RESOURCES Section Parameters

Parameter	Description	Sample Value	Meaning of Sample Value
`IPCKEY`	The address of shared memory.	`39210`	Indicates a number unique to this application on this system.
`MAXSERVERS`	IPC limit for the number of server processes.	`20`	Allows up to 20 active server processes for this application.

Table 3-1 RESOURCES Section Parameters (Continued)

Parameter	Description	Sample Value	Meaning of Sample Value
`MAXINTERFACES`	The IPC limit for the number of interfaces.	`25`	Allows up to 25 CORBA interfaces in the Bulletin Board interface tables. The `MAXINTERFACES` parameter is specific to the BEA WebLogic Enterprise system.
`MAXSERVICES`	The IPC limit for the number of services offered.	25	Allows up to 25 services to be advertised. On BEA WebLogic Enterprise systems, each CORBA interface is mapped to a BEA Tuxedo service. If you are using JavaServers, each JavaServer instance advertises two additional services.
`MAXOBJECTS`	The IPC limit for the number of CORBA objects.	`800`	Allows up to 800 BEA WebLogic Enterprise active CORBA objects in the Active Object Map tables in the Bulletin Board. The `MAXOBJECTS` parameter is specific to the BEA WebLogic Enterprise system.
`MASTER`	The administration site (`MASTER`) for boot and shutdown.	`SITE1, SITE2`	Specifying `LMID SITE1` means the machine is the master. If `LMID SITE2` is specified, the machine is the backup.
`MODEL`	Application architecture, which indicates a single machine or multiple machines application.	`MP`	Indicates that this application has more than one machine in the configuration.
`OPTIONS`	The options used.	`LAN, MIGRATE`	Indicates a networked application, and that the machine and servers can be migrated to alternate processors.
`SECURITY`	The level of security.	`APP_PW`	Indicates that this is a secure application; clients are required to supply a password to join.

Table 3-1 RESOURCES Section Parameters (Continued)

Parameter	Description	Sample Value	Meaning of Sample Value
`AUTHSVC`	The name of an application authentication service invoked by the system for each client joining the system.	`"AUTHSVC"`	Indicates that in addition to the password, clients must pass authentication from a service called `"AUTHSVC"`.
`SYSTEM_ACCESS`	The default mode used by BEA Tuxedo system libraries within application processes to gain access to a BEA Tuxedo system's internal tables.	`PROTECTED`, `FASTPATH`, `NO_OVERRIDE`	Specifying `PROTECTED` means the application code does not attach to shared memory.
`LDBAL`	Server load balancing enabled.	`Y`	Indicates that load balancing is on. This value is always `Y` in BEA WebLogic Enterprise systems; that is, setting `LDBAL` to `N` is ignored in the BEA WebLogic Enterprise system.

The following sections describe how to set the `RESOURCES` section parameters.

Setting the Shared Memory Address

You set the address of shared memory using the `IPCKEY` parameter. The BEA WebLogic Enterprise system uses this parameter to allocate application IPC resources so 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 a single processor mode, this key names the Bulletin Board; in a multiprocessor mode, this key names the message queue of the DBBL.

The `IPCKEY` parameter is:

- Required and must appear in the configuration file.

- Used to access the Bulletin Board and other IPC resources.
- An integer in the range 32,769 to 262,143.
- Unique. No other application on the system may use this specific value for its `IPCKEY`.

Specifying the Master Machine

You must specify a master machine for all configurations (`MASTER`). The master machine controls the booting and administration of the entire application. This machine is specified using a Logical Machine Identifier (`LMID`). This is an alphanumeric name chosen by the administrator. (`LMID`s are discussed further in the section "Configuring Machines" on page 3-24.)

Two `LMID`s are specified if migration of the master site is to be allowed. If it is necessary to bring down the master site without shutting down the application, it is necessary to specify the backup master site.

The `MASTER` parameter:

- Is required and controls booting and administration.
- Requires two `LMID`s for migration to back up the MASTER machine.

Setting the Application Type

Among the architectural decisions you need to make for a BEA WebLogic Enterprise or a BEA Tuxedo application are the following:

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

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 supported).

Table 3-2 lists the characteristics for the MODEL and OPTIONS parameters.

Table 3-2 Model and Options Parameter Characteristics

Parameter	Characteristics
MODEL	■ It is a required parameter. ■ A value of SHM indicates a single machine with global shared memory. ■ A value of MP indicates multiple machines or a nonglobal shared memory multiprocessor.
OPTIONS	■ It is a comma-separated list of application configuration options. ■ A value of LAN indicates a local area network. ■ A value of MIGRATE enables server migration.

Note: No OPTIONS are specified for the SHM model.

Defining Access Control (BEA Tuxedo Servers)

The BEA WebLogic Enterprise system provides security features, but does not support access control lists (ACLs) at this time. This section applies only to BEA Tuxedo servers.

You can provide basic access to a BEA Tuxedo application using the following three parameters:

- UID—the user ID of the administrator. The value is a numeric value corresponding to the UNIX system user ID of the person who boots and shuts down the system.
- GID—the numeric Group ID of the administrator.
- PERM—an octal number that specifies the permissions to assign to the IPC resources created when the application is booted. This provides the first level of

security to protect the BEA Tuxedo system IPC structures against unauthorized access. The default is 0666, which gives read/write access to all. These values should be specified for production applications.

Note: If the `UID` and `GID` parameters are not specified, they default to the IDs of the person who runs the `tmloadcf`(1) command on the configuration, unless they are overridden in the `MACHINES` section.

Table 3-3 lists the `UID`, `GID`, and `PERM` parameters characteristics.

Table 3-3 Access Control Parameters Characteristics

Parameter	Characteristics
`UID`	The user ID of the administrator. The default is the ID of the person who runs `tmloadcf`(1). Example: `UID=3002` On Windows NT, this value is always 0.
`GID`	The group ID of the administrator. The default is the ID of the person who runs `tmloadcf`(1). Example: `GID=100` On Windows NT, this value is always 0.
`PERM`	The permissions for access to IPC structures. The default is 0666. Example: `PERM=0660` On Windows NT, this value is always 0.

Note: You can overwrite values on remote machines.

Defining IPC Limits

Because most 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 consumed to excess; if they are set too small, the process fails when the limits are eclipsed.

Currently, the following tunable parameters are related to IPC sizing in the `RESOURCES` section:

- `MAXACCESSERS`—the maximum number of overall processes allowed to be attached to the BEA WebLogic Enterprise or BEA Tuxedo system at one site. This number is not the sum of all processes, but is equal to the number at the site that has the most processes. The default is 50. (You can overwrite `MAXACCESSERS` on a per-machine basis in the `MACHINES` section.)

  The `MAXACCESSERS` parameter sets the maximum number of concurrent accessors of a BEA WebLogic Enterprise system. Accessors include native and remote clients, servers, and administration processes.

  A single-threaded server counts as one accessor.

  For multithreaded BEA WebLogic Enterprise JavaServers, you must account for the number of worker threads that each server is configured to run. A *worker thread* is a thread that is started and managed by the BEA WebLogic Enterprise Java software, as opposed to threads started and managed by an application program. Internally, BEA WebLogic Enterprise Java manages a pool of available worker threads. When a client request is received, an available worker thread from the thread pool is scheduled to execute the request. When the request is completed, the worker thread is returned to the pool of available threads.

  For a multithreaded JavaServer, the number of accessors can be up to twice the maximum number of worker threads that the server is configured to run, plus one for the server itself. However, to calculate a `MAXACCESSERS` value for a BEA WebLogic Enterprise system running multithreaded servers, **do not** simply double the existing `MAXACCESSERS` value of the whole system. Instead, you add up the accessors for each multithreaded server.

  For example, assume that you have three multithreaded JavaServers in your system. JavaServer A is configured to run three worker threads. JavaServer B is configured to run four worker threads. JavaServer C is configured to run five worker threads. The accessor requirement of these servers is calculated by using the following formula:

  ```
  [(3*2) + 1]  +  [(4*2) + 1]  +  [(5*2) + 1] = 27 accessors
  ```

- `MAXSERVERS`—the maximum number of server processes in the application, including all the administrative servers (for example, BBL and TMS). It is the sum of the server processes at all sites. The default is 50.

- `MAXINTERFACES`—on a BEA WebLogic Enterprise system, the maximum number of CORBA interfaces to be accommodated in the interface table of the

Bulletin Board. Valid values are from 0 to 32766. If not specified, and if the BEA WebLogic Enterprise system is licensed for the domain, the default is 100. If the BEA WebLogic Enterprise system is not licensed, any nonzero value is replaced with a value of zero.

Note: All instances of an interface occupy and reuse the same slot in the interface table in the Bulletin Board. For example, if server `SVR1` advertises interfaces `IF1` and `IF2`, server `SVR2` advertises interfaces `IF2` and `IF3`, and server `SVR3` advertises interfaces `IF3` and `IF4`, the interface count is 4 (not 6) when calculating `MAXINTERFACES`.

- `MAXOBJECTS`—on a BEA WebLogic Enterprise system, the maximum number of active CORBA objects to be accommodated in the Active Object Table for a particular machine at one time. Valid values are from 0 to 32766. If not specified, and if the BEA WebLogic Enterprise system is licensed for the domain, the default is 1000. The `RESOURCES` value for this parameter can be overridden in the `MACHINES` section on a per-machine basis. If the BEA WebLogic Enterprise system is not licensed, any nonzero value is replaced with a value of zero.

- `MAXSERVICES`—the maximum number of different services that can be advertised in the application. It is the sum of all services in the system. The default is 100.

  Note: On BEA WebLogic Enterprise systems, each CORBA interface is mapped to a BEA Tuxedo service. Make sure you account for the number of services generated.

  Note: On BEA WebLogic Enterprise systems, each JavaServer instance advertises five additional services. If you are running JavaServers, you may need to increase the value of `MAXSERVICES` to take account of these additional services.

The cost incurred by increasing `MAXACCESSERS` is one additional semaphore per site per accessor. 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 especially important to pay attention to the value of `MAXACCESSERS`.

Note: Two additional parameters, MAXGTT and MAXCONV, affect shared memory. For details, see the UBBCONFIG(5) reference page in the *BEA Tuxedo Reference Manual*.

Table 3-4 lists the characteristics for the MAXACCESSERS, MAXSERVERS, MAXINTERFACES, MAXOBJECTS, and MAXSERVICES parameters.

Table 3-4 IPC Sizing Parameters Characteristics

Parameter	Characteristics
MAXACCESSERS	Number of processes on the site that is running the most processes. You can overwrite the value on a per-machine basis in the MACHINES section. The cost is one additional semaphore per accesser.
MAXSERVERS	Maximum number of server processes in an application (sum of all sites). The cost is a small amount of shared memory.
MAXINTERFACES (BEA WebLogic Enterprise servers)	Maximum number of CORBA interfaces advertised in the application (sum of all active interfaces in the domain). The cost is a small amount of shared memory. Default is 100.
MAXOBJECTS (BEA WebLogic Enterprise system)	Maximum number of CORBA objects in an application (sum of all objects in the domain). The cost is a small amount of shared memory. Default is 1000. You can overwrite the value on a per-machine basis in the MACHINES section.
MAXSERVICES	Maximum number of BEA Tuxedo services advertised in the application (sum of all sites). The cost is a small amount of shared memory. Default is 100. On BEA WebLogic Enterprise systems, each CORBA interface is mapped to a BEA Tuxedo service. Make sure you account for the number of services generated.

Enabling Load Balancing

Load balancing is always enabled on BEA WebLogic Enterprise systems. On BEA Tuxedo systems, use `LDBAL=Y` to enable load balancing.

Note: For more information about load balancing, see the section "Enabling Load Balancing" on page 3-73.

Setting Buffer Type and Subtype Limits

You can control the number of buffer types and subtypes allowed in the application with the `MAXBUFTYPE` and `MAXBUFSTYPE` parameters, respectively. The default for `MAXBUFTYPE` is 16. Unless you are creating many user-defined buffer types, you can omit `MAXBUFTYPE`. However, if you intend to use many different VIEW subtypes, you may want to set `MAXBUFSTYPE` to exceed its current default of 32.

Table 3-5 lists the characteristics of the `MAXBUFTYPE` and `MAXBUFSTYPE` parameters.

Table 3-5 Buffer Type and Subtype Sizing Characteristics

Parameter	Characteristics
`MAXBUFTYPE`	Maximum number of buffer types allowed in the system. Default is 16. Example: `MAXBUFTYPE 20`
`MAXBUFSTYPE`	Maximum number of buffer subtypes allowed in the system. Default is 32. Example: `MAXBUFSTYPE 40`

Setting the Number of Sanity Checks and Timeouts

You can set the number of times the administrative server (BBL) will periodically check the sanity of servers local to its machine. In addition, you can set the number of timeout periods for blocking messages, transactions, and other system activities.

You use the SCANUNIT parameter to control the granularity of such checks and timeouts. Its value (in seconds) can be a positive multiple of 5. Its default is 10.

You use the SANITYSCAN parameter to specify how many SCANUNITs elapse between sanity checks of the servers. The value of SANITYSCAN * SCANUNIT cannot exceed 300. The default value of SANITYSCAN * SCANUNIT is approximately 120 seconds.

Example: Setting Sanity Checks and Timeouts

A SCANUNIT of 10 and a BLOCKTIME of 3 allows 30 seconds before the client application times out. The BLOCKTIME default is set so that BLOCKTIME * SCANUNIT is approximately 60 seconds. The time is a total of the following times:

- Time waiting to get on the queue
- Time waiting on the queue
- Time for service processing
- Time on the network

Characteristics of the SCANUNIT, SANITYSCAN, and BLOCKTIME Parameters

Table 3-6 lists the SCANUNIT, SANITYSCAN, and BLOCKTIME parameters characteristics.

Table 3-6 SCANUNIT, SANITYSCAN, and BLOCKTIME Characteristics

Parameter	Characteristics
SCANUNIT	Establishes granularity of check intervals and timeouts. Value must be in multiples of 5 seconds. The default is 10. Example: SCANUNIT 10
SANITYSCAN	Frequency at which the BBL checks the server (in SCANUNIT intervals). SCANUNIT * SANITYSCAN must not exceed 300. Default of SCANUNIT * SANITYSCAN is approximately 120 seconds. Example: SANITYSCAN 3

Table 3-6 SCANUNIT, SANITYSCAN, and BLOCKTIME Characteristics

Parameter	Characteristics
BLOCKTIME	Timeout for blocking messages. SCANUNIT * BLOCKTIME must not exceed 300. Default of SCANUNIT * BLOCKTIME is approximately 60 seconds. Example: BLOCKTIME 1

Setting Conversation Limits (BEA Tuxedo Servers)

You can specify the maximum number of conversations on a machine with the MAXCONV parameter.

Note: The MAXCONV parameter applies only to the BEA Tuxedo servers.

The MAXCONV parameter has the following characteristics:

- It is the maximum number of simultaneous conversations per machine.
- Its value must be greater than or equal to 0 and less than 32766.
- 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 value in the MACHINES section.

Setting the Security Level

You can set three levels of security using the following parameters:

- PERM parameter—sets the first or lowest-level permission to write to the application queues.
- SECURITY parameter—sets the second-level permission. As a minimum, this level requires that the client supply a password when joining the application. This password is checked against the password supplied by the system administrator when the TUXCONFIG file is generated from the UBBCONFIG file.

Optionally, this level can also require user authorization and access control list privileges.

Note: For details about the supported security parameters, see *Using Security* in the BEA WebLogic Enterprise online documentation.

- `AUTHSVC` parameter—sets the third-level permission. This sends the client's request to join the application to an authentication service. This level requires the second level of `SECURITY` to be present. The authentication service may be the default supplied by the BEA Tuxedo system or it may be a service, such as a Kerberos service, supplied by another vendor.

Table 3-7 lists the `SECURITY` and `AUTHSVC` parameters characteristics.

Table 3-7 Security Level Parameters Characteristics

Parameter	Characteristics
`Security`	Accepted values are: `NONE` (default), `APP_PW`, `USER_AUTH`, `ACL`, and `MANDATORY_ACL`. The `ACL` and `MANDATORY_ACL` parameters are not supported and are ignored on machines using the BEA WebLogic Enterprise CORBA API. Default is `NONE`. Example: `SECURITY APP_PW`
`AUTHSVC`	The name of the authentication service. `SECURITY APP_PW` must be specified. Default is no authentication service. Client authentication with Kerberos is possible. Example: `AUTHSVC` ‘‘`AUTHSVC`’’

Setting Parameters of Unsolicited Notification (BEA Tuxedo Servers)

This section applies only to BEA Tuxedo servers.

You can set the default method for clients to receive unsolicited messages using the NOTIFY parameter. The client, however, can override this setting in the TPINIT structure when tpinit() is called.

The following three methods can be set for clients:

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

Two types of signals can be generated: SIGUSR1 and SIGUSR2. The USIGNAL parameter allows the administrator to choose the type of signal. The default is SIGUSR2. In applications that choose notification by signals, any MS-DOS client workstations are switched automatically to DIPIN.

Table 3-8 lists the NOTIFY and USIGNAL parameters characteristics.

Table 3-8 Unsolicited Notification Parameters Characteristics

Parameter	Characteristics
NOTIFY	Value of IGNORE means clients should ignore unsolicited messages. Value of DIPIN means clients should receive unsolicited messages by dip-In. Value of SIGNAL means clients should receive unsolicited messages by signals. Default is DIPIN. Example: NOTIFY SIGNAL
USIGNAL	Value of SIGUSR1 means notify clients with this type of signal. Value of SIGUSR2 means notify clients with this type of signal. Default is SIGUSR2. Example: USIGNAL SIGUSR1

Protecting Shared Memory

You can shield system tables kept in shared memory from application clients or servers using the SYSTEM_ACCESS parameter. This option is useful when applications are being developed because faulty application code can inadvertently corrupt shared memory with a bad pointer. When the application is fully debugged and tested, this option could then be changed to allow for faster responses. The following are the options for this parameter:

- PROTECTED—BEA WebLogic Enterprise or BEA Tuxedo libraries compiled with application code will not attach to shared memory while executing system code.
- FASTPATH—BEA WebLogic Enterprise or BEA Tuxedo libraries will attach to shared memory at all times.
- NO_OVERRIDE—the selected option cannot be changed either by the client in the TPINIT structure of the `tpinit()` call or in the SERVERS section for servers.

Table 3-9 lists the PROTECTED, FASTPATH, and NO_OVERRIDE parameters characteristics.

Table 3-9 Shared Memory Protection Parameters Characteristics

Parameter	Characteristics
PROTECTED	Internal structures in shared memory will not be corrupted inadvertently by application processes.
FASTPATH (default)	Application processes will join with access to shared memory at all times.

Example: `SYSTEM_ACCESS PROTECTED, NO_OVERRIDE`

Configuring Machines

This section explains how to define parameters for each processor, or *machine*, on which your application runs.

Identifying Machines in the MACHINES Section

Every machine in an application must have a `MACHINES` section entry in the configuration file and it must be the second section in the file. The `MACHINES` section contains the following information specific to each machine in the application:

- The mapping of the machine *address* to a logical identifier (`LMID`). (Required)
- The location of the configuration file (`TUXCONFIG`). (Required)
- The location of the installed BEA WebLogic Enterprise or BEA Tuxedo software (`TUXDIR`). Note that the `TUXDIR` parameter is used on BEA WebLogic Enterprise systems. Use it to identify the top-level location where you installed the BEA WebLogic Enterprise system, such as `c:\wledir`. (Required)
- The location of the application servers (`APPDIR`). (Required)
- The location of the application log file (`ULOGPFX`). (Optional)
- The location of the environment file (`ENVFILE`). (Optional)
- The maximum number of active CORBA objects to be accommodated in the Active Object Table for this processor (`MAXOBJECTS`). (Optional)

Note: For a particular machine, you can override the `UID`, `GID`, `PERM`, `MAXACCESSERS`, `MAXCONV`, and `MAXOBJECTS` values that were specified in the `RESOURCES` section.

Example: MACHINES Section

The following example provides a sample `MACHINES` section of a configuration file:

```
*MACHINES
gumby          LMID=SITE1
               TUXDIR=”/wledir”
               APPDIR=”/home/apps/mortgage”
               TUXCONFIG=”/home/apps/mortgage/tuxconfig”
               ENVFILE=”/home/apps/mortgage/ENVFILE”
               MAXOBJECTS=700
               ULOGPFX=”/home/apps/mortgage/logs/ULOG”
               MAXACCESSERS=100
```

Parameters in a Sample MACHINES Section

Table 3-10 lists the `MACHINES` section parameters characteristics.

Table 3-10 MACHINES Section Parameter Characteristics

Parameter	Description
`gumby`	The machine name obtained with the command `uname -n` on UNIX systems. On Windows NT systems, see the Computer Name value in the Network Control Panel.
`LMID=SITE1`	The logical machine identifier of the machine *gumby*.
`TUXDIR`	The double quoted string of the full path to the installed BEA WebLogic Enterprise or BEA Tuxedo software.
`APPDIR`	The string of the full path to the application directory, enclosed in double quotes.
`TUXCONFIG`	The string of the full pathname of the configuration file, enclosed in double quotes.
`ENVFILE`	The string of the full pathname of a file containing environment information, enclosed in double quotes.
`MAXOBJECTS`	Override the system wide value defined in `RESOURCES` with 700.
`ULOGPFX`	The string of the full pathname prefix of the log file, enclosed in double quotes.
`MAXACCESSERS`	Override the system-wide value with 100 for this machine.
`MAXCONV`	Override the system-wide value with 15 for this machine.

Reserving the Physical Address and Machine ID

You initially define the address in the address portion, which is the basis for a `MACHINES` section entry. All other parameters in the entry describe the machine specified by the 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 and specifies a logical name used to designate the computer whose address has just been provided. It may be any alphanumeric value, and must be unique among other machines in the application.

The address and machine ID and the LMID parameter have the following characteristics:

- The address and machine ID are specified in the following way:

  hostname LMID=*logical_machine_name*
- *hostname* identifies the physical machine.
- The format of the LMID parameter is LMID=*logical_machine_name*.
- The LMID is the logical machine name for a physical processor.
- LMID is alphanumeric and must be unique within the MACHINES section.

Identifying the Location of the Configuration File

You identify the configuration file location and file name for the machine with TUXCONFIG, a required parameter. The TUXCONFIG parameter is enclosed in double quotes and represents the full pathname up to 64 characters. The path specified must be the same as the environment variable, TUXCONFIG; otherwise, the tmloadcf(1) will not compile the binary file.

The TUXCONFIG parameter has the following characteristics:

- The syntax of the TUXCONFIG parameter is TUXCONFIG=*"<tuxconfig>"*.
- This parameter identifies the location of the configuration file and filename (though it should remain TUXCONFIG for convention purposes) for the machine.
- The full pathname for TUXCONFIG can be up to 64 characters.
- The value of TUXCONFIG must match the TUXCONFIG environment variable.

Identifying the Locations of the System and Application Software

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

Table 3-11 lists the `TUXDIR` and `APPDIR` parameters characteristics.

Table 3-11 System Software Location Parameters Characteristics

Parameter	Characteristics
`TUXDIR`	The syntax requires the full pathname in a string enclosed in double quotes: `TUXDIR="<`*`TUXDIR`*`>"`. Identifies the location of the BEA WebLogic Enterprise or BEA Tuxedo software. Is a required parameter.
`APPDIR`	The syntax requires the full pathname in a string enclosed in double quotes: `APPDIR="<`*`APPDIR`*`>"`. Identifies the location of application servers. Is a required parameter. Becomes the current working directory of server processes.

Identifying the User Log File Location

The user log file contains warning and informational messages, as well as error messages that describe the nature of any ATMI error with a return code of `TPESYSTEM` and `TPEOS` (that is, underlying system errors). The user can use this log to track application-related errors. By default, the file is named `ULOG.`*`mmddyy`* where *`mmddyy`* is the month, date, and two-digit year. By default, the file is written into the `APPDIR`.

You can override the default directory and prefix by specifying the `ULOGPFX` parameter which is the absolute pathname of the application log file, without the date. For example, this may be set to `APPDIR/logs/ULOG` so that logs collect in a particular directory. In a networked application, a central log can be maintained by specifying a remote directory that is mounted on all machines.

The `ULOGPFX` parameter has the following characteristics:

- The syntax of the `ULOGPFX` parameter is a string enclosed in double quotes: `ULOGPFX=`*"<ULOGPFX>"*.
- The `ULOGPFX` defaults to `<APPDIR>/ULOG`.

Examples:
```
ULOGPFX="/usr/appdir/logs/ULOG"
ULOGPFX="/mnt/usr/appdir/logs/BANKLOG"
```

Specifying Environment Variable Settings for Processes

With the `ENVFILE` parameter, you can specify a file that contains environment variable settings for all processes to be booted by the BEA WebLogic Enterprise or BEA Tuxedo system. The system sets `TUXDIR` and `APPDIR` for each process, so these variables should not be specified in this file. You can specify settings for the following variables because they affect an application's operation. Most of these settings apply only to BEA Tuxedo servers, as noted.

- `FIELDTBLS`, `FLDTBLDIR` (BEA Tuxedo servers)
- `VIEWFILES`, `VIEWDIR` (BEA Tuxedo servers)
- `TMCMPLIMIT` (BEA Tuxedo servers)
- `TMNETLOAD`

The `ENVFILE` parameter has the following characteristics:

- The syntax 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 WebLogic Enterprise or BEA Tuxedo system.
- You can set the environment variables but do not set `TUXDIR` and `APPDIR`.

- The `ENVFILE` parameter is optional and all settings must be hard coded. No evaluations such as `FLDTBLDIR=$APPDIR` are allowed.
- The format is `VARIABLE=`*`string`*.

Overriding System-wide Parameters

Table 3-12 lists the system-wide parameters you can override for a specific machine.

Note: You can override values on remote as well as local machines.

Table 3-12 System-wide Parameters That Can Be Overridden

Parameter	Characteristics
`UID`	The user ID of the administrator. The default is the ID of the person who runs `tmloadcf`(1). Example: `UID=3002` On Windows NT, `UID` value is always 0.
`GID`	The group ID of the administrator. The default is the ID of the person who runs `tmloadcf`(1). Example: `GID=100` On Windows NT, this value is always 0.
`PERM`	The permissions for access to IPC structures. The default is 0666. Example: `PERM=0660` On Windows NT, this value is always 0.
`MAXACCESSERS`	Number of processes on the site that is running the most processes. You can overwrite the value on a per-machine basis in the `MACHINES` section. The cost is one additional semaphore per accessor.

Table 3-12 System-wide Parameters That Can Be Overridden (Continued)

Parameter	Characteristics
`MAXOBJECTS` (BEA WebLogic Enterprise system)	Maximum number of active CORBA objects in an application on any machine (sum of all active CORBA objects on the machine). The cost is a small amount of shared memory. Default is 1000. You can overwrite the value on a per-machine basis in the `MACHINES` section.
`MAXGTT`	Maximum number of simultaneous global transactions in which a particular machine can be involved.

Configuring Groups

You can use `GROUPS` to group servers together logically. These groupings can later be used to access resource managers, and for server group migration. The `GROUPS` section of the configuration file contains the definition of server groups. You must define at least one server group for a machine to have an application server running on it. If no group is defined for a machine, the machine can still be part of the application and you can run the administrative command `tmadmin`(1) from that site.

For nontransactional, nondistributed systems, groups are relatively simple. You only need to define the basic mapping of group name to group number and logical machine of each group.

Specifying a Group Name, Number, and LMID

The group name is an alphanumeric name by which the group is identified. It must have a unique group number (`GRPNO`). Each group must reside entirely on one logical machine (`LMID`). The `LMID` is also mandatory.

Sample GROUPS Section

The example `GROUPS` section in Listing 3-2 is from the `UBBCONFIG` file in the BEA WebLogic Enterprise University sample Production application. In this sample, the groups specified by the `RANGES` identifier in the `ROUTING` section of the `UBBCONFIG` file need to be identified and configured.

The Production sample specifies four groups: `ORA_GRP1, ORA_GRP2, APP_GRP1,` and `APP_GRP2`. These groups mst be configured, and the machines on which they run on must be identified.

Listing 3-2 Example GROUPS Section

```
*GROUPS

APP_GRP1
   LMID = SITE1
   GRPNO = 2
   TMSNAME = TMS

APP_GRP2
   LMID = SITE1
   GRPNO = 3
   TMSNAME = TMS

ORA_GRP1
   LMID = SITE1
   GRPNO = 4

OPENINFO = "ORACLE_XA:Oracle_XA+Acc=P/scott/tiger+SesTm=100+LogDir=.+MaxCur=5"

   CLOSEINFO = ""
   TMSNAME = "TMS_ORA"

ORA_GRP2
   LMID = SITE1
   GRPNO = 5

OPENINFO = "ORACLE_XA:Oracle_XA+Acc=P/scott/tiger+SesTm=100+LogDir=.+MaxCur=5"

CLOSEINFO = ""
TMSNAME = "TMS_ORA"
```

The preceding example shows how the `ORA_GRP1, ORA_GRP2, APP_GRP1,` and `APP_GRP2` groups are configured. See the section “Example: Factory-based Routing (BEA WebLogic Enterprise Servers)” on page 3-76 to understand how the names in the `GROUPS` section match the group names specified in the `ROUTING` section. This match is critical for the routing function to work correctly. Also, any change in the way groups are configured in an application must be reflected in the `ROUTING` section.

Note: The Production sample application packaged with the BEA WebLogic Enterprise software is configured to run entirely on one machine. However, you can easily configure this application to run on multiple machines by specifying the other machines in the `LMID` parameter. This step assumes that you specify the `MODEL MP` parameter in the `RESOURCES` section.

Encrypting Passwords in OPENINFO

Passwords for server groups can be stored in the `UBBCONFIG` file in encrypted form using the `tmloadcf` utility.

To secure a password in the `UBBCONFIG` file, you initially enter a string of five or more continuous asterisks at the place in the `OPENINFO` statement where a password is to go. The asterisks are a placeholder for the password. For example:

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

When `tmloadcf` encounters this string, it prompts the user to create a password. For example:

```
>tmloadcf -y e:/wle5/samples/atmi/bankapp/xx
Password for OPENINFO (SRVGRP=BANKB1):
```

The password is stored in the `TUXCONFIG` in encrypted form. To place the encrypted password in the `UBBCONFIG` file, use `tmunloadcf` to generate a `UBBCONFIG` file. When `tmunloadcf` is run, the encrypted password is written into the `OPENINFO` string in the `UBBCONFIG` file 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. Instead, the `tmloadcf` command uploads the encrypted password back into the system.

Note: The `UBBCONFIG` file with the encrypted form of the password may be uploaded back into the system only once; subsequent attempts will fail.

Use of encrypted passwords is recommended for production environments.

Configuring Servers

This following paragraphs explain the `SERVERS` section parameters that you need to define to configure server processes.

Note: Administrators and programmers who are working in a Java environment should see the section "Starting JavaServer" on page 3-39.

Identifying Server Information in the SERVERS Section

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. Server-specific information includes the following:

- A server name, group, and numeric identifier (`SRVGRP`, `SRVID`)
- Command-line options (`CLOPT`)
- Parameters to determine the booting order and number of servers to boot (`SEQUENCE`, `MIN`, `MAX`)
- A server-specific environment file (`ENVFILE`)
- `JavaServer` information (JAR file and options)
- Server queue-related information (`RQADDR`, `RQPERM`, `REPLYQ`, `RPPERM`) (BEA Tuxedo servers)
- Restart information (`RESTART`, `RCMD`, `MAXGEN`, `GRACE`)
- Server designation as a conversational server (`CONV`) (BEA Tuxedo servers)
- Overriding of system-wide shared memory access (`SYSTEM_ACCESS`)
- Setting Security Parameters for ISL Servers

Command-line options supported by the BEA Tuxedo system are described on the `servopts`(5) reference page in the *BEA Tuxedo Reference Manual*.

Table 3-13 provides a sample of parameters and their values in the `SERVERS` section of the configuration file.

Table 3-13 SERVERS Section Parameters

Parameter Example	Meaning
`RESTART=Y`	Restart the servers.
`MAXGEN=5`	The `MAXGEN` parameter specifies a number greater than 0 and less than 256 that controls the number of times the server can be started within the period specified by the `GRACE` parameter. The default is 1. If the server is to be restartable, `MAXGEN` must be greater than or equal to 2. The number of restarts is at most *`number`* minus `1` times. `RESTART` must be `Y` or `MAXGEN` is ignored.
`GRACE=3600`	If `RESTART` is `Y`, the `GRACE` parameter specifies the time period (in seconds) during which this server can be restarted as `MAXGEN` minus `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`	There is no reply queue.
`CLOPT=”-A”`	Specify `-A` on the command line of each server.
`ENVFILE=”/usr/home/envfile”`	Read environment settings from the file `ENVFILE`.
`SYSTEM_ACCESS=FASTPATH`	Specifies the default mode used by system libraries within application processes to gain access to the BEA WebLogic Enterprise or BEA Tuxedo system’s internal tables. Valid access types are `FASTPATH` or `PROTECTED`. `FASTPATH` specifies that the internal tables should be accessible by the libraries via shared memory for fast access. **Note:** Always use `FASTPATH` when you start a `JavaServer`. `PROTECTED` specifies that while the internal tables are accessible by system libraries via shared memory, the shared memory for these tables is not accessible outside of the system libraries. `NO_OVERRIDE` can be specified (alone, or in conjunction with `FASTPATH` or `PROTECTED`) to indicate that the mode selected cannot be overridden by an application process. If `SYSTEM_ACCESS` is not specified, the default mode is determined by the setting of the `SYSTEM_ACCESS` keyword in the `RESOURCES` section.

Defining Server Name, Group, and ID

You initially define the server name entry in the `SERVERS` section entry. The server name is the name of an executable file built with**:**

- `buildserver`(1) (BEA Tuxedo systems)
- `buildobjserver` (BEA WebLogic Enterprise C++ server applications)
- `buildjavaserver` (BEA WebLogic Enterprise Java server application)

You must provide each server with a group identifier (`SRVGRP`). This is set to the name specified in the beginning of a `GROUPS` section entry. You must also provide each server process in a given group with a unique numeric identifier (`SRVID`). Every server must specify a `SRVGRP` and `SRVID`. Because the entries describe machines to be booted and not just applications, it is possible that in some cases the same server name will display in many entries.

Table 3-14 lists the `servername`, `SRVGRP`, and `SRVID` parameters characteristics:

Table 3-14 Servername, SRVGRP, and SRVID Parameters Characteristics

Parameter	Characteristics
`servername`	Identifies the executable to be booted. Is built with `buildserver`(1) on BEA Tuxedo systems, or with `buildobjserver` (C++) or `buildjavaserver` (Java) on BEA WebLogic Enterprise systems. Is required, but may not be unique.
`SRVGRP`	Identifies the group affiliation. The group name from a `GROUPS` section entry. Is required.

Using Server Command-Line Options

The server may need to obtain information from the command line. The `CLOPT` parameter lets you specify command-line options that can change some defaults in the server.

Note: On BEA Tuxedo systems only, you alternatively can 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()`.

On BEA WebLogic Enterprise systems, the standard `main()` of a server parses the set of options ending with the argument `--`; it passes the remaining user-defined options to `tpsvrinit()` on BEA Tuxedo servers, the `Server::initialize` operation on BEA WebLogic Enterprise C++ servers, or the `Server.initialize` method on BEA WebLogic Enterprise Java servers.

Table 3-15 provides a partial list of the available options.

Table 3-15 Partial List of Command-Line Options

Option	Function
`-o` *`filename`*	Redirects standard output to file *`filename.`*
`-e` *`filename`*	Redirects standard error to file *`filename.`*
`-f` *`filename`*	Specifies a nondefault location, name, or both of an Interface Repository. This option can only be used for BEA WebLogic Enterprise Interface Repository servers.
`-s` *`services`*	Advertises services (BEA Tuxedo servers only).
`-s` x,y,z	An example that advertises services `x`, `y`, and `z`.
`-s` x,y,`z:`*`funcname`*	An example that advertises services `x`, `y`, and `z`, but processes requests for those services with function *`funcname`*. This is called aliasing a function name.
`-r`	An example that specifies that the server should log the services performed.
`-A`	The default for `CLOPT` is `-A`, which tells the server to advertise all the services built into it with `buildserver(1)` or `buildobjserver` or `buildjavaserver.`

Note: You can find other standard `main()` options in the `servopts`(5) reference page in the *BEA Tuxedo Reference Manual*.

Server Command-Line Options

The following options apply to both BEA WebLogic Enterprise and BEA Tuxedo servers:

- The syntax is `CLOPT=”`*`servopts -- application_opts`*`”`.
- This is an optional parameter with a default of `-A`.

The following options apply only to BEA Tuxedo servers:

- Both `main()` and `tpsvrinit()` use server command-line options.
- The `servopts`(5) options are passed to `main()`.
- The application options are passed to `tpsvrinit()`.

A `BANKAPP` example is `CLOPT=”-A -- -T 10"`.

Starting JavaServer

In the BEA WebLogic Enterprise Java system, a server application is represented by a Java Archive (JAR). The JAR must be loaded in the Java Virtual Machine (JVM) to be executed. This JVM must execute in a BEA WebLogic Enterprise server to be integrated in a BEA WebLogic Enterprise application. By default, the server that loads the JVM is called `JavaServer`.

You include the options to start a `JavaServer` application in the `SERVERS` and `MODULES` sections of the application’s `UBBCONFIG` file.

WebLogic Enterprise provides support for the Java HotSpot Server VM on Windows NT platforms. If the Java HotSpot Server VM is installed, the JavaServer will load it by default. If you want to bypass the Java HotSpot VM after it is installed, set the `WLE_JVM` environment varible equal to classic or specify the -classic option for the JavaServer. Note that for Windows NT platforms, the `JAVA_HOME` environment variable must be set to the directory path where you installed the JDK software.

See the section “Required Order in Which to Boot Servers (BEA WebLogic Enterprise Servers)” on page 3-49 for important information about starting the BEA WebLogic Enterprise servers in the correct order.

Threading Options

The BEA WebLogic Enterprise Java system supports the ability to configure multithreaded BEA WebLogic Enterprise applications written in Java. A multithreaded BEA WebLogic Enterprise Java server can service multiple object requests simultaneously, while a single-threaded BEA WebLogic Enterprise Java server runs only one request at a time.

You can establish the number of threads for a Java server application by using the `-M` options to the `JavaServer` parameter in the `SERVERS` section. The `-M` options are described in the section "BEA WebLogic Enterprise JavaServer Options" on page 3-45.

For related information about the `MAXACCESSERS` parameter, see the section "Defining IPC Limits" on page 3-15.

Running the BEA WebLogic Enterprise Java server in multithreaded mode or in single-threaded mode is transparent to the application programmer. In the current version of BEA WebLogic Enterprise Java, you should not establish multiple threads in your object implementation code.

The potential for a performance gain from a multithreaded `JavaServer` depends on:

- The application pattern.
- Whether the application is running on a single-processor machine or a multiprocessor machine.

If the application is running on a single-processor machine and the application is merely CPU-intensive but without I/O or other external delays, in most cases the multithreaded `JavaServer` will not perform better. In fact, due to the overhead of switching between threads, the multithreaded `JavaServer` in this configuration may perform worse than a single-threaded `JavaServer`.

A performance gain is more likely with a multithreaded `JavaServer` when the application has some delays or is running on a multiprocessor machine.

Note: If your application uses JNI code to access ATMI, `JavaServer` must be configured as single-threaded.

Check that `SYSTEM_ACCESS=FASTPATH` is set for the JavaServer. Do not use `SYSTEM_ACCESS=PROTECTED` with **`JavaServer`**. (If `SYSTEM_ACCESS` is not specified in the `SERVERS` section, the default mode is determined by the setting of the `SYSTEM_ACCESS` keyword in the `RESOURCES` section.)

If your application is sending messages to the `ULOG`, it is not helpful to use the process ID to distinguish among the different threads. Instead, you can include in each message the object ID, the thread name, and (if your object is transactional) the transaction ID.

JavaServer Parameters

When you start `JavaServer`, the parameters are:

```
JavaServer
       SRVTYPE=JAVA
       SRVGRP=group
       SRVID=number
       MAXEJBCACHE=number
       EJBCACHE_FLUSH=number

       CLOPT="-A -- [java_options]"
```

Note: The JavaServer `MODULE`, *`archive_file`*, and *`options`* parameters were deprecated in BEA WebLogic Enterprise 5.1. To replace the `MODULE` parameter, the `MODULES` section was added to the `UBBCONFIG` file. For a description of the `MODULES` section, refer to "Configuring Modules" on page 3-57.

These `JavaServer` parameters are used as follows:

- `SRVTYPE`—the type of server. If the name of the server is `JavaServer` (not case sensitive), or at least one `MODULE` statement is specified, the default value for `SRVTYPE` is `JAVA`.
- `SRVGRP`—the name of the BEA WebLogic Enterprise group in the `GROUPS` section of the `UBBCONFIG` file.
- `SRVID`—a numeric identifier of the server in the group.
- `MAXEJBCACHE`—the capacity of the EJB cache. You use this parameter to specify the maximum number of beans that can be cached (that is, not passivated) at one time. This parameter is used for performance tuning. The following values can be specified:

Note: By default, if `MAXEJBCACHE` is not specified, the cache size is set to the larger of 10 or the number of threads (`-M` option) + 1.

n	The maximum number of beans that can be contained in the cache where *n* is greater than zero.
0	Disables the cache and results in the same behavior as in BEA WebLogic Enterprise 5.0.

Note: It is very important that you set the cache accurately for your EJB application as it directly affects performance. For more information, see "Using the EJB Cache Size for Tuning and Scaling" on page 3-43.

- `EJBCACHE_FLUSH`—the number of minutes between EJB cache flushes. At the interval specified by this value, all beans that are not currently involved in a transaction or a method invocation are passivated and their memory is released. You can specify the following values:

n	The flush time in minutes, where *n* is greater than zero.
`0`	No time-determined flush. The beans are passivated only when the cache becomes full. This is the default if `EJBCACHE_FLUSH` is not set.

- `java-options`—the `JavaServer` command-line options (`CLOPT`) are Java Virtual Machine (JVM) options, similar to the options that are passed to the `java` interpreter command. These options are described in the sections "Standard Java Virtual Machine Options" on page 3-44, "BEA WebLogic Enterprise JavaServer Options" on page 3-45, and "Nonstandard Java Virtual Machine Options" on page 3-46.

Example of CORBA JavaServer Entry

```
*SERVERS
   JavaServerXA
       SRVGRP=BANK_GROUP1
       SRVID=8
       CLOPT="-A -- -M 10"
       RESTART=N

*MODULES
   BankApp
     SRVGRP=BANK_GROUP1
     SRVID=8
```

```
    FILE="BankApp.jar"
    ARGS ="TellerFactory_1"
```

In this example, the JavaServer that implements the Bankapp's `TellerFactory_1` and Teller interfaces are started. The `-M 10` option enables multithreading for the JavaServer and specifies 10 as the maximum number of worker threads that a particular instance of JavaServer can support. The `FILE="Bankapp.jar"` option specifies the name of the JAR file that contains the implementation of the TellerFactory and Teller CORBA interfaces. The `ARGS="TellerFactory_1"` option specifies a TellerFactory name which is passed to the `com.beasys.Tobj.Server.initialize` method.

A worker thread is a thread that is started and managed by the BEA WebLogic Enterprise Java software, as opposed to threads started and managed by an application program. Internally, BEA WebLogic Enterprise Java manages a pool of available worker threads. When a client request is received, an available worker thread from the thread pool is scheduled to execute the request. When the request is completed, the worker thread is returned to the pool of available threads.

Example of EJB JavaServer Entry

```
JavaServer
       SRVTYPE=JAVA
       SRVGRP="APP_GRP"
       SRVID=5
       CLOPT="-A"
       MAXEJBCACHE=15
       EJBCACHE_FLUSH=1440
```

Using the EJB Cache Size for Tuning and Scaling

Sizing the bean cache correctly is very important. If you are using multithreaded servers, the cache should at least be the number of threads in the server. If the cache is smaller than the number of threads, you could get the CacheFullException exception because all the beans (one per thread) are active in a method invocation.

The optimum number of beans to maintain in the cache should be based on how many beans can be active simultaneously in the server process. This number is determined by:

- How long a bean will remain active in a JavaServer before it can be removed or is dormant.

- The maximum number of worker threads the JavaServer can support.

The guidelines for tuning and scaling the bean cache are as follows:

- While a bean is cached in a server process, all requests for the bean return to the server process that has cached the bean. Caching a bean effectively disables load balancing for the bean. The advantage of caching a bean is that caching saves a lot of activation and passivation, which involves input and output, especially in the case of stateful session beans.

  Every bean is passivated immediately after creation to give an opportunity to balance the load. This approach incurs at least two I/O cycles during the lifecycle of a stateful bean, even though the lifecycle may be relatively short. This could change in the future based on customer input regarding the pattern in which stateful beans are used.

  In general, BEA recommends not flushing the cache frequently. However, after a cache flush, all the beans that are not active (in a method or a transaction) are again available for load balancing to servers supporting the bean.

- If a server process in which a cached bean exists crashes, there is no recovery of that bean’s state.

- Every bean cache entry potentially uses an Active Object Map (AOM) entry. The default of 1,000 objects in the AOM parameter specified in the `UBBCONFIG` file may be insufficient if you have many server processes with many large caches.

Standard Java Virtual Machine Options

The standard Java virtual machine options are shown in the following list.

`-cp, -classpath path`
: Specifies the path `JavaServer` uses to look up classes. Overrides the default or the `CLASSPATH` environment variable if it is set.

`-verbose, -verbose:class`
: Causes `JavaServer` to print a message to the user log each time a class file is loaded.

`-verbose:gc`
: Causes the garbage collector to print messages in the user log whenever it frees memory.

`-verbose:jni`
: Prints JNI-related messages in the user log, including information about which native methods have been linked and warnings about excessive creation of local references.

`-DpropertyName=newValue`
: Redefines a property value. `propertyName` is the name of the property whose value you want to change and `newValue` is the value to change it to.

BEA WebLogic Enterprise JavaServer Options

The following JavaServer options are provided by the BEA WebLogic Enterprise system:

`-Dwle.dynamic`
: Enables hot (runtime) deployment, undeployment, and redeployment of Enterprise JavaBeans on `JavaServers`. By default, this feature is disabled.

Note: If hot deployment is not enabled on the `JavaServer`, attempts to use the `tmadmin` commands `addmodule`, `removemodule`, and `changemodule` will fail and no change will take effect.

`-M` *`number`*
: Enables multithreading for the `JavaServer` and specifies the maximum number of worker threads that a particular instance of `JavaServer` can support. The largest number that you can specify is `500`.

`-M 0` (zero) means that there are no worker threads and that all application code is executed in the single infrastructure thread.

`-M 1` is not useful because there is only one worker thread, which does not provide significant processing parallelism.

If *`number`* is a negative decimal, the server will revert to single-threaded mode. If *`number`* is larger than `500`, the server will use a maximum of 500 worker threads. In all cases, if *`number`* is invalid, the BEA WebLogic Enterprise software logs a warning message to the application's `ULOG` file.

`-noredirect`
: Causes the `System.out` and `System.err` streams to be redirected to the `$APPDIR/stdout` and `$APPDIR/stderr` files, respectively. If `-noredirect` is not specified, the `System.out` and `System.err` streams are redirected to the user log (`ULOG`).

-classic

Specifies that if the Java HotSpot Server VM is installed, the JavaServer should bypass it and load the Classic VM instead. If the `WLE_JVM` environment variable is set equal to hotspot, the -classic option will take precedence.

-hotspot

Specifies that if the Java HotSpot Server VM is installed, the JavaServer should load it. If the `WLE_JVM` environment variable is set equal to classic, the -hotspot option will take precedence.

Note: BEA WebLogic Enterprise supports both classic Java VM and Java HotSpot Server VM on Microsoft Windows NT systems and supports only Java VM on UNIX systems.

Nonstandard Java Virtual Machine Options

The Java Virtual Machine (JVM) in JDK 1.2 supports the nonstandard options in the following list. To display the nonstandard Java virtual machine options, use the `java -X` command at a system prompt.

`-Xdebug`

Allows the Java debugger, `jdb`, to attach itself to this `JavaServer` session. For example:

```
CLOPT = "-A -- -Xbootclasspath:d:\jdk1.2\lib\tools.jar;d:\jdk1.2\jre\lib\rt.jar
-Djava.compiler=NONE -Xdebug BankApp.jar TellerFactory_1"
```

Note: When `-Xdebug` is specified in the command-line options, `JavaServer` prints a password in the user log, which must be used when starting the debugging session.

`-Xmxx`

Sets the maximum size of the memory allocation pool (the garbage collected heap) to `x`. The default is 16 megabytes of memory. The value of `x` must be greater than or equal to 1000 bytes. The maximum memory size must be greater than or equal to the startup memory size (specified with the `-Xms` option, default 16 megabytes). By default, `x` is measured in bytes. You can specify `x` in kilobytes or megabytes by appending the letter `k` for kilobytes or the letter `m` for megabytes.

`-Xmsx`

Sets the startup size of the memory allocation pool (the garbage collected heap) to `x`. The default is 1 megabyte of memory. The value of `x` must be greater than 1000 bytes. The startup memory size must be less than or equal to the maximum memory size (specified with the `-Xmx` option, default 16 megabytes). By default, `x` is measured in bytes. You can specify `x` in kilobytes or megabytes by appending the letter `k` for kilobytes or the letter `m` for megabytes.

`-Xnoclassgc`
: Turns off garbage collection of Java classes. By default, the Java interpreter reclaims space for unused Java classes during garbage collection.

`-Xbootclasspath:bootclasspath`
: Specifies a semicolon-separated list of directories, JAR archives, and ZIP archives to search for boot class files. These are used in place of the boot class files included in the JDK version 1.2 software.

`-Xrs`
: Reduces the use of operating system signals.

`-Xcheck:jni`
: Performs additional check for Java Native Interface (JNI) functions.

`-Xrunhprof[:help]|[:`*`suboption`*`=`*`value`*`,...]`
: Enables CPU, heap, or monitor profiling. This option is typically followed by a list of comma-separated *`suboption`*`=`*`value`* pairs. Run the command `java -Xrunhprof:help` to obtain a list of suboptions and their default values.

JavaServer Options

`-stat`
: If set, enables run-time statistics gathering by various parts of `JavaServer`. Enabling statistics gathering can have adverse impacts on performance.

`-jdbclog`
: If set, causes JDBClog information to be written into the `ULOG`.

Setting the Server Boot Order

You can specify the server boot sequence with the `SEQUENCE` parameter, using a number in the range of 1 to 10,000. A server given a smaller `SEQUENCE` value is booted before a server with a larger value. If two servers have the same `SEQUENCE` value, they are booted simultaneously (that is, the second server can be started before the first server is finished booting).

If no servers specify `SEQUENCE`, servers are booted in the order of their appearance within the `SERVERS` section. If there is a mixture of sequenced and unsequenced servers, the sequenced servers are booted first. Servers are shut down in reverse order of the boot sequence.

This is an optional parameter. The `SEQUENCE` parameter may be helpful in a large application where control over the order is important. Also, the parallel booting may speed the boot process.

Warning: On a BEA WebLogic Enterprise system, there is a strict order in which the system Event Broker, the FactoryFinder object, and the application factories must be booted. A BEA WebLogic Enterprise application program will not boot if the order is changed. See the section “Required Order in Which to Boot Servers (BEA WebLogic Enterprise Servers)” on page 3-49 for details.

You can boot multiple servers using the `MIN` parameter, which is a shorthand method of booting. The servers all share the same server options. On a BEA Tuxedo system, if you specify `RQADDR`, the servers will form an `MSSQ` set (not supported on a BEA WebLogic Enterprise system). The default for `MIN` is 1.

You specify the maximum number of servers that can be booted with the `MAX` parameter. The `tmboot`(1) command boots up to `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 large applications to keep the size of the configuration file manageable. Allowances for MAX values must be made in the IPC resources.

Characteristics of the SEQUENCE, MIN, and MAX Parameters

Table 3-16 lists the SEQUENCE, MIN, and MAX parameters characteristics.

Table 3-16 SEQUENCE, MIN, and MAX Parameters Characteristics

Parameter	Characteristics
SEQUENCE	Is an optional parameter with a numeric range of 1 to 10,000. Smaller values are booted before larger values. The same values can be booted in parallel. Omitted values are booted in the order that they appear in the SERVERS section. Sequenced parameters are booted before any unsequenced parameters.
MIN	Represents the minimum number of servers to boot during run time. If RQADDR is specified and MIN>1, an MSSQ set is created. MSSQ sets apply only to the BEA Tuxedo system. All instances have the same server options. SRVIDs are SRVID + n - 1. The default is 1.
MAX	Represents the maximum number of servers to boot. Defaults to MIN.

Required Order in Which to Boot Servers (BEA WebLogic Enterprise Servers)

The following is the correct order in which to boot the servers on a BEA WebLogic Enterprise system. A BEA WebLogic Enterprise application program will not boot if the order is changed.

1. The system Event Broker, TMSYSEVT.
2. The TMFFNAME server with the -N option and the -M option, which starts the NameManager service (as a Master). This service maintains a mapping of application-supplied names to object references.

3. The TMFFNAME server with the -N option only, to start a Slave NameManager service.
4. The TMFFNAME server with the -F option, to start the FactoryFinder object.
5. The application JavaServers and C++ servers that are advertising factories.

Listing 3-3 shows the order in which servers are booted for the BEA WebLogic Enterprise University Basic application, which is one of the sample applications included with the BEA WebLogic Enterprise software. This SERVERS section is excerpted from an edited version of the ubb_b.nt configuration file.

Listing 3-3 Edited SERVERS Section from a University Sample UBBCONFIG

```
*SERVERS
    # By default, restart a server if it crashes, up to 5 times
    # in 24 hours.
    #
    DEFAULT:
        RESTART = Y
        MAXGEN  = 5

    # Start the BEA Tuxedo System Event Broker. This event broker
    # must be started before any servers providing the
    # NameManager Service
    #
    TMSYSEVT
        SRVGRP  = SYS_GRP
        SRVID   = 1

    # TMFFNAME is a BEA WebLogic Enterprise provided server that
    # runs the NameManager and FactoryFinder services.

    # The NameManager service is a BEA WebLogic Enterprise-specific
    # service that maintains a mapping of application-supplied names
    # to object references.

    # Start the NameManager Service (-N option). This name
    # manager is being started as a Master (-M option).
    #
    TMFFNAME
        SRVGRP  = SYS_GRP
        SRVID   = 2
        CLOPT   = "-A -- -N -M"

    # Start a slave NameManager Service
```

```
#
TMFFNAME
    SRVGRP  = SYS_GRP
    SRVID   = 3
    CLOPT   = "-A -- -N"

# Start the FactoryFinder (-F) service
#
TMFFNAME
    SRVGRP  = SYS_GRP
    SRVID   = 4
    CLOPT   = "-A -- -F"

# Start the interface repository server
#
TMIFRSVR
    SRVGRP  = SYS_GRP
    SRVID   = 5

# Start the university server
#
univb_server
    SRVGRP  = ORA_GRP
    SRVID   = 6
    RESTART = N

# Start the listener for IIOP clients
#
# Specify the host name of your server machine as
# well as the port. A typical port number is 2500
#
ISL
    SRVGRP  = SYS_GRP
    SRVID   = 7
    CLOPT   = "-A -- -n //TRIXIE:2500"
```

In the example, after the `TMSYSEVT` and `TMFFNAME` servers are started, servers are started for:

- An Interface Repository. For information about this feature and the command-line options (`CLOPT` parameter), see Chapter 8, “Managing Interface Repositories (BEA WebLogic Enterprise Systems).”

- The `univb_server`, for the University Basic sample application. For details about the sample applications, see the *Guide to the University Sample Applications*.
- An Internet Inter-ORB Protocol (IIOP) Server Listener (also known as an ISL). For information about this feature and the `CLOPT` parameter, see Chapter 12, "Managing Remote Client Applications (BEA WebLogic Enterprise Systems)."

Note: When migrating or shutting down and restarting groups or machines for any reason, if there are *active* slave NameManagers in other groups, be sure to organize your `UBBCONFIG` file so that a FactoryFinder or a slave NameManager is *never* restarted before the master NameManager is *active*. For example, if you have a FactoryFinder in the same group as the master NameManager, arrange the order of these servers in the `UBBCONFIG` file so the master NameManager is started first.

Identifying Server Environment File Location

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

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 for one server only.
- For overlapping variables, the setting in the `SERVERS` section `ENVFILE` overrides the setting in the `MACHINES` section `ENVFILE`.

Identifying Server Queue Information

Server queue information controls the creation of, and access to, server message queues. On a BEA Tuxedo system, you can create multiple server single queue (`MSSQ`) sets using the `RQADDR` parameter. For any given server, you can set this parameter to an alphanumeric value. Those servers that offer the same set of services can consolidate their services under one message queue, providing automatic load balancing. You can do this by specifying the same value for all members of the `MSSQ` set.

Note: MSSQ sets are not supported on a BEA WebLogic Enterprise system.

MSSQ Example (BEA Tuxedo Servers)

An MSSQ set must include servers that offer the same set of services only. The `MSSQ` set is similar to a situation at a bank. If you have four tellers, one line may be formed and everyone is assured of the most equitable wait in line. Understandably, the loan teller is not included because some people do not want loans on a given day. Similarly, `MSSQ` sets are not allowed if the participant servers offer different services from one another.

The `RQPERM` parameter allows you to specify the permissions of server request queues, similar to the UNIX system convention (for example, 0666). This 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. This is done 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

Table 3-17 lists the `RQADDR`, `RQPERM`, `REPLYQ`, and `RPPERM` parameters characteristics.

Table 3-17 MSSQ Set Parameter Characteristics

Parameter	Characteristics
`RQADDR`	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. **Note:** MSSQ sets are specific to the BEA Tuxedo system.

Table 3-17 MSSQ Set Parameter Characteristics (Continued)

Parameter	Characteristics
RQPERM	Represents the permissions on a request queue. If no parameter is specified, the permissions of the Bulletin Board, as specified by PERM in the RESOURCES section, is used. If no value is specified there, the default of 0666 is used. This opens your application to possible use by any login on the system.
REPLYQ	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, REPLYQ should be set to Y 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.
RPPERM	Assigns permissions to the reply queue. This parameter is useful only when REPLYQ=Y. If requests and replies are read from the same queue, only RQPERM is needed; RPPERM 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 booted will not be 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 specifies a command to be performed in parallel with restarting a server. This command must be an executable file. The option lets you take some action when a server is being restarted. For example, mail could be 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. The `RESTART` and associated parameters should have different settings during the testing phase than they do during production.

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

Table 3-18 lists the `RESTART`, `RCMD`, `MAXGEN`, and `GRACE` parameters characteristics.

Table 3-18 Server Restart Parameters Characteristics

Parameter	Characteristics
`RESTART`	A setting of `Y` enables a server to restart. The default is `N`.
`RCMD`	Specifies a command to be performed in parallel with restarting a server. Lets you take an action when a server is being restarted.
`MAXGEN`	Represents the maximum number of server lives in a specific interval. It defaults to 1; the maximum is 256.
`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.

Specifying a Conversational Server (BEA Tuxedo Servers)

If a server is a conversational server (that is, it establishes a connection with a client), 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.

This feature is specific to BEA Tuxedo servers.

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 Security Parameters for ISL Servers

The IIOP Listener (ISL) process listens for remote clients requesting a connection. The ISL process is specified in one entry as a server supplied by the BEA WebLogic Enterprise system.

The Secure Socket Layer (SSL) protocol defines how processes can communicate in a secure manner over IIOP. Use the -s option on the ISL command to set the required parameters. You only need to set these parameters if you are using the SSL protocol, which is installed in the BEA WebLogic Enterprise Security Pack.

Table 3-19 lists the SSL parameters characteristics.

Table 3-19 ISL and SSL Parameters Characteristics

Parameter	Characteristics
SEC_PRINCIPAL_NAME	Specifies the identity of the IIOP Listener/Handler.
SEC_PRINCIPAL_LOCATION	Specifies the location of the private key for the IIOP Listener/Handler.
SEC_PRINCIPAL_PASSWORD	Specifies the phrase for the private key of the IIOP Listener/Handler.

For more information about setting these parameters, see *Using Security*.

Defining Server Access to Shared Memory

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

This parameter overrides the value specified in the RESOURCES section unless the NO_OVERRIDE value was specified. In this case, the parameter is ignored.

The SYSTEM_ACCESS parameter has the following characteristics:

- A value of PROTECTED indicates that the server may not attach to shared memory outside of the 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 RESOURCES value.

Configuring Modules

When you use BEA WebLogic Enterprise 5.1 or later software to write applications in Java, you specify the JavaServer parameters in the SERVERS and MODULES sections of the UBBCONFIG file.

Notes: In BEA WebLogic Enterprise 5.1 and later software, a MODULES section was added to the UBBCONFIG file to replace the MODULE line(s) and the *archive_file* and *options* parameters in the JavaServer command-line options (CLOPT) in the SERVERS section. The MODULES section must follow the SERVERS section in the UBBCONFIG file.

The MODULES section format is as follows:

```
*MODULES
<module_name>     SRVGRP=<group_name>
                  SRVID=<server_id>
                  FILE=<filename>
                  [ARGS=<arguments>]
                  [CLASSPATH=<local_classpath>]
[<module_name2>    ...]
```

The MODULES section parameters are as follows:

`<module_name>`
: Identifies this module within this domain. This is a logical name that will be used when referring to this module (required parameter).

`SRVGRP=<group_name>`
: Specifies the name of the server group (required parameter).

`SRVID=<server_id>`
: Specifies the ID of the server (required parameter).

`FILE=<filename>`
: Specifies the JAR file that contains the EJB or CORBA descriptor, the remote and home interfaces, the implementation classes for the remote and home interfaces, the classes for the stubs and keys, and so on (required parameter). The JAR file typically contains the class that actually implements the module's business logic, but it is not required to do so.

`ARGS=<arguments>`
: Provides values that will be used to initialize the module (optional parameter).

`CLASSPATH=<local_classpath>`
: Indicates a "local class path" that can specify additional classes that may be required by the main JAR (optional parameter). For example, this parameter can be used for third party utility classes, business libraries, and so on. This parameter follows the standard Java class path semantics and is searched after searching the system/server class path and the main JAR of the module. For the BEA WebLogic Enterprise 5.1 release, only JAR or ZIP files may be specified.

  For all files, if an absolute path is given, that path will be used. If a relative path is specified, it will be relative to the `APPDIR` specified earlier in the `UBBCONFIG` file. Since `APPDIR` represents one or more paths (separated by ";"), all path combinations are checked when a relative path is used and the first match is used.

Note: The `UBBCONFIG` parser will recognize illegal fields and inform the user. The order of the fields does not matter.

The `MODULES` section syntax allows a module to be deployed to a specific server group and server, for example:

```
<module_name>  SVRGRP=<group>  SRVID=<id>  FILE=<jar_file>
```

Or to a subset of servers in a particular group, for example:

```
 <module_name>  SVRGRP=<group>  SRVID=<id1>  FILE=<jar_file>
<module_name>  SVRGRP=<group>   SRVID=<id2>  FILE=<jar_file>
```

In the last case, note that it is permissible to repeat the module name as long as different server groups and/or server ids are used. In the case where the exact same module name, server group, and server id occur more than once, `tmloadcf` will indicate an error. To minimize repetition and reduce the possibility of error, the use of the `DEFAULT` section is encouraged. For example:

```
*MODULES
   DEFAULT: SVRGRP=<group> FILE=<jar_file>
   <module_name> SVRID=<id1>
   <module_name> SVRID=<id2>
```

Note: For compatibility purposes, when the `tmloadcf` command encounters a BEA WebLogic Enterprise 5.0 version of a JavaServer `MODULE` in a `UBBCONFIG` file, it is converted to BEA WebLogic Enterprise 5.1 format and stored in the `TUXCONFIG` file. The module name is constructed based on the JAR file (without the `.jar` extension). When a `UBBCONFIG` file is regenerated using `tmunloadcf`, it will always use the BEA WebLogic Enterprise 5.1 format.

Example of a MODULES Section for an EJB JavaServer

Listing 3-4 shows an example of a `MODULES` section for an EJB JavaServer.

Listing 3-4 EJB JavaServer Example

```
*MODULES
   ejb_basic_statefulSession
     SRVGRP=APP_GRP
     SRVID=5

   FILE="D:\test\ejb\basic\statefulSession\ejb_basic_statefulSession.jar"
```

Note: The order in which the EJB JavaServer parameters are specified does not matter.

Configuring JDBC Connection Pools (BEA WebLogic Enterprise System)

The `JDBCCONNPOOLS` section applies only to the BEA WebLogic Enterprise system. This section must be placed after the `SERVERS` section in the configuration file. This section is used to configure connection pooling for Java Database Connectivity (JDBC). Pooling of JDBC connections is provided by the BEA WebLogic Enterprise infrastructure to conserve resources and improve performance. Each entry in the section represents a JDBC connection pool. This section has the following characteristics:

- For JDBC drivers version 2.0 or later (the default), the entries in this section start with the names of connection pools.
- The `SRVID` and `SRVGRP` attributes must refer to a JavaServer that is specified in the `SERVERS` section.
- Only the `SRVGRP`, `SRVID`, `MAXCAPACITY`, and `CAPACITYINCR` attributes are required for entries. `TESTTABLE` must be specified if `REFRESH` is specified, or if `TESTONRELEASE` or `TESTONRESERVE` are set to `Y`.
- Some attributes are dependent on the version of the JDBC driver.

Table 3-20 lists JDBC connection pool entries attributes.

Table 3-20 JDBC Connect Pool Entries Attributes

Attribute	Value Required? /Default	Allowable Values	Description
`SRVID`	Yes / N/A	A number from 1 to 30001	A server ID listed in the `SERVERS` section. In conjunction with SRVGRP, this attribute identifies the JavaServer for which the connection pool is being configured.
`SRVGRP`	Yes / N/A	A string of up to 30 characters	Name of a server group for identifying the JavaServer for which the connection pool is being configured.

Table 3-20 JDBC Connect Pool Entries Attributes (Continued)

Attribute	Value Required? /Default	Allowable Values	Description
`DRIVER`	Yes for JDBC versions earlier than 2.0 / None	A string of up to 256 characters	The Java class name in the case of a driver that is not JDBC 2.0-compliant.
`URL`	Yes for JDBC driver versions prior to 2.0 / None	A string of 0 to 256 characters	URL for a JDBC driver that is not JDBC 2.0-compliant.
`DBNAME`	No / None	A string of 0 to 30 characters	The database name.
`DBUSER`	No / None	A string of 0 to 30 characters	The user account name that will access the database for this BEA WebLogic Enterprise application.
`DBPASSWORD`	No / None	A string of 0 to 64 characters	The user password for the user account that will access the database for this BEA WebLogic Enterprise application.This can be specified as clear text or it can be encrypted using `tmloadcf`.
`USERROLE`	No / None	A string of 0 to 30 characters	The SQL role of the user account that will access the database for this BEA WebLogic Enterprise application.
`DBHOST`	No / None	A string of 0 to 30 characters	The hostname of the database server.
`DBNETPROTOCOL`	No / None	A string of 0 to 30 characters	The network protocol used to communicate with the database.
`DBPORT`	No / None	A number from 0 to 65535	The port number used for database connections.

Table 3-20 JDBC Connect Pool Entries Attributes (Continued)

Attribute	Value Required? /Default	Allowable Values	Description
`PROPS`	Yes for JDBC driver versions prior to 2.0 / None	A string of 0 to 256 characters	The vendor-specific properties of the JDBC driver. This information can be encrypted. See “Encrypting Passwords in OPENINFO” on page 3-34.
`ENABLEXA`	No / N	`Y` or `N`	Indicates whether the connection pool will be used with an XA-compliant driver. For applications using the BEA WebLogic Enterprise JDBC/XA driver, this value must be set to `Y`.
`CREATEONSTARTUP`	No / Y	`Y` or `N`	Indicates whether the connection pool will be created when the server is started. Otherwise the pool is created when the first request arrives.
`LOGINDELAY`	No / 0	Any number 0 or greater	The number of seconds to wait between each attempt to open a connection to the database. Some database servers cannot handle multiple requests for connections in rapid succession. This property allows you to build in a small delay to allow the database server to catch up.
`INITCAPACITY`	No / The default is the value of the `CAPACITYINCR` parameter	Any number 0 or greater	The number of connections initially supported in the connection pool. This should not exceed the value of the `MAXCAPACITY` parameter.
`MAXCAPACITY`	Yes / None	Any number 0 or greater	The maximum number of connections supported by the connection pool.
`CAPACITYINCR`	Yes / None	Any number 0 or greater	The number of connections added to the pool when the current limit is exceeded, but the maximum capacity is not yet reached.
`ALLOWSHRINKING`	No / N	`Y` or `N`	Indicates that the connection pool’s number of connections can return to the initial capacity, after expanding to meet demands. Shrinking only closes unused connections.

Table 3-20 JDBC Connect Pool Entries Attributes (Continued)

Attribute	Value Required? /Default	Allowable Values	Description
`SHRINKPERIOD`	No / 15	Any number 1 or greater	The length of time (in minutes) during which the JavaServer shrinks the pool to its initial capacity if additional connections are not used.
`TESTTABLE`	Yes if `REFRESH` is specified or if the parameter `TESTONRELEASE` or `TESTONRESERVE` is set to `Y`. Default: None	A string of 0 to 256 characters	The name of a database table that is used to test the viability of connections in the connection pool. The `query select count(*) from TESTTABLE` is used to test a connection. The table must exist and be accessible to the database user for the connection.
`REFRESH`	No / None	Any number 0 or greater	Defines a time interval (in minutes) for tests performed on the connection pool. This parameter is used in conjunction with the `TESTTABLE` parameter to enable automatic refreshes of connections in pools. At the specified interval, each unused connection in the pool is tested by executing an SQL query on the connection. If the test fails, the connection's resources are dropped and a new connection is created to replace it.
`TESTONRESERVE`	No / N	`Y` or `N`	Indicates whether the JavaServer tests a connection after removing it from the pool and before giving it to the client.
`TESTONRELEASE`	No / N	`Y` or `N`	Indicates whether the JavaServer tests a connection before returning it to the connection pool. If all connections in a pool are in use and a client is waiting for a connection, the client will wait longer while the connection is tested. This feature requires that a value be set for the `TESTTABLE` parameter.

Table 3-20 JDBC Connect Pool Entries Attributes (Continued)

Attribute	Value Required? /Default	Allowable Values	Description
WAITFORCONN	No / If WAITTIMEOUT is specified, the default is N. If WAITTIMEOUT is not specified, the default is Y.	Y or N	Indicates whether an application waits indefinitely for a connection if none is currently available. If set to N, the request for a connection returns to caller.
WAITTIMEOUT	No / None	Any number 0 or greater	Defines the interval (in seconds) for an application to wait for a connection to become available. WAITFORCONN and WAITTIMEOUT are mutually exclusive.

Encrypting DBPASSWORD and PROPS

Both DBPASSWORD and PROPS specify sensitive data that you may want to encrypt. Values for these attributes can be encrypted in the UBBCONFIG file using the tmloadcf and tmunloadcf utilities.

To store a value for DBPASSWORD or PROPS in encrypted form, you initially use a text editor to enter a string of five or more continuous asterisks in the parameter value in place of the password in the UBBCONFIG file. This string of asterisks is a placeholder for the password. The following is a sample DBPASSWORD statement illustrating this:

```
DBPASSWORD="*******"
```

When tmloadcf encounters this string of asterisks, it prompts the user to select a password. For example:

```
>tmloadcf -y e:/wle5/samples/atmi/bankapp/xx
DBPASSWORD ("pool2" SRVGRP=GROUP1 SRVID=5):
```

After entering the password, tmloadcf stores the password in the TUXCONFIG in encrypted form. If you use tmunloadcf to generate a UBBCONFIG file, the encrypted password entered is written into the DBPASSWORD statement in the UBBCONFIG file with @@ as delimiters. The following is a sample DBPASSWORD statement generated by tmunloadcf:

```
DBPASSWORD="@@A0986F7733D4@@"
```

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

Use of encrypted passwords is only recommended for production environments. Clear-text passwords can be used during application development.

Configuring Services (BEA Tuxedo System)

This section applies only to BEA Tuxedo systems. For information relevant to BEA WebLogic Enterprise systems, see the section "Configuring Interfaces (BEA WebLogic Enterprise Servers)" on page 3-68.

Note: Although each BEA WebLogic Enterprise interface is mapped to a BEA Tuxedo service, you do not have to configure these services in the `SERVICES` section of the application's `UBBCONFIG` file. As the administrator, you only need to account for the generated services in the `MAXSERVICES` parameter in the `RESOURCES` section. For more information, see the section "Defining IPC Limits."

Identifying BEA Tuxedo Services in the SERVICES Section

You indicate specific information about BEA Tuxedo services in your application in the `SERVICES` section of the configuration file. Such information, for nontransactional, nondistributed applications, 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`)

Sample SERVICES Section

The following example provides a sample 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 set to 50; the one service declared is a RINGUP service that accepts a ringup VIEW as its required buffer type.

Enabling Load Balancing

If you set the RESOURCES section parameter LDBAL to Y, server load balancing occurs. A LOAD factor is assigned to each service performed, which 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 attempt to constantly 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 across sites.

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

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.

Controlling the Flow of Data by Service Priority

You can control the flow of data in an application by assigning service priorities using the `PRIO` parameter. 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. The system dequeues every tenth request in `FIFO` order to prevent a message from waiting indefinitely on the queue.

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

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`.

Specifying Different Service Parameters for Different Server Groups

You can specify different load, priority, or other service-specific parameters for different server groups. To do this, you should repeat the service's entry for each group with different values for the `SRVGRP` parameter.

Sample SERVICES Section

The following example provides a sample `SERVICES` section of a configuration file:

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

This example assigns different service-specific parameters to two different server groups. Service A assigns 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`.

Specifying a List of Allowable Buffer Types for a Service

Using the BUFTYPE parameter, you can tune a service to check buffer types independently of the actual service code. This parameter specifies a list of allowable buffer types for a service. Its syntax is a semicolon-separated list of types in the format `type[:subtype[,subtype]]`. The subtype may be set to `*` to allow all subtypes.

If the BUFTYPE parameter for a service is set to ALL, then this service accepts all buffer types. If this parameter is not specified, the default is ALL.

Examples of the BUFTYPE Parameter

Table 3-21 lists the BUFTYPE parameters characteristics.

Table 3-21 BUFTYPE Parameters Characteristics

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

Configuring Interfaces (BEA WebLogic Enterprise Servers)

This section applies only to the BEA WebLogic Enterprise system.

The BEA WebLogic Enterprise software has an INTERFACES section in the UBBCONFIG file. In this section, you define application-wide default parameters for CORBA or EJB interfaces used by the application. For a CORBA interface participating in factory-based routing, you define the interface names and specify the name of the routing criteria that the BEA WebLogic Enterprise system should apply to each interface. Factory-based routing is a feature that lets you distribute processing to specific server groups. Factory-based routing is not currently supported for EJB.

In addition to defining the `INTERFACES` section, you must specify routing criteria in the `ROUTING` section and the names of groups in the `GROUPS` section when you implement factory-based routing. For details about the parameters and more information about factory-based routing, see the section "Configuring Routing" in this chapter.

Specifying CORBA Interfaces in the INTERFACES Section

You indicate specific information about CORBA interfaces used by your application in the `INTERFACES` section of the configuration file. There are no required parameters. CORBA interfaces need not be listed if no optional parameters are desired. The `INTERFACES` section includes the following types of information:

- Whether transactions should be started automatically (`AUTOTRAN`) (CORBA only)
- The routing criteria to be used for factory-based routing for this CORBA interface (`FACTORYROUTING`) (CORBA only)
- Load balancing information (`LOAD`)
- Assignment of priorities to interfaces (`PRIO`)
- Different service parameters for different server groups (`SRVGRP`)
- Timeout value for transactions associated with this CORBA interface (`TRANTIME`)
- Timeout value for processing a method for this CORBA or EJB interface (`TIMEOUT`)

Table 3-22 lists the `AUTOTRAN`, `FACTORYROUTING`, `LOAD`, `PRIO`, `SRVGRP`, `TRANTIME`, and `TIMEOUT` parameters characteristics.

Table 3-22 INTERFACES Section Parameters Characteristics

Parameter	Characteristic
`AUTOTRAN = {Y | N }`	For each CORBA interface, set `AUTOTRAN` to `Y` if you want a transaction to start automatically when an operation invocation is received. `AUTOTRAN=Y` has no effect if the interface is already in transaction mode. The default is `N`. The effect of specifying a value for `AUTOTRAN` is dependent on the transactional policy specified by the system designer in the implementation configuration file (ICF) or Server Description File (XML) for the interface. This transactional policy will become the transactional policy attribute of the associated `T_IFQUEUE MIB` object at run time. The only time this value actually affects the behavior of the application is if the system designer specified a transaction policy of *optional*. **Note:** To work properly, this feature may be dependent on personal communication between the system designer and the system administrator. If the system administrator sets this value to `Y` without prior knowledge of the ICF or XML parameters set by the programmer, the actual run-time effort of the parameter might be unknown. **Note:** `AUTOTRAN=Y` is not supported for EJB.
`FACTORYROUTING = criterion-name`	**Note:** Specify the name of the routing criteria to be used for factory-based routing for this CORBA interface. You must specify a `FACTORYROUTING` parameter for interfaces requesting factory-based routing. This feature is not supported for EJB.
`LOAD = number`	This is an arbitrary number between 1 and 100 that represents the relative load that the CORBA interface is expected to impose on the system. The numbering scheme is relative to the `LOAD` numbers assigned to other CORBA interfaces used by this application. The default is 50. The number is used by the BEA WebLogic Enterprise system to select the best server to route the request.
`PRIO = number`	Specify the `dequeuing` priority number for all methods of the CORBA interface. The value must be greater than 0 and less than or equal to 100. 100 is the highest priority. The default is 50.
`SRVGRP = server-group-name`	Use `SRVGRP` to indicate that any parameter defined in this portion of the `INTERFACES` section applies to the interface within the specified server group. For a given CORBA interface, this feature lets you define different parameter values in different server groups.

Table 3-22 INTERFACES Section Parameters Characteristics (Continued)

Parameter	Characteristic
`TRANTIME = number`	If `AUTOTRAN` is set to `Y`, you must set the `TRANTIME` parameter, which is the transaction timeout in seconds, for the transactions to be computed. The value must be greater than or equal to zero and must not exceed `2,147,483,647` (2^{31} - 1), or about 70 years. A value of `0` (zero) implies there is no timeout for the transaction. (The default is `30` seconds.) This feature is not supported for EJB.
`TIMEOUT=number`	The amount of time, in seconds, to allow for processing of a method for this CORBA interface. The values must be greater than or equal to 0. A value of 0 indicates that the interface cannot time out. A timed-out method causes the server processing the method for the interface to terminate with a `SIGKILL` event. You should consider specifying a timeout value for the longest-running method for the interface.

Specifying FACTORYROUTING Criteria (CORBA only)

For each CORBA interface, the `INTERFACES` section specifies what kinds of criteria the interface routes on. The `INTERFACES` section specifies the routing criteria via an identifier, `FACTORYROUTING`.

University Sample

The University Production sample application demonstrates how to code factory-based routing (see Listing 3-5). You can find the `UBBCONFIG` files (`ubb_p.nt` or `ubb_p.mk`) for this sample in the directory where the BEA WebLogic Enterprise software is installed. Look in the `\samples\corba\university\production` subdirectory.

Listing 3-5 Production Sample INTERFACES Section

```
*INTERFACES

   "IDL:beasys.com/UniversityP/Registrar:1.0"
        FACTORYROUTING = STU_ID

   "IDL:beasys.com/BillingP/Teller:1.0"
        FACTORYROUTING = ACT_NUM
```

The preceding example shows the fully qualified interface names for the two interfaces in the University Production sample. The `FACTORYROUTING` identifier specifies the names of the routing values, which are `STU_ID` and `ACT_NUM`, respectively.

To understand the connection between the `INTERFACES` `FACTORYROUTING` parameter and the `ROUTING` section, see the section “Example: Factory-based Routing (BEA WebLogic Enterprise Servers)” on page 3-76.

Bankapp Sample

Listing 3-6 shows how factory-based routing is specified in the Bankapp sample application.

Listing 3-6 Bankapp Sample Factory-based Routing

```
*INTERFACES
        "IDL:BankApp/Teller:1.0"
         FACTORYROUTING=atmID

*ROUTING
     atmID
           TYPE = FACTORY
           FIELD = "atmID"
           FIELDTYPE = LONG
           RANGES = "1-5:BANK_GROUP1,
                     6-10: BANK_GROUP2,
                       *:BANK_GROUP1
```

In this example, the `IDL:Bankapp/Teller` interface uses a factory-based routing scheme called `atmID`, as defined in the `ROUTING` section. In the `ROUTING` section, the sample indicates that the processing will be distributed across two groups. `BANK_GROUP1` processes interfaces used by the application when the `atmID` field is between 1 and 5, or greater than 10. `BANK_GROUP2` processes interfaces used by the application when the `atmID` field is between 6 and 10, inclusive.

Enabling Load Balancing

In BEA WebLogic Enterprise systems, load balancing is always enabled.

A `LOAD` factor is assigned to each CORBA interface invoked, which keeps track of the total load of CORBA interfaces that each server process has performed. Each interface 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 CORBA interface requested.

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

Controlling the Flow of Data by Interface Priority

You can control the flow of data in a BEA WebLogic Enterprise client or server application by assigning interface priorities using the `PRIO` parameter. For instance, Server 1 offers Interfaces A, B, and C. Interfaces A and B have a priority of `50` and Interface C has a priority of `70`. An interface 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. The system dequeues every tenth request in `FIFO` order to prevent a message from waiting indefinitely on the queue.

The `PRIO` parameter has the following characteristics:

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

Specifying Different Interface Parameters for Different Server Groups

You can specify different load, priority, or other interface-specific parameters for different server groups. To do this, you should repeat the interface's entry for each group with different values for the `SRVGRP` parameter.

Configuring Routing

The `ROUTING` section of `UBBCONFIG` allows the full definition of the routing criteria named in the `INTERFACES` section (for BEA WebLogic Enterprise factory-based routing) or in the `SERVICES` section (for BEA Tuxedo data-dependent routing).

For more information about using these parameters to implement factory-based routing or data-dependent routing, see Chapter 5, "Distributing Applications."

Defining Routing Criteria in the ROUTING Section

Table 3-23 identifies the information required for an entry in the `ROUTING` section.

Table 3-23 ROUTING Section Parameters Characteristics

Parameter	Characteristics
`criterion_name`	This is a string value with a maximum length of 15 characters. For BEA Tuxedo data-dependent routing, the routing criterion name that you specified as the `ROUTING` parameter in the `SERVICES` section. For BEA WebLogic Enterprise factory-based routing, the routing criteria name that you specified as the `FACTORYROUTING` parameter in the `INTERFACES` section.
`TYPE`	Specifies the routing type. The default is `TYPE=SERVICE` to ensure that existing `UBBCONFIG` files used in BEA Tuxedo environments continue to work properly. Use `TYPE=FACTORY` if you are implementing factory-based routing for a BEA WebLogic Enterprise interface.

Table 3-23 ROUTING Section Parameters Characteristics (Continued)

Parameter	Characteristics
`FIELD`	The name of the buffer field on which the routing is to be done. In BEA Tuxedo data-dependent routing, the name of an `FML` field (for `FML` buffers) or `VIEW` structure element name (for `VIEW` buffers). This is the actual field that is used to route the message. It may be of any data type. In BEA WebLogic Enterprise factory-based routing, this value specifies the name of the routing field. The maximum length is 30 characters. It must correspond to a field name specified for factory-based routing in a factory's call to `TP::create_object_reference` (C++) or `com.beasys.Tobj.TP::create_object_reference` (Java) for the interface.
`FIELDTYPE`	Specifies the type of the routing field. Field types supported are: `SHORT` $-2^{15} \ldots 2^{15} - 1$ (16 bit) `LONG` $-2^{31} \ldots 2^{31} - 1$ (32 bit) `FLOAT` IEEE single-precision floating point numbers `DOUBLE` IEEE double-precision numbers `CHAR` A single character; an 8-bit quantity `STRING` A null-terminated character array

`RANGES`	The limits assigned to each criteria. The syntax is:
	`RANGES="[val1[-val2]:group1]`
	`[,val3[-val4]:group2]...[,*:groupn]"`
	`val1` is a value, `val1-val2` is a range, `group<n>` is either a group name or the wildcard character (*) denoting all group names. `val` can be a numeric literal, a string enclosed in single quotes (' '), `MIN` or `MAX`; a wildcard in place of a range is *Catch-All*, or *No Limit* to the number of ranges.

`BUFTYPE`	For BEA Tuxedo data-dependent routing, 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` and `VIEW` types can be used for routing. The syntax is the same as the `SERVICES` section, a semicolon-separated list of `type:subtype[,subtype]`. You can specify only one type for a routing criteria. This restriction limits the number of buffer types allowed in routing services.

Specifying Range Criteria in the ROUTING Section

The RANGES parameter provides the actual mapping between field value and group name. Its syntax is as follows:

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

where *val1*, and so on, are values of that field and *group<n>* 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, what to do if the data does not fall into any range yet specified. *val1* would be a numeric literal for numeric fields, and would be enclosed in single quotes (‘ ’) for STRING or CARRAY fields. The field values MIN and MAX (not enclosed in quotes) are provided to allow *machine minimum and maximum* data values to be expressed. There is no limit to the number of ranges that may be specified, but all routing information is stored in shared memory and incurs a cost there.

Note: Overlapping ranges are allowed, but will map to the first group. For example: RANGES=”0-5:Group1,3-5:Group2”, a range value of 4 would route to Group1.

Example: Factory-based Routing (BEA WebLogic Enterprise Servers)

The University Production sample application demonstrates how to implement factory-based routing. You can find the ubb_p.nt or ubb_p.mk UBBCONFIG files for this sample in the directory where the BEA WebLogic Enterprise software is installed. Look in the \samples\corba\university\production subdirectory.

The following INTERFACES, ROUTING, and GROUPS sections from the ubb_b.nt configuration file show how you can implement factory-based routing in a BEA WebLogic Enterprise application.

The INTERFACES section lists the names of the interfaces for which you want to enable factory-based routing. For each interface, this section specifies what kinds of criteria the interface routes on. This section specifies the routing criteria via an identifier, FACTORYROUTING, as in the example in Listing 3-7.

Listing 3-7 Production Sample INTERFACES Section

```
*INTERFACES

   "IDL:beasys.com/UniversityP/Registrar:1.0"
        FACTORYROUTING = STU_ID

   "IDL:beasys.com/BillingP/Teller:1.0"
        FACTORYROUTING = ACT_NUM
```

The preceding example shows the fully qualified interface names for the two interfaces in the Production sample in which factory-based routing is used. The `FACTORYROUTING` identifier specifies the names of the routing values, which are `STU_ID` and `ACT_NUM`, respectively.

The `ROUTING` section specifies the following data for each routing value:

- The `TYPE` parameter, which specifies the type of routing. In the Production sample, the type of routing is factory-based routing. Therefore, this parameter is defined to `FACTORY.`
- The `FIELD` parameter, which specifies the variable name that the factory inserts as the routing value. In the Production sample, the field parameters are `student_id` and `account_number`, respectively.
- The `FIELDTYPE` parameter, which specifies the data type of the routing value. In the Production sample, the field types for `student_id` and `account_number` are long.
- The `RANGES` parameter, which associates a server group with a subset of the valid ranges for each routing value.

Listing 3-8 shows the `ROUTING` section of the `UBBCONFIG` file used in the Production sample application.

Listing 3-8 Production Sample ROUTING Section

```
*ROUTING

   STU_ID
      FIELD     = "student_id"
      TYPE      = FACTORY
      FIELDTYPE = LONG
      RANGES    = "100001-100005:ORA_GRP1,100006-100010:ORA_GRP2"

   ACT_NUM
      FIELD     = "account_number"
      TYPE      = FACTORY
      FIELDTYPE = LONG
      RANGES    = "200010-200014:APP_GRP1,200015-200019:APP_GRP2"
```

The preceding example shows that Registrar objects for students with IDs in one range are instantiated to one server group, and Registrar objects for students with IDs in another range are instantiated in another group. Likewise, Teller objects for accounts in one range are instantiated to one server group, and Teller objects for accounts in another range are instantiated in another group.

The groups specified by the `RANGES` identifier in the `ROUTING` section of the `UBBCONFIG` file need to be identified and configured. For example, the Production sample specifies four groups: `ORA_GRP1, ORA_GRP2, APP_GRP1,` and `APP_GRP2`. These groups need to be configured, and the machines on which they run need to be identified.

Listing 3-9 shows the `GROUPS` section of the Production sample `UBBCONFIG` file. Notice how the names in the `GROUPS` section match the group names specified in the `ROUTING` section; this is critical for factory-based routing to work correctly. Furthermore, any change in the way groups are configured in an application must be reflected in the `ROUTING` section. (Note that the Production sample packaged with the BEA WebLogic Enterprise software is configured to run entirely on one machine. However, you can easily configure this application to run on multiple machines.)

Listing 3-9 Production Sample GROUPS Section

```
*GROUPS

APP_GRP1
   LMID = SITE1
   GRPNO = 2
   TMSNAME = TMS

APP_GRP2
   LMID = SITE1
   GRPNO = 3
   TMSNAME = TMS

ORA_GRP1
   LMID = SITE1
   GRPNO = 4

OPENINFO = "ORACLE_XA:Oracle_XA+Acc=P/scott/tiger+SesTm=100+LogDir=.+MaxCur=5"

   CLOSEINFO = ""
   TMSNAME = "TMS_ORA"

ORA_GRP2
   LMID = SITE1
   GRPNO = 5

OPENINFO = "ORACLE_XA:Oracle_XA+Acc=P/scott/tiger+SesTm=100+LogDir=.+MaxCur=5"

CLOSEINFO = ""
TMSNAME = "TMS_ORA"
```

Example: Factory-based Routing in the Bankapp Sample Application (BEA WebLogic Enterprise Servers)

Listing 3-10 shows how the `INTERFACES` section extends the Bankapp sample application to use factory-based routing. The sample included with the BEA WebLogic Enterprise software does not contain these parameter settings.

Listing 3-10 Bankapp Sample INTERFACES Section

```
*INTERFACES
        "IDL:BankApp/Teller:1.0"
         FACTORYROUTING=atmID

*ROUTING
     atmID
            TYPE = FACTORY
            FIELD = "atmID"
            FIELDTYPE = LONG
            RANGES = "1-5:BANK_GROUP1,
                      6-10: BANK_GROUP2,
                      *:BANK_GROUP1

*GROUPS
      SYS_GRP
             LMID        = SITE1
                  GRPNO       = 1
      BANK_GROUP1
                  LMID        = SITE1
                  GRPNO       = 2
      BANK_GROUP2
                  LMID        = SITE1
                  GRPNO       = 3
```

In this example, the `IDL:Bankapp/Teller` interface employs a factory-based routing scheme called `atmID`, as defined in the `ROUTING` section. In the `ROUTING` section, the example indicates that the processing will be distributed across the following two server groups:

- `BANK_GROUP1` processes interfaces used by the application when the `atmID` field is between 1 and 5 (inclusive), or greater than 10.
- `BANK_GROUP2` processes interfaces used by the application when the `atmID` is between 6 and 10, inclusive.

Configuring Network Information

You can configure network groups in the NETGROUPS and NETWORK sections of an application's UBBCONFIG file.

Note: For specific information about the tasks involved, see Chapter 6, "Building Networked Applications."

Specifying Information in the NETGROUPS Section

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 a pair of machines may be assigned. 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 releases of the BEA Tuxedo system prior to Release 6.4 are assigned to the DEFAULTNET network group.

Table 3-24 lists the NETGRPNO, NETPRIO, NETGROUP, MAXNETGROUPS, and MAXPENDINGBYTES parameters characteristics.

Table 3-24 NETGROUPS Section Parameters Characteristics

Parameter	Required/ Optional	Description
NETGRPNO = *numeric_value*	Required	A unique network group number that you must assign to use in failover and failback situations. If this entry describes DEFAULTNET, the numeric value must be zero. Communication with pre-version 6.4 releases of the BEA Tuxedo system use only DEFAULTNET.

Table 3-24 NETGROUPS Section Parameters Characteristics (Continued)

Parameter	Required/ Optional	Description
`NETPRIO =` *`numeric_value`*	Optional	The priority of this network group. A pair of machines in multiple network groups of the same priority communicates simultaneously over the circuits with the highest priority. If all network circuits of a certain priority are torn down by the administrator or by network conditions, the next lowest priority circuit is used. Retries of the higher priority circuits are attempted. This value must be greater than zero and less than 8,192. If not specified, the default is 100. **Note:** In version 6.4 of the BEA Tuxedo system, parallel data circuits are prioritized by the network group number (`NETGRPNO`) parameter within the priority group number. In future releases, a different algorithm/mechanism may be used to prioritize parallel data circuits.
`NETGROUP =` *`string_value`*	Required	The network group associated with this network entry. All network entries with a `NETGROUP` parameter of `DEFAULTNET` are represented in the `T_MACHINE` class, while `NETWORK` entries associated with any other `NETGROUP` are represented in the `T_NETMAP` class to interoperate with previous releases.
`MAXNETGROUPS`	Optional	Allows more netgroups to be defined than the default (8).
`MAXPENDINGBYTES`	Optional	`MAXPENDINGBYTES` enables you to configure the maximum size of data waiting for the network to become available. 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`. Because connections were always synchronous in version 6.3 of the BEA Tuxedo system, situation (1) above did not apply.

Sample NETGROUPS 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. Figure 3-1 shows machines A through E in the networks for which they have network addresses.

Figure 3-1 Example of a Network Grouping (netgrp1.gif)

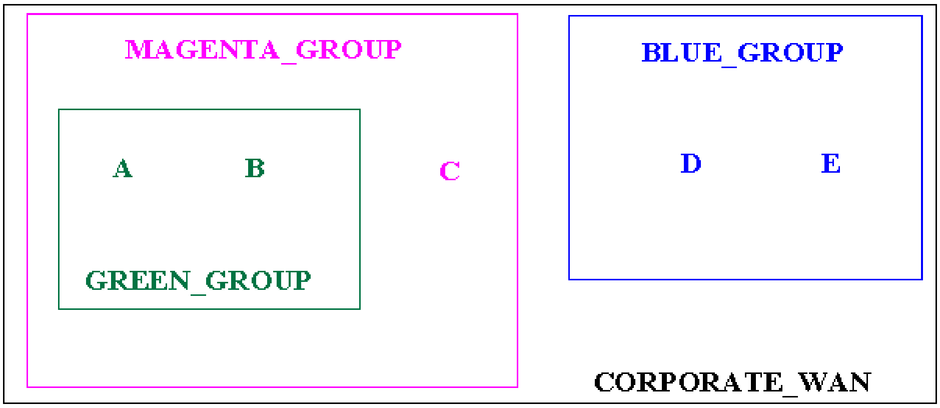


Table 3-25 lists which machines have addresses for which groups.

Table 3-25 Machine Addresses and Groups

Machines	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 the 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 the 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 Listing 3-11.

Listing 3-11 Sample NETGROUPS and NETWORK Sections

```
*NETGROUPS

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

*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"
```

4 Starting and Shutting Down Applications

This chapter describes how to ensure that your application is ready to be booted, how to boot it, and how to shut it down. There are also procedures that help you resolve some problems you may run into when you first begin to start and shut down your BEA WebLogic Enterprise or BEA Tuxedo applications.

This topic includes the following sections:

- Starting Applications
- Shutting Down Applications
- Using tmshutdown
- Clearing Common Problems

Starting Applications

Before you start an application, make sure you have completed all of the tasks in the prerequisite checklist, described in the following section.

Prerequisite Checklist

Complete the following tasks before booting your application:

1. Set Environment Variables.
2. Create TUXCONFIG.
3. Propagate the Software.
4. Create a TLOG Device.
5. Start tlisten at All Sites (MP environments).

Set Environment Variables

Set and export variables `TUXDIR`, `TUXCONFIG`, `PATH`, and `LD_LIBRARY_PATH` so that they are in your environment as the system is booted. For example:

```
TUXDIR=<pathname to installed BEA WebLogic Enterprise or
         BEA Tuxedo directory>
TUXCONFIG=<pathname where TUXCONFIG should go>
PATH=$PATH:$TUXDIR/bin
LD_LIBRARY_PATH=<pathname to shared libraries>
export TUXDIR TUXCONFIG PATH LD_LIBRARY_PATH
```

Replace text within angle brackets (< >) with values for your installation. Other environment variables can be specified in an `ENVFILE` (see `ubbconfig`(5)).

On AIX, `LIBPATH` must be set instead of `LD_LIBRARY_PATH`. On HP UX, `SHLIB_PATH` must be set instead of `LD_LIBRARY_PATH`. On Windows NT, no variable for shared libraries is required.

For BEA WebLogic Enterprise Java, verify that the following environment variables were defined by the Java installation procedure:

- `JAVA_HOME`, the directory where the JDK is installed.
- `CLASSPATH`, which must point to the location of the BEA WebLogic Enterprise Java ARchive (JAR) file (which contains all the class files), and the location of the message catalogs.
- `TUXDIR`, the directory where the BEA WebLogic Enterprise software is installed.

Then, use the new environment variables when you add to your system's PATH, as shown in the following platform-specific examples.

Windows NT system example:

```
set JAVA_HOME=c:\jdk1.2

set CLASSPATH=.;%TUXDIR%\udataobj\java\jdk\wle.jar;%TUXDIR%\locale\java\wle

set PATH=%JAVA_HOME%\bin;%JAVA_HOME%\jre\bin;%JAVA_HOME%\jre\bin\classic;
%TUXDIR%\lib;%TUXDIR%\bin;%PATH%
```

Solaris system example:

```
JAVA_HOME=/usr/kits/jdk1.2

CLASSPATH=.:$TUXDIR/udataobj/java/jdk/wle.jar:$TUXDIR/locale/java/wle

PATH=$JAVA_HOME/bin:$TUXDIR/bin:$PATH

LD_LIBRARY_PATH=$JAVA_HOME/jre/lib/sparc/native_threads:
$JAVA_HOME/jre/lib/sparc/classic:$JAVA_HOME/jre/lib/sparc:$TUXDIR/lib

THREADS_FLAG=native

export JAVA_HOME CLASSPATH PATH LD_LIBRARY_PATH THREADS_FLAG
```

During the deployment step, you must also define the environment variables `TUXCONFIG` and `APPDIR`. These variables are described in subsequent sections of this chapter.

Create TUXCONFIG

`TUXCONFIG` is a binary version of the text configuration file. The `tmloadcf`(1) command converts the configuration file to binary form and writes it to the location given in the `TUXCONFIG` variable.

Enter the command as follows:

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

You may want to consider the following options before you create `TUXCONFIG`:

`-c`
: Calculates the minimum IPC resources of the configuration.

`-n`
: Performs a syntax check only; report errors.

The `-c` and `-n` options do not load the `TUXCONFIG` file.

UNIX IPC resources are platform specific. If you use the `-c` option, check the platform data sheet for your platform in Appendix A of the WebLogic Enterprise or Tuxedo *Installation Guide* to judge whether you need to make some changes. If you do want to change IPC resources, refer to "Defining IPC Limits" on page 3-15.

If the `-n` option indicates syntax errors in the configuration file, correct the errors before you proceed.

For *`ubbconfig_file`*, substitute the fully qualified name of your configuration file.

When you are ready to create the `TUXCONFIG` file, you may want to consider the following options:

`-b`
: Limits the size of the `TUXCONFIG` file.

`-y`
: Overwrites the existing `TUXCONFIG` file without prompting for permission.

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

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

Creating Encrypted Passwords

If you have set up the `UBBCONFIG` file for storing server group or JDBC database passwords in encrypted form, when you run `tmloadcf` to load this `UBBCONFIG` file, you are prompted to create the passwords. `tmloadcf` then stores the passwords in encrypted form in the `TUXCONFIG` file.

To force encryption of a password for a server group, for example, you would write a continuous string of five or more asterisks into the `OPENINFO` string in the `UBBCONFIG` file in the place where the password is to go. For example,

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

When `tmloadcf` encounters a continuous string of five or more asterisks, it will prompt the user to enter a password, for example:

```
>tmloadcf -y e:\wle5\samples\atmi\bankapp\xx
Password for OPENINFO (SRVGRP=BANKB1):
```

The values for the `DBPASSWORD` and `PROPS` atttibutes for JDBC connection pools can also be encrypted in the same manner. Refer to Chapter 3, “Creating a Configuration File”.

Note: If you use the `tmunloadcf` command to convert the binary version of the configuration file back into a `UBBCONFIG` file, an encrypted password is written into the `UBBCONFIG` file with `@@` as delimiters. When a password is stored in the `UBBCONFIG` file in encrypted form, `tmloadcf` does not prompt the user to create a password. The following is a sample `OPENINFO` statement generated by `tmunloadcf`:

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

Propagate the Software

`TUXCONFIG` is automatically propagated by the BEA WebLogic Enterprise or BEA Tuxedo system to all machines in your configuration when you run `tmboot`(1), but there are other files that need to be present on all machines. Table 4-1 is a list of files and directories needed for a networked application.

Table 4-1 Propagating Directories or Files

Directory or File	Comments
`APPDIR`	The directory named in the `APPDIR` variable must be created on each node. It is helpful if this directory has the same pathname on all nodes.
Executables	Application servers must be built once for each platform type, and must be manually propagated to other machines of that platform (that is, BEA WebLogic Enterprise or BEA Tuxedo does not do this automatically). Store the executables in `APPDIR,` or in a directory pointed to in a `PATH` variable in `ENVFILES` in the `MACHINES` section.
Field tables View files	Depending on the requirements of application services (that is, if FML or VIEWS buffer types are used), field tables and view description files must be manually propagated to machines where they are used, then recompiled. Use `mkfldhdr`(1) to make a header file out of a field table file; use `viewc`(1) to compile a view file. The FML field tables and VIEW description files should be available through the environment variables `FLDTBLDIR`, `FIELDTBLS`, `VIEWDIR`, and `VIEWFILES`, or their 32-bit equivalents.
`tlisten`	The `tlisten` process must be started on each machine of a networked BEA WebLogic Enterprise or BEA Tuxedo application. See `tlisten`(1). The `tlisten` process must be started before the application is booted. **Note:** You must define `TUXDIR`, `TUXCONFIG`, `APPDIR`, and other relevant environment variables before starting `tlisten`.

Create a TLOG Device

To enable distributed transaction processing, several parameters in the `MACHINES` section of the configuration file are used to define a global transaction log (`TLOG`). You must create the device list entry for the `TLOGDEVICE` on each machine where a `TLOG` is needed. It can be done before or after `TUXCONFIG` has been loaded, but must be done before the system is booted.

Follow these steps to create an entry in the Universal Device List (UDL) for the `TLOG` device.

1. On the master node with the application inactive, invoke `tmadmin -c`. The `-c` option brings `tmadmin` up in configuration mode.
2. Enter:

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

   where

   `-z` *`config`* specifies the full pathname for the device where the UDL should be created (that is, where the `TLOG` will reside).The value of *`config`* should match the value of the `TLOGDEVICE` parameter in the `MACHINES` section. If *`config`* is not specified, it defaults to the value of the variable `FSCONFIG` (which points to the application's databases).

   `-b` *`blocks`* specifies the number of blocks to be allocated on the device.
3. Repeat steps 1 and 2 on each node of your application that is expected to be involved with global transactions.

If the `TLOGDEVICE` is mirrored between two machines, step 3 is not required on the paired machine. To be recoverable, the `TLOG` should preferably be on a device that can be mirrored. Because the `TLOG` is too small (typically,100 pages) to warrant having a whole disk partition to itself, the expectation is that the `TLOG` will be stored on the same raw disk slice as the application's databases. `FSCONFIG` is the environment variable used by the system. Therefore, the `tmadmin` `crdl` command defaults to `FSCONFIG`.

Start tlisten at All Sites

To have a networked application, a listener process must be running on each machine.

This step is required if you are running the application on more than one machine, as established by the `MODEL MP` parameter in the `RESOURCES` section of the application's `UBBCONFIG` file.

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

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

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

The options to this command are as follows:

`-d` *`device`*

The full pathname of the network device. For the BEA WebLogic Enterprise system and BEA Tuxedo system version 6.4 or later, this option is not required. For earlier versions of the BEA Tuxedo system (version 6.3 and earlier), some network providers (TCP/IP, for example) require this information.

`-l` *`nlsaddr`*

The network address, as specified for this machine (*`LMID`*) in the `NETWORK` section of the configuration file. `nlsaddr`, can be specified in any of the formats that can be specified for the `NADDR` parameter in the same section. If the address has the form `0xhex-digits` or `\\xhex-digits`, it must contain an even number of valid hexadecimal digits.

TCP/IP addresses may be in the //#.#.#.#:*`port`* format or the `//machine-name:port` format.

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

`-u` *`uid-#`* or *`uid-name`*

This parameter can be used to have the `tlisten` process run as the indicated user. This option is required if the `tlisten`(1) command is run by `root` on a remote machine.

`-z` *`[bits]`*

This parameter is specific to BEA Tuxedo systems. When establishing a network link between a BEA Tuxedo administrative process and `tlisten`, it requires at least this minimum level of encryption. Zero (0) means no encryption, while 40 and 128 specify the length (in bits) of the encryption key. If this minimum level of encryption cannot be met, link establishment fails. The default is zero.

-Z [bits]

This parameter is specific to BEA Tuxedo systems. When establishing a network link between a BEA Tuxedo administrative process and `tlisten`, allow encryption up to this level. Zero (0) means no encryption, while 40 and 128 specify the length (in bits) of the encryption key. The default is 128. The `-z` and `-Z` options are available only if either the International or Domestic BEA Tuxedo Security Add-on Package is installed.

Booting the Application

Once the preliminaries have been successfully completed, you are ready to bring up the application, as described in the following section.

Using tmboot

The user who created the `TUXCONFIG` file is considered the administrator of the application. Only this user can execute `tmboot`(1).

The application is normally booted from the machine designated as the `MASTER` in the `RESOURCES` section of the configuration file, or the `BACKUP MASTER` acting as the `MASTER`. The `-b` option allows some deviation from this rule.

For `tmboot`(1) to find executables, the BEA WebLogic Enterprise or BEA Tuxedo system processes, such as the BBL, must be located in `$TUXDIR/bin`. Application servers should be in `APPDIR,` as specified in the configuration file.

When booting application servers, `tmboot`(1) uses the `CLOPT`, `SEQUENCE`, `SRVGRP`, `SRVID`, and `MIN` parameters from the configuration file.

Application servers are booted in the order specified by their `SEQUENCE` parameter, if `SEQUENCE` is used. If `SEQUENCE` is not specified, servers are booted in the order in which they appear in the configuration file.

The command line should look something like the following (this is a greatly simplified example):

```
$ tmboot  [-g grpname] [-o sequence] [-s server] [-S] [-A] [-y]
```

Table 4-2 describes the `tmboot` options.

Table 4-2 tmboot Options

Option	Meaning
`-g` *`grpname`*	Boots all TMS and application servers in groups using this *`grpname`* parameter.
`-o` *`sequence`*	Boots all servers in the order shown in their `SEQUENCE` parameter.
`-s server-name`	Boots individual servers.
`-S`	Boots all servers listed in the `SERVERS` section.
`-A`	Boots all administrative servers for machines listed in the `MACHINES` section. This ensures that the `DBBL`, `BBL`, and `BRIDGE` processes are started in the proper order.
`-y`	Provides an automatic "yes" response to the prompt that asks if all administrative and application servers should be booted. This prompt appears only if no options that limit the scope of the command (`-g` *`grpname`*, for example) are specified.

There are many more options than are shown in the example. For a complete listing of the `tmboot` options, see the `tmboot`(1) reference page in the *BEA Tuxedo Reference Manual*.

Default Boot Sequence for a Small Application

The following scenario shows the order of processing when booting a two-machine configuration. This is not a procedure that you have to initiate; it is what the software does if you enter the following command:

```
prompt> tmboot -y
```

1. `tmboot` comes up on the `MASTER` site and processes the `TUXCONFIG` file, creating a “to do” list for itself.

2. `tmboot` boots the `DBBL` on the `MASTER` machine.

3. `tmboot` boots the `BBL` on the `MASTER` machine, which creates the shared memory Bulletin Board.

4. `tmboot` boots the `BRIDGE` on the `MASTER` machine, which establishes its listening address.
5. `tmboot` establishes a connection with the remote site `tlisten` process and propagates the `TUXCONFIG` file to the remote site if the file is not already there.
6. `tmboot` boots a `BSBRIDGE`. The `BSBRIDGE` establishes a connection back to the `BRIDGE` process on the `MASTER` machine.
7. `tmboot` boots a `BBL`. The `BBL` creates the local Bulletin Board and sends a request to the `DBBL` via the `BSBRIDGE`, to register it as a server. The reply from the `DBBL` contains a complete copy of the `MASTER` Bulletin Board and the `BBL` updates its Bulletin Board with the information.
8. `tmboot` boots a `BRIDGE`. The `BRIDGE` establishes a connection back to the `BRIDGE` on the `MASTER` site, at which point `tmboot` tells the `BSBRIDGE` to go away, since it is no longer needed.
9. `tmboot` can then boot the application servers.
10. `tmboot` boots the local application servers first, then boots the remote application servers.
11. `tmboot` is now finished processing and terminates gracefully.

Optimized Boot Sequence for Large Applications

The boot sequence recommended for larger applications is shown here. This sequence boots entire machines in a single step, rather than taking all the steps used to boot two machines in the default sequence. The optimized sequence can be explained as follows:

1. Boot the entire `MASTER` machine first. This is done by using the `-M -l` combination.
2. Boot the entire remote machine. This is done by using the `-B -l` combination.

This method is faster because the number of system messages is far smaller. In large applications (more than 50 machines), this method generally reduces boot time by 50%.

In a configuration with a slow network, boot time can be improved by first booting the machines that have higher speed connections to the `MASTER` machine.

Shutting Down Applications

The `tmshutdown`(1) command is used to shut down an application.

The `tmshutdown`(1) command is the inverse of the `tmboot`(1) command. It shuts down part or all of the BEA WebLogic Enterprise or BEA Tuxedo application.

When the entire application is shut down, `tmshutdown`(1) removes the IPC resources associated with the BEA WebLogic Enterprise or BEA Tuxedo system.

The options used by `tmboot`(1) for partial booting (`-A`, `-g`, `-I`, `-S`, `-s`, `-l`, `-M`, `-B`) are supported in `tmshutdown`(1). Note that the `-B` option, which allows `tmboot` to be used from a non-`MASTER` machine, is not supported for `tmshutdown`; the `tmshutdown` command must be entered from the `MASTER` (or `BACKUP MASTER`) machine.

If servers are to be migrated, the `-R` option must be used. This option shuts down the servers without removing the Bulletin Board entries.

If a node is partitioned, `tmshutdown`(1) with the `-P` *`lmid`* option can be run on the partitioned machine to shut down the servers on that machine.

`tmshutdown`(1) will not shut down the administrative server BBL on a machine that has clients attached. The `-c` option can be used to override this feature. This option is required when a machine must be brought down immediately and the administrator has been unable to contact the clients.

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

Always check the details of a command such as `tmshutdown`(1) in the *BEA Tuxedo Reference Manual* to make sure you have the most recent information on available options.

Using tmshutdown

The user creating the `TUXCONFIG` file is considered to be the administrator of the application. Only this user can execute `tmshutdown`(1).

The application can be shut down only from the machine designated as `MASTER` in the configuration file. When the `BACKUP MASTER` is acting as the `MASTER`, it is considered to be the `MASTER` for shutdown purposes.

The only exception to this rule is a partitioned machine. By using the `-p` option, the administrator can run the command from the partitioned machine to shut down the application at that site.

Application servers are shut down in the reverse order specified by their `SEQUENCE` parameter, or by reverse order of their appearance in the configuration file. If some servers have `SEQUENCE` numbers and others do not, the unnumbered servers are the first to be shut down, followed by the application servers with `SEQUENCE` numbers (in reverse order). Finally, administrative servers are shut down.

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

Clearing Common Problems

There are several problems that you may encounter when first working with the BEA WebLogic Enterprise system. This section lists and discusses some of the common startup and shutdown problems.

Common Startup Problems

When starting your first BEA WebLogic Enterprise application you might encounter evidence of a problem in the form of a message to `ULOG`, a message to your screen, or both, as follows:

- TLOG Not Created
- Server Not Built Correctly
- Incorrect OPENINFO String
- Unable to Propagate BEA WebLogic Enterprise System

TLOG Not Created

If the transaction log (`TLOG`) fails to get created, a message is sent to the user log (`ULOG`).

The message includes the message catalog name, the unique message number within the catalog, and the reason for the failure. For example, one such message is:

```
CMDTUX 142 ERROR: Identifier for TLOGNAME must be <= len characters in length
```

`TLOGNAME` cannot be more than 30 characters long.

You can avoid problems of this kind if you check the syntax of the `TLOG` parameters in the `MACHINES` section of the `UBBCONFIG` file (see `ubbconfig`(5)).

The `TLOG` also might not get created for the following reasons:

- The person entering the command may lack the proper authority to do so.
- File permissions may not let you write to the device.
- There is not enough space to create the file.

Server Not Built Correctly

A server may not start correctly if either:

- `buildobjserver` fails, or
- `buildobjserver` succeeds but the server comes up with the wrong services

buildobjserver Failure

You should try to detect an error in this area before you attempt to boot a BEA WebLogic Enterprise application. `buildobjserver` is used to compile application code, combining the interfaces to be offered by a server into the executable module. If the code fails to compile, the causes can be that an incorrect compiler was specified, the needed libraries were not found, needed interface modules were not found, there is a problem in the code, and so forth. Pay close attention to the error messages and consult Creating C++ Server Applications and Creating Java Server Applications.

Server Comes Up with Wrong Services (BEA Tuxedo Systems)

Problems in this area can often be attributed to an incorrect `CLOPT` parameter for the server (`CLOPT` is an abbreviation for "command-line options"). The `CLOPT` parameter is assigned in the `SERVERS` section of the `UBBCONFIG` file. It carries command-line options that apply to a server when the server is booted. The options are defined on the `servopts`(5) reference page. Refer to this page and the `ubbconfig`(5) reference page for help on debugging the problem.

Another cause for a server coming up with the wrong services could be an incorrect specification of services when the server is built. While services are usually in a module of code that has a mnemonic name, there is no requirement that this be the case. Service *a*, for example, may actually be performed by function *x*, which could lead to an error.

Incorrect OPENINFO String

The `OPENINFO` string is specified in the `GROUPS` section of the `UBBCONFIG` file. It contains information needed by servers in the group when they try to open an application database. There is a very specific form for the information that is agreed to by vendors of XA-compliant database management systems. The information is stored in the BEA WebLogic Enterprise or BEA Tuxedo system file `$TUXDIR/udataobj/RM`.

Note: After changing the `OPENINFO` string, BEA recommends that you reboot the servers that use this resource manager (RM).

To clear a problem:

1. Check the *System Messages* for an explanation of the error message.

   If this does not resolve the problem, go to step 2.

2. Check the syntax of the `OPENINFO` parameter as specified in the `GROUPS` section of `ubbconfig`(5).

   If the problem persists, go to step 3.

3. Look in `$TUXDIR/udataobj/RM` to see how the information for your DBMS needs to be specified.

Unable to Propagate BEA WebLogic Enterprise System

In a networked application, there are several reasons why the system may not be able to propagate the `TUXCONFIG` file. The generic message is as follows:

```
cannot propagate TUXCONFIG file
```

The following are possible reasons for the failure:

- No listener on the remote machine
- Mismatched address specifications for the listener on the remote machine
- Group ID or the user ID are not the same on both machines
- Access (permissions) problems
- Cannot overwrite an existing `TUXCONFIG` on the slave (remote) machine

Table 4-3 shows a possible solution for each propagation problem.

Table 4-3 Possible Solutions to Propagation Failure

Problem	Solution
Application fails to boot	If `tlisten` password security is enabled, check that the `tlisten` passwords match on both machines. The match is required.
Listener process not started on remote machine	Check that the `TUXDIR, TUXCONFIG, APPDIR`, and other relevant environment variables are set on the remote machine, before starting the listener. Then use the `tlisten`(1) command to start the listener.
Listener started at address different from the `NLSADDR` in the configuration file	Correct the listener address and rerun the `tlisten`(1) command.
Group ID or the user ID are not the same on both machines	Change the IDs to be the same or specify the correct IDs in the `MACHINES` section for that machine.

Table 4-3 Possible Solutions to Propagation Failure

Problem	Solution
Wrong permissions on files/directories on remote machine	Change the permissions to the appropriate values.

Common Shutdown Problems

The two most common problems encountered when shutting down applications are shown with solutions in Table 4-4.

Table 4-4 Two Common Shutdown Problems and Their Solutions

Problem	Solution
Shutting down administrative servers before application servers	The BEA WebLogic Enterprise or BEA Tuxedo system does not allow this action because the administrative servers are needed even after all application servers are shut down. If you want to shut down a machine, you must shut down the application servers before the administrative servers. Use the `tmshutdown` `-l`, `-S`, `-s`, `-g`, and `-I` options before `-A`, `-M`, and `-B`.
Unable to shut down a machine with clients attached	As a rule, the BEA WebLogic Enterprise or BEA Tuxedo system does not allow this. However, if the client cannot be contacted, the `-c` option can be used to shut down the `BBL` while it still has clients attached. There are consequences to client applications that must be considered before taking this action. Try using the `tmshutdown -w` *`delay`* option to shut down servers forcibly after *`delay`* seconds, or use the `tmshutdown -c` option to shut down the `BBL,` even though it has clients attached.

CHAPTER

5 Distributing Applications

For a detailed discussion of distributing applications administrative information, see the chapter *Distributing Applications* in *BEA WebLogic Enterprise Tuning and Scaling*.

6 Building Networked Applications

This topic includes the following sections:

- Terms and Definitions
- Configuring Networked Applications
- Example: A Network Configuration
- Example: A Network Configuration with Multiple Netgroups
- Running a Networked Application

Terms and Definitions

asynchronous connections
: Virtual circuits set up to execute independently of each other or asynchronously. An asynchronous connection does not block the processing of working circuits while attempts are being made to reconnect failed circuits. The BEA Tuxedo system BRIDGE allows the use of nonfailing network paths by listening and transferring data using multiple network address endpoints.

failover and failback
: Network failover occurs when a redundant unit seamlessly takes over the network load for the primary unit. Some operating system and hardware bundles transparently detect a problem on one network card and have a spare

automatically replace it. When done quickly enough, application-level TCP virtual circuits have no indication a fault happened.

In the BEA WebLogic Enterprise or BEA Tuxedo system, data flows over the highest available priority circuit. If 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 is called failover.

When a higher priority circuit becomes available, the data flow is shifted to flow over the higher priority circuit. This is called failback.

When a failover condition is detected, all higher priority circuits are retried periodically. After connections to all network addresses have been tried and failed, connections are tried again the next time data needs to be sent between machines.

multiple listening addresses

Having addresses available on separate networks means that even if one virtual circuit is disrupted, the other circuit can continue undisturbed. Only a failure on all configured networks makes reconnection of the BRIDGES impossible. For example, when a high priority network fails, its load can be switched to an alternate network that has a lower priority. When the higher priority network returns to service, the network load returns to it.

parallel data circuits

Parallel data circuits enable data to flow simultaneously on more than one circuit. When you configure parallel data circuits, network traffic is scheduled over the circuit with the largest network group number (`NETGRPNO`). When this circuit is busy, the traffic is scheduled automatically over the circuit with the next lower network group number. When all circuits are busy, data is queued until a circuit is available.

Note: Alternate scheduling algorithms may be introduced in future releases.

Configuring Networked Applications

To configure a networked application, make these changes in the configuration file.

1. Check the following settings in the `RESOURCES` section:
   - Make sure `MODEL` is set to `MP`.

     `MP` stands for multiprocessor and enables the other networking parameters.
   - Make sure `OPTIONS` is set to `LAN`.

     `LAN` specifies that communication between machines is via a Local Area Network (as opposed to being between two or more processors in a single machine).
   - Use the `MAXNETGROUPS` parameter to set a limit on the number of `NETGROUPS` that can be defined.

     The default is 8; the upper limit 8192.
2. Check the following settings in the `MACHINES` section:
   - `TYPE=`*`string`*. Specifying *`string`* for the machines in your network allows the system to bypass encode/decode processing when messages are transmitted between machines of the same `TYPE`.

     When you identify machines as being of the same `TYPE`, encode/decode processing is not needed. If you have, say, nine SPARC machines and one HP machine, specify `TYPE=` *`string`* only for the HP; for the SPARC machines, the default null string identifies them as being of the same type.
   - `CMPLIMIT=`*`remote,local`*. The `CMPLIMIT` setting specifies thresholds for the point at which message compression should begin. A threshold is a number from 0 to `MAXLONG`. It sets the minimum byte size for a message to be compressed before being sent over the network. For example:

     ```
     CMPLIMIT=1024
     ```

     This parameter specifies that any message greater than 1024 bytes bound for a remote location should be compressed. The absence of a second number means that local messages are never compressed. Compression thresholds can also be specified with the variable `TMCMPLIMIT`. See also the discussion in `tuxenv`(5) of the variable `TMCMPPRFM`. It sets the degree of compression in a range of 1 to 9.
   - `NETLOAD=`*`number.`* Assigns an application-specific number to be added to a remote service's `LOAD` number. The result is used by the system to evaluate whether the request should be processed locally or sent to a remote machine.
3. Check the following settings in the `NETGROUPS` section:

- `NETGROUP.` The name assigned by the application to the particular group. The name can be up to 30 characters long. One group (that includes all machines on the network) must be named `DEFAULTNET`.
- `NETGRPNO=`*`number.`* If this is `DEFAULTNET`, `NETGRPNO` must be zero; for any other group the number can be from 1 to 8192. This parameter is required.
- `NETPRIO=`*`number`*. Assigning a priority to a `NETGROUP` helps the software determine which network connection to use. The number must be between 0 and 8192. Assign higher priority to your faster circuits; give your lowest priority to `DEFAULTNET`.

4. Check the following settings in the `NETWORK` section:

- `LMID.` This Logical Machine Identifier must match one of the entries in the `MACHINES` section. It associates this particular `NETWORK` section entry with one of the application's machines.
- `NADDR=`*`string.`* This network address is the listening address for the `BRIDGE` process on this `LMID`. There are four valid formats for specifying this address. See the `NETWORK` section of `ubbconfig`(5).
- `NLSADDR=`*`string.`* This parameter is the network address for the `tlisten` process on this `LMID`. Valid formats are the same as the valid formats for `NADDR`.
- `NETGROUP=`*`string.`* This must be a `NETWORK` group name previously specified in the `NETGROUPS` section. If not specified, it defaults to `DEFAULTNET`.

Example: A Network Configuration

The following example illustrates the configuration of 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"
```

Example: A Network Configuration with Multiple Netgroups

The hypothetical First State Bank has a network of five machines (A-E). It serves the bank’s business best interest to have four network groups and to have each machine belong to two or three of the four groups.

Note: Configuration of multiple NETGROUPS has both hardware and system software prerequisites that are beyond the scope of this document. For example, NETGROUPS commonly requires machines with more than one directly attached network. Each TCP/IP symbolic address must be identified in the `/etc/hosts` file or in the DNS (Domain Name Services). In the example that follows, addresses in the form “`//A_CORPORATE:5345`” assume that the string “`A_CORPORATE`” is in the `/etc/hosts` file or in DNS.

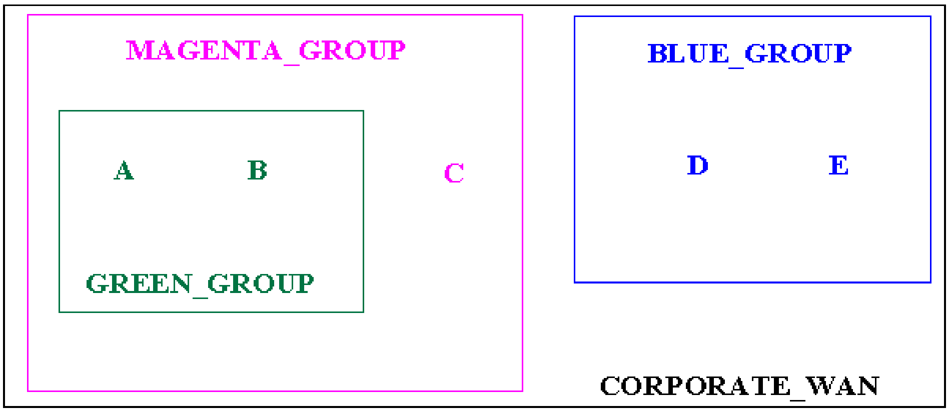
The four groups in the First State Bank example are as follows:

- `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. Figure 6-1 illustrates group assignments for the network.

Figure 6-1 Example of a Network Grouping



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 among the 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.

The UBBCONFIG File for the Network Example

To set up the configuration described in the preceding section, the First State Bank administrator defined 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 = 100
MAGENTA_GROUP NETGRPNO = 125        NETPRIO = 200
GREEN_GROUP   NETGRPNO = 13         NETPRIO = 200

*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"
```

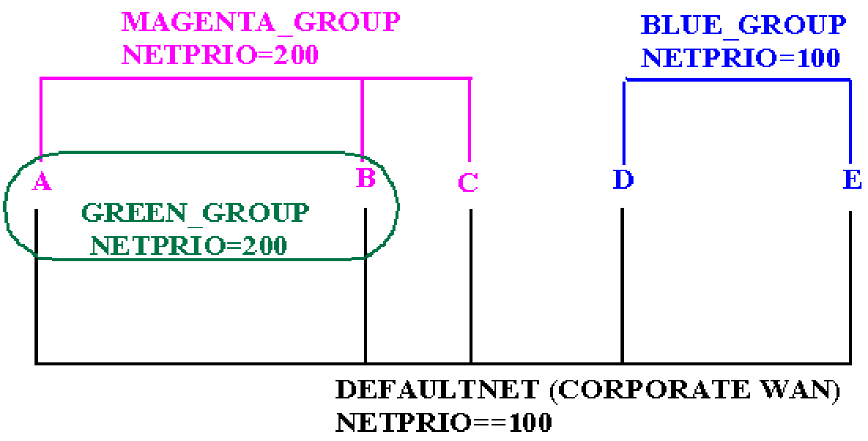
Assigning Priorities for Each Network Group

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

- Data flows over the highest available priority circuit.
- If 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 this higher priority circuit.
- 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.

Figure 6-2 illustrates how the First State Bank administrator can assign priorities to the network groups.

Figure 6-2 Assigning Priorities to Network Groups



The UBBCONFIG Example Considerations

You can specify the value of NETPRIO for DEFAULTNET just as you do for any other netgroup. If you do not specify a NETPRIO for DEFAULTNET, a default of 100 is used, as in the following example:

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

For DEFAULTNET, the value of the network group number must be zero; any other number is invalid. If the BLUE_GROUP's network priority is commented out, the priority defaults to 100. Network group number entries are unique. Some of the network priority values are equal, as in the case of MAGENTA_GROUP and GREEN_GROUP (200).

Each network address is associated by default with the network group, DEFAULTNET. It may be specified explicitly for uniformity or to associate the network address with another netgroup.

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

In this case, MAGENTA_GROUP and GREEN_GROUP have the same network priority of 200. Note that a lower priority network, such as DEFAULTNET, could be a charge-per-minute satellite link.

Running a Networked Application

For the most part, the work of running a BEA WebLogic Enterprise or BEA Tuxedo networked application takes place in the configuration phase. Once you have defined the network for an application and you have booted the system, the software automatically takes care of running the network for you.

In this section, we discuss some aspects of running a networked application to give you a better understanding of how the software works. Knowledge of how the software works can often make configuration decisions easier.

Scheduling Network Data Over Parallel Data Circuits

If you have configured a networked application that uses parallel data circuits, scheduling network data proceeds as follows:

- The `BRIDGE` listens on more than one address and may send data simultaneously on parallel data circuits, thus making the `BRIDGE` more frequently available and making error recovery faster.
- When you configure parallel data circuits, the software 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 lower network group number. When all circuits are busy, data is queued until a circuit is available.
- The software guarantees that conversational messages are kept in the correct sequence by binding the conversation connection to one particular data circuit.
- If your application requires that all messages be kept in sequence, the application must be programmed to keep track of the sequence for nonconversational messages. If this is your design, you might elect not to configure parallel data circuits.
- The `BRIDGE` sends a message to destination machine X by writing the message to a virtual circuit and delegating to the operating system the responsibility for sending it. The operating system retains a copy of pending messages. If a network error occurs, however, pending messages are lost.

Figure 6-3 is a flow diagram that illustrates how the `BRIDGE` processes data by priority.

Figure 6-3 Flow of Data over the BRIDGE

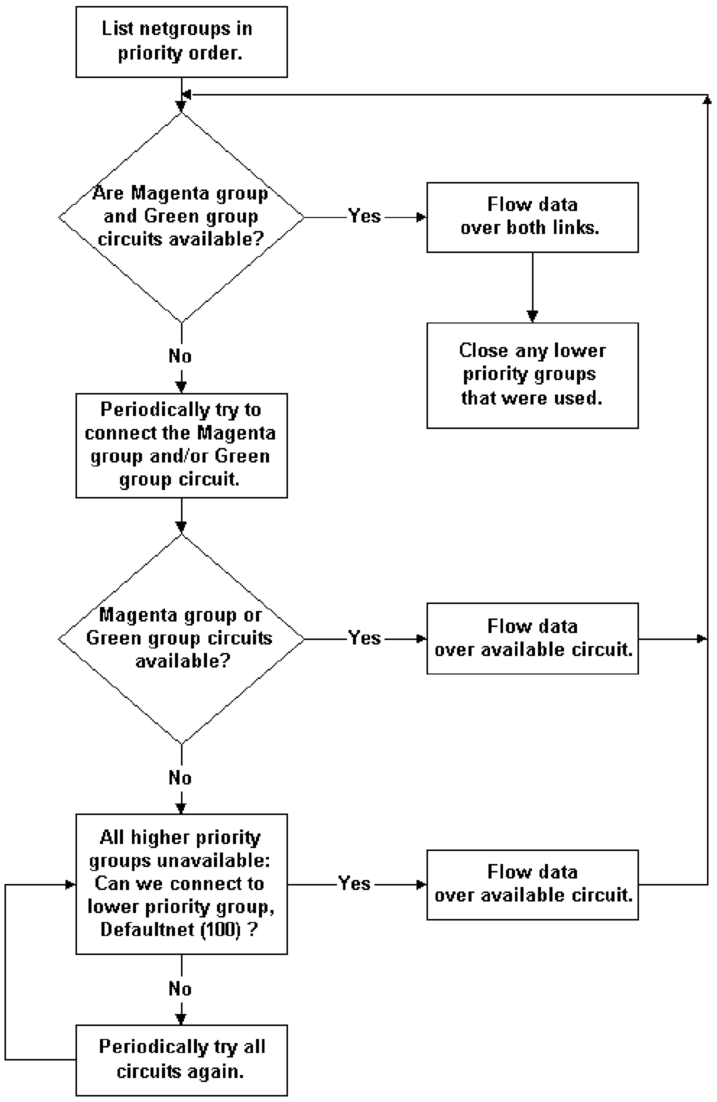

List netgroups in priority order.
Are Magenta group and Green group circuits available?
Yes
Flow data over both links.
Close any lower priority groups that were used.
No
Periodically try to connect the Magenta group and/or Green group circuit.
Magenta group or Green group circuits available?
Yes
Flow data over available circuit.
No
All higher priority groups unavailable: Can we connect to lower priority group, Defaultnet (100) ?
Yes
Flow data over available circuit.
No
Periodically try all circuits again.

Figure 6-3 illustrates the flow of data when machine A attempts to contact machine B. First, the `BRIDGE` determines which network groups are common to both machine A and machine B. They are the `MAGENTA_GROUP`, the `GREEN_GROUP`, and the `DEFAULTNET`.

The highest priority network addresses originate from the network groups with the highest network priority. Network groups with the same `NETPRIO` value flow network data in parallel. All network groups with a higher priority than that of the network groups that are flowing data are retried, periodically.

Once network connections have been established with different `NETPRIO` values, no further data is scheduled for the lower priority connection. The lower priority connection is disconnected in an orderly fashion.

Network Data in Failover and Failback

Data flows over the highest available priority circuit. If 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 is called failover.

When a higher priority circuit becomes available, data flow is restored to the higher priority circuit. This is called failback.

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.

Using Data Compression for Network Data

When data is sent between processes of an application, you can elect to have it compressed. Several aspects of data compression are described in the sections that follow.

Taking Advantage of Data Compression

Data compression is useful in most applications and is in fact vital to supporting large configurations. Following is a list of recommendations for when to use data compression and for how the limits should be set.

When should I set remote data compression and what setting should be used?

You should always use remote data compression as long as all of your sites are running BEA Tuxedo Release 4.2.1 or later. The setting used depends on the speed of your network. In general, you can separate the decision into high-speed (for example, Ethernet) and low-speed (for example, X.25) networks.

High-speed networks. Set remote data compression to the lowest limit for BEA WebLogic Enterprise or BEA Tuxedo generated file transfers (see note below on file transfers). That is, compress only the messages that are large enough to be candidates for file transfer either on the sending site or on the receiving site. Note that each machine in an application may have a different limit and the lowest limit should be chosen.

Low-speed networks. Set remote data compression to zero on all machines; that is, compress all application and system messages.

When should I set local data compression and what setting should be used?

You should always set local data compression for sites running BEA Tuxedo Release 4.2.1 or later, even if they are interoperating with pre-4.2.1 sites. The setting should be the local limit for file transfers generated by the BEA Tuxedo system (see note below). This setting enables you to avoid file transfers in many cases that might otherwise have required a transfer, and greatly reduces the size of files used if file transfers are still necessary.

Note: For high-traffic applications that involve a large volume of timeouts and discarding of messages due to queue blocking, you may want to set local compression to always occur, thus lowering the demand of the application on the queuing subsystem.

Setting the Compression Level

An environment variable, `TMCMPPRFM`, can be used to set the level of compression. This variable adds further control to data compression by allowing you to go beyond the simple choice of “compress or do not compress” that is provided by `CMPLIMIT.` You can specify any of nine levels of compression. The `TMCMPPRFM` environment variable takes as its value a single digit in the range of 1 through 9. A value of 1 specifies the lowest level of compression; 9 is the highest. When a low number is specified, the compression routine does its work more quickly. (See `tuxenv`(5) in the *BEA Tuxedo Reference Manual* for details.)

Balancing Network Request Loads

If load balancing is on (LDBAL set to Y in the RESOURCES section of the configuration file), the BEA WebLogic Enterprise or BEA Tuxedo system attempts to balance requests across the network. Because load information is not updated globally, each site will have its own view of the load at remote sites. This means the local site views will not all be the same.

The TMNETLOAD environment variable (or the NETLOAD parameter in the MACHINES section) can be used to force more requests to be sent to local queues. The value expressed by this variable is added to the remote values to make them appear to have more work. This means that load balancing can be on, but that local requests will be sent to local queues more often.

NETLOAD

The NETLOAD parameter affects the load balancing behavior of a system when a service is available on both local and remote machines. NETLOAD is a numeric value (of arbitrary units) that is added to the load factor of services remote from the invoking client. This provides a bias for choosing a local server over a remote server.

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

Another enhancement to load balancing is local idle server preference. Requests are preferentially sent to a server on the same machine as the client, assuming it offers the desired service and is idle. This decision overrides any load balancing considerations, since the local server is known to be immediately available.

SPINCOUNT

SPINCOUNT determines the number of times a process tries to get the shared memory latch before the process stops trying. Setting SPINCOUNT to a value greater than 1 gives the process that is holding the latch enough time to finish.

Using Link-level Encryption (BEA Tuxedo Servers)

Note: This section is specific to BEA Tuxedo servers; however, see the note below for benefits to BEA WebLogic Enterprise servers.

Link-level encryption (LLE) is the encryption of messages going across network links. This functionality is provided in the BEA Tuxedo system Security Package, which is offered in two versions: U.S./Canada and International. The difference between the two versions consists solely in the number of bits of the 128-bit encryption key that remain private. The U.S./Canada version has a key length of 128 bits; the International version now has an effective key length of 56 bits.

The Security Package allows encryption of data that flows over BEA Tuxedo system network links. The objective is to ensure data privacy, so a network-based eavesdropper cannot learn the content of BEA Tuxedo system messages or application-generated messages.

Link-level encryption applies to the following types of BEA Tuxedo links:

- Workstation client to `WSH`
- `BRIDGE` to `BRIDGE`
- Administrative utilities (`tmboot`, `tmshutdown`, `tmadmin`, and so forth) to `tlisten`
- Domains gateway to Domains gateway

Note: Link-Level Encryption is currently a BEA Tuxedo system feature; however, a BEA WebLogic Enterprise-only customer can benefit from this feature in the following ways:

- `BRIDGE` to `BRIDGE` links
- Administrative utilities (`tmboot`, `tmshutdown`, `tmadmin`, and so forth) to `tlisten`
- Domains gateway to Domains gateway
- BEA Administration Console

Note: Administration Console now supports up to 128-bit encryption for the data communication between the applet and the Administration Console server (`wgated` process). This encryption level is irrespective of the strength of the WebLogic Enterprise encryption level. You can downgrade the encryption to 0-bit, 40-bit, or 56-bit by specifying the parameter `ENCRYPTBITS` in the Administration Console configuration file `webgui.ini`.

How LLE Works

Link-level encryption control parameters and underlying communication protocols are different for various link types, but there are some common themes, as follows:

- A Connecting process begins the communication session.
- An Accepting process receives the initial connection.
- Both processes are aware of the link-level encryption feature, and both have two configuration parameters. (This statement is not true if the processes are interoperating between releases, in which case the older release’s lack of encryption capability is implicitly assumed.)
- The first configuration parameter is the minimum encryption level a process will accept. The value is a number representing the key length: 0, 40, or 128 bits.
- The second configuration parameter is the maximum encryption level a process is willing to support. The value of this parameter is expressed as 0, 40, or 128 bits.
- For convenience, we denote the two parameters as (`MIN`, `MAX`). So (40,128) means that a process will accept at least a 40-bit encryption key but would prefer a 128-bit key, if possible.
- LLE is point-to-point, which means that your data may be encrypted/decrypted many times as it flows over network links.

Encryption Key Size Negotiation

The first step in negotiating the key size is for the two processes to agree on the largest common key size supported by both. This negotiation need not be encrypted or hidden.

Once encryption is negotiated, it remains in effect for the lifetime of the network connection.

A preprocessing step temporarily reduces the maximum key size parameter configured to agree with the installed software's capabilities. This must be done at link negotiation time, because at configuration time it may not be possible to verify a particular machine's installed encryption package. For example, the administrator may configure (0, 128) encryption for an unbooted machine that has only a 40-bit encryption package installed. When the machine actually negotiates a key size, it should represent itself as (0, 40). In some cases this may cause a run-time error; for example (128, 128) is not possible with a 40-bit encryption package.

In some cases, international link level is upgraded automatically from 40 bits to 56 bits. The encryption strength upgrade requires that both sides of a network connection are running BEA Tuxedo Release 6.5 software, with the optional U.S./Canada or International Encryption Security Add-on Package installed. You can verify a server machine's encryption package by running the `tmadmin -v` command. Both machines must also be configured to accept 40-bit encryption. When these conditions are met, the encryption strength is upgraded automatically to 56 bits.

Table 6-1 shows the outcome for all possible combinations of min/max parameters.

Table 6-1 Encryption Key Matrix

Inter-Process Negotiation Results	**(0,0)**	**(0,40)**	**(0, 128)**	**(40, 40)**	**(40,128)**	**(128, 128)**
(0,0)	0	0	0	ERROR	ERROR	ERROR
(0,40)	0	56	56	56	56	ERROR
(0,128)	0	56	128	56	128	128
(40,40)	ERROR	56	56	56	56	ERROR
(40,128)	ERROR	56	128	56	128	128
(128,128)	ERROR	ERROR	128	ERROR	128	128

Note: Shaded cells show the result of an automatic upgrade from 40-bit to 56-bit encryption when both machines are running BEA Tuxedo Release 6.5. When communicating with an older release, encryption remains at 40-bit strength in the shaded cells.

MINENCRYPTBITS/MAXENCRYPTBITS

When a network link is established to the machine identified by the `LMID` for the current entry, the `MIN` and `MAX` parameters are used to specify the number of significant bits of the encryption key. `MINENCRYPTBITS` says, in effect, "at least this number of bits are meaningful." `MAXENCRYPTBITS`, on the other hand, says, "encryption should be negotiated up to this level." The possible values are 0, 40, and 128. A value of zero means no encryption is used, while 40 and 128 specify the number of significant bits in the encryption key.

The BEA Tuxedo system U.S./Canada security package permits use of up to 128 bits; the International package allows specification of no more than 56 bits.

How to Change Network Configuration Parameters

Use `tmconfig`(1) to change configuration parameters while the application is running. In effect, `tmconfig` is a shell-level interface to the BEA Tuxedo system Management Information Base (MIB). See the `tmconfig`(1), `MIB`(5), and `TM_MIB`(5) reference pages in the *BEA Tuxedo Reference Manual*.

CHAPTER

7 Configuring Transactions

For a detailed discussion of transactions administrative information, see the chapter *Administering Transactions* in *BEA WebLogic Enterprise Using Transactions*.

CHAPTER

8 Managing Interface Repositories (BEA WebLogic Enterprise Systems)

This topic, which is specific to BEA WebLogic Enterprise systems, includes the following sections:

- Administration Considerations
- Using Administration Commands to Manage Interface Repositories
- Configuring the UBBCONFIG File to Start One or More Interface Repository Servers

An Interface Repository contains the interface descriptions of the CORBA objects that are implemented within the BEA WebLogic Enterprise domain. Administration of the Interface Repository is done using tools specific to BEA WebLogic Enterprise servers. These tools allow you to create an Interface Repository, populate it with definitions specified in Object Management Group Interface Definition Language (OMG IDL), and then delete interfaces. You may need to configure the system to include an Interface Repository server by adding entries in the application's `UBBCONFIG` file.

For related programming information, see the *CORBA Java Programming Reference* or the *CORBA C++ Programming Reference*.

Administration Considerations

As an administrator, you need to determine whether an Interface Repository is required. Not all systems require it. If an Interface Repository is required, you need to create and populate a repository database. The repository database is created and populated using the `idl2ir` command.

If an Interface Repository is required, you need to answer the following questions:

- How many Interface Repository servers will be required?
- Will the Interface Repository database(s) be replicated?
- Will there be shared access to the Interface Repository database(s)?
- What procedures will be followed for updating the Interface Repository?

You can configure the system to have one or more Interface Repository servers. At least one Interface Repository server needs to be configured if any of the clients use Dynamic Invocation Interface (DII) or ActiveX.

There are two reasons to have more than one server: performance and fault tolerance. From a performance point of view, the number of Interface Repository servers is a function of the number of DII and ActiveX clients. From a fault tolerance point of view, the number of Interface Repository servers needed is determined by the configuration of the system, and the degree of failure protection required.

In systems with more then one Interface Repository server, you must decide whether to have replicated databases, shared databases, or a combination of the two. There are advantages and disadvantages to each configuration. Replicated Interface Repository databases allow for local file access that can potentially increase performance.

The main problem with replicated databases is updating them. All the databases must be identical and this requires the starting and stopping of Interface Repository servers. Having the Interface Repository database mounted and shared eliminates this problem, but this has performance implications and introduces a single point of failure. A combination of the two alternatives is also possible.

Using Administration Commands to Manage Interface Repositories

Use the following commands to manage the Interface Respository for a BEA WebLogic Enterprise domain:

- `idl2ir`
- `ir2idl`
- `irdel`

Prerequisites

Before executing a BEA WebLogic Enterprise command, you must ensure the `bin` directory is in your defined path, as follows:

On Windows NT

```
set path=%TUXDIR%\bin;%path%
```

On UNIX

For c shell (csh): `set path = ($TUXDIR/bin $path)`

For Bourne (sh) or Korn (ksh):
```
PATH=$TUXDIR/bin:$PATH
export PATH
```

To set environment variables:

On Windows NT

```
set var=value
```

On UNIX

For c shell:

```
setenv var value
```

For Bourne and Korn (sh/ksh):

```
var=value
export var
```

Creating and Populating an Interface Respository

Use the `idl2ir` command to create an Interface Repository and load interface definitions into it. If no repository file exists, the command creates it. If the repository file does exists, the command loads the specified interface definitions into it. The format of the command is as follows:

```
idl2ir [options] definition-filename-list
```

For a detailed description of this command, see the *Commands, System Processes, and MIB Reference* in the BEA WebLogic Enterprise online documentation.

Note: If you want changes to be visible, you must restart the Interface Repository servers.

Displaying or Extracting the Content of an Interface Repository

Use the `ir2idl` command to display the content of an Interface Repository. You can also extract the OMG IDL statements of one or more interfaces to a file. The format of the command is as follows:

```
ir2idl [options] [interface-name]
```

For a detailed description of this command, see the *Commands, System Processes, and MIB Reference* in the BEA WebLogic Enterprise online documentation.

Deleting an Object from an Interface Repository

Use the `irdel` command to delete the specified object from the Interface Repository. Only interfaces not referenced from another interface can be deleted. By default, the repository file is `repository.ifr`. The format of the command is as follows:

```
irdel [-f repository-name]  [-i id] object-name
```

For a detailed description of this command, see the *Commands, System Processes, and MIB Reference* in the online documentation.

Note: If you want changes to be visible, you must restart the Interface Repository servers.

Configuring the UBBCONFIG File to Start One or More Interface Repository Servers

For each application that uses one or more Interface Repositories, you must start one or more of the Interface Repository servers provided by the BEA WebLogic Enterprise system. The server name is `TMIFRSVR`. You can add one or more entries for `TMIFRSVR` to the `SERVERS` section of the application's `UBBCONFIG` file.

By default, the `TMIFRSVR` server uses the Interface Repository file `repository.ifr` in the first pathname specified in the `APPDIR` environment variable. You can override this default setting by specifying the `-f filename` option on the command line options (`CLOPT`) parameter.

The following example shows a `SERVERS` section from a sample `UBBCONFIG` file. Instead of using the default file `repository.ifr` in the default directory (`$APPDIR`) where the application resides, the example specifies an alternate file and location, `/usr/repoman/myrepo.ifr`.

Note: Other server entries are shown in the following sample to emphasize that the order in which servers are started for BEA WebLogic Enterprise applications is critical. A BEA WebLogic Enterprise application will not boot if the order is changed. For more information, see the section "Required Order in Which to Boot Servers (BEA WebLogic Enterprise Servers)" on page 3-49 in Chapter 3, "Creating a Configuration File." Notice that the `TMIFRSVR` Interface Repository server is the fifth server started.

```
*SERVERS

    # Start the BEA Tuxedo System Event Broker
    TMSYSEVT
        SRVGRP  = SYS_GRP
        SRVID   = 1

    # Start the NameManager (master)
        SRVGRP  = SYS_GRP
        SRVID   = 2
        CLOPT   = "-A -- -N -M"

    # Start the NameManager (slave)
    TMFFNAME
        SRVGRP  = SYS_GRP
        SRVID   = 3
        CLOPT   = "-A -- -N"

    # Start the FactoryFinder (-F)
    TMFFNAME
        SRVGRP  = SYS_GRP
        SRVID   = 4
        CLOPT   = "-A -- -F"

    # Start the interface repository server
    TMIFRSVR
        SRVGRP  = SYS_GRP
        SRVID   = 5
        RESTART=Y
        MAXGEN=5
        GRACE=3600
        CLOPT="-A -- -f /usr/repoman/myrepo.ifr"
```

For a description of the `TMIFRSVR -f filename` parameter, refer to the *Commands, System Processes, and MIB Reference* in the BEA WebLogic Enterprise online documentation. In addition to the `CLOPT -f filename` parameter, the `TMIFRSVR` parameter can contain other parameters (those that are not specific to the BEA Tuxedo system) in the `SERVERS` section of an application's `UBBCONFIG` configuration file. See the section "Configuring Servers" on page 3-34 for details about parameters such as `SRVGRP`, `SRVID`, `RESTART`, `MAXGEN`, and `GRACE`.

9 Configuring Multiple Domains (BEA WebLogic Enterprise Systems)

BEA WebLogic Enterprise domains are an extension of BEA Tuxedo domains. A domain is a construct that is entirely administrative. There are no programming interfaces that refer to domains. Everything concerning domains is done by configuration files; only an administrator is aware of domains.

This topic includes the following sections:

- Configuring Multiple Domains
- Types of Domain Configurations
- Examples: Configuring Multiple Domains

Overview of Multiple Domains

In the versions 4.0 and 4.1 releases of the BEA WebLogic Enterprise software, a domain was an administrative unit that was entirely self-contained and that described one application. The concept of application in those earlier versions is that of a "logical application" that covers the entire domain. The logical application might well be made up of several individual subapplications with little or no interactions. Only servers described in the domain were available to the applications. In this context, it is correct to say that BEA WebLogic Enterprise version 4.0 and 4.1 systems consisted of only one “local domain.”

Since BEA WebLogic Enterprise software was capable of having only one domain, there was no reason to consider reasons for grouping services one way or another. There was only one way: everything goes into the (single) local domain. However, an enterprise can have many different kinds of applications, be geographically dispersed, and be organized into different areas of responsibility. There might be many separate domains. Each domain is a separately administered unit. Perhaps it is organized for geographical considerations (all the machines in a given location). Perhaps it is organized on departmental grounds within an enterprise (accounting, manufacturing, shipping, and so on).

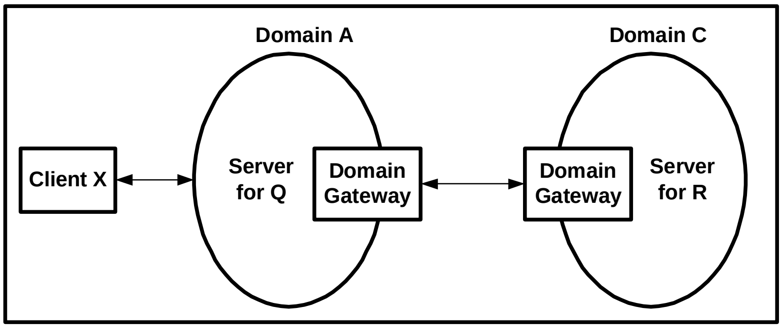
Eventually, an enterprise wants the different applications in those domains to be able to cooperate. It is often impossible to expand a single domain to encompass the enterprise. However, the size of an expanded domain in terms of the number of machines and services would be impractical. Since a single domain must be administered as a whole, the configuration would rapidly become huge and require more effort in administering than in developing and implementing applications.

Therefore, to keep a domain relatively compact for administration, there must be a way to separate applications into multiple domains and still allow applications in one domain to access services in other domains. This capability for interdomain communication is what is generically called “BEA WebLogic Enterprise domains.”

Interdomain Communication

Figure 9-1 shows a simple multiple-domain configuration.

Figure 9-1 Multiple-domain Configuration



The following steps describe single-domain communication between Client X and Domain A:

1. Client X connects to Domain A using the Bootstrap object. The client application uses the Bootstrap object to locate a FactoryFinder and then uses the FactoryFinder to ask for a factory for objects of type Q. (The FactoryFinder call is itself an invocation on Domain A.)
2. When the FactoryFinder returns a factory, the client then invokes that factory in Domain A.
3. The factory returns a reference to an object of type Q, called Q1.
4. The client now invokes on object Q1 in Domain A.

Note: Throughout all of these steps, the client does not know where any of the objects are, or which domains they are in. It might not even know that there is something called a domain. The administrative actions for connecting a client to Domain A are relatively simple for a client, because the client is a simple machine and has very little infrastructure; it stands alone for the most part. Indeed, the connection to a WebLogic Enterprise domain is the primary administration for a client. The actual administrative chore is setting the address of the ISL that is in Domain A.

For multiple-domain communication, Q1 needs the services of Object R1, which is in Domain C; therefore, object Q1 must execute operations similar to those described in steps 1 through 4 above, but across domain boundaries. The actual steps are as follows:

1. Object Q1 uses a Bootstrap object to locate a FactoryFinder and then uses the FactoryFinder to ask for a factory for objects of type R.
2. When the FactoryFinder returns a reference to a factory in Domain C, Object Q1 invokes that factory.
3. The factory returns a reference to an object of type R, called R1.
4. Object Q1 invokes on Object R1.

Note: As with Client X, there must be some administration to allow Object Q1 to get at the factories and objects in Domain C. As Figure 9-1 shows, the mechanism for communication between domains is a domain gateway. A domain gateway is a system server in a domain.

A system server is different than a user-written server because it is provided as part of the WebLogic Enterprise product; other system servers are the name servers, FactoryFinders, and ISLs. A domain gateway is somewhat similar in concept to an ISL because it is the "contact" point for a domain. It is different from an ISL, however, because a domain gateway connects to another domain gateway, which is itself a contact point for a domain; that is, a domain gateway's job is to connect to another domain gateway. Thus, the pair of domain gateways cooperate to make sure that invocation on objects that inhabit different domains are routed to the correct domain.

For domain gateways to operate in this manner, they must be configured properly. That configuration is the subject of the following sections.

Functions of Multiple-domain Configuration Elements

The following elements work together to accomplish the configuration of multiple domains:

- `UBBCONFIG` file

  The `UBBCONFIG` file names a domain and identifies the group and service entry for a domain gateway server. No attributes of domain gateways are specified in the `UBBCONFIG` file; all such attributes are in the `DMCONFIG` file.

- Domain configuration file

  The domain configuration file (`DMCONFIG`) describes the remote domains that are connected to the local domain. If there is no `DMCONFIG` file, there are no connections.

- FactoryFinder domain configuration file

  One FactoryFinder domain configuration file (`factory_finder.ini`) is required for each domain that is connected to one or more other domains. If a domain is not connected to another domain, there is no need for this file.

  This file specifies which factories can be searched for or found across domain boundaries. You must carefully coordinate the `factory_finder.ini` file with the `DMCONFIG` so that they both have information about the same connected domains and provide the same connectivity.

- Invocation of an object in a remote domain

  The whole point of the “BEA WebLogic Enterprise Domains” feature is for a application in one domain to be able to make an invocation on an object in another domain, without either the client or server applications being aware that domains are a factor. Configuration information is intended to allow such invocations to cross domain boundaries and to hide the fact of those boundaries from applications.

  Being able to make an invocation on a reference for an object in a remote domain depends on a satisfactory set of three configuration files—the `UBBCONFIG`, `DMCONFIG`, and `factory_finder.ini` files—for each domain and on the coordination of two of those configuration files—the `DMCONFIG` and `factory_finder.ini` files—between domains. As the number of domains grows, the coordination effort grows.

- References to objects in a remote domain

  Any object reference may specify a local domain or a remote domain. A reference to a remote domain typically happens when a FactoryFinder returns a reference to a factory in a remote domain. It also happens when that factory, in turn, creates and returns a reference to an object in that remote domain (although, of course, the reference is local to the domain of the factory).

Note: Applications are not aware of the domain of an object reference. Applications cannot find out what domain an object reference refers to. Thus, invocations on an object reference for a remote domain are transparent to the application. This transparency allows administrators the freedom to configure services in individual domains and to spread resources across multiple domains. If applications were to include information about domains, changing configurations would require that the applications be rewritten as well.

- FactoryFinders

  For a server in a local domain to obtain an object reference to an object in another domain, the application uses the same FactoryFinder pattern as it does for objects in the local domain. The application uses the same pattern because it is not aware that the factory finder returns a reference to a factory in another domain. The configuration files hide this fact.

  Once an object reference has been obtained via a FactoryFinder or factory, the object reference can be passed anywhere; that is, passed to objects in the local domain, returned to a client, or passed to another domain.

Configuring Multiple Domains

You use the following three configuration files to configure multiple domains:

- The `UBBCONFIG` file
- The domain configuration (`DMCONFIG`) file, and
- The FactoryFinder domain configuration file (`factory_finder.ini`).

The UBBCONFIG File

You must specify the following parameters in the `UBBCONFIG` file to configure multiple domains:

- Domain name
- Gateway group
- Gateway service

Domain Name

Though not required for single domains (that is, standalone domains), a domain that is connected to another domain must have a `DOMAIN ID`. You specify this parameter in the `RESOURCES` section of the `UBBCONFIG` file, as follows:

```
DOMAIN ID = <domain-name>
```

The `<domain-name>` must be 1 to13 characters long. For example:

```
DOMAIN ID = headquarters
```

`<domain-name>` is the name that will be referenced in the `DM_REMOTE_SERVICES` and `DM_LOCAL_SERVICES` sections of the related `DMCONFIG` file. In that file, the `<domain-name>` will be referenced as:

```
"//<domain-name>"
```

The quotes are part of the reference. The slashes (`//`) mean that the name applies to BEA WebLogic Enterprise domains, rather than to BEA Tuxedo domains. For example:

```
"//headquarters"
```

Note: Every domain in an enterprise must have a unique `<domain-name>`.

Gateway Group and Service

As with every other system service, there must be a group and a service name specified for a gateway. For example, the `GROUPS` section might contain:

```
LGWGRP     GRPNO=4     LMID=LDOM
```

In this example, `LGWGRP` is a name chosen by a user (perhaps an abbreviation for "Local Gateway Group").

The service name for a domain gateway is `GWTDOMAIN` and must be associated, like every other group, with a server group and a server ID. You specify the service name in the `SERVERS` section associated with the server group name chosen. For example:

```
GWTDOMAIN SRVGRP=LGWGRP SRVID=1
```

This tells the BEA WebLogic Enterprise server that a domain gateway is to be used and that additional information is found in the `DMCONFIG` file.

The Domain Configuration (DMCONFIG) File

There is one DMCONFIG file per domain. It describes the relationship between the local domain (the domain in which the DMCONFIG file resides) and remote domains (any other domains). The DMCONFIG file contains domain information for BEA Tuxedo domains and for BEA WebLogic Enterprise domains.

The sections below concentrate on the information that applies to BEA WebLogic Enterprise domains. In other documentation for the DMCONFIG file, the communication between local and remote domains is based on BEA Tuxedo services, a concept not used in BEA WebLogic Enterprise. For BEA WebLogic Enterprise, the “service” name is the name of another BEA WebLogic Enterprise domain that can service BEA WebLogic Enterprise requests.

The DMCONFIG file consists of up to eight parts, but one part, DM_ROUTING, does not apply to BEA WebLogic Enterprise domains. The other seven parts refer to BEA WebLogic Enterprise domains, but many of the BEA Tuxedo parameters are not used. Those seven parts are: DM_RESOURCES, DM_LOCAL_DOMAINS, DM_REMOTE_DOMAINS, DM_LOCAL_SERVICES, DM_REMOTE_SERVICES, DM_ACCESS_CONTROL, and DM_TDOMAIN.

The following sections refer to the sample DMCONFIG file shown in Listing 9-1.

Listing 9-1 Sample DMCONFIG File

```
#
# BEA WebLogic Enterprise DOMAIN CONFIGURATION FILE
#
*DM_RESOURCES
VERSION=Experimental8.9

*DM_LOCAL_DOMAINS
LDOM GWGRP=LGWGRP TYPE=TDOMAIN DOMAINID="MUTT"

*DM_REMOTE_DOMAINS
TDOM1 TYPE=TDOMAIN DOMAINID="JEFF"

*DM_TDOMAIN
LDOM   NWADDR="//MUTT:2507"
TDOM1  NWADDR="//JEFF:3186"

*DM_LOCAL_SERVICES
"//MUTT"
```

```
*DM_REMOTE_SERVICES
"//JEFF"    RDOM=TDOM1
```

DM_RESOURCES

The DM_RESOURCES section can contain a single field, VERSION. It is not checked by software; it is provided simply as a place where users can enter a string that may have some documentation value to the application.

```
*DM_RESOURCES
VERSION=Experimental8.9
```

DM_LOCAL_DOMAINS

The DM_LOCAL_DOMAINS section specifies some attributes for gateways into the local domain from the outside. The section must have an entry for each gateway group defined in the UBBCONFIG fle that will provide access to the local domain from other domains. Each entry specifies the parameters required for the domain gateway processes running in that group.

Entries have the form:

```
LDOM required-parameters [optional-parameters]
```

where LDOM is an identifier used to refer to the gateway to the local domain. LDOM must be unique among all LDOM and RDOM entries across the enterprise (that is, among the set of domains connected to each other). Note that LDOM is not the same name as the <domain-name> or the gateway group that is specified in the UBBCONFIG file. Rather, LDOM is a name used only within the DMCONFIG file to provide an extra level of insulation from potential changes in the UBBCONFIG file (changes in UBBCONFIG will affect only this one part of DMCONFIG).

The following are required parameters:

GWGRP = identifier

This parameter specifies the name of a gateway server group (the name provided in the `UBBCONFIG` file) representing this local domain.

`TYPE = TDOMAIN`

The `TYPE` parameter is required to specify the use of domains for BEA WebLogic Enterprise.

`DOMAINID = string`

The `DOMAINID` parameter is used to identify the local domain for the purposes of security. The gateway server group in `GWGRP` uses this string during any security checks. It has no required relationship to the `<domain-name>` found in the `RESOURCES` section of the `UBBCONFIG` file. `DOMAINID` must be unique across both local and remote domains. The value of `string` can be a sequence of characters (for example, `“BA.CENTRAL01”`), or a sequence of hexadecimal digits preceded by 0x (for example, `“0x0002FF98C0000B9D6”`). `DOMAINID` must be 32 octets or fewer in length. If the value is a string, it must be 32 characters or fewer (counting the trailing null).

For example, the lines

```
*DM_LOCAL_DOMAINS
LDOM  GWGRP=LGWGRP  TYPE=TDOMAIN  DOMAINID="MUTT"
```

identify `LDOM` as an access point to the local domain. It is associated with the service group `LGWGRP` (as specified in the `UBBCONFIG` file). If the gateway is ever involved in a domain-to-domain security check, it goes by the name `MUTT`.

Optional parameters describe resources and limits used in the operation of domain gateways. For a description of these parameters, refer the `dmconfig`(5) reference page in the *BEA Tuxedo Reference Manual*.

DM_REMOTE_DOMAINS

The `DM_REMOTE_DOMAINS` section specifies some attributes for gateways to remote domains. The section has an entry for each `UBBCONFIG` file-defined gateway group that will send requests to remote domains. Each entry specifies the parameters required for the domain gateway processes running in that group.

Entries have the form:

```
RDOM required-parameters
```

where RDOM is an identifier used to refer to the gateway providing access to the remote domain. RDOM must be unique among all LDOM and RDOM entries across the enterprise (that is, among the set of domains connected to each other). Note that RDOM is not the same name as the <domain-name> or the gateway group that is specified in the UBBCONFIG file. Rather, RDOM is a name used only within the DMCONFIG to provide an extra level of insulation from potential changes in UBBCONFIG (changes in UBBCONFIG will affect only this one part of DMCONFIG).

The required parameters are:

TYPE = TDOMAIN
: The TYPE parameter is required to specify the use of domains for BEA WebLogic Enterprise.

DOMAINID = string
: The DOMAINID parameter is used to identify the remote domain for the purposes of security. The gateway uses this string during any security checks. DOMAINID has no required relationship to the <domain-name> found in the RESOURCES section of the UBBCONFIG file. DOMAINID must be unique across both local and remote domains. The value of string can be a sequence of characters (for example, "BA.CENTRAL01"), or a sequence of hexadecimal digits preceded by "0x" (for example, "0x0002FF98C0000B9D6"). DOMAINID must be 32 octets or fewer in length. If the value is a string, it must be 32 characters or fewer (counting the trailing null).
Entries associated with a remote domain can be specified more than once. The first one specified is considered to be the primary address, which means it is the first one tried when a connection is being attempted to a remote domain. If a network connection cannot be established using the NWADDR of the primary entry, the NWADDR associated with the secondary entry is used. (NWADDR is the physical address; see the DM_TDOMAIN section.)
For example, the lines

```
*DM_REMOTE_DOMAINS
TDOM1 TYPE=TDOMAIN DOMAINID="JEFF"
```

identify TDOM1 as the access point name of a gateway. If the gateway is ever involved in a domain-to-domain security check with a partner gateway, the gateway expects that partner to go by the name JEFF.

DM_TDOMAIN

The DM_TDOMAIN section defines the network addressing information for gateways implementing BEA WebLogic Enterprise domains. There should be one entry for each domain gateway that accepts requests from remote domains, and one entry for each domain gateway that sends requests to remote domains.

The format of each entry is:

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

where DOM is an identifier value used to identify either a local domain access point (LDOM in the DM_LOCAL_DOMAINS section) or a remote domain access point (RDOM in the DM_REMOTE_DOMAINS section).

The following parameter is required:

NWADDR = string

This parameter specifies the network address associated with a local domain or a remote domain. If the association is with a local domain, the NWADDR is used to accept connections from other domains. If the association is with a remote domain, the NWADDR is used to initiate a connection. This parameter specifies the network address to be used by the process as its listening address. The listening address for a domain gateway is the means by which it is contacted by other gateway processes participating in the application. If string has the form "0xhex-digits" or "\\xhex-digits", it must contain an even number of valid hex digits. These forms are translated internally into a character array containing TCP/IP addresses. The addresses may also be in either of the following two forms:

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

In the first of these formats, hostname is resolved to a TCP/IP host address at the time the address is bound, using the locally configured name resolution facilities accessed via gethostbyname(3c). The "#.#.#.#" is the dotted decimal format, where each # represents a decimal number in the range 0 to 255.

Port_number is a decimal number in the range 0 to 65535 (the hexadecimal representations of the string specified). For example:

```
*DM_TDOMAIN
  LDOM  NWADDR="//MUTT:2507"
  TDOM1 NWADDR="//JEFF:3186"
```

Continuing the example from above, the first entry specifies a gateway with the domain access name of `LDOM` (meaning that it corresponds to the local gateway group `LGWGRP`, specified in `UBBCONFIG`). Since `LDOM` was defined in `DM_LOCAL_DOMAINS`, that means the gateway is configured to accept requests from other domains. It listens on the address `"//MUTT:2507"`. Similarly, the second entry is for the domain access name `TDOM1`, which appears in `DM_REMOTE_DOMAINS`, transferring requests to a remote domain. In this case, the gateway associated with `TDOM1` sends requests to the address `"//JEFF:3186"`.

For a description of the optional parameters, refer to the `dmconfig`(5) reference page in the *BEA Tuxedo Reference Manual*.

DM_REMOTE_SERVICES

The `DM_REMOTE_SERVICES` section specifies additional attributes for gateways to remote domains. The format of each entry is:

```
service  RDOM=<rdom-name>
         [LDOM=<ldom-name>]
         [TRAN_TIME=...]
```

where `service` is of the form:

```
"//<domain-name>"
```

This `<domain-name>` is the name that occurs `RESOURCES` section of the `UBBCONFIG` file as `<domain-name>`. Each entry specifies an `rdom-name` and, optionally, an `ldom-name`. The gateway uses the attributes for those entries for establishing a gateway pair for BEA WebLogic Enterprise domain communication. Gateways operate in pairs. At boot time, the local domain uses attributes of `rdom-name` (the address specified in the `DM_TDOMAIN` section) to establish a connection to a gateway in the other domain. If security is used, the other attributes of `rdom-name` and `ldom-name` are used for mutual authentication. At run time, when BEA WebLogic Enterprise determines that a request must travel to domain `<domain-name>`. It uses the gateway specified by `rdom-name` to send the request to another domain.

Most often, `<domain-name>` is the name of the domain specified in the address of the `rdom-name`. In that situation, when the request ends up at the other end of the gateway, it is served in that domain. For example:

```
*DM_REMOTE_SERVICES
   "//JEFF"    RDOM=TDOM1
```

In this case, the domain name `JEFF` is located at the address `"//JEFF:3186"`. That address might or might not have a `UBBCONFIG` file that specifies its domain name as `JEFF`. If it does, the request can be serviced immediately.

It is possible to have entries that send requests for the specified `domain-name` to an intermediary domain that acts as a pass-through for routing purposes.

The remaining optional parameter, `TRANTIME = integer`, specifies the default timeout value, in seconds, for a transaction automatically started for the associated service. The value must be greater than or equal to 0 (zero) and less than 2147483648. The default is 30 seconds. A value of 0 (zero) implies the maximum timeout value for the machine.

DM_LOCAL_SERVICES

The `DM_LOCAL_SERVICES` section specifies additional attributes for gateways that accept requests into the local domain from the outside.

Lines within this section have the form:

```
service  [LDOM=<ldom-name>]
         [ACL=...]
```

where `service` is of the form:

```
"//<domain-name>"
```

This `<domain-name>` is the name that occurs in the `RESOURCES` section of the `UBBCONFIG` file as `<domain-name>`. Most likely this is the name of the domain in which the gateway resides, meaning that this (local) domain accepts BEA WebLogic Enterprise requests from other domains. It is also possible (but not necessary, except for purposes of security) to have an entry that accepts requests for a different domain name in the case where the local domain acts as a pass-through for routing purposes.

Notice that exported services inherit the properties specified for the service in an entry in the `SERVICES` section of the `TUXCONFIG` file, or their defaults. Some of the properties that may be inherited are `LOAD`, `PRIO`, `AUTOTRAN`, `ROUTING`, `BUFTYPE`, and `TRANTIME`.

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

For example, the lines:

```
*DM_LOCAL_SERVICES
"//MUTT"
```

state that this domain accepts requests destined for the domain with name `MUTT`.

DM_ACCESS_CONTROL

The `DM_ACCESS_CONTROL` section specifies the access control lists used by a local domain. Lines in this section are of the form:

```
ACL_NAME required parameters
```

where `ACL_NAME` is an (identifier) name used to identify a particular access control list; it must be 15 characters or less in length.

The only required parameter is:

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

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

Note: The `factory_finder.ini` and `DMCONFIG` files must be coordinated; that is, if the `factory_finder.ini` file declares another domain to have accessible factories, there must be a way in `DMCONFIG` to get to that domain.

The factory_finder.ini File

Administrators are required to identify any factory objects that can be used in the current (local) /Domain, but that are resident in a different (remote) /Domain. You identify these factories in a FactoryFinder domain configuration file, also referred to as the `factory_finder.ini` file. This is an ASCII file that can be created and updated using a text editor.

The factory_finder.ini file can be used to identify remote CORBA factories and remote EJB Home interfaces that can be used in the local domain.

The format of the `factory_finder.ini` file is modeled after the syntax used to describe /Domains, and is shown below:

```
*DM_REMOTE_FACTORIES
  "local_factory_id.factory_kind"
    DOMAINID="domain_id"
    RNAME="remote_factory_id.factory_kind"
    ...

[*DM_LOCAL_FACTORIES]
  ["factory_id.factory_kind"]
  ...
```

Sample syntax for CORBA factory objects is as follows:

```
*DM_REMOTE_FACTORIES
    "AccountFactory.FactoryKind"
    "DOMAINID="MyAccountFactoryDomain"
    RNAME="MyAccountFactory.FactoryKind
```

where: `AccountFactory` is the name used to register the factory in the local domain's FactoryFinder, `MyAccountFactoryDomain` is the name of the remote domain, `MyAccountFactory` is the name used to register the factory in the remote domain's FactoryFinder.

Sample syntax for EJB Home interfaces in the Java Naming and Directory Interface (JNDI) is as follows:

```
*DM_REMOTE_FACTORIES
   "AccountHome.FactoryKind"
   DOMAINID="MyAccountHomeDomain"
   RNAME="MyAccountHome.FactoryKind"
```

where: `AccountHome` is the name used to register the EJB Home interface in the local domain's JNDI, `MyAccountHomeDomain` is the name of the remote domain, and `MyAccountHomeDomain` is the name used to register the EJB Home interface in the remote domain's JNDI.

The Master NameManager reads the `factory_finder.ini` file when the process is started. The reason for starting the Master NameManager affects which portions of the `factory_finder.ini` file are processed. If the Master NameManager is being started as part of booting an application, the initialization mode, the entire contents of the file is processed. As a result, the information in the `DM_REMOTE_FACTORIES` section results in entries being added for the factory objects or EJB Home interfaces being imported.

On the other hand, if the Master NameManager is being restarted as a result of a process failure, only the `DM_LOCAL_FACTORIES` section of the file is read. This section of the `factory_finder.ini` file must be re-read to reload the information that is used to restrict the exportation of certain factory objects or EJB Home interfaces into another domain.

Note: Since the Master NameManager reads the `factory_finder.ini` file only when the process is started, there is no way to update the Master NameManager (for example, when a new domain with factory objects to be imported needs to be added) without shutting down the Master NameManager.

A `factory_finder.ini` file applies to the domain in which it resides. It contains two sections: the `DM_REMOTE_FACTORIES` section and the `DM_LOCAL_FACTORIES` section. Either section can be absent or contain nothing.

The following sections provide more information on how to use the `DM_REMOTE_FACTORIES` section and the `DM_LOCAL_FACTORIES` section.

DM_REMOTE_FACTORIES

The `DM_REMOTE_FACTORIES` section provides information about the factory objects or EJB Home interfaces that are available in remote domains and that are imported so that applications in the local domain can use them. Identifiers for remote factory objects or EJB Home interfaces are listed in this section. The identifier, under which the object is registered, including a `kind` value of `“FactoryInterface”`, must be listed in this section. For example, the entry for a remote factory object to be registered by the TP Framework with the identifier `Teller` in domain `“Norwest”` would be specified as:

```
*DM_REMOTE_FACTORIES
  "Teller.FactoryInterface"
    DOMAINID="Norwest"
    RNAME="BankTeller.FactoryInterface"
```

If the `RNAME` is not specified, the `factory_kind` must be specified in the factory name and the factory name must be enclosed in quotation marks; otherwise, the NameManager is not able to locate the appropriate factory. An entry that does not contain a `factory_kind` value is not defaulted with a value of `“FactoryInterface”`. The following example shows a factory object to be registered with the identifier `Teller` in domain `“Norwest”`. Note the absence of the `RNAME` specification, the specification of the `factory_kind` value, and the quotation marks around the factory name.

```
*DM_REMOTE_FACTORIES
 "Teller.FactoryInterface"
    DOMAINID="Norwest"
```

Because the identities of factories in a multidomain configuration may collide, the factory identifier and the `RNAME` parameters allow you to specify alternative identities, or “aliases,” in the local domain for remote factories. Listing 9-2 shows two examples of a remote factory that is registered by the TP Framework with the identifier `BankTeller` in domain `“Norwest”`. In both examples, the factory is made available in local domain with an alias of `Teller`.

Listing 9-2 Assigning an Alias to a Remote Factory

```
#EXAMPLE 1:

*DM_REMOTE_FACTORIES
   Teller
     DOMAINID="Norwest"
     RNAME=”BankTeller.FactoryInterface”

#EXAMPLE 2:

*DM_REMOTE_FACTORIES
   "Teller.FactoryInterface"
     DOMAINID="Norwest"
     RNAME="BankTeller.FactoryInterface"
```

You can also assign multiple aliases to the same remote factory. In the example shown in Listing 9-3, the remote factory will be registered in the local domain with two aliases: `Teller` and `BankTeller`.

Listing 9-3 Assigning Multiple Aliases to a Remote Factory

```
*DM_REMOTE_FACTORIES
   "Teller.FactoryInterface"
     DOMAINID="Norwest"
     RNAME="BankTeller.FactoryInterface"
   "BankTeller.FactoryInterface"
     DOMAINID="Norwest"
     RNAME="BankTeller.FactoryInterface"
```

Usage Note: In multidomain configurations, factory object and ejb home interface identifiers must be unique across domains in the enterprise.

In a multidomain configuration, two different domains must *not* have a factory objects or EJB Home interfaces with the same `factory_id.factory_kind` identifier, for example: `"Teller.FactoryInterface"`.

If the same identifier, or name, is used in two domains, the software behavior varies according to the version of the BEA WebLogic Enterprise software.

- In releases prior to 5.1, the BEA WebLogic Enterprise software allows the first server in a domain to register the factory without issuing an error message. If two factories with the same name are registered in a domain, the Master NameManager fails.
- In the 5.1 release, the BEA WebLogic Enterprise software generates an error and writes it to the ULOG.

Note: In a single domain configuration, BEA WebLogic Enterprise supports multiple factories objects or EJB Home interfaces with the same name. This type of configuration is allowed so as to achieve load-balancing.

There are two ways to ensure that your identifiers, or names, are unique across domains and thus avoid this problem:

1. Use unique identifiers throughout the enterprise. This may mean keeping a master list of all identifiers.
2. In the `factory_finder.ini` file, use the `RNAME` parameter so that an alias is used by the local NameManager. (This also means that local clients will have to be modified to use the alias to access the remote factory object or EJB Home interface.) Listing 9-2 shows an example of a `factory_finder.ini` file that uses the `RNAME` parameter to create an alias.

DM_LOCAL_FACTORIES

The `DM_LOCAL_FACTORIES` section specifies factory objects or EJB Home interfaces in the local domain that are available to be exported to other domains. This section can be used in the following ways:

- If the `DM_LOCAL_FACTORIES` section does not exist in a `factory_finder.ini`, or exists but is empty, all factory objects and EJB Home interfaces in the local domain are available to remote domains. This allows administrators an easy means to make local factory objects or EJB Home interfaces available to remote domains without having to provide an entry for every factory object or EJB Home interface in the local domain.

- If the `DM_LOCAL_FACTORIES` section exists in a `factory_finder.ini` file but contains the reserved keyword `"NONE"`, none of the factory objects or EJB Home interfaces in the local domain are available to remote domains. This allows administrators to restrict access without having to provide an entry for every factory object or EJB Home interface in the local domain.

The identifier, or name, under which the factory object or EJB Home interface is registered, including a kind value of `"FactoryInterface"`, must be listed in this section. For example, the entry for a factory object to be registered by the TP Framework with the identifier `Teller` would be specified as:

```
*DM_LOCAL_FACTORIES
  "Teller.FactoryInterface"
```

The `factory_kind` must be specified for the NameManager to locate the appropriate factory object or EJB Home interface. An entry that does not contain a `factory_kind` value is not defaulted with a value of `"FactoryInterface"`. This allows for the use of the CORBA NamingService.

The `factory_finder.ini` file specifies that the process of finding a factory can be exported to a remote domain by including a section beginning with `"*DM_REMOTE_FACTORIES"`. In other words, including this section means that the local domain can find factories in a remote domain.

An entry into the file for domain `A` might be:

```
*DM_REMOTE_FACTORIES
 fA.FactoryInterface DOMAINID=B
```

This means that a request in domain `A` to find a factory with the identifier `fA` can be satisfied by the Factory Finder in domain `B`. Of course, the `UBBCONFIG` and `DMCONFIG` files for the two domains must also be set up so that there are connected domain gateways between the two domains.)

An alternate form of the entry is:

```
CDE.FactoryInterface DOMAINID=B RNAME=fA.FactoryInterface
```

This means that a request in domain A to find a factory with the identifier `"CDE"` will be satisfied by the FactoryFinder in domain `B` using the ID `fA`. This is sometimes called an alias.

Note: The factory ID must have "`.FactoryInterface`" at the end. For simplicity, when talking about test configurations, we will leave that off, but it should appear in the file.

For more information about the `factory_finder.ini` file, see description of the factory_finder.ini file in the *Commands, System Processes, and MIB Reference*.

Local Factories

A domain can specify which of its factories can be accessed by other domains. This is specified in a section beginning with `*DM_LOCAL_FACTORIES`. If the `factory_finder.ini` file does not exist, or if it exists and this section does not appear, or is empty, *all* local factories can be accessed by remote domains. If the section exists and contains the keyword `None`, none of the local factories are exportable; that is, none are allowed to be found by a remote Factory Finder. If the section exists, it can contain a list of factories available to remote domains. For example,

```
*DM_LOCAL_FACTORIES
fA.FactoryInterface
fB.FactoryInterface
```

This specifies that factories `fA` and `fB` are findable from other domains. All factories other than factories explicitly listed are not findable. Unlike remote factories, there is no provision for an alias with local factories.

Note: The `factory_finder.ini` and `DMCONFIG` files must be coordinated, that is, if the `factory_finder.ini` file declares another domain to have accessible factories, there must be a way in `DMCONFIG` to get to that domain.

Types of Domain Configurations

When using the multiple domains feature, you can configure two types of configurations: directly connected domains and indirectly connected domains. You, as the administrator, configure both types using the domain configuration file, `DMCONFIG`.

Directly Connected Domains

It is possible for every domain in an enterprise to have a gateway to every other domain it might use. Such a configuration has the advantage that a request goes directly to the target domain, with the minimum of delay. Such an “n-way” configuration is quite reasonable when the number of domains is small, but each new domain requires two new gateways. At some point, an administrator may consider a different configuration, giving up speed of delivery for ease of management of domain connections. This is when the ability to configure indirectly connected domains becomes advantageous.

Indirectly Connected Domains

An administrator should consider what the likely traffic patterns are. Domains that have only occasional interactions are candidates for gateway removal. Since there will still be interactions, it must still be possible to reach the other domain. The technique used is to route the request through an intermediate domain that does have direct access to the target domain. For example, we might have three domains, `A`, `B`, and `C`. Domains `A` and `B` are directly connected and domains `B` and `C` are directly connected, but `A` and `C` are not directly connected (See Figure 9-2). For domains `A` and `C` to communicate, they must use domain `B` as the intermediary. Therefore, the `DMCONFIG` file for domain `A` must state that it is possible to connect to domain `C` by going through domain `B` (and vice versa). That is, the connectivity is:

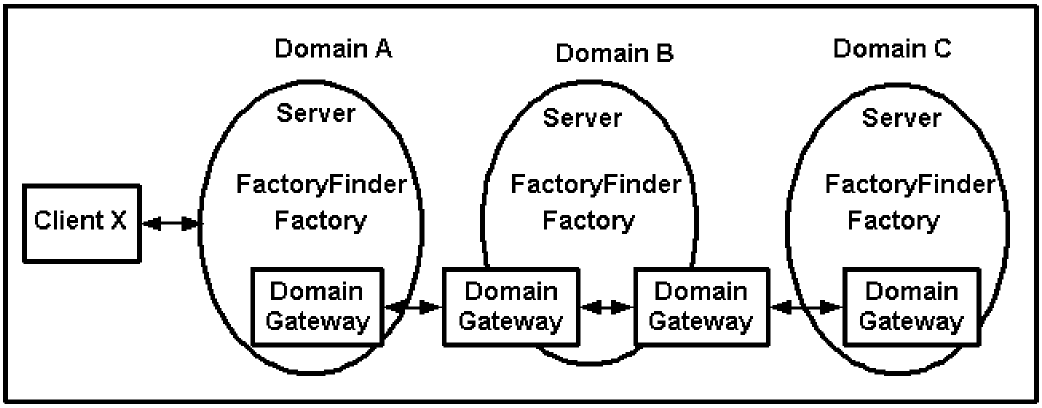
```
Domains       A     <->     B     <->    C
Gateways          GAB  GBA        GBC  GCB
```

Domain `A` has a gateway process, `GAB` (the Gateway from `A` to `B`), that connects to domain `B`. The domain `A` `DMCONFIG` file states that `GAB` acts as a gateway to two domains, domains `B` and `C`. The `DMCONFIG` file for domain `C` has a similar configuration, stating that `GCB` is connected to `B` and `A`. The `DMCONFIG` file for domain `B` has two gateway processes, one which connects to `A` (`GBA`) and one which connects to `C` (`GBC`). This is called an indirect connection.

Given this indirect connection, when a server in `A` invokes a request on an object in `C`, BEA WebLogic Enterprise knows that it can send the request to gateway `GAB`. The BEA WebLogic Enterprise gateway does not know that its partner gateway in `B` cannot service the request itself, but that is acceptable. Once the request is in domain `B`, it is routed through `GBC` to `C`, which can service the request. Thus, the request is serviced with one extra hop.

It is even possible for the two gateways in domain `B` to be a single gateway, so that there is not an extra hop within `B`. In effect, the same processing occurs in domain `B`, but it all occurs within a single gateway process.

Figure 9-2 Indirectly Connected Domains



Examples: Configuring Multiple Domains

The following sections provide examples of how to configure directly connected domains.

Note: These examples are provided for informational purposes only. If you want to use these examples, you will have to change the `APPDIR`, `TUXCONFIG`, and `TUXDIR` variables to match your environment. Also, you will have to substitute appropriate information wherever text is enclosed by left (<) and right (>) angle brackets (for example, <App Server Name>) and delete the angle brackets.

Sample UBBCONFIG Files

Listing 9-4, Listing 9-5, and Listing 9-6 show the `UBBCONFIG` files for three directly connected domains: Here, There, and Yonder.

Note: To use these files, you must replace *`host`* with the name of the local machine.

Listing 9-4 UBBCONFIG File for the Here Domain

```
#
#     Copyright (c) 1999 BEA Systems, Inc.
#     All rights reserved
#
#
#
# RESOURCES
#
*RESOURCES
    IPCKEY    123312
    DOMAINID  HereD
    MASTER    LAPP
    MODEL     SHM
    LDBAL     N


#
# MACHINES
#
*MACHINES
    <host>
           LMID=LAPP
           APPDIR="/tst1/wle4.2/test_dom/t07:
                    /tst1/wle4.2/dec_unix/wlemdomai"
           TUXCONFIG="/tst1/wle4.2/test_dom/tuxconfig"
           TUXDIR="/lclobb/lc"
           MAXWSCLIENTS=10
#
# GROUPS
#
*GROUPS
    DEFAULT:   LMID=LAPP
    ICEGRP     GRPNO=11 OPENINFO=NONE
    GROUP1     GRPNO=21 OPENINFO=NONE
    LDMGRP     GRPNO=3
    LGWGRP     GRPNO=4
#
# SERVERS
#
*SERVERS
    DEFAULT:   CLOPT="-A"
    DMADM      SRVGRP=LDMGRP SRVID=1
    GWADM      SRVGRP=LGWGRP SRVID=1
    GWTDOMAIN  SRVGRP=LGWGRP SRVID=2
    TMSYSEVT   SRVGRP=ICEGRP SRVID=1
    TMFFNAME   SRVGRP=ICEGRP SRVID=2
                   CLOPT="-A -- -N -M -f <FF ini file for Here>"
    TMFFNAME   SRVGRP=ICEGRP SRVID=3 CLOPT="-A -- -N"
    TMFFNAME   SRVGRP=ICEGRP SRVID=4 CLOPT="-A -- -F"
    <App Server Name>    SRVGRP=GROUP1 SRVID=2
    ISL        SRVGRP=GROUP1 SRVID=1
                    CLOPT="-A -- -d /dev/tcp -n //<host>:<port>"

#
# SERVICES
#
*SERVICES
```

Listing 9-5 UBBCONFIG File for the There Domain

```
#
#      Copyright (c) 1999 BEA Systems, Inc.
#      All rights reserved
#
# RESOURCES
#
*RESOURCES
    IPCKEY     133445
    DOMAINID   ThereD
    MASTER     LAPP1
    MODEL      SHM
    LDBAL  N
#
# MACHINES
#
*MACHINES
    <host>
         LMID=LAPP1
         APPDIR="D:\test_dom\t07;D:\Iceberg\qa\orb\bld\wlemdomain"
         TUXCONFIG="D:\test_dom\tuxconfig"
         TUXDIR="D:\Iceberg"
         MAXWSCLIENTS=10
#
# GROUPS
#
*GROUPS
    DEFAULT   LMID=LAPP1
    ICEGRP    GRPNO=11  OPENINFO=NONE
    GROUP1    GRPNO=21  OPENINFO=NONE
    LDMGRP    GRPNO=3
    LGWGRP    GRPNO=4
#
# SERVERS
#
*SERVERS
    DEFAULT:   CLOPT="-A"
    DMADM      SRVGRP=LDMGRP SRVID=1
    GWADM      SRVGRP=LGWGRP SRVID=1
    GWTDOMAIN  SRVGRP=LGWGRP SRVID=2
    TMSYSEV    SRVGRP=ICEGRP SRVID=1
    TMFFNAME   SRVGRP=ICEGRP SRVID=2
                 CLOPT="-A -- -N -M -f <FF ini file for There>"
    TMFFNAME   SRVGRP=ICEGRP SRVID=3 CLOPT="-A -- -N"
    TMFFNAME   SRVGRP=ICEGRP SRVID=4 CLOPT="-A -- -F"
    <App Server Name>     SRVGRP=GROUP1 SRVID=2
    ISL        SRVGRP=GROUP1 SRVID=1
                  CLOPT="-A -- -d /dev/tcp -n //<host>:<port>"
#
# SERVICES
#
*SERVICES
```

Listing 9-6 UBBCONFIG File for the Yonder Domain

```
#      Copyright (c) 1999 BEA Systems, Inc.
#      All rights reserved
#
# RESOURCES
#
*RESOURCES
    IPCKEY    123334
    DOMAINID  YonderD
    MASTER    LAPP
    MODEL     SHM
    LDBAL     N
#
# MACHINES
#
*MACHINES
    <host>
          LMID=LAPP
          APPDIR="/tst1/wle4.2/test_dom/t07p:
                             /tst1/wle4.2/<host3>/wlemdomain"
          TUXCONFIG="/tst1/wle4.2/test_dom/<host3>/tuxconfig"
          TUXDIR="/lclobb/lc"
          MAXWSCLIENTS=10
#
# GROUPS
#
*GROUPS
    DEFAULT:   LMID=LAPP
    ICEGRP     GRPNO=11 OPENINFO=NONE
    GROUP1     GRPNO=21 OPENINFO=NONE
    LDMGRP     GRPNO=3
    LGWGRP     GRPNO=4
#
# SERVERS
#
*SERVERS
    DEFAULT:  CLOPT="-A"
    DMADM     SRVGRP=LDMGRP SRVID=1
    GWADM     SRVGRP=LGWGRP SRVID=1
    GWTDOMAIN SRVGRP=LGWGRP SRVID=2
    TMSYSEVT  SRVGRP=ICEGRP SRVID=1
    TMFFNAME  SRVGRP=ICEGRP SRVID=2
                  CLOPT="-A -- -N -M"
    TMFFNAME  SRVGRP=ICEGRP SRVID=3 CLOPT="-A -- -N"
    TMFFNAME  SRVGRP=ICEGRP SRVID=4 CLOPT="-A -- -F"
    <App Server Name>  SRVGRP=GROUP1 SRVID=2
    ISL       SRVGRP=GROUP1 SRVID=1
                  CLOPT="-A -- -d /dev/tcp -n //<host>:<port>"
#
# SERVICES
#
*SERVICES
```

Sample DMCONFIG File

Listing 9-7, Listing 9-8, and Listing 9-10 show the `DMCONFIG` files for three directly connected domains: Here, There, and Yonder.

Note: To use Listing 9-7 in a multidomain configuration, you must replace *`host1`* with the name of the local machine for the Here domain, replace *`host2`* with the name of the local machine for the There domain, and replace *`host3`* with the name of the local machine for the Yonder domain,

Listing 9-7 DMCONFIG File for the local machine in the Here Domain in a Three-Domain Configuration

```
#
#Copyright (c) 1999 BEA Systems, Inc.
#All rights reserved
#
#
# Tuxedo DOMAIN CONFIGURATION FILE
#
*DM_RESOURCES

    VERSION=U22

#
# DM_LOCAL_DOMAINS
#
*DM_LOCAL_DOMAINS

    LDOM1  GWGRP=LGWGRP  TYPE=TDOMAIN  DOMAINID="HereG"

#
# DM_REMOTE_DOMAINS
#
*DM_REMOTE_DOMAINS

    TDOM1    TYPE=TDOMAIN DOMAINID="ThereG"
    TDOM2    TYPE=TDOMAIN DOMAINID="YonderG"


#
# DM_TDOMAIN
#
*DM_TDOMAIN
```

```
    LDOM1   NWADDR="//<host1>:<tcpport>"
    TDOM1   NWADDR="//<host2>:<tcpport>"
    TDOM2   NWADDR="//<host3>:<tcpport>"
#
# DM_LOCAL_SERVICES
#
*DM_LOCAL_SERVICES
    "//HereD"
#
# DM_REMOTE_SERVICES
#
*DM_REMOTE_SERVICES

    "//ThereD     "RDOM=TDOM1
    "//YonderD    "RDOM=TDOM2
```

Note: To use Listing 9-8 in a multidomain configuration, you must replace *`host1`* with the name of the local machine for the There domain, replace *`host2`* with the name of the local machine for the Here domain, and replace *`host3`* with the name of the local machine for the Yonder domain,

Listing 9-8 DMCONFIG File for the There Domain in a Three-Domain Configuration

Listing 9-9 `#`

```
#Copyright (c) 1999 BEA Systems, Inc.
#All rights reserved
#
#
# Tuxedo DOMAIN CONFIGURATION FILE
#
*DM_RESOURCES
```

```
    VERSION=U22

#
# DM_LOCAL_DOMAINS
#
*DM_LOCAL_DOMAINS
```

```
    LDOM1  GWGRP=LGWGRP  TYPE=TDOMAIN  DOMAINID="ThereG"
#
# DM_REMOTE_DOMAINS
#
*DM_REMOTE_DOMAINS
    TDOM1    TYPE=TDOMAIN DOMAINID="HereG"
    TDOM2    TYPE=TDOMAIN DOMAINID="YonderG"


#
# DM_TDOMAIN
#
*DM_TDOMAIN
    LDOM1   NWADDR="//<host1>:<tcpport>"
    TDOM1   NWADDR="//<host2>:<tcpport>"
    TDOM2   NWADDR="//<host3>:<tcpport>"
#
# DM_LOCAL_SERVICES
#
*DM_LOCAL_SERVICES
    "//ThereD"
#
# DM_REMOTE_SERVICES
#
*DM_REMOTE_SERVICES
    "//HereD     "RDOM=TDOM1
    "//YonderD    "RDOM=TDOM2
```

Note: To use Listing 9-10 in a multidomain configuration, you must replace `host1` with the name of the local machine for the Yonder domain, replace `host2` with the name of the local machine for the Here domain, and replace `host3` with the name of the local machine for the There domain,

Listing 9-10 DMCONFIG File for the Yonder Domain in a Three-Domain Configuration

Listing 9-11 `#`

```
#Copyright (c) 1999 BEA Systems, Inc.
#All rights reserved
#
#
# Tuxedo DOMAIN CONFIGURATION FILE
#
*DM_RESOURCES
```

```
    VERSION=U22

#
# DM_LOCAL_DOMAINS
#
*DM_LOCAL_DOMAINS

    LDOM1  GWGRP=LGWGRP  TYPE=TDOMAIN  DOMAINID="YonderG"

#
# DM_REMOTE_DOMAINS
#
*DM_REMOTE_DOMAINS

    TDOM1    TYPE=TDOMAIN DOMAINID="HereG"
    TDOM2    TYPE=TDOMAIN DOMAINID="ThereG"


#
# DM_TDOMAIN
#
*DM_TDOMAIN

    LDOM1   NWADDR="//<host1>:<tcpport>"
    TDOM1   NWADDR="//<host2>:<tcpport>"
    TDOM2   NWADDR="//<host3>:<tcpport>"
#
# DM_LOCAL_SERVICES
#
*DM_LOCAL_SERVICES
    "//YonderG"
#
# DM_REMOTE_SERVICES
#
*DM_REMOTE_SERVICES

    "//HereD     "RDOM=TDOM1
    "//ThereD    "RDOM=TDOM2
```

Sample factory_finder.ini File

This section shows the `factory_finder.ini` files for the Here and There domains. The Yonder domain does not require a `factory_finder.ini` file.

Listing 9-12 factory_finder.ini File for the Here Local Domain

```
#Copyright (c) 1999 BEA Systems, Inc.
#All rights reserved
#
# Factory Finder Initialization file for Domain "Here".
# This is the local Domain.
#
# DM_LOCAL_FACTORIES
#
*DM_LOCAL_FACTORIES
    "AFactory.FactoryInterface"
#
# DM_REMOTE_FACTORIES
#
*DM_REMOTE_FACTORIES
    "AFacYonder.FactoryInterface"
       DOMAINID="YonderD"
       RNAME="AFactory.FactoryInterface"

    "BFactory.FactoryInterface"
       DOMAINID="YonderD"
```

Listing 9-13 factory_finder.ini File for the There Remote Domain

```
#
#Copyright (c) 1999 BEA Systems, Inc.
#All rights reserved
#
# Factory Finder Initialization file for Domain "There".
#This is a remote domain.
#
# DM_LOCAL_FACTORIES
#
*DM_LOCAL_FACTORIES
    "AFactory.FactoryInterface"
```

```
#
# DM_REMOTE_FACTORIES
#
*DM_REMOTE_FACTORIES
    "AFacYonder.FactoryInterface"
DOMAINID="YonderD"
RNAME="AFactory.FactoryInterface"
 "BFactory.FactoryInterface"
DOMAINID="YonderD"
```

CHAPTER

10 Working with Multiple Domains (BEA Tuxedo Systems)

This chapter describes the task of administering services across multiple Domains by using the BEA Tuxedo Domains feature. For information about configuring WebLogic Enterprise domains, refer to Chapter 9, “Configuring Multiple Domains (BEA WebLogic Enterprise Systems).”

This topic includes the following sections:

- Benefits of Using BEA Tuxedo System Domains
- What Is the Domains Gateway Configuration File?
- Configuring Local and Remote Domains
- Example of a Domains-based Configuration
- Ensuring Security in Domains
- Routing Service Requests to Remote Domains

Benefits of Using BEA Tuxedo System Domains

Using Domains provides the following benefits:

- Scalability and modular growth—programmers can structure their application for modularity, isolation of failures, and independent growth. Interoperation with other transaction processing applications is achieved easily by adding to the Domains configuration the description of the interfaces (that is, services) used by a remote application.
- Transparency and independence—applications are totally unaware of service distribution. A service may be available on the same machine, on another machine in the local domain, or on a remote domain. Client application programmers do not need to know the implementation changes made to a service, the location of a service, network addresses, and so on.
- Aliasing capability—this allows you to define a mapping between the service names used by a remote application and the service names used by the local application, allowing for easy integration of applications that use different naming schemes.
- Transaction management and reliability—the Domains feature us integrated with the BEA Tuxedo system transaction management capabilities.
- Availability—you can specify alternate destinations to handle failure conditions.
- Security—an access control list (ACL) facility is provided to restrict access to local services from a particular set of remote domains. Domains also provides encryption and password verification.

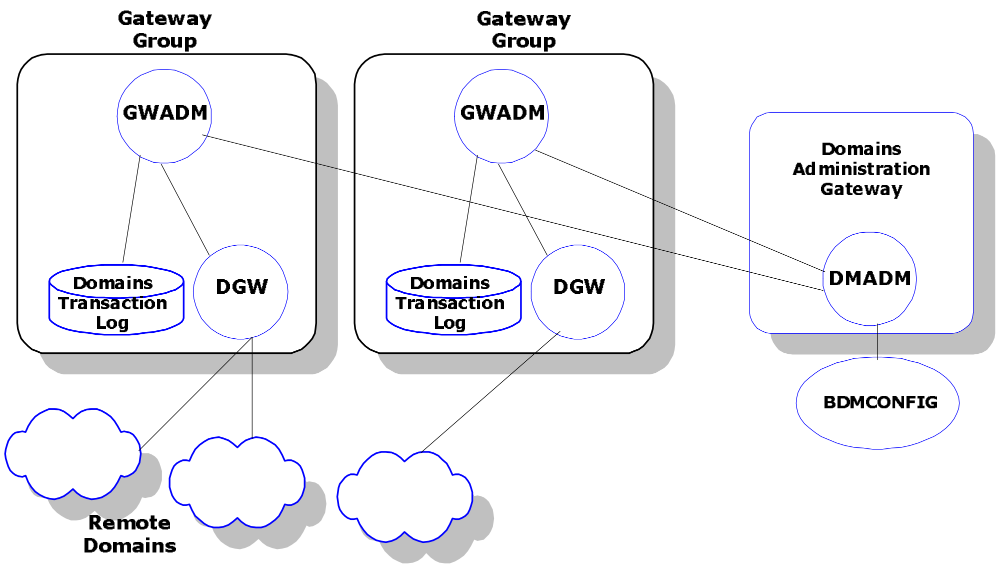
What Is the Domains Gateway Configuration File?

All domain configuration information is stored in a binary file, called the `BDMCONFIG` file. You can create and edit the domain gateway configuration file (`DMCONFIG` file), with any UNIX text editor. You can update the compiled `BDMCONFIG` file while the system is running by using the `dmadmin`(1) command when using Domains. There must be one `BDMCONFIG` file per BEA Tuxedo application.

A BEA Tuxedo system domain gateway is a server supplied by the BEA Tuxedo system that enables access to and from remote domains. Domains provides a gateway administrative server (`GWADM`) that enables run-time administration of the Domains gateway group, and a Domains administrative server (`DMADM`) that enables run-time administration of the Domains configuration information (`BDMCONFIG`). You enable remote domain access by specifying a gateway group and a domain administration group in the `GROUPS` section of the `TUXCONFIG` file, and by adding entries for the gateway and the two administrative servers in the `SERVERS` section.

In Figure 10-1, `DGW` is the domain gateway; `GWADM` is the gateway administrative server; `DMADM` is the Domains administrative server; and `BDMCONFIG` is the Domains gateway configuration file.

Figure 10-1 BEA Tuxedo Domains Gateway



Components of the DMCONFIG File

Table 10-1 describes the sections of the `DMCONFIG` file.

Table 10-1 DMCONFIG Sections Descriptions

Section	Purpose
`DM_LOCAL_DOMAINS`	Describes the environment for a particular domain gateway group. You can use multiple entries in this section to define multiple gateway groups within a single BEA Tuxedo application.
`DM_REMOTE_DOMAINS`	Identifies the remote domains that clients and servers of this Domains application can access.
`DM_LOCAL_SERVICES`	Describes the set of services in this domain which remote domains can access.
`DM_REMOTE_SERVICES`	Describes the set of services provided by remote domains that are accessible from this domain.

Table 10-1 DMCONFIG Sections Descriptions (Continued)

Section	Purpose
`DM_ROUTING`	Specifies criteria for data-dependent routing used by gateways to route service requests to specific remote domains.
`DM_ACCESS_CONTROL`	Specifies a named list (the Access Control List) of remote domains permitted to access a particular service.
`DM_<dmtype>`	Defines the specific parameters required for a particular Domains instance. Currently, the value of *`dmtype`* can be `OSITP`, `SNA`, or `TDOMAIN`. (This chapter focuses only on `TDOMAIN`.) You must specify each domain type in a section of its own.

Configuring Local and Remote Domains

To configure a local domain and a remote domain, perform the following tasks:

- Set environment variables
- Build a local application configuration file and a local domain gateway configuration file
- Build a remote application configuration file and a remote domain gateway configuration file

Setting Environment Variables

You need to set the following environment variables for the application to be configured successfully:

- `TUXDIR`—the root directory (for example, `/opt/tuxedo`)
- `TUXCONFIG`—the application configuration file (for example, `lapp.tux` or `rapp.tux`)
- `BDMCONFIG`—the Domains gateway configuration file (for example, `lapp.bdm` or `rapp.bdm`)

- `PATH`—must include `$TUXDIR/bin`
- `LD_LIBRARY_PATH`—must include `$TUXDIR/lib`

On AIX, `LIBPATH` must be set instead of `LD_LIBRARY_PATH`. On HP UX, `SHLIB_PATH` must be set instead of `LD_LIBRARY_PATH`. On Windows NT, no variable for shared libraries is required.

Examples

```
$ TUXDIR=/opt/tuxedo

$ PATH=$TUXDIR/bin:$PATH

$ LD_LIBRARY_PATH=$TUXDIR/lib:$LD_LIBRARY_PATH

$ export TUXDIR PATH LD_LIBRARY_PATH
```

Building a Local Application Configuration File and a Local Domains Gateway Configuration File

Build a local application configuration file using `tmloadcf`(1), and a local domain gateway configuration file using `dmloadcf`(1). The local application configuration file (`lapp.ubb`) contains the information necessary to boot the local application. This file is compiled into a binary data file (`lapp.tux`), using `tmloadcf`(1).

The local domain gateway configuration file (`lapp.dom`) contains the information used by domain gateways for communications with other domains. This file is compiled into a binary data file (`lapp.bdm`), using `dmloadcf`(1).

```
$ cd /home/lapp

$ TUXCONFIG=/home/lapp/lapp.tux; export TUXCONFIG

$ tmloadcf -y lapp.ubb

$ BDMCONFIG=/home/lapp/lapp_bdm; export BDMCONFIG

$ dmloadcf -y lapp.dom

$ tmboot -y
```

Building a Remote Application Configuration File and a Remote Domains Gateway Configuration File

Build a remote application configuration file and a remote domain gateway configuration file. The remote application configuration file (`rapp.ubb`) contains the information used by domain gateways for communication with other domains. This file is compiled into a binary data file (`rapp.tux`).

The remote domain gateway configuration file (`rapp.dom`) contains the information used by domain gateways to initialize the context required for communications with other domains. This configuration file is similar to the local domain gateway configuration file. The difference is in which services are exported and imported. This file is compiled into a binary data file (`rapp.bdm`).

```
$ cd /home/rapp

$ TUXCONFIG=/home/rapp/rapp.tux; export TUXCONFIG

$ tmloadcf -y rapp.ubb

$ BDMCONFIG=/home/rapp/rapp_bdm; export BDMCONFIG

$ dmloadcf -y rapp.dom

$ tmboot -y
```

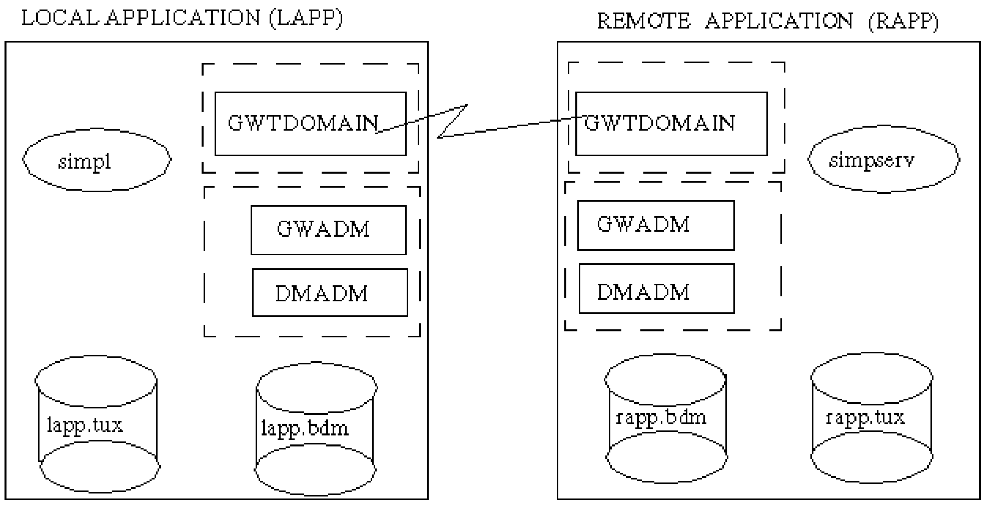
Once you create both the local and remote domains, you can then boot the application using `tmboot`(1). The order in which the two domains are booted does not matter. Monitor the applications with `dmadmin`(1).

Once both applications are booted, a client in the local application can call the `TOUPPER` service residing in the remote application.

Example of a Domains-based Configuration

The Domains example, illustrated in Figure 10-2 and throughout this chapter, consists of two applications, both of which are based on the `Simpapp` example provided with the BEA Tuxedo system. The first application is called `lapp` for “local application;” the second application is called `rapp` for “remote application.” `lapp` is configured to allow its clients to access a service called `TOUPPER`, which is advertised in `rapp`.

Figure 10-2 A Local and a Remote Application (simpapp)



Defining the Local Domains Environment

For the sample local application configuration file (`lapp.ubb`) shown in Listing 10-1, only the required parameters are defined. Default settings are used for the other parameters.

The following two server groups are defined:

- The first contains the domain administrative server (`DMADM`).
- The second contains the gateway administrative server (`GWADM`) and the domain gateway (`GWTDOMAIN`).

The following three servers are defined:

- `DMADM`—the domain administrative server enables run-time administration of the configuration information required by domain gateway groups. This server provides run-time administration of the binary domain configuration file and supports a list of registered gateway groups. (There must be only one instance of `DMADM` per Domains application.)

- GWADM—the gateway administrative server enables run-time administration of a particular Domains gateway group. This server gets domain configuration information from the DMADM server. It also provides administrative functionality and transaction logging for the gateway group.
- GWTDOMAIN—the domain gateway server enables access to and from remote Domains. It allows for interoperability of two or more BEA Tuxedo domains. Information about the local and remote services it needs to export and import is included in the domain configuration file. The domain gateway server should always be configured with REPLYQ=N.

Listing 10-1 Example of a Local Application Configuration File

```
# lapp.ubb
#
*RESOURCES
IPCKEY         111111

MASTER         LAPP
MODEL          SHM

*MACHINES
giselle

               LMID=LAPP
               TUXDIR="/opt/tuxedo"
               APPDIR="/home/lapp"
               TUXCONFIG="/home/lapp/lapp.tux"


*GROUPS

LDMGRP         GRPNO=1 LMID=LAPP
LGWGRP         GRPNO=2 LMID=LAPP

*SERVERS

DMADM          SRVGRP=LDMGRP SRVID=1
GWADM          SRVGRP=LGWGRP SRVID=1
GWTDOMAIN      SRVGRP=LGWGRP SRVID=2 REPLYQ=N

*SERVICES
```

Defining the Local and Remote Domains, Addressing, and Imported and Exported Services

For the sample local domain gateway configuration file (`lapp.dom`), shown in Listing 10-2, only the required parameters are defined. Default settings are used for the other parameters.

The `DM_LOCAL_DOMAIN` section identifies the local domains and their associated gateway groups. This section has one entry (`LAPP`) and specifies the parameters required for the domain gateway processes in that group, as follows:

- `GWGRP` specifies the name of the gateway server group as specified in the application.
- `TYPE` of `TDOMAIN` indicates that the local domain will be communicating with another BEA Tuxedo domain. Other options are SNA and OSI.
- `DOMAINID` identifies the name of the Domains gateway and must be unique across all Domains.

The `DM_REMOTE_DOMAINS` section identifies the known set of remote Domains and their characteristics. This section has one entry (`RAPP`). `TYPE` is used to classify the type of Domains. `DomainsID` is a unique domain identifier.

The `DM_TDOMAIN` section defines the addressing information required by the BEA Tuxedo Domains feature. Following are entries in the section for each local and remote domain specified in this configuration file:

- `NWADDR` specifies either the network address to accept connections from other BEA Tuxedo Domains (local Domains entry), or the network address to connect to other BEA Tuxedo Domains (remote Domains entry).

The `DM_LOCAL_SERVICES` section provides information about the services that are exported. This section has no entries because no services are being exported.

The `DM_REMOTE_SERVICES` section provides information about the services that are imported. The `TOUPPER` service is imported so that it can be accessed by clients in the local domains.

Listing 10-2 Example of a Local Domains Gateway Configuration File

```
#
# lapp.dom
#
*DM_LOCAL_DOMAINS

LAPP         GWGRP=LGWGRP
             TYPE=TDOMAIN
              DOMAINID=”111111"

*DM_REMOTE_DOMAINS

RAPP         TYPE=TDOMAIN
              DOMAINID=”222222"

*DM_TDOMAIN

LAPP          NWADDR=”//mach1:5000"

RAPP          NWADDR=”//mach2:5000"

*DM_LOCAL_SERVICES

*DM_REMOTE_SERVICES

TOUPPER
```

Defining the Remote Domains Environment

For the sample remote application configuration file (`rapp.ubb`), shown in Listing 10-3, only the required parameters are defined. Default settings are used for the other parameters.

The following three server groups are defined:

- The first server group (`SRVGP=RDMGRP`) contains the Domains administrative server (`DMADM`).
- The second server group (`SRVGP=RGWGRP`) contains the gateway administrative server, `GWADM`, and the Domains gateway, `GWTDOMAIN`.
- The third server group (`SRVGP=APPGRP`) contains the application server `simpserv`.

The following four servers are defined:

- `DMADM`—the Domains administrative server
- `GWADM`—the gateway administrative server
- `GWTDOMAIN`—the Domains gateway server
- `simpserv`—the simple application server that advertises the `TOUPPER` service, which converts strings from lowercase to uppercase characters

Listing 10-3 Example of a Remote Application Configuration File

```
# rapp.ubb
#
*RESOURCES
IPCKEY        222222

MASTER        RAPP

MODEL        SHM

*MACHINES

juliet

             LMID=RAPP
             TUXDIR="/opt/tuxedo"
              APPDIR="/home/rapp"
               TUXCONFIG="/home/rapp/rapp.tux"

*GROUPS

RDMGRP        GRPNO=1 LMID=RAPP
RGWGRP        GRPNO=2 LMID=RAPP
APPGRP         GRPNO=3 LMID=RAPP

*SERVERS

DMADM         SRVGRP=RDMGRP SRVID=1
GWADM         SRVGRP=RGWGRP SRVID=1
GWTDOMAIN     SRVGRP=RGWGRP SRVID=2 REPLYQ=N
simpserv       SRVGRP=APPGRP SRVID=1

*SERVICES
TOUPPER
```

Defining the Exported Services

For the sample remote domain gateway configuration file (rapp.dom), shown in Listing 10-4, only the required parameters are defined. Default settings are used for the other parameters.

This configuration file is similar to the local domain gateway configuration file. The difference is in which services are exported and imported.

The DM_LOCAL_SERVICES section provides information about the services exported by each local domain. In this example, the TOUPPER service is exported and included in the DM_LOCAL_SERVICES section. No service is imported so there are no entries in the DM_REMOTE_SERVICES section.

Listing 10-4 Example of a Remote Domains Gateway Configuration File

```
# rapp.dom
#

*DM_LOCAL_DOMAINS

RAPP        GWGRP=RGWGRP
            TYPE=TDOMAIN
             DOMAINID=”222222"

*DM_REMOTE_DOMAINS

LAPP        TYPE=TDOMAIN
             DOMAINID=”111111"

*DM_TDOMAIN

RAPP         NWADDR=”//mach2:5000"

LAPP         NWADDR=”//mach1:5000"

*DM_LOCAL_SERVICES
TOUPPER
*DM_REMOTE_SERVICES
```

Using Data Compression Between Domains

Data compression is useful in most applications and vital to supporting large configurations. When data is sent between Domains, you can elect to compress it for faster performance. This is configured by setting the `CMPLIMIT` parameter in the `dmconfig`(5). See Chapter 6, "Building Networked Applications," for more information on data compression.

Ensuring Security in Domains

Because Domains can exist under diverse ownership, multiple ways are offered to enable you to provide sufficient security:

- Local Domains—provides a first level of security. A partial view of the application (that is, a subset of services) can be made available to remote domains. This partial view is defined by including the corresponding services in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file.
- Domains Passwords—authentication techniques are required to ensure the proper *identity* of each remote domain. Domains provides a facility for the definition of passwords on a per-remote-domain basis. This is configured by setting `SECURITY=DM_PW` in `dmconfig`(5).
- Access Control—access control provides another level of security in which you can restrict access to services within a local domain such that only selected remote domains can execute these services. This is configured in the `DM_ACCESS_CONTROL` section of the `dmconfig`(5).
- Link-level Encryption—encryption can be used across domains to ensure data privacy, so a network-based eavesdropper cannot learn the content of BEA Tuxedo messages or application-generated messages from domain gateway to domain gateway. This is configured by setting `MINENCRYPTBITS` and `MAXENCRYPTBITS` in the `dmconfig`(5). (See Chapter 6, "Building Networked Applications," for more information.)

Creating a Domain Access Control List (ACL)

To create a domain ACL, you must specify the name of the domain ACL and a list of the remote domains that are part of the list (the Domain Import List) in the `DM_ACCESS_CONTROL` section of the `DMCONFIG` file. Table 10-2 describes these two fields.

Table 10-2 Domain ACL Fields

Fields	Description
Domain ACL name	The name of this ACL. A valid name consists of a string of 1-30 characters, inclusive. It must be printable and it may not include a colon, a pound sign, or a new line character. An example is: `ACLGRP1`
Domain import `VIEW` list	The list of remote domains that are granted access for this access control list. A valid value in this field is a set of one or more comma-separated strings. An example is: `REMDOM1`,`REMDOM2`,`REMDOM3`

Routing Service Requests to Remote Domains

Information for data-dependent routing used by gateways to route service requests (to specific remote domains) is provided in the `DM_ROUTING` section of the `DMCONFIG` file. The `FML32`, `VIEW32`, `FML`, `VIEW`, `X_C_TYPE`, and `X_COMMON` typed buffers are supported. To create a routing table for a domain, you must specify the buffer type for which the routing entry is valid, the name of the routing entry and field, and the ranges and associated remote domain names of the routing field. Table 10-3 describes these fields.

Table 10-3 Routing Table Fields

Fields	Description

Buffer type	A list of types and subtypes of data buffers for which this routing entry is valid. The types may include `FML32`, `VIEW32`, `FML`, `VIEW`, `X_C_TYPE`, or `X_COMMON`. No subtype can be specified for type `FML`; subtypes are required for the other types. The * (or *wildcard*) value 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. 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.
	
	Valid values for *`type`* are:
	`[:subtype1[,subtype2 . . .]][;type2[:subtype3[`
	`        ,subtype4 . . .]]] . . .`
	where the maximum length is 256 characters over 32 *`type`*/*`subtype`* combinations.
	
	Valid values for *`subtype`* are names that may not include semicolons, colons, commas, or asterisks.
	
	An example is `FML`.

Domain routing criteria	The name (identifier) of the routing entry. A valid value is any string of 1-15 characters, inclusive. An example is `ROUTTAB1`.
Routing field name	The name of the routing field. This field is assumed to be a field name that is identified in an FML field table (for `FML` buffers) or an `FML VIEW` table (for `VIEW`, `X_C_TYPE`, or `X_COMMON` buffers). A valid value is an identifier string that is 1-30 characters, inclusive. An example is `FIELD1`.

Table 10-3 Routing Table Fields (Continued)

Fields	Description
Ranges	The ranges and associated remote domain names (`RDOM`) for the routing field. The routing field can be of any data type supported in `FML`. A numeric routing field must have numeric range values, and a string routing field must have string range values. String range values for string, carray, and character field types must be placed inside a pair of single quotes and cannot be preceded by a sign. Short and long integer values are a string of digits, optionally preceded by a plus or minus sign. Floating point numbers are of the form accepted by the C compiler or `atof()`as follows: an optional sign, then a string of digits optionally containing a decimal point, then an optional `e` or `E` followed by an optional sign or space, followed by an integer. When a field value matches a range, the associated `RDOM` value specifies the remote domains to which the request should be routed. An `RDOM` value of * indicates that the request can go to any remote domain known by the gateway group. Valid values are a comma-separated ordered list of range/`RDOM` pairs where a *range* is one of two types: (a) a single value (signed numeric value or character string in single quotes); or (b) a range of the form *lower-upper* (where *lower* and *upper* are both signed numeric values or character strings in single quotes). Note that *lower* must be less than or equal to *upper*. Within a range/`RDOM` pair, the range is separated from the `RDOM` by a colon (`:`). `MIN` can be used to indicate the minimum value for the data type of the associated `FIELD`; for strings and carrays, it is the null string; for character fields, it is `0`; for numeric values, it is the minimum numeric value that can be stored in the field. `MAX` can be used to indicate the maximum value for the data type of the associated `FIELD`; for strings and carrays, it is effectively an unlimited string of octal-255 characters; for a character field, it is a single octal-255 character; for numeric values, it is the maximum numeric value that can be stored in the field. Thus, `MIN - -5` is all numbers less than or equal to -5 and `- MAX` is the set of all numbers greater than or equal to 6. The meta-character * (*wildcard*) in the position of a range indicates any values not covered by the other ranges previously seen in the entry; only one wildcard range is allowed per entry and it should be last (ranges following it are ignored). An example is `1-100:REMDOM3`.

11 Managing Workstation Clients (BEA Tuxedo Systems)

This chapter is specific to the BEA Tuxedo system. If you are using the BEA WebLogic Enterprise system and you need to configure remote clients or the Internet Inter-ORB Protocol (IIOP) Listener/Handler, see Chapter 12, "Managing Remote Client Applications (BEA WebLogic Enterprise Systems)" for more information.

This topic includes the following BEA Tuxedo sections:

- Workstation Terms
- What Is a Workstation Client?
- Setting Environment Variables
- Setting the Maximum Number of Workstation Clients
- Configuring a Workstation Listener (WSL)
- Modifying the MACHINES Section to Support Workstation Clients

Workstation Terms

Workstation	Workstation Extension—the workstation product that is an extension of the base BEA Tuxedo system.
DLL	Dynamic Link Libraries—a collection of functions grouped into a load module that is dynamically linked with an executable program at run time for a Microsoft Windows or an OS/2 application.
WSC	Workstation Client—a client process running on a remote site.
WSH	Workstation Handler—a client process running on an application site that acts as a surrogate on behalf of the WSC.
WSL	Workstation Listener—a server process running on an application site that listens for WSCs to connect.

What Is a Workstation Client?

The Workstation Extension 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, or a Bulletin Board. All communication between the client and the application takes place over the network.

The client process can be running UNIX, MS-DOS, Windows, or OS/2. The client has access to the ATMI interface for clients. The networking behind the calls is transparent to the user. The client process registers with the system and has the same status as a native client. The client can do the following:

- Send and receive messages
- Begin, end, or commit transactions

- Send and receive unsolicited messages
- Pass application security (on a mandatory basis)
- Communicate information about remote clients through the `tmadmin`(1) command

Note: A client process communicates with the native domain through the WSH rather than through a `BRIDGE` process.

Illustration of an Application with Two Workstation Clients

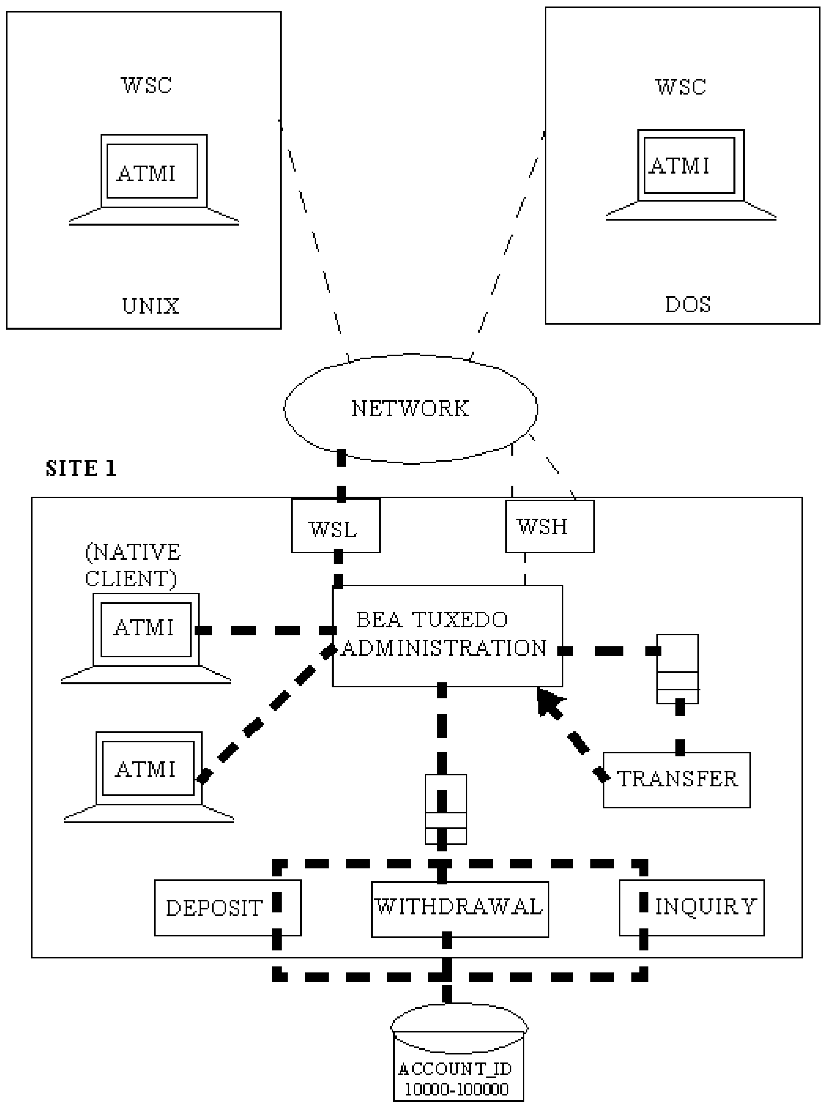
Figure 11-1 shows an example of an application with two WSCs connected. The client on the left is running on a UNIX system workstation, while the client on the right is running on an MS-DOS workstation. Both WSCs are communicating with the application through the WSH process. Initially, both joined by communicating with the WSL (indicated by the heavily dashed line).

The administrative servers and the application servers are located entirely on `SITE1`. Any request by a WSC to access the resource manager (RM) is sent over the network to the WSH. This process sends the request to the appropriate server and sends the reply back to the WSC.

The application is running in SHM mode. If the application was distributed over several nodes, the procedure would be very similar. The WSC would communicate with one WSH, and the request would be sent to a `BRIDGE` process, which would forward it to the correct node.

Note: As used in this book, the term “resource manager” refers to an entity that interacts with the BEA Tuxedo system and implements the XA standard interfaces. The most common example of a resource manager is a database. Resource managers provide transaction capabilities and permanence of actions; they are the entities accessed and controlled within a global transaction.

Figure 11-1 A Bank Application with Two Workstation Clients



How the Workstation Client Connects to an Application

A Workstation client connects to an application in the following manner:

1. The client connects to the WSL process using a known network address. This is initiated when the client calls either `tpchkauth()` or `tpinit()`. The WSL returns the address of a WSH to the client.
2. The WSL process sends a message to the WSH process informing it of the connection request.
3. The WSC connects to the WSH. (All further communication between the WSC and the application takes place through the WSH.)

Setting Environment Variables

Eight environment variables can be used to pass information to the system. All are optional except `TUXDIR` and `WSNADDR`. Defaults are available for all except `WSENVFILE`:

- `TUXDIR`—this contains the location of the BEA Tuxedo software on this workstation. It must be set for the client to connect.
- `WSNADDR`— this contains the network address of the WSL that the client wants to contact. This must match the address of a WSL process, as specified in the application configuration file.
- `WSDEVICE`—this contains the network device to be used. The default is an empty string. `WSDEVICE` must be set if TLI is being used.
- `WSENVFILE`—this contains the name of a file in which all environment variables may be set. There is no default for this variable.
- `WSTYPE`—this contains the machine type. If the value of `WSTYPE` matches the value of `TYPE` in the configuration file for the WSL machine, no encoding/decoding is performed. The default is the empty string. Keep in mind, when deciding whether to use the default, that a value of “empty string” will match any other “empty string” value. Be sure to specify the value of `WSTYPE` whenever that value does not match the value of `TYPE` on the WSL machine.

- `WSRPLYMAX`—this contains the amount of core memory to be used for buffering application replies. The default is 32,000 bytes.
- `TMPDIR`—this contains the directory in which to store replies when the `WSRPLYMAX` limit has been reached. The default is the working directory.
- `APP_PW`—this contains the password in a secure application. Clients that run from scripts can get the application password from this variable.

Setting the Maximum Number of Workstation Clients

To join Workstation clients to 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 accessers through a single accesser slot and, therefore, requires only one semaphore. This points out an additional benefit of the Workstation extension. By putting more clients out on workstations and off 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 `MAXACCESSERS`, native clients and servers fail at `tpinit()` time. The following table describes the `MAXWSCLIENTS` parameter.

Parameter	Description
MAXWSCLIENTS	Specifies the maximum number of WSCs that may connect to a node. The default is 0. If not specified, WSCs may not connect to the machine being described. The syntax is `MAXWSCLIENTS=`*number*.

Configuring a Workstation Listener (WSL)

Workstation clients access your application through the services of a WSL process and one or more WSH processes. The WSL and WSH are specified in one entry as a server supplied by the BEA Tuxedo system, although they are separate processes. The WSL can support multiple Workstation clients and 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 Workstation 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 list the Workstation Listener (WSL) processes in the `SERVERS` section of the `UBBCONFIG` file. Use the same syntax you use when listing a server.

Format of the CLOPT Parameter

Use the command-line option string (`CLOPT`) to pass information to a WSL process. 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]"
```

The `-A` value indicates that the WSL is to be booted to offer all its services. This is a default, but it is shown to emphasize the distinction between system-supplied servers and application servers. The latter can be booted to offer only a subset of their available services. The `--` syntax marks the beginning of a list of parameters that are passed to the WSL after the latter has been booted.

Command-line Options of the CLOPT Parameter

You can specify the following command-line options in the `CLOPT` string after the `--` (double minus signs):

- `-n` *`netaddr`* is the network address that WSCs use to contact the listener. The WSC must set the environment variable (`WSNADDR`) to this value. This is a required parameter.
- [`-d` *`device`*] is the network device name. This is an optional parameter because some transport interfaces (sockets) do not require it. However, it is required if the provider is TLI.
- [`-t` *`timeout`*] allows more time for a client to join when there is a large number of clients attempting to join simultaneously. The value is multiplied by the `SCANUNIT` parameter. The default is 3 in a nonsecure application, and 6 in an application with security on it.
- [`-w` *`name`*] is the name of the WSH process that should be booted for this listener. The default is WSH, which is the name of the handler provided. If another handler process is built with the `buildwsh`(1) command, that name is specified here.
- [`-m` *`number`*] specifies the minimum number of handlers that should be booted and always available. The default is 0.
- [`-M` *`number`*] specifies the maximum number of handlers that can be booted. The default is the value of `MAXWSCLIENTS` for that node divided by the multiplexing value.
- [`-x` *`number`*] specifies the maximum number of clients that a WSH can multiplex at a time. The default is 10 and the value must be greater than 0.
- [`-T` *`client-timeout`*] specifies the inactive client timeout option. The inactive client timeout is the time (in minutes) allowed for a client to stay idle. If a client

does not make any requests within this time period, the WSH disconnects the client. If this argument is not given or is set to 0, the timeout is infinite.

- [`-p` *`minwshport`*] [`-P` *`maxwshport`*] specifies 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 *`minwshport`* and 65535 for *`maxwshport`*.

Modifying the MACHINES Section to Support Workstation Clients

Listing 11-1 shows an example of how you can add the Workstation feature to the `bankapp` application.

Listing 11-1 UBBCONFIG Configuration

```
MACHINES
SITE1
              ...
              MAXWSCLIENTS=150

              ...
SITE2

              ...
              MAXWSCLIENTS=0
              ...

SERVERS
              ...
WSL SRVGRP=”BANKB1" SRVID=500 RESTART=Y
           CLOPT=”-A -- -N 0x0002ffffaaaaaaaa \
           -d /dev/tcp -m 5 -M 30 -x 5"

           ...
```

Notice the following specifications in the `MACHINES` and `SERVERS` sections:

- The `MACHINES` section shows the default `MAXWSCLIENTS` as being overridden for two sites. For `SITE1`, the default is raised to 150, while it is lowered to 0 for `SITE2`, which will not have WSCs connected to it.
- The `SERVERS` section shows a WSL process listed 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 `0x0002ffffaaaaaaaa` (`-N`).
  - The network provider will be `/dev/tcp` (`-d`).
  - 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`).

CHAPTER

12 Managing Remote Client Applications (BEA WebLogic Enterprise Systems)

This chapter explains how to configure connections from remote client applications to CORBA objects via the standard Internet Inter-ORB Protocol (IIOP). This chapter is specific to BEA WebLogic Enterprise servers.

This topic includes the following sections:

- Terms and Definitions
- Remote Client Overview
- Setting Environment Variables
- Setting the Maximum Number of Remote Clients
- Configuring a Listener for a Remote Client
- Modifying the UBBCONFIG File to Support Remote Clients
- Configuring Outbound IIOP for Remote Joint Client/Servers
- Using the ISL Command to Configure Outbound IIOP Support

Terms and Definitions

The following terms are used in this chapter.

DLL
: Dynamic Link Libraries. These are a collection of functions grouped into a load module that is dynamically linked with an executable program at run time for a Microsoft Windows or an OS/2 application.

IIOP
: Internet Inter-ORB Protocol (IIOP). IIOP is basically TCP/IP with some CORBA-defined message exchanges that serve as a common backbone protocol.

ISH
: IIOP Handler. This is a client process running on an application site that acts as a surrogate on behalf of the remote client.

ISL
: IIOP Listener. This is a server process running on an application site that listens for remote clients requesting connection.

server
: A server hosted on a machine in a BEA WebLogic Enterprise domain. A server is built with the BEA WebLogic Enterprise `buildobjserver` command. Servers implement BEA WebLogic Enterprise functionality, such as security, transactions, and object state management.

Note: In BEA WebLogic Enterprise 4.0, servers could only make invocations on other servers inside the BEA WebLogic Enterprise domain. In BEA WebLogic Enterprise 4.2, the servers can make invocations on any server, inside or outside a BEA WebLogic Enterprise domain.

native client
: A client located within a BEA WebLogic Enterprise domain, using the BEA WebLogic Enterprise ORB to make invocations on objects either inside or outside the BEA WebLogic Enterprise domain. A native client's host contains the BEA WebLogic Enterprise administrative and infrastructure components, such as `tmadmin`, FactoryFinder, and ISL/ISH. Native clients use the environmental objects to access BEA WebLogic Enterprise objects.

You build native clients with either the `buildobjclient` command or Java client commands.

Note: In BEA WebLogic Enterprise 4.0, a native client could not make invocations on objects outside the BEA WebLogic Enterprise domain.

remote client

A client not located within a BEA WebLogic Enterprise domain. A remote client can use the BEA WebLogic Enterprise ORB to make invocations on objects either inside or outside the BEA WebLogic Enterprise domain. A remote client's host does not contain BEA WebLogic Enterprise administrative and infrastructure components, such as `tmadmin`, FactoryFinder, and ISL/ISH; it does contain supporting software (the BEA WebLogic Enterprise ORB) that allows remote clients to invoke objects. Remote clients use the environmental objects to access BEA WebLogic Enterprise objects. You build remote clients with either the `buildobjclient` command or the Java client commands.

native joint client/server

A process that has two purposes: 1) to execute code acting as the starter for some business actions and 2) to execute method code for invocations on objects.
A joint client/server located within a BEA WebLogic Enterprise domain. You build C++ native joint client/servers with the `buildobjclient` command. Java native joint client servers are not supported.

Note: In BEA WebLogic Enterprise 4.0 and 4.1, a client could not act as a server.

Note: The server role of the native joint client/server is considerably less robust than that of an server. It has none of the BEA WebLogic Enterprise administrative and infrastructure components, such as `tmadmin`, FactoryFinder, and ISL/ISH (hence none of BEA WebLogic Enterprise's scalability and reliability attributes), it does not use the BEA WebLogic Enterprise TP Framework, and it requires more direct interaction between the client and the ORB.

remote joint client/server

A process that has two purposes: 1) execute code acting as the starter for some business actions and 2) execute method code for invocations on objects. A joint client/server located outside a BEA WebLogic Enterprise domain. The joint client/server does not use the BEA WebLogic Enterprise TP Framework and requires more direct interaction between the Client and the ORB. You build remote joint client/servers with the `buildobjclient` command or the Java client commands.

Note: In BEA WebLogic Enterprise 4.0, a remote client could not act as a server.

Note: A joint client/server is different from a server that acts as a client as part of its server role. Once the server completes processing of an invocation, it returns to dormancy. A joint client/server is always in the active mode, executing code not related to a server role; the server role temporarily interrupts the active client role, but the client role is always resumed.

Note: The server role of the remote joint client/server is considerably less robust than that of a server. Neither the client nor the server has any of the BEA WebLogic Enterprise administrative and infrastructure components, such as `tmadmin`, FactoryFinder, and ISL/ISH (hence, none of BEA WebLogic Enterprise's scalability and reliability attributes).

BEA WebLogic Enterprise object

A CORBA object that is implemented using TP Framework and that implements security, transactions, and object state management. BEA WebLogic Enterprise objects are implemented in servers; that is, it is in a BEA WebLogic Enterprise domain and uses the BEA WebLogic Enterprise infrastructure.

Callback object

A CORBA object supplied as a parameter in a client's invocation on a target object. The target object can make invocations on the callback object either during the execution of the target object or at some later time (even after the invocation on the target object has been completed). A callback object might be located inside or outside a BEA WebLogic Enterprise domain.

Note: In BEA WebLogic Enterprise 4.0 and 4.1, callback objects existed but were not named as such; they could have implementations only in the BEA WebLogic Enterprise domain; that is, they could be located only in a server (as a CORBA object).

Remote Client Overview

In this chapter, the term remote client means BEA WebLogic Enterprise client applications that you deployed on systems that do not have the full BEA WebLogic Enterprise server software installed. This means that no administration or application servers are running there and that no Bulletin Board is present. All communication between the client and the application takes place over the network. The types of clients are:

- CORBA C++ client
- CORBA Java client
- ActiveX client
- RMI clients
- Jolt clients
- Tuxedo Workstations (/WS) clients

A client process can be running UNIX or Microsoft Windows. The CORBA and ActiveX clients have access to the CORBA ORB interface. RMI clients have access to Enterprise JavaBeans (EJBs). Jolt and Tuxedo /WS clients have access to Tuxedo services. The networking behind the calls is transparent to the user. The client process registers with the system and has the same status as a native client. The client can do the following:

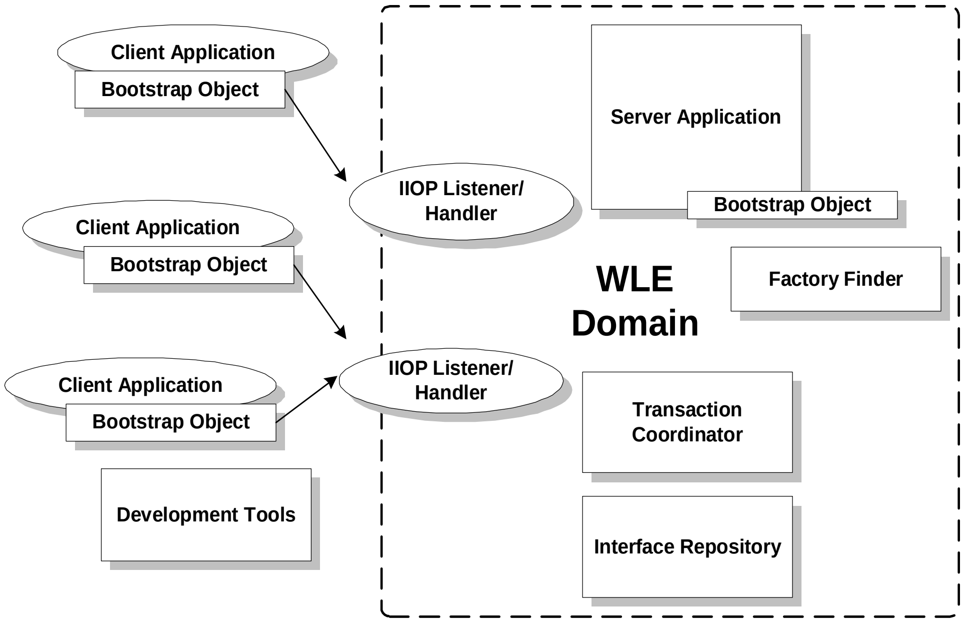
- Invoke methods on remote objects
- Begin, roll back, or commit transactions
- Be required to pass application security

Note: A client process communicates with the native domain through the ISH.

Illustration of an Application with Remote Clients

Figure 12-1 shows an example of an application with remote clients connected. Any request by a remote client to access the CORBA server application is sent over the network to the ISH. This process sends the request to the appropriate server and sends the reply back to the remote client.

Figure 12-1 Bank Application with Remote Clients



How the Remote Client Connects to an Application

The client connects to the ISL process in the IIOP Listener/Handler using a known network address. This is initiated when the client calls the Bootstrap object constructor. The ISL process uses a function that is specific to the operating system to pass the connection directly to the selected ISH process. To the client application, there is only one connection. The client application does not know, or need to know, that it is now connected to the ISH process.

Setting Environment Variables

For CORBA C++ clients, environment variables can be used to pass information to the system, as follows:

- `TUXDIR`—this contains the location of the BEA WebLogic Enterprise client software on this remote client. It must be set for the client to connect.
- `TOBJADDR`—this contains the network address of the ISL that the client wants to contact. This must match the address of an ISL process as specified in the application configuration file.

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

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

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

  On UNIX systems, use the `uname -n` command on the host system to determine the capitalization used. On Windows NT systems, see the host system's Network control panel to determine the capitalization used. Or use the environment variable `COMPUTERNAME`. For example:

  ```
  echo %COMPUTERNAME%
  ```

Setting the Maximum Number of Remote Clients

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

`MAXWSCLIENTS` tells the BEA WebLogic Enterprise system at boot time how many accesser slots to reserve exclusively for remote clients. For native clients, each accesser slot requires one semaphore. However, the ISH process (executing on the native platform on behalf of remote clients) multiplexes remote client accessers through a single accesser slot and, therefore, requires only one semaphore. This points out an additional benefit of the remote extension. By putting more clients out on remote systems and taking them off 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 remain to accommodate native clients as well as servers. Do not specify a value for `MAXWSCLIENTS` greater than `MAXACCESSERS`. The following table describes the `MAXWSCLIENTS` parameter.

Parameter	**Description**
`MAXWSCLIENTS`	Specifies the maximum number of remote clients that may connect to a machine. The default is 0. If a value is not specified, remote clients may not connect to the machine being described. The syntax is `MAXWSCLIENTS=`*`number`*.

Configuring a Listener for a Remote Client

Remote clients access your application through the services of an ISL process and one or more ISH processes. The ISL is specified in one entry as a server supplied by the BEA WebLogic Enterprise system. The ISL can support multiple remote clients and acts as the single point of contact for all the remote clients connected to your application at the network address specified on the ISL command line. The listener schedules work for one or more remote handler processes. An ISH process acts as a surrogate within the administrative domain of your application for remote clients on remote systems. The ISH uses a multiplexing scheme to support multiple remote clients concurrently.

To join remote clients to an application, you must list the ISL processes in the `SERVERS` section of the `UBBCONFIG` file. The processes follow the same syntax for listing any server.

Format of the CLOPT Parameter

You use the following `ISL` command-line options (`CLOPT`) to pass information to the ISL process for remote clients. The format of the `CLOPT` parameter is as follows:

```
ISL SRVGRP=”identifier”
    SRVID="number"
    CLOPT="[ -A ] [ servopts options  ] -- -n netaddr
    [ -C {detect|warn|none} ]
    [ -d device ]
    [ -K {client|handler|both|none} ]
    [ -m minh ]
    [ -M maxh ]
    [ -T client-timeout]
    [ -x mpx-factor ]
    [ -H external-netaddr"
```

For a detailed description of the command-line options (`CLOPT`), see the ISL command in the *Command, System Processes, and MIB Reference*.

Modifying the UBBCONFIG File to Support Remote Clients

Listing 12-1 shows a sample `UBBCONFIG` file to support remote clients, as follows:

- The `MACHINES` section shows the default `MAXWSCLIENTS` as being overridden for two sites. For `SITE1`, the default is raised to 150, while it is lowered to 0 for `SITE2`, which does not have remote clients connected to it.
- The `SERVERS` section shows an ISL process listed for group `BANKB1`. Its server ID is 500 and it is marked as restartable.
- The command line options show the following:
  - The IIOP Listener/Handler will advertise all of its services (`-A`).
  - The IIOP Listener/Handler will listen at host `TRIXIE` on port 2500.
  - The network provider is `/dev/tcp` (`-d`).
  - The minimum number of ISH processes to boot is 5 (`-m`).
  - The maximum number of ISH processes to boot is 30 (`-M`).
  - Each handler can have a maximum of 5 clients connected at any one time (`-x`).

Listing 12-1 Sample UBBCONFIG File Configuration

```
*MACHINES
SITE1
              ...
              MAXWSCLIENTS=150
              ...
SITE2
              ...
              MAXWSCLIENTS=0
              ...
*SERVERS
              ...
ISL SRVGRP=”BANKB1" SRVID=500 RESTART=Y
           CLOPT=”-A -- -n //TRIXIE:2500 -d /dev/tcp
```

```
          -m 5 -M 30 -x 5"
...
```

Configuring Outbound IIOP for Remote Joint Client/Servers

Support for outbound IIOP provides native clients and servers acting as native clients the ability to invoke on a remote object reference outside of the BEA WebLogic Enterprise domain. This means that calls can be invoked on remote clients that have registered for callbacks, and objects in remote servers can be accessed.

Administrators are the only users who interact directly with the outbound IIOP support components. Administrators are responsible for booting the ISLs with the correct startup parameters to enable outbound IIOP to objects not located in a connected client. Administrators may need to adjust the number of ISLs they boot and the various startup parameters to obtain the best configuration for their installation's specific workload characteristics. They have the option of booting the ISLs with the default parameters.

Note: In this release, outbound IIOP is not supported for transactions or security.

Functional Description

Outbound IIOP support is required to support client callbacks. In version 4.0 and version 4.1 releases of the BEA WebLogic Enterprise software, the ISL/ISH was an inbound half-gateway. Outbound IIOP support adds the outbound half-gateway to the ISL/ISH (see Figure 12-2).

There are three types of outbound IIOP connections available, depending on the version of GIOP supported by the native server and the remote joint client/server application:

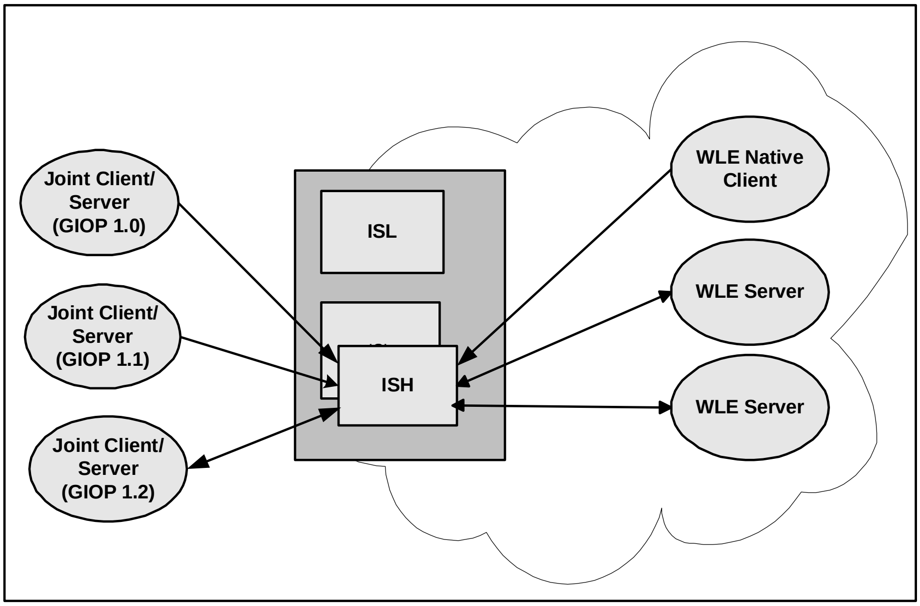
- Bidirectional—outbound IIOP reusing the same connection (supported only for BEA WebLogic Enterprise 4.2 and later C++ GIOP 1.2 servers, clients, and joint client/servers and WebLogic Enterprise 5.0 and later Java GIOP 1.2 servers).
- Asymmetric—outbound IIOP via a second connection (supported for GIOP version 1.0, GIOP 1.1, and GIOP 1.2 servers, clients, and joint client/server applications).
- Dual-paired connection—outbound IIOP (supported for GIOP 1.0 servers, clients, and joint client/server applications).

Note: GIOP 1.2 is supported only by BEA WebLogic Enterprise 4.2 and later C++ clients, servers, and joint client/servers and WebLogic Enterprise 5.0 and later Java servers. GIOP 1.2 is not supported by BEA WebLogic Enterprise 4.0 or 4.1 clients and joint client/servers. The Java clients and joint client/servers only support GIOP 1.0. The BEA WebLogic Enterprise 4.0 and 4.1 C++ clients and servers only support GIOP 1.0 and 1.1.

Bidirectional and dual-paired connection outbound IIOP provides outbound IIOP to object references located in joint client/servers connected to an ISH. Asymmetric outbound IIOP provides outbound IIOP to object references *not* located in a joint client/server connected to an ISH, and also allows BEA WebLogic Enterprise clients to invoke on any object reference, not only object references located in clients currently connected to an ISH.

Each type of outbound IIOP is described in more detail in the following sections.

Figure 12-2 Joint Client/Server IIOP Connections Supported


Joint Client/ Server (GIOP 1.0)
Joint Client/ Server (GIOP 1.1)
Joint Client/ Server (GIOP 1.2)
ISL
ISH
WLE Native Client
WLE Server
WLE Server

Bidirectional Outbound IIOP

With bidirectional outbound IIOP, the following operations are executed (see Figure 12-3):

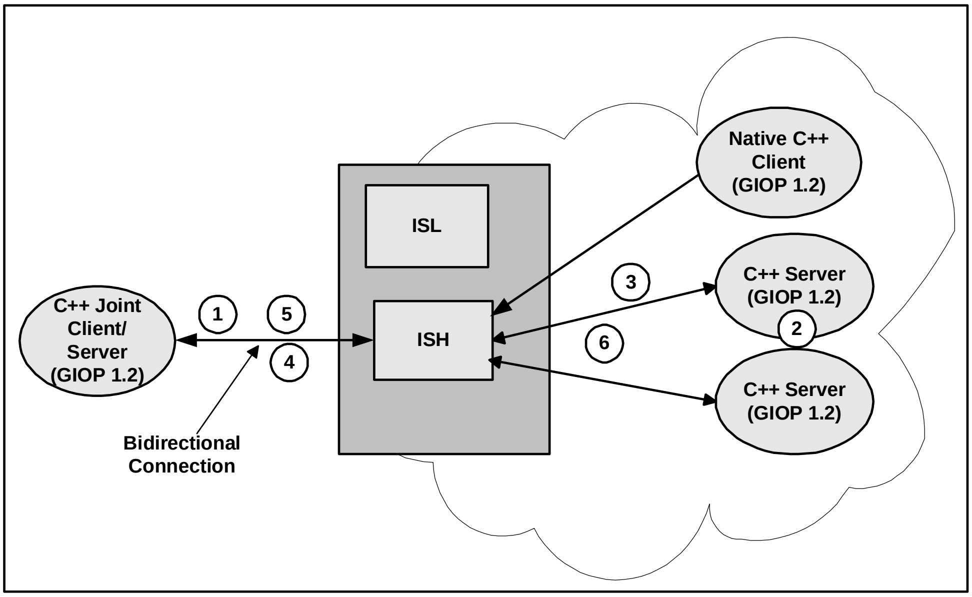
1. A client creates an object reference and invokes on a BEA WebLogic Enterprise server. The client ORB identifies the connection as being bidirectional using the service context. The service context travels with the message to the BEA WebLogic Enterprise server.

2. When unmarshaling the object reference, the BEA WebLogic Enterprise server compares the host/port in the service context with the host/port in the object reference. If they match, the ORB adds the ISH client information needed for routing to the ISH. This client information travels with the object reference whenever it is passed to other BEA WebLogic Enterprise servers.

3. At some point in time, a BEA WebLogic Enterprise server or native client invokes on the object reference, and the routing code invokes on the appropriate ISH, given the client information.

4. The ISH sends the request to the client over the same client connection.
5. The client executes the method and sends the reply back to the ISH via the client connection.
6. The ISH receives the reply and sends it to the BEA WebLogic Enterprise server.

Figure 12-3 Bidirectional Connection



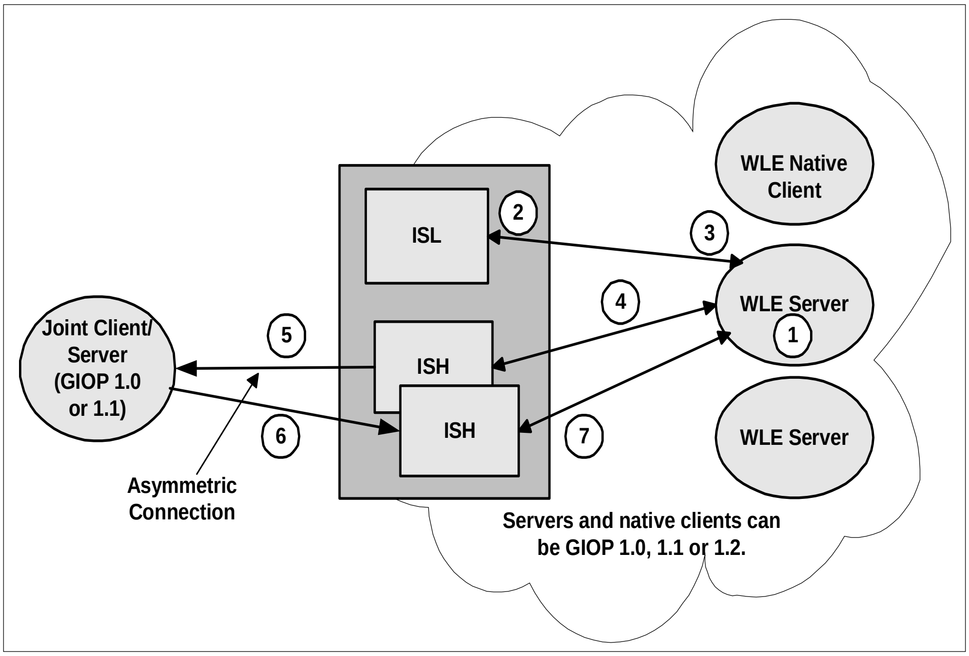
Asymmetric Outbound IIOP

With asymmetric outbound IIOP, the following operations are executed (see Figure 12-4):

1. A server gets an object reference from some source. It could be a naming service, a `string_to_object`, or it could be passed in through a client, but not located in that client. Since the object reference is not located in a client connected to an ISH, the outgoing call cannot be made using the bidirectional method. The BEA WebLogic Enterprise server invokes on the object reference.
2. On the first invoke, the routing code invokes a service in the ISL and passes in the host/port.

3. The ISL selects an ISH to handle the outbound invoke and returns the ISH information to the BEA WebLogic Enterprise server.
4. The BEA WebLogic Enterprise server invokes on the ISH.
5. The ISH determines which outgoing connection to use to send the request to the client. If none is connected, the ISH creates a connection to the host/port.
6. The client executes the method and sends the reply back to the ISH.
7. The ISH receives the reply and sends it to the BEA WebLogic Enterprise server.

Figure 12-4 Asymmetric Outbound IIOP


WLE Native Client
ISL
2
3
WLE Server
1
4
Joint Client/ Server (GIOP 1.0 or 1.1)
5
ISH
ISH
6
7
WLE Server
Asymmetric Connection
Servers and native clients can be GIOP 1.0, 1.1 or 1.2.

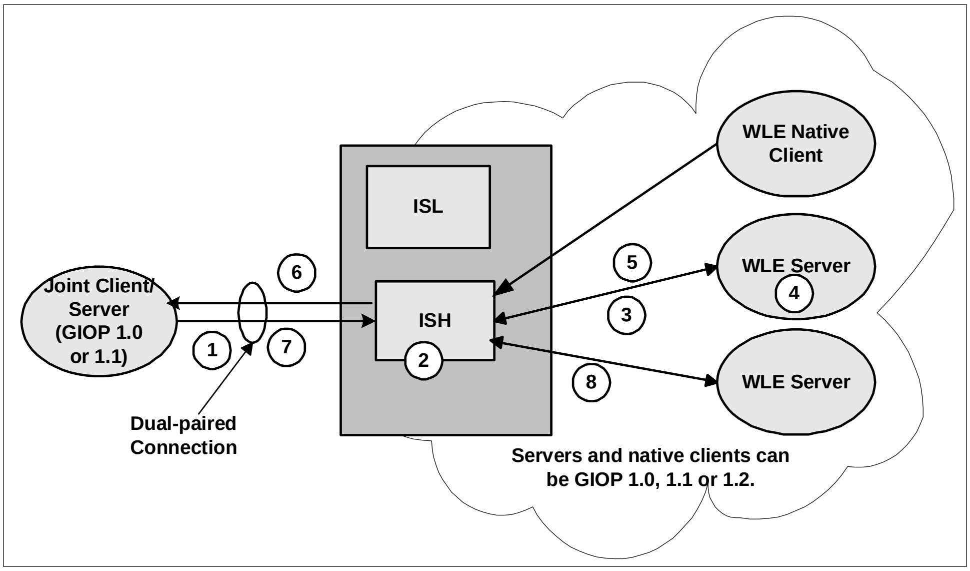
Dual-paired Connection Outbound IIOP

With dual-paired connection outbound IIOP, the following operations are executed (see Figure 12-5):

1. A client creates an object reference and calls the Bootstrap function (`register_callback_port`) and passes the object reference.

2. The ISH gets the host/port from the IOR and stores it with the client context.
3. The client invokes on a BEA WebLogic Enterprise server and passes the object reference. From the `register_callback_port` call, the ISH creates a service context containing the host/port. The service context travels with the message to the BEA WebLogic Enterprise server.
4. When unmarshaling the object reference, the BEA WebLogic Enterprise server compares the host/port in the service context with the host/port in the object reference. If they match, the ORB adds the ISH client information to the object reference. This client information travels with the object reference whenever it is passed to other BEA WebLogic Enterprise servers.
5. At some point in time, a BEA WebLogic Enterprise server or native client invokes on the object reference. The routing code invokes on the appropriate ISH, passing the client information.
6. The ISH creates a second connection to the client. It sends the request to the client over the second connection.
7. The client executes the method and sends the reply back to the ISH via the first client connection.
8. The ISH receives the reply and sends it to the BEA WebLogic Enterprise server. If the client disconnects from the ISH, the second connection is also disconnected.

Figure 12-5 Dual-paired Connections Outbound IIOP


ISL
ISH
Joint Client/ Server (GIOP 1.0 or 1.1)
WLE Native Client
WLE Server
WLE Server
1
2
3
4
5
6
7
8
Dual-paired Connection
Servers and native clients can be GIOP 1.0, 1.1 or 1.2.

How the Routing Code Finds an ISL

The steps to finding an ISL are as follows:

1. A service is advertised in each ISL.
2. The routing code invokes on that service name.

   Note: Normal BEA Tuxedo routing is used to find an ISL.

3. An idle ISL on the same machine is always chosen, if available. If not available, NETLOAD ensures that a local ISL is chosen most often.

   Note: Some invokes may be made to ISLs on nonlocal machines.

Using the ISL Command to Configure Outbound IIOP Support

Outbound IIOP support is used when a native C++ or Java client, or a server acting as a native client, invokes on an object reference that is a remote object reference. The routing code recognizes that the object reference is from a nonBEA WebLogic Enterprise ORB or from a remote BEA WebLogic Enterprise joint client/server.

Types of Object References

There are two kinds of remote object references:

- Object references created by BEA WebLogic Enterprise remote joint client/servers outside of the BEA WebLogic Enterprise domain.
- Object references created by other vendors' servers.

Both are detected by the routing code and sent to the outbound IIOP support for handling.

User Interface

The user interface to outbound IIOP support is the command-line interface for booting the ISL process(es).

Command-line options to configure the outbound IIOP processing were added to the ISL command in a previous release of the BEA WebLogic Enterprise software. These options enable support for asymmetric IIOP to object references not located in clients connected to an ISH.

Additional command-line options were added in the WebLogic Enterprise 5.1 release for SSL and Secure Connection Pools. Listing 12-2 shows the ISL command syntax and highlights the new options.

Listing 12-2 ISL Command-line Options

```
ISL SRVGRP="identifier"

    SRVID="number"

    CLOPT="[ -A ] [ servopts options ] -- -n netaddr
           [ -C {detect|warn|none} ]
           [ -d device ]
           [ -K {client|handler|both|none} ]
           [ -m minh ]
           [ -M maxh ]
           [ -T Client-timeout]
           [ -x mpx-factor ]
           [-H external-netaddr]
#Options for outbound IIOP
           [-O]
           [-o outbound-max-connections]
           [-s Server-timeout]
           [-u out-mpx-users] "
#NEW options for SSL
           [a]
           [-R renegotiation-interval]
           [-S secure-port]
           [-v {detect | warn | none}]
           [-z [0|40|56|128]]
           [-Z [0|40|56|128]]
#NEW options for Secure connection Pools
           [-E principal_nane]”
```

For a detailed description of the `CLOPT` command-line options, see the ISL command in the *Command, System Processes, and MIB Reference*.

CHAPTER

13 Managing Queued Messages (BEA Tuxedo System)

This chapter, which is specific to the BEA Tuxedo system, describes how to configure the BEA Tuxedo Queued Message Facility for your application, and how to manage the facility when the application goes into production.

This topic includes the following sections:

Terms and Definitions

The following terms are used in this chapter.

/Q
: A short name for the BEA Tuxedo Queued Message Facility

`QMCONFIG`
: An environment variable that holds the name of the device (file) where /Q queue space is located.

queue
: A named stable storage area where service requests from client processes or responses from application servers are stored.

queue space
: A collection of queues that can be administered as a unit.

request queue
: A space associated with an application server where service requests are placed for processing by the server.

`TMQUEUE`
: A BEA Tuxedo system server that accepts messages from a `tpenqueue()` call and places them on a /Q queue.

`TMQFORWARD`
: A BEA Tuxedo system server that dequeues a message from a /Q queue and forwards the message to an application server.

`TMS_QM`
: A BEA Tuxedo system server that manages transactions for /Q.

Overview of the BEA Tuxedo Queued Message Facility

The BEA Tuxedo Queued Message Facilityallows messages to be queued to stable storage for later processing. Primitives are added to the BEA Tuxedo system application-transaction manager interface (ATMI), that provide for messages to be added to or read from stable-storage queues. Reply messages and error messages can be queued for later return to client programs. An administrative command interpreter is provided for creating, listing, and modifying the queues. Prewritten servers are included to accept requests to enqueue and dequeue messages, to forward messages from the queue for processing, and to manage the transactions that involve the queues.

Administrative Tasks

The BEA Tuxedo system administrator is responsible for defining servers and creating queue space and queues like those shown between the vertical dashed lines in Figure 13-1.

The administrator must define at least one queue server group with `TMS_QM` as the transaction manager server for the group.

Two additional system-provided servers need to be defined in the configuration file. These servers perform the following functions:

- The message queue server, `TMQUEUE`(5), is used to enqueue and dequeue messages. This provides a surrogate server for doing message operations for clients and servers, whether or not they are local to the queue.
- The message forwarding server, `TMQFORWARD`(5), is used to dequeue and forward messages to application servers. The BEA Tuxedo system provides routines for servers that handle server initialization and termination, allocate buffers to receive and dispatch incoming requests to service routines, and route replies to the correct destination. All of this processing is transparent to the application.

- Existing servers do not dequeue their own messages or enqueue replies. One goal of /Q is to be able to use existing servers to service queued messages without change. The `TMQFORWARD` server, for example:
  - Dequeues a message from one or more queues in the queue space.
  - Forwards the message to a server that has a service with the same name as the queue.
  - Waits for the reply.
  - Queues the success reply or failure reply on the associated reply or failure queues (assuming the originator specified a reply or failure queue).

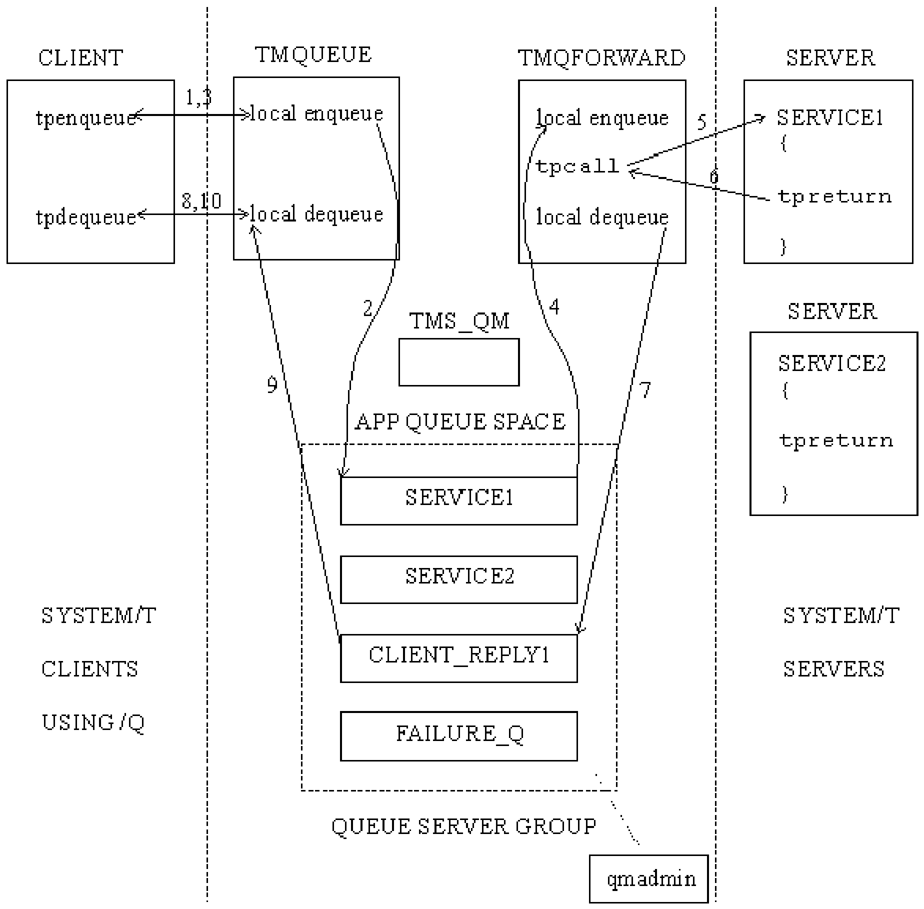
Also, the administrator must create a queue space using the queue administration program, `qmadmin`(1). The queue space contains a collection of queues. In Figure 13-1, for example, four queues are present within the queue space named `APP`. There is a one-to-one mapping of queue space to queue server group since each queue space is a resource manager (RM) instance and only a single RM can exist in a group.

The notion of queue space allows for reducing the administrative overhead associated with a queue by sharing the overhead among a collection of queues in the following ways:

- The queues in a queue space share the stable storage area for messages.
- A single message queue server, such as `TMQUEUE` in Figure 13-1, can be used to enqueue and dequeue messages for multiple queues within a single queue space.
- A single message forwarding server, such as `TMQFORWARD` in Figure 13-1, can be used to dequeue and forward messages for multiple queues within a single queue space.
- A single transaction manager server, such as `TMS_QM` in Figure 13-1, can be used to complete transactions for multiple queues within a single queue space.
- The administrator can define a single server group in the application configuration for the queue space by specifying the group in `UBBCONFIG` or by using `tmconfig`(1) to add the group dynamically.
- Finally, when the administrator moves messages between queues within a queue space, the overhead is less than if the messages were in different stable storage areas, because a one-phase commit can be done.

Figure 13-1 shows how the BEA Tuxedo Queued Message Facility works. The queue spaces and queues shown between the vertical dashed lines must be defined by the system administrator.

Figure 13-1 Overview of the Queued Message Facility



In Figure 13-1 (Steps 1, 2, and 3), a client enqueues a message to the `SERVICE1` queue in the APP queue space using `tpenqueue()`. Optionally, the names of a reply queue and a failure queue can be included in the call to `tpenqueue()`. In Figure 13-1 they are the queues `CLIENT_REPLY1` and `FAILURE_Q`. The client can specify a "correlation identifier" value to accompany the message. This value is persistent across queues so that any reply or failure message associated with the queued message can be identified when it is read from the reply or the failure queue.

The client can use the default queue ordering (for example, a time after which the message should be dequeued), or can specify an override of the default queue ordering (asking, for example, that this message be put at the top of the queue or ahead of another message on the queue). The call to `tpenqueue()` sends the message to the `TMQUEUE` server, the message is queued to stable storage, and an acknowledgment (step 3) is sent to the client. The acknowledgment is not seen directly by the client, but can be assumed when the client gets a successful return. (A failure return includes information about the nature of the failure.)

A message identifier assigned by the queue manager is returned to the application. The identifier can be used to dequeue a specific message. It can also be used in another `tpenqueue()` to identify a message already on the queue that the subsequent message should be enqueued ahead of.

Before an enqueued message is made available for dequeuing, the transaction in which the message is enqueued must be committed successfully.

When the message reaches the top of the queue, the `TMQFORWARD` server dequeues the message and forwards it, via `tpcall()`, to a service with the same name as the queue name. In Figure 13-1 the queue and the service are both named `SERVICE1`; steps 4, 5, and 6 show the transfer and return of the message. The client identifier and the application authentication key are set to the client that caused the message to be enqueued; they accompany the dequeued message as it is sent to the service.

When the service returns a reply, `TMQFORWARD` enqueues the reply (with an optional user-return code) to the reply queue (step 7 in Figure 13-1). Sometime later, the client uses `tpdequeue()` to read from the reply queue (`CLIENT_REPLY1`), and to get the reply message (steps 8, 9, and 10 in Figure 13-1). Messages on the reply queue are not automatically cleaned up; they must be dequeued, either by an application client or server, or by a `TMQFORWARD` server.

Part of the task of defining a queue is specifying the order for messages on the queue. Queue ordering can be time-based, priority based, `FIFO` or `LIFO`, or a combination of these sort criteria. The administrator specifies one or more of these criteria for the queue, listing the most significant criteria first. `FIFO` or `LIFO` can be specified only as the least significant sort criteria. Messages are put on the queue according to the specified sort criteria, and dequeued from the top of the queue.

The administrator can configure as many message queuing servers as are needed to keep up with the requests generated by clients for the stable queues.

Data-dependent routing can be used to route between multiple server groups with servers offering the same service.

For housekeeping purposes, the administrator can set up a command to be executed when a threshold is reached for a queue that does not routinely get drained. The threshold can be based on the bytes, blocks, or percentage of the queue space used by the queue, or the number of messages on the queue. The command set up by the administrator might boot a `TMQFORWARD` server to drain the queue or send mail to the administrator for manual handling.

Setting the QMCONFIG Environment Variable

The environment variable `QMCONFIG` must be set and exported before work can be done to create a queue space. A BEA Tuxedo system application uses a Universal Device List (UDL). The `QMCONFIG` variable must contain the full pathname of the device list, such as the path shown in the following example.

```
$ QMCONFIG = /dev/rawfs; export QMCONFIG
```

The commands provided by `qmadmin`, (the /Q administrative interface), will not work unless this location is defined. The information can be furnished on the command line as well as in the environment variable. If it is specified in both places, the information on the command line takes precedence.

Using qmadmin, the /Q Administrative Interface

/Q has an administrative program, `qmadmin`(1), that is used to create and administer queues. The following sections include a sampling of the available commands. For a complete list of `qmadmin` commands, refer to the `qmadmin`(1) reference page in the *BEA Tuxedo Reference Manual*.

Creating an Application Queue Space and Queues

Complete the following four steps to create an application queue space and queues.

1. Create an entry in the UDL with the `qmadmin` `crdl` command. The device may be created on a raw slice or in a UNIX file. For example:

   ```
   qmadmin        # to start the qmadmin command
   crdl device offset size
   ```

   where *`device`* is the same device named in the `QMCONFIG` variable; *`offset`* is the block number within the UDL where space may begin to be allocated (the first entry must have an offset of 0), and *`size`* is the number of blocks to allocate. To make the example more realistic, it might be like the following:

   ```
   crdl /dev/rawfs 500 500
   ```

   which says create an entry on the device `/dev/rawfs` 500 blocks from the start of the UDL and allocate 500 blocks. Implicit in this request is the presence of an existing entry, since the offset 0 is not specified. If you enter `crdl` without arguments, the software prompts you for information. You can create up to 25 entries on a device list.

2. Create a queue space on the device. This will be a space on the device that will contain a collection of queues. Space is created with the `qmadmin qspacecreate` command.

   `qspacecreate` *`queue_space_name ipckey pages queues trans procs\`*
   `messages errorq inityn`

   If you enter `qspacecreate` without arguments, the software prompts you for information. This is probably the better choice for this command because the prompts explain the information you need to provide. The following is an example from the `qmadmin`(1) reference page.

   ```
       > qspacecreate
       Queue space name: myqueuespace
       IPC Key for queue space: 42000
       Size of queue space in disk pages: 50000
       Number of queues in queue space: 30
       Number of concurrent transactions in queue space: 20
       Number of concurrent processes in queue space: 30
       Number of messages in queue space: 20000
       Error queue name: ERRORQ
       Initialize extents (y, n [default=n]): y
       Blocking factor [default=16]: 16
   ```

   The IPC key value must be unique and different from the value specified in the `RESOURCES` section. The number of disk pages specified as the size of the queue space varies from application to application and depends on the number of queues, the number of messages to be handled and the size of the messages. The specification for the number of concurrent processes in the queue space must be large enough to include four or five possible BEA Tuxedo system processes.

3. Open the queue space.

   `qopen` *`queue_space_name`*

   The queue space has to be open for you to proceed.

4. Create individual queues within the queue space. Queues are created with the `qmadmin qcreate` command, as follows.

   `qcreate` *`queue_name qorder out-of-order retries delay high low`*

   This is another command where it is better to allow the software to prompt you for information. The following is an example from `qmadmin`(1) (using mostly default values where available).

   ```
   >qcreate Queue name: service1 queue order (priority, time, fifo,
   lifo): fifo out-of-ordering enqueuing (top, msgid,
   [default=none]):none retries [default=0]: 0 retry delay in
   ```

```
seconds [default=0]: 0 High limit for queue capacity warning (b
for bytes used, B for blocks used, % for percent used, m for
messages [default=100%]): 100% Reset (low) limit for queue
capacity warning [default=0%]: 50% queue capacity command:
/usr/app/bin/mailadmin myqueuespace service1
```

`Retries` specifies the number of times the system attempts to enqueue the message.

We recommend that you read the `qmadmin`(1) reference page in the *BEA Tuxedo Reference Manual* carefully and that you also read the "Administration" chapter of the *BEA Tuxedo System /Q Guide.* The parameters that you enter for the `qcreate` command control the way the queue operates for your application. Of particular importance is the choice for the order in which messages are placed on the queue (they are always removed from the top).

Modifying the Configuration File

In addition to creating a queue space and queues, the system administrator needs to associate these resources with the BEA Tuxedo Queued Message Facilityapplication by editing the configuration file as described in the remaining sections of this chapter.

The configuration changes involve making an entry in the `GROUPS` section for the group that owns the queue and the transaction server (`TMS_QM`), and listing (in the `SERVERS` section) the two servers (`TMQUEUE` and `TMQFORWARD`).

Note: The chronological order of these specifications is not critical. The configuration file can be created either before or after the queue space is defined. The important thing is that the configuration must be defined and queue space and queues must be created before the facility can be used.

Associating a Queue with a Group

A server group must be defined for each queue space the application expects to use. In addition to the standard requirements of a group name tag and a value for `GRPNO`, the `TMSNAME` and `OPENINFO` parameters need to be set, as shown in the following example.

```
TMSNAME=TMS_QM
```

and

`OPENINFO="TUXEDO/QM:`*`device_name`*`:`*`queue_space_name`*`"`

(See the `ubbconfig`(5) reference page in the *BEA Tuxedo Reference Manual* for details.)

`TMS_QM` is the name for the transaction manager server for the BEA Tuxedo Queued Message Facility. In the `OPENINFO` parameter, `TUXEDO/QM` is the literal name for the resource manager as it appears in `$TUXDIR/udataobj/RM.` The values for *`device_name`* and *`queue_space_name`* are instance-specific and must be set to the path name for the universal device list and the name associated with the queue space, respectively.

The following example includes some of the detail.

```
*GROUPS
QUE1
LMID = SITE1 GRPNO = 2
TMSNAME = TMS_QM TMSCOUNT = 2
OPENINFO = “TUXEDO/QM:/dev/rawfs:myqueuespace”
```

Note the use of quotation marks around the information for `OPENINFO.` We recommend using quotation marks in this way to protect your entries in the configuration file.

Listing the /Q Servers in the SERVERS Section

Three servers are provided with the BEA Tuxedo Queued Message Facility. One is the TMS server, `TMS_QM`, that is the transaction manager server for the /Q resource manager. `TMS_QM` is defined in the `GROUPS` section of the configuration file.

The other two, `TMQUEUE`(5) and `TMQFORWARD`(5), provide services to users. They must be defined in the `SERVERS` section of the configuration file, as follows.

```
*SERVERS
TMQUEUE SRVGRP=QUE1 SRVID=1 CLOPT="-s QSPACENAME:TMQUEUE - - "
TMQFORWARD SRVGRP=QUE1 SRVID=5 CLOPT="- - -I 2 -q STRING"
```

The application can also create its own queue servers. If the functionality of `TMQFORWARD`, for example, does not fully meet the needs of the application, you might want to have a special server written. You might, for example, create a server that dequeues messages moved to the error queue, which `TMQFORWARD` does not do.

CHAPTER

14 Securing Application

For a detailed discussion of securing applications, see *Using Security* in the BEA WebLogic Enterprise online documentation.

CHAPTER

15 Monitoring a Running System

After your application is up and running, as an administrator you must ensure that it meets the performance, availability, and security standards of your company. To perform this task, you need to monitor the resources (such as shared memory), activities (such as transactions), and potential problems (such as security breaches) in your configuration, and take any corrective actions that are necessary.

To help you meet this responsibility, the BEA WebLogic Enterprise and BEA Tuxedo systems provide tools that enable you to oversee both system events and application events. This chapter explains how to use these tools to keep your application performing fast, correctly, and securely.

This topic includes the following sections:

- Overview of System and Application Data
- Monitoring Methods
- Using the tmadmin Command Interpreter
- Running tmadmin Commands
- Monitoring a Running System with tmadmin
- Example: Output from tmadmin Commands
- Case Study: Monitoring Run-time bankapp

Overview of System and Application Data

This section describes the types of data available for monitoring a running system and explains how to use that data.

Components and Activities for Which Data Is Available

Your BEA WebLogic Enterprise or BEA Tuxedo system maintains parameter settings and generates statistics for the following system components:

- Clients
- Conversations
- Groups
- Message queues
- Networks
- Servers
- Services
- Transactions
- Interfaces
- JDBC Connection Pools

Where the Data Resides

To ensure that you have the information necessary for monitoring your system, the BEA WebLogic Enterprise or BEA Tuxedo system provides the following data repositories:

- `UBBCONFIG`—an ASCII file in which you define the *parameters* of your system and application.

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

This chapter describes the data stored in the `UBBCONFIG` file and in the Bulletin Board, and provides instructions for monitoring that data. For a description of the log files, see Chapter 16, “Monitoring Log Files.”

How You Can Use the Data

The administrative data provided by your BEA WebLogic Enterprise or BEA Tuxedo system lets you monitor a multitude of potential trouble areas on your system. For example, this data lets you:

- Tune the running system based on actual loads
- Detect security breaches

You can also set up your system so that it is able to use the statistics in the Bulletin Board to make decisions and to modify system components dynamically, without user intervention. With proper configuration, your system may be able to perform tasks such as the following (when indicated by Bulletin Board statistics):

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

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

Static and Dynamic Data

There are two types of administrative data available on every running BEA WebLogic Enterprise and BEA Tuxedo system: *static* and *dynamic*.

Static Data

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

At times you will need to check the static data about your configuration. For example:

- Suppose you want to add a large number of machines and you are concerned that by doing so you may exceed the maximum number of machines allowed in your configuration—or, to be precise, allowed in the machine tables of the Bulletin Board. To look up the maximum number of machines allowed, check the current values of the system-wide parameters for your configuration (one of which is `MAXMACHINES`).
- Suppose you think you may be able to improve your application's performance by tuning your system. To determine whether tuning is required, you need to check on the amount of local IPC resources currently available.

Dynamic Data

Dynamic data consists of information that changes in realtime, that is, while an application is running. For example, the *load* (the number of requests sent to a server) and the state of various configuration components (such as servers) change frequently. Dynamic data is kept in the Bulletin Board and in JavaServers and the Active Object Map (AOM).

You will need to check the dynamic data about your configuration frequently. For example:

- Suppose performance is degraded and you want to know whether you have enough servers running to accommodate the number of clients currently connected.
  - Check the numbers of running servers and connected clients
  - Check the load on one or more servers
  - Check the numbers of actively running threads

These numbers will help you determine whether adding more servers is likely to improve performance.

- Suppose you receive complaints from multiple users about slow response when making particular requests of your application. Checking load statistics may help you determine whether it is appropriate to increase the value of `BLOCKTIME`.

Monitoring Methods

To monitor a running application, you need to keep track of the dynamic aspects of your configuration and sometimes check the static data. Thus, you need to be able to watch the Bulletin Board on an ongoing basis and consult the `UBBCONFIG` file when necessary. Both the BEA WebLogic Enterprise and BEA Tuxedo systems provide the following ways to monitor this data, as shown in the following table.

You Can Use the . . .	By . . .	For Instructions, See . . .
`tmadmin` command	Entering commands after a prompt	This chapter
AdminAPI	Using the MIB (and the commands described in this chapter) to write programs that monitor your run-time application	Chapter 21, “Event Broker/Monitor (BEA Tuxedo Systems)” *BEA Tuxedo Reference Manual*, Section 5
BEA Administration Console Web-based GUI	Using a graphical interface	The Help accessed directly from the GUI

The preferred method depends on your level of experience and the type of information you need to view.

If you are an experienced administrator (and have shell programming expertise), you may prefer to write programs that automate your most frequently run commands.

If you are not an experienced UNIX user, you may be most comfortable using the Web-based GUI.

If you examine the `RESOURCES` section of the `UBBCONFIG` file using the `tmadmin` command, you can see only the current values; the defaults are not displayed.

If you decide to monitor your system at run time using the `tmadmin` command interpreter, continue reading; this chapter describes `tmadmin` and explains how to use it.

Using the tmadmin Command Interpreter

The `tmadmin` command is an interpreter for a large set of commands that let you view and modify a Bulletin Board and its associated entities.

Note: `tmadmin` is supported on UNIX and Windows NT platforms.

This section provides step-by-step information about:

- What happens during a typical `tmadmin` session, which includes descriptions of the operating modes for `tmadmin` sessions and instructions for invoking them, a table showing the system requirements for access to various `tmadmin` commands, and descriptions of the `tmadmin` meta-commands: commands that help you make the best—and most efficient—use of the `tmadmin` commands.
- A procedure that you can follow to run `tmadmin` for most tasks.

  Detailed instructions for individual tasks are provided in later sections of this chapter.

How a tmadmin Session Works

How might you want to use `tmadmin` to modify your system while it is running? Consider the following sample scenario. Suppose you want to check the current values for all the parameters listed in the Bulletin Board, such as maximum number of servers and services. You can do this by running the `tmadmin` command, `bbparms`.

1. A `tmadmin` session starts when you (the administrator) enter the `tmadmin` command at a shell prompt. The shell prompt ($) is replaced by the `tmadmin` prompt (>) which is used until you quit `tmadmin`.

```
$ tmadmin [operating_mode_option]
>
```

You can request one of three operating modes on the command line: the default mode (which allows you to view and change the Bulletin Board and associated entities), read-only mode (`-r`), or configuration mode (`-c`).

2. `tmadmin` verifies that the configuration is running. If the configuration is not running, the following message is displayed:

   ```
   No bulletin board exists. Entering boot mode
   >
   ```

3. `tmadmin` checks the `TUXCONFIG` environment variable (and the optional `TUXOFFSET` variable if it is set) to get the location where the configuration file has been loaded. (Be sure you have defined the `TUXCONFIG` environment variable before beginning a `tmadmin` session.)

4. `tmadmin` enters the Bulletin Board in one of the following three states, depending on which operating mode you have requested.

   - If you have requested the default operating mode (`tmadmin` with no options), `tmadmin` enters the Bulletin Board as an administrative process, allowing you to view and make changes to configuration components and/or activities listed in the Bulletin Board.
   - If you have requested read-only mode (`tmadmin -r`), `tmadmin` enters the Bulletin Board as a client instead of as an administrator. This mode is useful if you want to leave the administrator slot unoccupied. (Only one `tmadmin` process can be the administrator at one time.) If the `-r` option is specified by a user other than the BEA WebLogic Enterprise or BEA Tuxedo administrator and security is turned on, the user is prompted for a password.
   - If you have requested configuration mode (`tmadmin -c`), `tmadmin` enters the Bulletin Board as an administrative process, allowing you to make changes to the configuration components and/or activities listed in the Bulletin Board. You can request configuration mode on any machine, whether the machine is active or inactive. (A machine is considered active if `tmadmin` can join the application as an administrative process or as a client, via a running BBL.)

5. The > prompt is displayed on your screen and you enter a `tmadmin` command.

   Not all `tmadmin` commands are available on every machine at all times. Which commands are available depends on several factors:

- The mode (read-only or configuration) of the current `tmadmin` session
- The current state of the configuration
- The type of machine on which you are working

For details, see the `tmadmin`(1) reference page in the *BEA Tuxedo Reference Manual*.

tmadmin Options

Whenever you start a `tmadmin` session, you have a choice of operating modes for that session: read-only mode, configuration mode, or the default operating mode. You can also generate a report of the BEA WebLogic Enterprise or BEA Tuxedo version and license numbers.

Read-only Mode

In this mode, you can view the data in the Bulletin Board, but you cannot make any changes. The advantage of working in read-only mode is that your administrator process is not tied up by `tmadmin`; the `tmadmin` process attaches to the Bulletin Board as a client, leaving your administrator slot available for other work.

To start a `tmadmin` session in read-only mode, specify the `-r` option on the command line:

```
$ tmadmin -r
```

Configuration Mode

In this mode, you can view the data in the Bulletin Board and, if you are the BEA Tuxedo application administrator, you can make changes. You can start a `tmadmin` session in configuration mode on any machine, including an inactive machine. On most inactive machines, configuration mode is required. (The only inactive machine on which you can start a `tmadmin` session without requesting configuration mode is the `MASTER` machine.)

To start a `tmadmin` session in configuration mode, specify the `-c` option on the command line:

```
$ tmadmin -c
```

Default Operating Mode

If you want to view and change Bulletin Board data during a `tmadmin` session, you must:

- Have administrator privileges (that is, your effective `UID` and `GID` must be those of the administrator).
- Invoke the command interpreter without any options:

```
$ tmadmin
```

Version Number and License Number Report

To find out which version of the BEA WebLogic Enterprise or BEA Tuxedo system you are running and to get the license number for it, specify the `-v` option on the command line:

```
$ tmadmin -v
```

After displaying the version and license numbers, `tmadmin` exits, even if you have specified `-c` or `-r` in addition to `-v`. When `-v` is requested, all other options are ignored.

tmadmin Metacommands

The `tmadmin` command interpreter is equipped with a set of metacommands, commands that help you use `tmadmin`. Table 15-1 lists the `tmadmin` metacommands.

Note: The tables and examples in this chapter include the abbreviated forms of the `tmadmin` command names.

Table 15-1 tmadmin Metacommands

Use This Command	Or Its Abbreviation	To...
`default`	`d`	Set defaults for arguments of other commands.
`dump`	`du`	Download the current Bulletin Board into a file.

Table 15-1 tmadmin Metacommands (Continued)

Use This Command	Or Its Abbreviation	To...
`echo`	`e`	Display input command lines.
`help`	`h`	Display command list or command syntax.
`paginate`	`page`	Pipe output of commands to a pager.
`quit`	`q`	Terminate the session.
`verbose`	`v`	Show output in verbose mode (a toggle key).
`!`*`shlcmd`*	(n/a)	Escape to the shell and run the specified shell command.
`!!`	(n/a)	Repeat the previous shell command.
<RETURN>	(n/a)	Repeat the last `tmadmin` command.

Default

The `default` metacommand (`d`) lets you set and unset defaults for the following frequently used parameters for most `tmadmin` commands: group name, server ID, machine, username, client name, queue address, service name, device blocks, device offset, JDBC connection pool name, and UDL configuration device path. For details, see the `tmadmin`(1) reference page in the *BEA Tuxedo Reference Manual*.

Note: You cannot assign defaults to any parameters for the `boot` and `shutdown` commands.

After defaults are set, they remain in effect until the session ends or until the parameters are reset to different values. The remainder of this section provides a list of instructions for checking, setting, and unsetting defaults.

- To check your current default settings, run the `default` metacommand without any options. Listing 15-1 shows an example of the report that is displayed when no parameters are set.

Listing 15-1 Default Output

```
> d
Default Settings:
     Group Name: (not set)
      Server ID: (not set)
     Machine ID: (not set)
     Queue Name: (not set)
    client Name: (not set)
   Service Name: (not set)
      User Name: (not set)
 Conn Pool Name: (not set)
         Blocks: 1000
         Offset: 0
           Path: (not set)
>
```

- To assign a new value as the default for a parameter, enter the `default` command, specifying the parameter, as follows:

  `default -`*`parameter new_value`*

  For example, to change the default of the service name to "teller," enter the following command:

  ```
  default -s teller
  ```

- To unset a default setting, run the `default` command with the appropriate option for the parameter in question, followed by the `*` wildcard argument:

  `default -`*`parameter`*` *`

  For example, to unset the default for the service name (specified with the `-s` argument), enter the following command:

  ```
  default -s *
  ```

  For most parameters, when you unset the default setting without specifying a new one, the result is that you have no default for that parameter. This generalization does not apply to the machine ID parameter, however.

  In a multiprocessor environment, the value of the machine ID can be a specific processor, the DBBL, or `all`. If the value of the machine ID is a specific processor, information is retrieved only from that processor. To remind you of this fact, the logical machine ID is added to the `tmadmin` session prompt (*`LMID`* >), as shown in Listing 15-2.

Listing 15-2 Prompt When Machine ID Is Set to a Specific Processor

```
                         # 1. default mid not previously set
> d -m SITE1             # 2. set SITE1 as default mid
SITE1 >                  # 3. prompt now shows default mid
```

If you unset the current default of the machine ID without specifying a new default, the DBBL is used automatically as the new default. In other words, if you enter:

```
default -m *
```

DBBL becomes the machine ID. You can also simply specify DBBL as the new machine by entering the following:

```
default -m DBBL
```

Optional and Required Arguments

Most `tmadmin` commands require explicit information about the resource on which the command is to act. Required arguments can always be specified on the command line, and can often be set via the `default` command, as well. `tmadmin` reports an error if the required information is not available from either source.

Some `tmadmin` statistical commands interpret unspecified default parameters as `all`.

Running tmadmin Commands

This section provides the basic procedure for running `tmadmin` commands. Commands for doing specific monitoring tasks through `tmadmin` are provided in the section "Monitoring a Running System with tmadmin" in this chapter.

Note: For complete details about `tmadmin`, see the `tmadmin`(1) reference page in the *BEA Tuxedo Reference Manual*.

To run the `tmadmin` commands:

1. Make sure the `TUXCONFIG` environment variable has been set.
2. Enter `tmadmin` in the appropriate operating mode.
   - For default mode (which allows you to view and change information listed in the Bulletin Board), do not specify any options.
   - For configuration mode, enter the `-c` option on the `tmadmin` command line.
   - For read-only mode, enter the `-r` option on the `tmadmin` command line.
3. When the `tmadmin` session prompt (>) is displayed, enter your first `tmadmin` command. Specify, on the command line, how much information from the Bulletin Board you want to have displayed.
   - For complete, detailed output, request verbose mode:

     `tmadmin_command` `-v`

     For example: `bbparms -v`
   - For abbreviated (sometimes truncated) output, request terse mode:

     `tmadmin_command` `-t`

     For example: `printjdbcconnpool -t`
4. After viewing the output of your first `tmadmin` command, continue entering `tmadmin` commands until you are ready to end the session.
5. End the `tmadmin` session by entering:

   ```
   quit
   ```

Monitoring a Running System with tmadmin

Table 15-2 provides a list of potential problems that you might want to check while monitoring your run-time system, along with a list of the `tmadmin` commands that enable you to perform such a check. The table also suggests follow-up actions you might take if the `tmadmin` command you run generates a particular type of output.

Note: For a comprehensive list of the `tmadmin` commands, see the `tmadmin`(1) reference page in the *BEA Tuxedo Reference Manual*.

Table 15-2 Commands for Monitoring Tasks

To Determine Whether . . .	Run This Command . . .	If . . .	Then . . .

Any servers are stalled in a service.	`$ tmadmin -r`
	`> printserver`
		The Current Service and Request fields do not change.	The server is spending excessive time on the current service.
			In a development environment, the server might be stalled in an infinite loop; you may want to stop it.

The load distribution is appropriate.	`$ tmadmin -r`
	`> printserver`
		The values in the Load Done field are not reasonably similar.	Check the layout of the MSSQs and the data-dependent routing.
			If the current servers have too heavy a load, you may want to boot more servers.

A particular service is doing any work.	`$ tmadmin -r`
	`> printservice`
		The value in the Requests Completed field is 0.	Data-dependent routing may be preventing requests from being sent to that server for that service. You can:
			■ Change the routing criteria or
			■ Move the service to another server.

An interface is doing any work.	`tmadmin -r`
	`>printinterface -v`
		The value in the Requests Done field is 0.	Factory-based routing may be preventing requests from being sent to that server for that interface. You can:
			■ Change the routing criteria or
			■ Move the interface to a different server

Table 15-2 Commands for Monitoring Tasks (Continued)

To Determine Whether . . .	Run This Command . . .	If . . .	Then . . .

The number of active objects is approaching the limit.	`tmadmin -r`
	`>bbparms`
	`>bbstats`
		The number of active objects reported by the `bbstats` command is close to the maximum number of objects as reported by the `bbparms` command.	You may need to increase the number of maximum objects.

An object is hung or slow in processing requests.	`tmadmin -r`
	`>printactiveobject`
		The value of the Reference Count field is greater than 1.	Clients are waiting for requests. The object may be hung or overloaded. You may need more objects performing functions currently assigned to this object.

Any clients are inactive.	`$ tmadmin -r`
	`> printclient`
		■ There has been no activity for a long time for a client, and
		■ Resources are needed.
			Tell the client—via a broadcast message—to exit.

The work is distributed in such a way that it is flowing smoothly through the system.	`$ tmadmin -r`
	`> printqueue`
		Some queues are always heavy and others are not.	Check the arrangement of services within servers, data-dependent routing, and/or queue organization.

A client is tying up a connection and preventing a server from doing any work for another client.	`$ tmadmin -r`
	`> printconn`
		A client is maintaining control of a connection and is not issuing any requests.	1. Suspend the client by using the client MIB. (We recommend using the BEA Administration Console for this task.)
			2. Terminate the client.

Table 15-2 Commands for Monitoring Tasks (Continued)

To Determine Whether . . .	Run This Command . . .	If . . .	Then . . .

A JDBC connection pool is overloaded.	`$tmadmin -v`
	`>printjdbcconnpool`
		The high-water mark (HWM) of connections in use is at or close to the maximum size, or connections in use is close to the maximum size and clients are waiting.	You may want to expand the maximum size of the pool.

The network is stable.	`$ tmadmin -r`
	`> printnet`
		A machine is no longer connected.	You may want to:
			1. Partition the machine (that is, take it off the network).
			2. Resolve the problem.
			3. Reconnect the machine.

You must manually commit or abort a transaction.	`$ tmadmin -r`
	`> printtrans`
		For example, the status is `TMGDECIDED`.	The first phase of the two-phase commit has completed successfully. This means you must find out why the second phase cannot be completed.
			
			For example, you may find that the coordinating TMS cannot complete the transaction because a participating site has gone down.

Your operating system resources (such as shared memory and semaphores) on a local machine are sufficient.	`$tmadmin -r`
	`> bbsread`
		You do not have sufficient resources in the operating system.	Increase the IPC resources (semaphores, shared memory segments, and so on) in the operating system.

Table 15-2 Commands for Monitoring Tasks (Continued)

To Determine Whether . . .	Run This Command . . .	If . . .	Then . . .

You want to keep the current values for system-wide parameters (in the `RESOURCES` section of your `UBBCONFIG` file).	`$ tmadmin -r`
	`> bbparms`
		You do not have sufficient resources for your application.	1. Stop the application.
			2. Configure additional IPC resources (assuming you have enough available) by increasing the values of relevant parameters (such as `MAXSERVERS` and `MAXCLIENTS`) in the `RESOURCES` section of the configuration file.
			3. Reboot the application.

Example: Output from tmadmin Commands

This section provides examples of output from the following `tmadmin` monitoring commands:

- `printqueue`
- `printconn`
- `printnet`
- `printtrans`
- `printjdbcconnpool`

Note: For a list of all 50 `tmadmin` commands, see the `tmadmin`(1) reference page in the *BEA Tuxedo Reference Manual*.

printqueue Output

Listing 15-3 shows output from the `printqueue` command lets you check the distribution of work in the `bankapp` application.

Listing 15-3 Printqueue Command Output

```
printqueue [qaddress]

tmadmin - Copyright © 1996-1999 BEA Systems, Inc.
Portions* Copyright 1986-1997 RSA Data Security, Inc.

>printqueue
```

Prog Name	Queue Name	# Serve	Wk Queued	# Queued	Ave. Len	Machine
TLR	28706	1	0	0	0.0	SITE1
TMS_SQL	BANKB1_T	2	0	0	0.0	SITE1
TLR	24946	1	0	0	0.1	SITE1
BAL	8533	1	0	0	0.0	SITE1
BAL	24915	1	0	0	0.0	SITE1
BTADD	28897	1	0	0	0.0	SITE1
XFER	4380	1	0	0	0.0	SITE1
XFER	28840	1	100	0	1.0	SITE1
TLR	12519	1	100	2	0.0	SITE1
BBL	24846	1	0	2	0.0	SITE1
ACCT	71	1	0	0	0.0	SITE1
TMS_SQL	BANKB3_T	2	0	0	0.0	SITE1
BAL	28958	1	0	0	0.0	SITE1
ACCT	254	1	0	0	0.0	SITE1
BTADD	12310	1	0	0	0.0	SITE1
XFER	16494	1	0	0	0.0	SITE1
TMS_SQL	BANKB2_T	2	0	0	0.0	SITE1
BTADD	8430	1	0	0	0.0	SITE1
ACCT	24641	1	0	0	0.0	SITE1

Note: By default, information is supplied for all queues. If you want your output to be limited to information about only one queue, specify the address for the desired queue.

The output of this command includes the following information:

In the Column Labeled . . .	You See . . .
`Prog Name`	The name of the executable to which the queue is connected.
`Queue Name`	The symbolic queue name (set to either the `RQADDR` parameter of `UBBCONFIG` or a randomly chosen value).
`# Serve`	The number of servers connected to the queue.
`Wk Queued`	The load factor of all requests currently queued.
`# Queued`	The actual number of requests queued.
`Ave. Len`	The average queue length. Not available in MP mode.
`Machine`	The LMID of the machine on which the queue is located.

printconn Data

Listing 15-4 shows verbose output from the `printconn` command shows that the client process has:

- Two conversations were initiated
- Control of both lines was maintained
- No requests were sent yet

Listing 15-4 Printconn Command Output

```
printconn [-m machine]

tmadmin - Copyright © 1996-1999 BEA Systems, Inc.
```

```
> echo
Echo now on.

> v
Verbose now on.

> pc

Originator
         Group/pid:    Client/29704
              LMID:    SITE1
             Sends:    0
Subordinate
   Group/server id:    Group1/2
              LMID:    SITE1
             Sends:    -
           Service:   TOUPPER1
Originator
         Group/pid:    Client/29704
              LMID:    SITE1
             Sends:    0
Subordinate
   Group/server id:    Group1/2
              LMID:    SITE1
             Sends:    -
           Service:    TOUPPER2
```

printnet Command Output

Listing 15-5 shows the output from the following procedure.

1. The `printnet` command was run. (The output shows the number of messages sent and received by both sites.)

2. The `BRIDGE` process at `SITE2` was stopped.

3. The `printnet` command was re-entered. (The output shows that `SITE2` is no longer connected to the master machine, `SITE1`.)

Listing 15-5 Printnet Command Output

```
printnet [-m machine_list]

tmadmin - Copyright © 1996-1999 BEA Systems, Inc. All rights
reserved.

> echo
Echo now on.

> pnw
SITE1     Connected To:   msgs sent   msgs received
SITE2       100103

SITE2     Connected To:   msgs sent   msgs received
SITE1       104             101

> pnw
SITE1     Connected To:   msgs sent   msgs received

           Could not retrieve status from SITE2

>
```

printtrans Command Output

The `printtrans` command reports statistics only for transactions that are currently in progress, specifically, statistics on the number of rollbacks, commits, and aborts that have been executed on your machine, group, or server.

Listing 15-6 shows the output produced by running the `printtrans` command in both terse and verbose modes:

- In terse mode, the `GTRID` (a unique string that identifies a transaction across an application) and the transaction state are shown.
- In verbose mode, information about timeouts and participants is added.

Note: The index shown in the example is used by the administrator to commit or abort the transaction.

Listing 15-6 Printtrans Command Output

```
printtrans [-m machine] [-g groupname]

tmadmin - Copyright © 1996-1999 BEA Systems, Inc.
Portions* Copyright 1986-1997 RSA Data Security, Inc.
All rights reserved.

> printtrans
>> index=0>gtrid=x0 x2bb8f464 x1
:   Machine id: SITE1, Transaction status: TMGACTIVE
    Group count: 1

> v
Verbose now on.

> pt
>> index=0>gtrid=x0 x2bb8d464 x1
:   Machine id: SITE1, Transaction status: TMGACTIVE
    Group count: 1, timeout: 300, time left: 299
    Known participants:
       group: GROUP1, status: TMGACTIVE, local, coord

>
```

printjdbcconnpool Command Output

The `printjdbcconnpool` command reports statistics on JDBC connection pools such as the maximum number of connections per pool, the number of connections in use, the number of clients waiting for a connection, and the high-water mark or highest number of connections used for a pool.

Listing 15-7 shows the output produced by running the `printjdbcconnpool` command in terse and verbose modes. In terse mode the maximum pool size, the current pool size, and the number of connections currently in use are shown. In verbose mode the number of clients waiting and the high-water mark are also shown.

Listing 15-7 Printjdbcconnpool Command Output

```
>printjdbcconnpool
Pool Name   Grp Name   Srv Id Size Max Size Used
----------- ---------- ------ ---- -------- ----
ejbPool     J_SRVGRP    101      1       15    0
Pool2       J_SRVGRP    102     10       30    3
```

Listing 15-8 shows is the verbose mode output for a single connnection pool:

Listing 15-8 Printjdbcconnpool Command Verbose Mode Output

```
             Pool Name: Pool2
              Group ID: J_SRVGRP
             Server ID: 102
                Driver: (none)
                   URL: (none)
         Database Name: Db
                  User: leia
                  Host: SITE1
              Password: mypwd
          Net Protocol: odbc
                  Port: 120
                 Props: (none)
             Enable XA: No
     Create On Startup: Yes
             Pool Size: 10
          Maximum Size: 30
    Capacity increment: 3
       Allow shrinking: Yes
       Shrink interval: 10 min(s)
           Login delay: 1 sec(s)
    Connections in use: 3
  Connections awaiting: 0
HWM connections in use: 5
            Test table: testtable
      Refresh interval: 20 sec(s)
   Test conn OnReserve: Yes
   Test conn OnRelease: No
```

Case Study: Monitoring Run-time bankapp

This section presents a sample configuration for a multiprocessor (MP) version of the `bankapp` application. This section also shows the output that was returned when the local IPC resources and system-wide parameters were checked by running the appropriate `tmadmin` commands.

Configuration File for bankapp

For this case study, we will use the configuration defined in the `UBBCONFIG` file shown in Listing 15-9.

Listing 15-9 UBBCONFIG File for bankapp (MP Version)

```
#Copyright (c) 1997, 1998 BEA Systems, Inc.
#All rights reserved

*RESOURCES
IPCKEY              80952
UID                 4196
GID                 601
PERM                0660
MAXACCESSERS        40
MAXSERVERS          35
MAXSERVICES         75
MAXCONV             10
MAXGTT              20
MASTER              SITE1,SITE2
SCANUNIT            10
SANITYSCA
       12
BBLQUERY            180
BLOCKTIME           30
DBBLWAIT            6
OPTIONS             LAN,MIGRATE
MODEL               MP
LDBAL               Y
#
*MACHINES
mchn1               LMID=SITE1
                    TUXDIR="/home/tuxroot"
```

```
                  APPDIR="/home/apps/bank"
                  ENVFILE="/home/apps/bank/ENVFILE"
                  TLOGDEVICE="/home/apps/bank/TLOG"
                  TLOGNAME=TLOG
                  TUXCONFIG="/home/apps/bank/tuxconfig"
                  TYPE="3B2"
                  ULOGPFX="/home/apps/bank/ULOG"
wgs386            LMID=SITE2
                  TUXDIR="/home2/tuxroot"
                  APPDIR="/home2/apps/bank"
                  ENVFILE="/home2/apps/bank/ENVFILE"
                  TLOGDEVICE="/home2/apps/bank/TLOG"
                  TLOGNAME=TLOG
                  TUXCONFIG="/home2/apps/bank/tuxconfig"
                  TYPE="386"
                  ULOGPFX="/home2/apps/bank/ULOG"
#
*GROUPS
DEFAULT:  TMSNAME=TMS_SQL   TMSCOUNT=2
# For NT/Netware,  :bankdb: becomes ;bankdb;
BANKB1              LMID=SITE1         GRPNO=1
         OPENINFO="TUXEDO/SQL:/home/apps/bank/bankdl1:bankdb:readwrite"
BANKB2              LMID=SITE2         GRPNO=2
        OPENINFO="TUXEDO/SQL:/home2/apps/bank/bankdl2:bankdb:readwrite"

*NETWORK
SITE1     NADDR="//mach1.beasys.com:1900"
          BRIDGE="/dev/tcp"
          NLSADDR="//mach1.beasys.com:1900"
SITE2     NADDR="//mach386.beasys.com:1900"
          BRIDGE="/dev/tcp"
          NLSADDR="//mach386.beasys.com:1900"
*SERVERS
#
DEFAULT:  RESTART=Y MAXGEN=5 REPLYQ=Y CLOPT="-A"

TLR      SRVGRP=BANKB1    SRVID=1    RQADDR=tlr1    CLOPT="-A -- -T 100"
TLR      SRVGRP=BANKB1    SRVID=2    RQADDR=tlr1    CLOPT="-A -- -T 200"
TLR      SRVGRP=BANKB2    SRVID=3    RQADDR=tlr2    CLOPT="-A -- -T 600"
TLR      SRVGRP=BANKB2    SRVID=4    RQADDR=tlr2    CLOPT="-A -- -T 700"
XFER      SRVGRP=BANKB1   SRVID=5
XFER      SRVGRP=BANKB2   SRVID=6
ACCT      SRVGRP=BANKB1   SRVID=7
ACCT      SRVGRP=BANKB2   SRVID=8
BAL       SRVGRP=BANKB1   SRVID=9
BAL       SRVGRP=BANKB2   SRVID=10
BTADD     SRVGRP=BANKB1
BTADD     SRVGRP=BANKB2   SRVID=12
AUDITC    SRVGRP=BANKB1   SRVID=13 CONV=Y MIN=1 MAX=10
BALC      SRVGRP=BANKB1   SRVID=24
BALC      SRVGRP=BANKB2   SRVID=25
#
```

```
*SERVICES
DEFAULT:  LOAD=50          AUTOTRAN=N
WITHDRAWAL         PRIO=50          ROUTING=ACCOUNT_ID
DEPOSIT            PRIO=50          ROUTING=ACCOUNT_ID
TRANSFER           PRIO=50          ROUTING=ACCOUNT_ID
INQUIRY            PRIO=50          ROUTING=ACCOUNT_ID
CLOSE_ACCT         PRIO=40          ROUTING=ACCOUNT_ID
OPEN_ACCT          PRIO=40          ROUTING=BRANCH_ID
BR_ADD             PRIO=20          ROUTING=BRANCH_ID
TLR_ADD            PRIO=20          ROUTING=BRANCH_ID
ABAL               PRIO=30          ROUTING=b_id
TBAL               PRIO=30          ROUTING=b_id
ABAL_BID           PRIO=30          ROUTING=b_id
TBAL_BID           PRIO=30          ROUTING=b_id
ABALC_BID          PRIO=30          ROUTING=b_id
TBALC_BID          PRIO=30          ROUTING=b_id

*ROUTING
ACCOUNT_ID         FIELD=ACCOUNT_ID
                   BUFTYPE="FML"
                   RANGES="10000-59999:BANKB1,
                            60000-109999:BANKB2,
                            *:*"
BRANCH_ID FIELD=BRANCH_ID
                   BUFTYPE="FML"
                   RANGES="1-5:BANKB1,
                            6-10:BANKB2,
                            *:*"
b_id               FIELD=b_id
                   BUFTYPE="VIEW:aud"
                   RANGES="1-5:BANKB1,
                            6-10:BANKB2,
                            *:*"
```

Output from Checking the Local IPC Resources

To check the local IPC resources for this configuration, a `tmadmin` session was started, and the `bbsread` command was run. The output of `bbsread` is shown in Listing 15-10.

Listing 15-10 bbsread Output

```
SITE1> bbsread

IPC resources for the bulleti
board o
machine SITE1:
SHARED MEMORY:           Key: 0x1013c38
SEGMENT 0:
                          ID: 15730
                        Size: 36924
         Attached processes: 12
      Last attach/detach by: 4181

This semaphore is the system semaphore
SEMAPHORE:               Key: 0x1013c38
                          Id: 15666
       |  semaphore |  current |   last   | # waiting |
       |   number   |  status  | accesser | processes |
       |-----------------------------------------------|
       |     0      |   free   |   4181   |     0     |
       |------------|----------|----------|-----------|
This semaphore set is part of the user-level semaphore
SEMAPHORE:               Key: IPC_PRIVATE
                          Id: 11572
       |  semaphore |  current |   last   | # waiting |
       |   number   |  status  | accesser | processes |
       |-----------------------------------------------|
       |      0     |  locked  |   4181   |     0     |
       |      1     |  locked  |   4181   |     0     |
       |      2     |  locked  |   4181   |     0     |
       |      3     |  locked  |   4181   |     0     |
       |      4     |  locked  |   4181   |     0     |
       |      5     |  locked  |   4181   |     0     |
       |      6     |  locked  |   4181   |     0     |
       |      7     |  locked  |   4181   |     0     |
       |      8     |  locked  |   4181   |     0     |
       |      9     |  locked  |   4181   |     0     |
       |     10     |  locked  |   4181   |     0     |
       |     11     |  locked  |   4181   |     0     |
       |     12     |  locked  |   4181   |     0     |
       |     13     |  locked  |   4181   |     0     |
       |------------|----------|----------|-----------|
```

Note: The display is the same with verbose mode on or off.

Output from Checking System-wide Parameter Settings

To check the current values of the system-wide parameters for this configuration, we started a `tmadmin` session and ran the `bbparms` command. The output of `bbparms` is shown in Listing 15-11.

Listing 15-11 Sample bbparms Output

```
> bbparms
Bulletin Board Parameters:
        MAXSERVERS: 50
       MAXSERVICES: 100
      MAXACCESSERS: 50
            MAXGTT: 100
           MAXCONV: 1
        MAXBUFTYPE: 16
       MAXBUFSTYPE: 32
        MAXOBJECTS: 1000
     MAXINTERFACES: 150
            IPCKEY: 35384
            MASTER: SITE1,SITE2
             MODEL: MP
             LDBAL: Y
           OPTIONS: LAN,MIGRATE
          SCANUNIT: 10
        SANITYSCAN: 12
          DBBLWAIT: 6
          BBLQUERY: 30
         BLOCKTIME: 6
  Shared Memory ID: 0
```

Note: The display is the same with verbose mode on or off.

CHAPTER

16 Monitoring Log Files

To help you identify error conditions quickly and accurately, the BEA WebLogic Enterprise and BEA Tuxedo systems provide you with two log files:

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

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

This topic includes the following sections:

- What is the ULOG?
- What Is the Transaction Log (TLOG)?
- Creating and Maintaining Logs
- Using Logs to Detect Failures

What is the ULOG?

The user log (`ULOG`) is a central event logger. All messages generated by the BEA WebLogic Enterprise or BEA Tuxedo system—error messages, warning messages, information messages, and debugging messages—can be written to this log. Application clients and servers can also write to the user log.

A new ULOG is created every day and there can be a different log on each machine. When a remote file system is being used, however, a ULOG can be shared across machines.

The ULOG also contains messages generated by the tlisten process. The tlisten process provides remote service connections for other machines. Each machine, including the master machine, should have a tlisten process running on it.

In previous BEA Tuxedo releases, tlisten had its own log file. Now, all messages from the tlisten are written in a ULOG format.

Note: It is possible that the tlisten might have its own ULOG file that is different from the application's ULOG file. This is because the tlisten process starts before any application is started. Therefore, tlisten writes the ULOG in the current directory where it was started, or in the directory in the ULOGPFX environment variable, if one was defined. The application's ULOG file is usually present in the $APPDIR directory.

Purpose

The purpose of the ULOG is to give you, the administrator, a record of the events on your system from which you can determine the cause of most BEA WebLogic Enterprise or BEA Tuxedo system and application failures.

How Is the ULOG created?

The ULOG is created by the BEA WebLogic Enterprise or BEA Tuxedo system whenever one of the following events occurs:

- A new configuration file is loaded
- An application is booted

With the exception of the RMI trace and JDBC log information, all messages generated by the BEA WebLogic Enterprise or BEA Tuxedo system are written to this log by default.

- To have the RMI trace information written to the ULOG file, you must set the -rmilog CLOPT attribute in the SERVERS section of the UBBCONFIG file.

- To have the JDBC log information written to the `ULOG` file, you must set the `-jdbclog` CLOPT attribute in the `SERVERS` section of the `UBBCONFIG` file.

For a discussion of these attributes, see “Configuring Servers” on page 3-34.

How Is the ULOG Used?

You can view the `ULOG`, an ASCII file, with any text editor.

When a message is written to the `ULOG` through the `tperrno` global variable, application clients and servers are notified, as follows:

- If `tperrno` is set to `TPESYSTEM` after returning from an ATMI call, you can conclude that:
    - A BEA WebLogic Enterprise or a BEA Tuxedo system error has occurred.
    - An error message has been placed in the user log.
- If `tperrno` is set to `TPEOS` after returning from an ATMI call, you can conclude that:
    - An operating system error has occurred.
    - An error message has been placed in the user log.

Message Format

A `ULOG` message consists of two parts: a tag and text. Each part consists of three strings, as shown in the following table.

This Part . . .	Consists of . . .
tag	A 6-digit string (*`hhmmss`*) representing the time of day (in terms of hour, minute, and second). Name of the machine (as returned, on UNIX systems, by the `uname -n` command). Name and identifier of the process that is logging the message.

This Part . . .	Consists of . . .
text	Message catalog name
	Message number
	System message

Consider the following example of a user log message.

```
121449.gumby!simpserv.27190: LIBTUX_CAT:262: std main starting
```

From the message tag we learn:

- The message was written into the log at around 12:15 P.M.
- The machine on which the error occurred was `gumby`.
- The message was logged by the `simpserv` process (which has an ID of 27190).

From the message text we learn:

- The message came from the `LIBTUX` catalog.
- The number of the message is 262.
- The message itself reads as follows: `std main starting.`

For more information about a message, note its catalog name and catalog number. With this information you can look up the message in the *System Messages* and *BEA Tuxedo System Message Manual*, which provide complete descriptions of all system messages.

Location

By default, the user log is called `ULOG.`*`mmddyy`* (where *`mmddyy`* represents the date in terms of month, day, and year) and it is created in the `$APPDIR` directory.

You can place this file in any location, however, by setting the `ULOGPFX` parameter in the `MACHINES` section.

What Is the Transaction Log (TLOG)?

The transaction log (`TLOG`) keeps track of global transactions during the commit phase. A global transaction is recorded in the `TLOG` only when it is in the process of being committed. The `TLOG` is used to record the reply from the global transaction participants at the end of the first phase of a two-phase-commit protocol. The `TLOG` records the decision about whether a global transaction should be committed or rolled back.

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

How Is the TLOG Created?

For instructions on creating a `TLOG`, see the section “Creating a Transaction Log (TLOG)” on page 16-7.

How Is the TLOG Used?

The `TLOG` file is used only by the Transaction Manager Server (TMS) that coordinates global transactions. It is not read by the administrator.

Location

The location and size of the `TLOG` are specified by four parameters that you set in the `MACHINES` section of the `UBBCONFIG` file: `TLOGDEVICE`, `LOGOFFSET`, `TLOGNAME`, and `TLOGSIZE`. (For descriptions of these parameters and instructions for assigning values to them, see “Creating a Transaction Log (TLOG)” on page 16-7.)

Creating and Maintaining Logs

The ULOG is generated by various BEA WebLogic Enterprise or BEA Tuxedo system processes; you do not need to create it. The TLOG, however, is not produced automatically; you must create it.

This section provides instructions for:

- Maintaining the ULOG
- Creating TLOGs

How to Assign a Location for the ULOG

To override the default location for your ULOG file, specify the desired location as the value of the ULOGPFX parameter in the MACHINES section of the UBBCONFIG file. (By default, the value of ULOGPFX is $APPDIR/ULOG.) The value you assign becomes the first part of the ULOG filename.

Listing 16-1 shows how you can override the default setting.

Listing 16-1 Overriding Default Settings in the MACHINES Section of Your UBBCONFIG File

```
MACHINES
gumby LMID=SITE1
TUXDIR=”/usr/tuxedo”
APPDIR=”/home/apps”
TUXCONFIG=”/home/apps/tuxconfig”
ULOGPFX=”/home/apps/logs/ULOG”
...
```

The following ULOG was created for SITE1 on 04/13/98.

```
/home/apps/logs/ULOG.041398
```

Creating a Transaction Log (TLOG)

To create a `TLOG`, you must complete the following procedure:

- Step 1: Assign Values to MACHINES Section Parameters
- Step 2: Create a UDL Entry
- Step 3 (optional): Allocate Space for a New Device on an Existing System
- Step 4: Create the Log

This section provides instructions for each step.

Step 1: Assign Values to MACHINES Section Parameters

Your first step is to assign values to four parameters in the `MACHINES` section of the `UBBCONFIG` file: `TLOGDEVICE`, `TLOGNAME`, `TLOGOFFSET`, and `TLOGSIZE`.

`TLOGDEVICE`

`TLOGDEVICE` specifies the device in the BEA Tuxedo file system that contains the transaction log. This can be the same device used by `TUXCONFIG`.

Note: Technically, there is no reason that `TLOGDEVICE` cannot be a separate VTOC file, but there are two reasons why it is not recommended: the `TLOG` is generally too small to justify devoting a raw disk segment to it, and creating `TLOGDEVICE` as a UNIX file leads to expensive delays when synchronous writes to the `TLOG` are required.

The `TLOG` is stored as a BEA WebLogic Enterprise or a BEA Tuxedo system VTOC table on the device named in this parameter. If the `TLOGDEVICE` parameter is not specified, there is no default; the BEA WebLogic Enterprise or BEA Tuxedo system assumes that no `TLOG` exists for the machine. If no `TLOG` exists for a given machine, the associated `LMID` cannot be used by server groups that participate in distributed transactions.

After `TUXCONFIG` has been created via `tmloadcf`, you must create a device list entry for the `TLOG` on each machine for which `TLOGDEVICE` is specified. This is done using the `tmadmin crdl` command. The BBL creates the log automatically the first time the system is booted.

`TLOGNAME`

`TLOGNAME` specifies the name of the Distributed Transaction Processing (DTP) transaction log for this machine. The default name is `TLOG`. If more than one transaction log exists on the same `TLOGDEVICE`, each transaction log must have a unique name. If a name is specified, it must not conflict with any other table specified on the configuration.

`TLOGOFFSET`

`TLOGOFFSET` specifies the offset in pages from the beginning of `TLOGDEVICE` to the start of the VTOC that contains the transaction log for this machine. The number must be greater than or equal to 0 and less than the number of pages on the device. The default value is 0.

`TLOGOFFSET` 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, `TLOGOFFSET` can be used to indicate a starting address relative to the address of the device.

`TLOGSIZE`

`TLOGSIZE` specifies the number of pages for the `TLOG`. The default is 100 pages. Once a global transaction is complete, `TLOG` records are no longer needed and are thrown away. The maximum number of pages that can be specified, subject to the amount of available space on `TLOGDEVICE`, is 2048 pages. Choosing a value is entirely application-dependent.

Listing 16-2 shows an example of the use of transaction log parameters.

Listing 16-2 Sample Transaction Log Parameters for a Specified Machine

```
MACHINES
gumby LMID=SITE1
...
TLOGDEVICE=”/home/apps/logs/TLOG”
TLOGNAME=TLOG
TLOGOFFSET=0
TLOGSIZE=100
...
```

Step 2: Create a UDL Entry

Next, create an entry in the Universal Device List (UDL) for the `TLOGDEVICE` on each machine that requires a `TLOG`. You can perform this step either before or after `TUXCONFIG` has been loaded, but you must do it before the system is booted.

To create an entry in the UDL for the `TLOG` device, complete the following procedure.

1. On the master machine (with the application inactive), enter the following.

   ```
   tmadmin -c
   ```

   You do not have to create `TLOG`s on any machine other than the master machine. The BEA WebLogic Enterprise or BEA Tuxedo system creates `TLOG`s on nonmaster machines (as long as a UDL exists on those machines) when the application is booted.

2. Enter the command:

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

   - `-z` *`config`* specifies the full path name for the device on which the UDL should be created (and where the `TLOG` will reside).
   - `-b` specifies the number of blocks to be allocated on the device.
   - *`config`* should match the value of the `TLOGDEVICE` parameter in the `MACHINES` section. If *`config`* is not specified, it defaults to the value of `FSCONFIG` (a BEA WebLogic Enterprise or a BEA Tuxedo system environment variable).

3. Repeat steps 1 and 2 on each machine of your application that will participate in global transactions.

   Note: If the `TLOGDEVICE` is mirrored between two machines, step 3 is not required on the paired machine.

During the boot process, the Bulletin Board Listener (BBL) initializes and opens the `TLOG`.

Step 3 (optional): Allocate Space for a New Device on an Existing System

In step 2 you created a new BEA WebLogic Enterprise or BEA Tuxedo file system that can be used to hold the `TLOG`. Sometimes, however, it is necessary to add new devices or space to an existing configuration or to check space usage. You can perform these tasks by running the command:

```
tmadmin -c
```

(You can run this command whether or not the system is booted.)

It is possible that the UDL exists on *`config`* but does not have sufficient space for the log. To allocate space on a new device to an existing BEA Tuxedo file system, enter:

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

where *`new_device`* specifies the full path name for the new device where space is to be allocated. This creates a new entry on the UDL and the space is available for any tables that are created on *`config.`* (For example, this procedure can be used for the `TUXCONFIG` file when there is not enough space for a modified configuration, for allocating a new `TLOG`, or for increasing the size of the `TLOG` by deleting an old log and then creating a larger one.) If you are running several commands using the current configuration, it is possible to set the default configuration by entering the `default` command (`d`), as follows:

```
d -z config
```

If you run this command, you will not need to enter the `-z` option after each command.

Under rare circumstances, a device does not start at offset 0. This might happen if space has been allocated on a device (less than the entire device) to a BEA Tuxedo file system, and more space on the same device is available to be allocated. In this case, you can allocate the second entry by entering the following command:

```
crdl -z config -b blocks -o new_device_offset new_device
```

Here, *`new_device_offset`* specifies the offset of the new space being allocated on the device. (Note that the option is a lowercase `o.`) In this case, since the first entry on the UDL is allocated at offset 0, `TLOGOFFSET` and/or `TUXOFFSET` are set to 0, instead of to the offset of the new device. (The BEA WebLogic Enterprise or BEA Tuxedo system needs to find the UDL, from which it can determine the offset of other available space.)

A second (and rarer) reason that a device does not start at offset 0 is that a single raw device is shared. This happens, for example, if a UNIX file system is followed by a BEA Tuxedo file system on the same device. (This situation is risky because the raw device must be writable by the BEA WebLogic Enterprise or BEA Tuxedo system administrator and it is possible to overwrite the UNIX file system.) If the first entry on the UDL does not start at offset 0 (as in this example), the device offset must be specified everywhere that the device is referenced. To allocate the entry, enter the following command:

```
crdl -z config -b blocks -o offset -O offset new_device
```

Here, *offset* is the offset of the space to be allocated for the BEA Tuxedo file system (UDL and tables). Note that the -o (lowercase o) specifies the offset of the UDL and -O (uppercase O) specifies the offset of the new device space being allocated, which in this case are the same. Any devices that are created subsequently on this configuration must use both the -o option with the offset of the first entry, and the -O option with the offset of the new entry. (The offset may be 0 if a new device is being specified.) Since the first entry on the UDL is not allocated at offset 0, TLOGOFFSET and/or TUXOFFSET must be set to the offset of the first entry. This is the only case in which TLOGOFFSET and TUXOFFSET must be set in the UBBCONFIG file, and the TUXOFFSET environment variable must be set when all BEA Tuxedo application and administrative processes are being run.

To list the current UDL, enter:

```
lidl -z config
```

where *config* was created using the above procedures. If the first entry was created with an offset other than 0, -o *offset* must be specified in addition to the configuration device. In verbose mode, this command lists not only the space initially allocated for each device entry, but also the amount of free space.

It is also possible to generate a list of the tables on the configuration by entering the following command:

```
livtoc -z config
```

Here *config* was creating using the above procedures. If the first entry was created with an offset other than 0, -o *offset* must be specified in addition to the configuration device. This command lists the table name, device number, offset within the device, and number of pages for each table. The first two tables are always VTOC and UDL. TUXCONFIG table names are of the form _*secname*_SECT, where *secname* is the name of a section in the UBBCONFIG file. The TLOG table name is based on the TLOG parameter in the UBBCONFIG file, and defaults to TLOG. In the rare case in which two applications share a single BEA Tuxedo file system for the TLOGDEVICE, the TLOG parameter must be different for each application.

Note: A BEA Tuxedo system file system is a file that is managed by BEA Tuxedo, which may be located on a raw disk or in an operating system file system. A BEA Tuxedo system file system contains one TUXCONFIG file and one or more TLOG files.

Because the table names for the `TUXCONFIG` file are fixed, it is not possible for two applications to share the same BEA Tuxedo file system for the `TUXCONFIG` file.

Step 4: Create the Log

To create the log, complete the following steps:

1. Make sure you have a `TUXCONFIG` file. (If you do not, the commands for creating the `TLOG` will fail.)
2. Start a `tmadmin` session by entering:

   ```
   tmadmin -c
   ```

3. At the `tmadmin` command prompt (>), enter:

   ```
   crlog [-m machine]
   ```

   where the value of *`machine`* is the `LMID` of a machine, as specified in `TUXCONFIG`.

Note: The `-m` option is shown as optional because it can be specified with the `default` (`d`) command of `tmadmin`. If you have not specified a machine with the `d` command, however, the `-m` option is required on the `crlog` command line.

Maintaining a TLOG

There are few tasks needed to maintain a transaction log (`TLOG`). Two of them are as follows:

- To reinitialize a `TLOG`, enter:

  ```
  inlog [-yes] [-m machine]
  ```

  The value of *`machine`* is the `LMID` of a machine, as specified in `TUXCONFIG`.

  Be careful when using this command: it will reinitialize the log even if there are outstanding transactions. The result could be inconsistent `TLOG`s, possibly causing transactions to abort.

- To destroy a `TLOG`, enter:

  ```
  dslog [-yes] [-m machine]
  ```

  The value of *`machine`* is the `LMID` of a machine, as specified in `TUXCONFIG`.

If the application is not active or if there are transactions still outstanding in the log, an error will be returned.

Note: The `-yes` and `-m` options are shown as optional because they can be specified with the `default` (`d`) command. If you have not specified a machine with the `d` command, however, the `-m` option is required on the `inlog` and `dslog` command lines.

Using Logs to Detect Failures

The BEA Tuxedo log files can help you detect failures in both your application and your system. This section provides instructions for analyzing the data in the logs.

Analyzing the User Log (ULOG)

Note: Although application administrators are responsible for analyzing user logs, application programmers may also consult the logs.

It is not unusual for multiple messages to be placed in the user log for a given problem. In general, the earlier messages will better reflect the exact nature of the problem.

Listing 16-3 shows how the `LIBTUX_CAT` message 358 identifies the exact nature of the problem, namely, that there are not enough UNIX system semaphores to boot the application.

Listing 16-3 Sample ULOG Messages

```
151550.gumby!BBL.28041: LIBTUX_CAT:262: std main starting
151550.gumby!BBL.28041: LIBTUX_CAT:358: reached UNIX limit on semaphore ids
151550.gumby!BBL.28041: LIBTUX_CAT:248: fatal: system init function ...
151550.gumby!BBL.28040: CMDTUX_CAT:825: Process BBL at SITE1 failed ...
151550.gumby!BBL.28040: WARNING: No BBL available on site SITE1.
        Will not attempt to boot server processes on that site.
```

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

Analyzing tlisten Messages

Keep the following guidelines in mind as you check the `tlisten` messages in your `ULOG`:

- A message is placed in the log every time the log is contacted.
- A sequence number is given to every accepted request.
- If you cannot boot your application and subsequently cannot find any `tlisten` messages in your `ULOG` file, one of the following problems may have occurred:
  - The `tlisten` process may not have been started.
  - The `tlisten` process may be listening on the wrong network address.

  To find out whether one of these errors has occurred, check the `ULOG` file.
- It is possible that the `tlisten` might have its own `ULOG` file that is different from the application's `ULOG` file. This is because the `tlisten` process starts before any application is started. Therefore, `tlisten` writes the `ULOG` in the current directory where it was started, or in the directory in the `ULOGPFX` environment variable, if one was defined. The application's `ULOG` file is usually present in the `$APPDIR` directory.

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

The `CMDTUX` catalog in the *BEA Tuxedo System Message Manual* contains the following information about `tlisten` messages:

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

Example

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

```
042398; 27909;CMDTUX_CAT: 615 INFO: Terminating tlisten process
```

This message was recorded on April 23, 1998. Its purpose is simply to provide information: the `tlisten` process is being terminated. No action is required.

Note: This message can be found in the `CMDTUX` catalog of the *BEA Tuxedo System Message Manual*.

Analyzing a Transaction Log (TLOG)

The `TLOG` is a binary file that contains only messages about global transactions that are in the process of being committed. You should never need to examine this file.

If you do need to view the `TLOG`, you must first convert it to ASCII format so that it is readable. The BEA Tuxedo system provides two `tmadmin` commands for this purpose:

```
dumptlog (dl) -z config [ -o offset ] [ -n name ]
     [ -g groupname ] filename
```

This command downloads (or dumps) the `TLOG` (a binary file) to an ASCII file.

```
loadtlog -m machine filename
```

This command uploads (or loads) an ASCII version of the `TLOG` into an existing `TLOG` (a binary file).

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

For more information about these `tmadmin` commands, see `tmadmin`(1) in the *BEA Tuxedo Reference*.

CHAPTER

17 Tuning Applications

For a detailed discussion of tuning applications administrative information, see the chapter *Tuning Applications* in *Tuning and Scaling* in the BEA WebLogic Enterprise online documentation.

CHAPTER

18 Migrating Applications

This topic includes the following sections:

- About Migration
- Migration Options
- Switching Master and Backup Machines
- Migrating a Server Group
- Migrating Machines
- Canceling a Migration
- Migrating Transaction Logs to a Backup Machine

Note: A migration requirement is that both the master and backup machines must be running the same release of the BEA WebLogic Enterprise software, or the same release of the BEA Tuxedo software.

About Migration

Whether you need to migrate all or portions of an application, the changes to the application setup must be made with minimal service disruption. Machines, networks, databases, the BEA WebLogic Enterprise or BEA Tuxedo system, and the application all need to be maintained. The BEA WebLogic Enterprise and BEA Tuxedo systems provide a way to migrate the applications so that they can be serviced.

The BEA WebLogic Enterprise and BEA Tuxedo systems offer migration tools that can also be used to recover from a machine crash, network partitions, database corruptions, BEA WebLogic Enterprise or BEA Tuxedo system problems, and application faults.

Migration Options

The following is a list of migration options:

- Switch master and backup machines
- Migrate a server group from its primary machine to its alternate machine
- Migrate all server groups from their primary machine to their alternate machine
- Cancel a migration
- Migrate a transaction log

By using a combination of these options and partitioned network recovery utilities, you can migrate entire machines.

Switching Master and Backup Machines

Server migration is the process of moving one or more servers from one machine to another. One special instance of this process is the ability to switch master and backup machines. This type of switching is done by migrating the DBBL from the master machine to the backup machine. While this procedure is most frequently used when a network is partitioned, it is also useful in situations that require you to shut down the master machine.

Use the `master` command to switch the master machine.

Command	Description
`master(m)`	Switches the master machine to the backup machine or the reverse

Use the `tmadmin`(1) `master` (`m`) command to switch master and backup machines when the master machine must be shut down for maintenance, or when the master machine is no longer accessible. Switching master and backup machines, however, is only a first step. In most cases, application servers need to be migrated to alternate sites, or the master machine needs to be restored. (These tasks are described in this chapter.)

How to Switch the Master and Backup Machines

To switch the master and backup machines, call the `tmadmin`(1) command interpreter with the `master` (`m`) command from the backup machine.

Examples: Switching Master and Backup Machines

The following examples illustrate how you can switch master and backup machines.

In the first example (Listing 18-1), the master machine is accessible from the backup machine, and the `DBBL` process is migrated from the master machine to the backup machine.

Listing 18-1 When the Master Machine Is Accessible from the Backup Machine

```
$ tmadmin
tmadmin - Copyright © 1999 BEA.  All rights reserved.
> master
are you sure? [y,n] y
Migrating active DBBL from SITE1 to SITE2, please wait...
DBBL has been migrated from SITE1 to SITE2
> q
```

In the second example (Listing 18-2), because the master machine is not accessible from the backup machine, the DBBL process is created on the backup machine.

Listing 18-2 When the Master Machine Is Not Accessible from the Backup Machine

```
$ tmadmin
> master
are you sure? [y,n]  y
Creating new DBBL on SITE2, please wait... New DBBL created on SITE2
> q
```

Migrating a Server Group

Use the following two tmadmin commands to migrate servers.

Use This Command	To
migrategroup(migg)	Migrate servers in a group to their alternate location
migratemach(migm)	Migrate servers by using LMIDs

The tmadmin(1) migrategroup (migg) command takes the name of a single server group as an argument. You must first shut down the servers to be migrated with the -R option (for example, tmshutdown -R -g GROUP1).

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

If transactions are being logged for the server involved in a group migration, you may need to move the TLOG to the backup machine, load it, and perform a warm start.

Migrating a Server Group When the Alternate Machine Is Accessible from the Primary Machine

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

1. Call `tmshutdown`(1) from the master machine with the `-R` and `-g` (`group_name`) options.
2. Run `tmadmin`(1) from the master machine.
3. Call the `migrategroup` (`migg`) command with *`group_name`* as the argument.
4. Migrate the transaction log, if necessary.
5. Migrate the application data, if necessary.

Migrating a Server Group When the Alternate Machine Is Not Accessible from the Primary Machine

To migrate a server group when the alternate machine is not accessible from the primary machine, complete the following steps.

1. Switch the master and backup machines, if necessary.
2. Run `tmadmin`(1) from the alternate machine.
3. Call the `pclean` (`pcl`) command with the primary machine as the argument.
4. Call the `migrategroup` (`migg`) command with *`group_name`* as the argument.
5. Call the `tmboot`(1) command to boot the server group.

Examples: Migrating a Server Group

Listing 18-3 and Listing 18-4 show how you can migrate a server group. In the first example, the alternate machine is accessible from the primary machine. In the second example, the alternate machine is not accessible from the primary machine.

Listing 18-3 When the Alternate Machine Is Accessible from the Primary Machine

```
$ tmshutdown -R -g GROUP1
Shutting down server processes...
Server ID = 1 Group ID = GROUP1 machine = SITE1: shutdown succeeded
1 process stopped.
$ tmadmin
> migg GROUP1
migg successfully completed
> q
```

Listing 18-4 When the Alternate Machine Is Not Accessible from the Primary Machine

```
$ tmadmin
> pclean SITE1
Cleaning the DBBL.
Pausing 10 seconds waiting for system to stabilize.
3 SITE1 servers removed from bulletin board
> migg GROUP1
migg successfully completed.
> boot -g GROUP1
Booting server processes ...
exec simpserv -A :
on SITE2 -> process id=22699 ... Started.
1 process started.
> q
```

Migrating Machines

Use the `tmadmin`(1) `migratemach` (`migm`) command to migrate all server groups from one machine to another when the primary machine must be shut down for maintenance or when the primary machine is no longer accessible.

The command takes one logical machine identifier as an argument. The `LMID` names the processor on which the server group(s) have been running. The alternate location must be the same for all server groups on the `LMID`. Servers on the `LMID` must specify `RESTART=Y` and the `MIGRATE` options must be specified in the `RESOURCES` section of the `UBBCONFIG` file. You must first shut down the server groups with the `tmshutdown`(1) `-R` option, and servers in the groups must be marked as restartable.

Migrating Machines When the Alternate Machine Is Accessible from the Primary Machine

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

1. Call `tmshutdown`(1) from the master machine with the `-R` and `-l` (`primary_machine`) options.
2. Run `tmadmin`(1) from the master machine.
3. Call the `migratemach` (`migm`) command with *`primary_machine`* as the argument.
4. Migrate the transaction log, if necessary.
5. Migrate the application data, if necessary.

Migrating Machines When the Alternate Machine Is Not Accessible from the Primary Machine

To migrate a machine when the alternate machine is not accessible from the primary machine, complete the following steps.

1. Switch the master and backup machines, if necessary.
2. Run `tmadmin`(1) from the alternate machine.
3. Call the `pclean` (`pcl`) command with *`primary_machine`* as the argument.
4. Call the `migratemach` (`migm`) command with *`primary_machine`* as the argument.
5. Call the `boot` (`b`) command to boot the server groups.

Examples: Migrating a Machine

Listing 18-5 and Listing 18-6 illustrate how you can migrate server groups. In the first example, the alternate machine is accessible from the primary machine. In the second example, the alternate machine is not accessible from the primary machine.

Listing 18-5 When the Alternate Machine Is Accessible from the Primary Machine

```
$ tmshutdown -R -l SITE1
Shutting down server processes...
Server ID = 1 Group ID = GROUP1 machine = SITE1: shutdown
succeeded 1 process stopped.
$ tmadmin
> migm SITE1
migm successfully completed
> q
```

Listing 18-6 When the Alternate Machine Is Not Accessible from the Primary Machine

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

Canceling a Migration

You can cancel a migration after a shutdown occurs, but before using the `migrate` command, by using the `-cancel` option with the `migrate` command.

You can cancel a migration in the following ways:

- By using the `tmadmin`(1) `migrategroup` (`migg`) `-cancel` command to cancel a server migration. Server entries are deleted from the Bulletin Board. You must reboot the servers once the migration procedure is canceled.
- By using the `tmadmin`(1) `migratemach` (`migm`) `-cancel` command to cancel a machine migration.

Example: A Migration Cancellation

Listing 18-7 illustrates how a server group and a machine can be migrated between their respective primary and alternate machines.

Listing 18-7 Canceling a Server Group Migration for Server Group GROUP1

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

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

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

Migrating Transaction Logs to a Backup Machine

To migrate transactions logs to a backup machine, complete the following steps.

1. Shut down the servers in all the groups that write to the log to stop additional writes to the log.
2. Dump the `TLOG` into an ASCII file by running the following command:

   `dumptlog [-z` *`config`*`] [-o` *`offset`*`] [-n` *`name`*`] [-g` *`groupname`*`]`

   Note: The `TLOG` is specified by the *`config`* and *`offset`* arguments. *`Offset`* defaults to `0` and *`name`* defaults to `TLOG`. If the `-g` option is chosen, only those records for which the `TMS` from *`groupname`* is the coordinator are dumped.

3. Copy `filename` to the backup machine.
4. Use `loadtlog -m machine ASCII_file` to read the name of the ASCII file into the existing `TLOG` for the specified machine.
5. Use `logstart machine` to force a warm start of the `TLOG`.
   (The information is read from the `TLOG` to create an entry in the transaction table in shared memory.)
6. Migrate the servers to the backup machine.

19 Dynamically Modifying Systems

The BEA WebLogic Enterprise and BEA Tuxedo systems allow you to make changes to your configuration without shutting it down. Without inconveniencing your users, you can suspend or resume interfaces or services, advertise or unadvertise services, and change interface or service parameters (such as `LOAD` and `PRIORITY`). If your configuration specifies interfaces or services as `AUTOTRAN`, it is also possible to change the timeout value associated with such transactions. Thus, you can adjust your system to reflect either current or expected conditions.

This topic includes the following sections:

- Dynamic Modification Methods
- Procedures for Dynamically Modifying Your System

Dynamic Modification Methods

You have a choice of two methods for making changes to your system while the system is running:

- The BEA Administration Console—a graphical user interface to the commands that perform administrative tasks, including dynamic system modification.
- The `tmadmin` command interpreter—a shell-level command with 50 subcommands for performing various administrative tasks, including dynamic system modification.

Because it is a graphical user interface, the BEA Administration Console is easier to use than the `tmadmin` command interpreter. If you prefer using a GUI, bring it up on your screen as soon as you are ready to begin an administrative task. The graphics and detailed procedures will guide you through any task you need to perform.

For instructions on using the `tmadmin` command interpreter, see Chapter 8, "Monitoring a Running System."

Instructions for dynamically modifying your system through `tmadmin` are provided in this chapter.

Procedures for Dynamically Modifying Your System

This section provides procedures for making the following types of changes, through `tmadmin`, while your system is running:

- Suspending and resuming services
- Advertising and unadvertising services
- Changing service parameters
- Changing the `AUTOTRAN` timeout value

Suspending and Resuming Services (BEA Tuxedo Servers)

This section provides instructions for suspending and resuming services and servers, and describes the results of these operations.

Note: The execution of the `suspend` and `resume` commands described in this section have minimal impact on the BEA Tuxedo system resources when compared with the resources gained by suspending a server.

Suspending Services

To suspend a server or a service, enter the `suspend` (or `susp`) command, as follows:

```
prompt> tmadmin
> susp
```

For example:

```
>susp -s toupper
```

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

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

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

Resuming Services

To resume a server or a service, enter the `resume` (or `res`) command, as follows:

```
prompt> tmadmin
> res
```

For example:

```
>res -s toupper
```

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

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

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

Advertising and Unadvertising Services (BEA Tuxedo Servers)

This section provides instructions for advertising and unadvertising services and servers, and describes the results of these operations.

Advertising Services

To advertise a service, enter the following command:

```
adv [{[-q queue_name] | [-g grpid] [-i srvid]}] service
```

Note: Although a service must be suspended before it may be unadvertised, you do not need to "unsuspend" a service before re-advertising it. If you simply advertise a service that has been suspended and unadvertised previously, the service will be unsuspended.

Unadvertising Services

To unadvertise a service, complete the following procedure:

1. Suspend the service.
2. Enter the following command:

   ```
   unadv [{[-q queue_name] | [-g grpid] [-i srvid]}] service
   ```

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

Changing Service Parameters (BEA Tuxedo Servers) or Interface Parameters (BEA WebLogic Enterprise Servers)

You can change the service and interface parameters for the following:

- A specific group ID/server ID combination
- A specific queue

The following table lists the names of the parameters for which you can change values dynamically, along with the commands for changing them.

To Change . . .	Enter the Following Command . . .
Load associated with the specified service or interface.	`chl [-m` *`machine`*`] {-q` *`qaddress`* `[-g` *`groupname`*`] [-i` *`srvid`*`] [-s` *`service`*`] | -g` *`groupname`* `-i` *`srvid`* `-s` *`service`* `| -I` *`interface`* `[-g` *`groupname`*`]} newload`
Dequeuing priority associated with the specified service or interface.	`chp [-m` *`machine`*`] {-q` *`qaddress`* `[-g` *`groupname`*`] [-i` *`srvid`*`] [-s` *`service`*`] | -g groupname -i srvid -s service | -I` *`interface`* `[-g` *`groupname`*`]}` *`newpri`*

You must specify a service name (after the `-s` option) for both commands.

Note: The `-s` option is listed as optional because the required value may be specified on the default subcommand line.

Changing the AUTOTRAN Timeout Value

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

```
chtt [-m machine] {-q qaddress [-g groupname] [-i srvid]
        [-s service] | -g groupname -i srvid -s service |
        -I interface [-g groupname]} newtlim
```

Note: Transaction timeouts begun by application clients using `tpbegin()` or `tx_set_transaction_timeout()` cannot be changed.

Suspending and Resuming Interfaces (BEA WebLogic Enterprise System)

This section provides instructions for suspending and resuming interfaces.

Note: The execution of the `suspend` and `resume` commands described in this section have minimal impact on the BEA WebLogic Enterprise system resources when compared with the resources gained by suspending a server.

Suspending an Interface

To suspend an interface, enter the `suspend` (or `susp`) command. For example:

```
tmadmin
>susp -i IDL:beasys.com/Simple:1.0
```

If an interface is suspended, a client will not be able to invoke a method on that interface until the interface is resumed.

Resuming an Interface

To resume an interface, enter the `resume` (or `res`) command. For example:

```
tmadmin
>res -i IDL:beasys.com/Simple:1.0
```

If a suspended interface is resumed, clients will be able to invoke methods on that interface.

CHAPTER

20 Dynamically Reconfiguring Applications

This topic includes the following topics:

- Introduction to Dynamic Reconfiguration
- Overview of the tmconfig Command Interpreter
- General Instructions for Running tmconfig
- Procedures
- Final Advice About Dynamic Reconfiguration

Introduction to Dynamic Reconfiguration

At times you will want to modify an application's configuration without having to shut it down. The BEA WebLogic Enterprise and BEA Tuxedo systems allow you to perform two types of dynamic reconfiguration of your application. You can do the following:

- Modify existing entries in your configuration file (`TUXCONFIG`)
- Add components by adding entries for them to your configuration file

Both types of change are implemented by editing `TUXCONFIG`. Because `TUXCONFIG` is a binary file, however, it cannot be edited through a simple text editor. For this reason, the BEA WebLogic Enterprise and BEA Tuxedo systems provide the following tools for configuration file editing:

- The BEA Administration Console is a graphical user interface (GUI) to the commands that perform administrative tasks, including dynamic system modification.
- The `tmconfig` command interpreter is a shell-level command with 50 subcommands for performing various administrative tasks, including dynamic system modification.

The BEA Administration Console is a graphical user interface to administrative tasks. You always have the choice between doing application administration tasks through this graphical interface or through a command-line interface. You can choose the working style most familiar and comfortable to you. When it comes to dynamic reconfiguration, however, we recommend using the BEA Administration Console. You will find the dynamic reconfiguration is easier when you use the BEA Administration Console instead of the `tmconfig` command interpreter.

The BEA Administration Console is not described in this document. Full descriptions of the GUI are available by accessing the Help directly from the GUI.

If you prefer to work on the command line, run the `tmconfig` command interpreter.

Note: We recommend that you keep a copy of the `tmconfig`(1) and `ubbconfig`(5) reference pages handy as you read this chapter. The input and output field names that correspond to `UBBCONFIG` parameters and reconfiguration restrictions are listed in `tmconfig`(1) and `TM_MIB`(5) in the *BEA Tuxedo Reference Manual*. These reference pages are the final authority on the semantics, range values, and validations of configuration parameters.

Overview of the tmconfig Command Interpreter

This section describes the following:

- What `tmconfig` does
- How `tmconfig` works

What tmconfig Does

The `tmconfig` command enables you to browse and modify the `TUXCONFIG` file and its associated entities, and to add new components (such as machines and servers) while your application is running.

When you modify your configuration file (`TUXCONFIG` on the `MASTER` machine), `tmconfig` performs the following tasks:

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

The `tmconfig` command runs as a BEA WebLogic Enterprise or a BEA Tuxedo system client.

Implications of Running as a Client

Keep in mind the following implications of the fact that `tmconfig` runs as a BEA WebLogic Enterprise or a BEA Tuxedo system client:

- `tmconfig` fails if it cannot allocate a `TPINIT` typed buffer.
- The *`username`* associated with the client is the login name of the user. (`tmconfig` fails if the user's login name cannot be determined.)
- For a secure application (that is, an application for which the `SECURITY` parameter has been set in the `UBBCONFIG` file), `tmconfig` prompts for the application password. If the application password is not provided, `tmconfig` fails.
- If `tmconfig` cannot register as a client, an error message containing *`tperrno`* is displayed and `tmconfig` exits. If this happens, check the user log to determine the cause. The most likely causes for this type of failure are:
  - The `TUXCONFIG` environment variable was not set correctly.

- The system was not booted on the machine on which `tmconfig` is being run.

- `tmconfig` ignores all unsolicited messages.
- The client name for the `tmconfig` process that is displayed in the output from `printclient` (a `tmadmin` command) will be `tpsysadm`.

How tmconfig Works

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

Listing 20-1 shows the menus and prompts that are displayed once you enter the `tmconfig` command, thus launching the session.

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

Listing 20-1 Menus and Prompts Displayed in a tmconfig Session

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

As shown here, you are asked to answer four questions:

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

This section discusses these four questions and defines possible answers to each.

Sections of the Configuration File

When you start a `tmconfig` session, the following menu of sections (of `TUXCONFIG`, the configuration file) is displayed.

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

Note: For details about these sections (including a list of configurable parameters for each section), see the `ubbconfig`(5) reference page in the *BEA Tuxedo Reference Manual*.

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

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

The default selection is the `RESOURCES` section, in which parameters that apply to your entire application are defined. To accept the default selection, simply press `ENTER` after the menu and colon (`:`) prompt.

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

tmconfig Operations

Next, a menu of operations that `tmconfig` can perform is displayed:

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

To select an operation, enter the appropriate number after the menu prompt. For example, to select the `UPDATE` section, enter 5, as follows:

```
6) CLEAR BUFFER 7) QUIT [1]: 5
```

Table 20-1 defines each operation.

Table 20-1 tmconfig Operations

Operation Number . . .	Called . . .	Performs the Following . . .
1	`FIRST`	Displays the first record from the specified section. No key fields are needed (they are ignored if they are in the input buffer). Using the `FIRST` operation can reduce the amount of typing that is needed. When adding a new entry to a section, instead of typing in all of the parameter names and values, use the `FIRST` operation to retrieve an existing entry for the `UBBCONFIG` section. Then, select the `ADD` operation and use the text editor to modify the parameter values.
2	`NEXT`	Displays the next record from the specified section, based on the key fields in the input buffer.
3	`RETRIEVE`	Displays the record (requested with the appropriate key field(s)) from the specified section.
4	`ADD`	Adds the indicated record in the specified section. Any fields not specified (unless required) take the default values specified in `ubbconfig`(5). (All default values and validations used by `tmloadcf`(1) are enforced.) The current value for all fields is returned in the output buffer. This operation can be done only by the BEA Tuxedo system administrator.
5	`UPDATE`	Updates the record specified in the input buffer in the selected section. Any fields not specified in the input buffer remain unchanged. (All default values and validations used by `tmloadcf`(1) are enforced.) The current values for all fields are returned in the input buffer. This operation can be done only by the BEA Tuxedo system administrator.
6	`CLEAR BUFFER`	Clears the input buffer (all fields are deleted). After this operation, `tmconfig` immediately prompts for the section again.
7	`QUIT`	Exits `tmconfig` gracefully (that is, the client is terminated). A value of `q` for any prompt allows you to exit `tmconfig`.

Output from tmconfig Operations

After `tmconfig` has executed an operation, the results (a return value and the contents of the output buffer) are displayed on the screen.

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

  ```
  Return value TAOK
  ```

  The following is the message in the `TA_STATUS` field:

  ```
  Operation completed successfully.
  ```

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

  ```
  Return value TAUPDATED
  ```

  The following is the message in the `TA_STATUS` field:

  ```
  Update completed successfully.
  ```

- If the operation failed, an error message is displayed:
  - If there is a problem with permissions or a BEA Tuxedo system communications error (rather than with the configuration parameters), one of the following return values is displayed: `TAEPERM`, `TAEOS`, `TAESYSTEM`, or `TAETIME`.
  - If there is a problem with a configuration parameter of the running application, the name of that parameter is displayed as the value of the `TA_BADFLDNAME` file, and the problem is indicated in the value of the `TA_STATUS` field in the output buffer. If this type of problem occurs, one of the following return values is displayed: `TAERANGE`, `TAEINCONSIS`, `TAECONFIG`, `TAEDUPLICATE`, `TAENOTFOUND`, `TAEREQUIRED`, `TAESIZE`, `TAEUPDATE`, or `TAENOSPACE`.

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

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

`TAESYSTEM`
: A BEA Tuxedo system error has occurred. The exact nature of the error is recorded in `userlog`(3c).

`TAEOS`
: An operating system error has occurred. The exact nature of the error is written to `userlog`(3c).

`TAETIME`
: A blocking timeout has occurred. The input buffer is not updated so no information is returned for retrieval operations. The status of update operations can be checked by doing a retrieval on the record that was being updated.

`TAERANGE`
: A field value is either out of range or invalid.

`TAEINCONSIS`
: A field value (or set of field values) is inconsistently specified. For example, an existing `RQADDR` value may be specified for a different `SRVGRP` and `SERVERNAME`.

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

`TAEDUPLICATE`
: The operation attempted to add a duplicate record.

`TAENOTFOUND`
: The record specified for the operation was not found.

`TAEREQUIRED`
: A field value is required but is not present.

`TAESIZE`
: A field value for a string field is too long.

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

`TAENOSPACE`
: The operation attempted to do an update but there was not enough space in the `TUXCONFIG` file and/or the Bulletin Board.

General Instructions for Running tmconfig

This section explains how to do the following:

- Set up your environment properly before starting a `tmconfig` session
- Walk through a `tmconfig` session

Preparing to Run tmconfig

Before you can start a `tmconfig` session, you must have the required permissions and set the required environment variables. For your convenience, you may also want to select a text editor other than the default. Complete the following procedure to ensure you have set up your working environment properly before running `tmconfig`.

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

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

Running tmconfig: A High-level Walk-through

This section provides a walk-through of a generic `tmconfig` session in which you modify entries in your configuration file.

1. Enter `tmconfig` after a shell prompt.

   ```
   $ tmconfig
   ```

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

   A menu of sections in the `TMCONFIG` file is displayed:

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

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

   A menu of possible operations is displayed:

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

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

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

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

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

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

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

   If you select yes, the specified editor is invoked and you can start adding or changing fields. The format of each field is:

   field_name<*tabs*>*field_value*

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

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

   Note: For details about valid input, see the following section ("Input Buffer Considerations"). For descriptions of the field names associated with each section of UBBCONFIG, see the TM_MIB(5) reference page in the *BEA Tuxedo Reference Manual* available on the online documentation CD.

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

```
Enter editor to correct?
```

5. Select n or y.

   If you decide not to correct the problem (by selecting n), the input buffer contains no fields. Otherwise, the editor is executed again.

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

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

6. Select n or y. The default choice (shown at the end of the prompt) is yes ([y]).

   - If you select no, the menu of sections is displayed again. (Return to step 2.)
   - If you select yes, tmconfig executes the requested operation and displays the following confirmation message:

     ```
     Return value TAOK
     ```

   The results of the operation are displayed on the screen.

   You have completed an operation on one section of TMCONFIG; you may now start another operation on the same section or on another section. To allow you to start a new operation, `tmconfig` displays, again, the menu of `TMCONFIG` sections (as shown in step 1).

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

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

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

Input Buffer Considerations

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

- If the value of a field you are typing extends beyond one line, you may continue it on the next line if you insert one or more tabs at the beginning of the second line. (The tab characters are dropped when your input is read into `tmconfig`.)
- An empty line consisting of a single newline character is ignored.
- If more than one line is provided for a particular field name, the first occurrence is used and other occurrences are ignored.
- To enter an unprintable character as part of the value of a field, or to start a field value with a tab, use a backslash followed by the two-character hexadecimal representation of the desired character (see the `ASCII`(5) reference page in a UNIX system reference manual). For example:
  - To insert a blank space, type \20.

- To insert a backslash, type \\.

Procedures

This section provides procedures for dynamically reconfiguring your application by making the following changes:

- Adding a new machine
- Adding a server to a running application
- Activating a newly configured server
- Adding a new group
- Changing the factory-based routing for an interface
- Changing the data-dependent routing (DDR) for the application
- Changing application-wide parameters
- Changing the application password

Adding a New Machine

To add a new machine, complete the following steps:

1. Start a `tmconfig` session.
2. Specify the `MACHINE` section of the configuration file (choice #2 in the list).
3. Request the `FIRST` operation; that is, request a display of the first entry in the `MACHINE` section. (This operation is the default choice; press `ENTER` to select it.)
4. Request the `ADD` operation (choice #4 in the list).
5. Specify new values for four key fields:
    - `TLOG`
    - `TA_LMID`

- TA_TYPE
- TA_PMID

Listing 20-2 illustrates a `tmconfig` session in which a machine is being added.

Listing 20-2 Adding a Machine

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

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

Enter editor to add/modify fields [n]?
Perform operation [y]?
Return value TAOK
Buffer contents:
TA_OPERATION            4
TA_SECTION              1
TA_OCCURS               1
TA_PERM                 432
TA_MAXACCESSERS         40
TA_MAXGTT               20
TA_MAXCONV              10
TA_MAXWSCLIENTS         0
TA_TLOGSIZE             100
TA_UID                  4196
TA_GID                  601
TA_TLOGOFFSET           0
TA_TUXOFFSET            0
TA_STATUS               LIBTUX_CAT:1137: Operation completed successfully
TA_PMID                 mchn1
TA_LMID                 SITE1
TA_TUXCONFIG            /home/apps/bank/tuxconfig
TA_TUXDIR               /home/tuxroot
TA_STATE                ACTIVE
TA_APPDIR               /home/apps/bank
TA_TYPE                 3B2
TA_TLOGDEVICE           /home/apps/bank/TLOG
TA_TLOGNAME             TLOG
TA_ULOGPFX              /home/apps/bank/ULOG
TA_ENVFILE              /home/apps/bank/ENVFILE

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

Enter editor to add/modify fields [n]? y
491
g/home/s//usr/p
TA_TUXCONFIG            /usr/apps/bank/tuxconfig
TA_TUXDIR               /usr/tuxroot
TA_APPDIR               /usr/apps/bank
TA_TLOGDEVICE           /usr/apps/bank/TLOG
TA_ULOGPFX              /usr/apps/bank/ULOG
TA_ENVFILE              /usr/apps/bank/ENVFILE
g/TLOG/d
/SITE1/s//SITE3/p
TA_LMID SITE3
/3B2/s//SPARC/p
TA_TYPE SPARC
/mchn1/s//mchn2/p
TA_PMID mchn2
w
412
q

Perform operation [y]?
Return value TAUPDATED
Buffer contents:
TA_OPERATION            2
TA_SECTION              1
TA_OCCURS               1
TA_PERM                 432
TA_MAXACCESSERS         40
TA_MAXGTT               20
TA_MAXCONV              10
TA_MAXWSCLIENTS         0
TA_TLOGSIZE             100
TA_UID                  4196
TA_GID                  601
TA_TLOGOFFSET           0
TA_TUXOFFSET            0
TA_STATUS               LIBTUX_CAT:1136: Update completed successfully
TA_PMID                 mchn2
TA_LMID                 SITE3
TA_TUXCONFIG            /usr/apps/bank/tuxconfig
TA_TUXDIR               /usr/tuxroot
TA_STATE                NEW
TA_APPDIR               /usr/apps/bank
TA_TYPE                 SPARC
TA_TLOGDEVICE
```

```
TA_TLOGNAME                  TLOG
TA_ULOGPFX                   /usr/apps/bank/ULOG
TA_ENVFILE                     /usr/apps/bank/ENVFILE
```

Adding a Server

To add a server, complete the following steps:

1. Start a `tmconfig` session.
2. Specify the `SERVERS` section of the configuration file (choice #4 in the list).
3. Request the `CLEAR BUFFER` operation (choice #6 in the list).
4. Request the `ADD` operation (choice #4 in the list).
5. Enter the text editor.
6. Specify new values for three key fields:
   - `TA_SERVERNAME`
   - `TA_SRVGRP`
   - `TA_SRVID`

Listing 20-3 illustrates a `tmconfig` session in which a server is added.

Listing 20-3 Adding a Server

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

```
TA_SERVERNAME            XFER
TA_SRVGRP                BANKB1
TA_SRVID                 5
.
w
28
q
Perform operation [y]?
Return value TAOK
Buffer contents:
TA_OPERATION             3
TA_SECTION               3
TA_OCCURS                1
TA_SRVID                 5
TA_SEQUENCE              0
TA_MIN                   1
TA_MAX                   1
TA_RQPERM                432
TA_RPPERM                432
TA_MAXGEN                5
TA_GRACE                 86400
TA_STATUS                LIBTUX_CAT:1137: Operation completed successfully
TA_SYSTEM_ACCESS         FASTPATH
TA_ENVFILE
TA_SRVGRP                BANKB1
TA_SERVERNAME            XFER
TA_CLOPT                 -A
TA_CONV                  N
TA_RQADDR
TA_REPLYQ                Y
TA_RCMD
TA_RESTART               Y
```

Activating a Newly Configured Server

Complete the following steps to add a newly configured server.

1. Start a `tmconfig` session.
2. Select the `MACHINES` section.
3. Using the `FIRST` and `NEXT` operations, select the entry for which you want to change the state from `NEW` to `ACTIVE`.

4. Select the `UPDATE` operation (choice #5 in the list).
5. Enter `y` (for yes) when prompted to say whether you want to start editing.
6. Change the value of the `TA_STATE` field from `NEW` to `ACTIVE`.
7. `tmconfig` displays the revised entry for the specified machine so you can review your change (and, if necessary, edit it).
8. If the revised entry is acceptable, select `QUIT` (choice #6 in the list) to end the `tmconfig` session.

Adding a New Group

To add a group, complete the following steps:

1. Start a `tmconfig` session.
2. Select the `GROUPS` section of the configuration file (choice #3 in the list).
3. Request the `CLEAR BUFFER` operation (choice #6 in the list).
4. Request the `ADD` operation (choice #4 in the list).
5. Enter `y` (for yes) when prompted to say whether you want to start editing.
6. Specify new values for three key fields:
   - `TA_LMID`
   - `TA_SRVGRP`
   - `TA_GRPNO`

Listing 20-4 illustrates a `tmconfig` session in which a group is added.

Listing 20-4 Adding a Group

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

```
Section: 1) RESOURCES, 2) MACHINES, 3) GROUPS 4) SERVERS
 5)SERVICES  6) NETWORK 7) ROUTING q) QUIT 9) WSL
 10) NETGROUPS 11) NETMAPS 12) INTERFACES [3]:
Operation: 1) FIRST 2) NEXT 3) RETRIEVE 4) ADD 5) UPDATE
   6) CLEAR BUFFER 7) QUIT [6]: 4
Enter editor to add/modify fields [n]? y
1
c
TA_LMID                  SITE3
TA_SRVGRP                GROUP3
TA_GRPNO                 3
.
w
42
q
Perform operation [y]?
Return value TAUPDATED
Buffer contents:
TA_OPERATION             2
TA_SECTION               2
TA_OCCURS                1
TA_GRPNO                 3
TA_TMSCOUNT              0
TA_STATUS                LIBTUX_CAT:1136: Update completed successfully
TA_LMID                  SITE3
TA_SRVGRP                GROUP3
TA_TMSNAME
TA_OPENINFO
TA_CLOSEINFO
```

Changing the Factory-based Routing (FBR) for an Interface

To change the factory-based routing for an interface, complete the following steps:

1. Start a `tmconfig` session.
2. Select the `ROUTING` section of the configuration file (choice #7 on the menu of configuration file sections).
3. Using the `FIRST` and `NEXT` operations, select the entry for which you want to change the FBR.

4. Select the `UPDATE` operation.
5. Enter `y` (for yes) when prompted to say whether you want to start editing.

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

6. Change the relevant fields to values such as those shown in the middle column in the following table:

Field	Sample Value	Meaning
`TA_ROUTINGNAME`	`STU_ID`	Name of the routing section.
`TA_FIELD`	`student_id`	The value of this field is subject to the criterion (specified in the `TA_RANGES` field); that is, the value of this field determines the routing result.
`TA_RANGES`	`100001-100050:ORA_GRP1,` `100051-*:ORA_GRP2`	The routing criterion being used.

The value of the `TA_RANGES` field is the routing criterion. For example, assume that our modest student enrollment before the update allowed for a routing criterion of student IDs between 100001-100005 to `ORA_GRP1`, and 100006-100010 to `ORA_GRP2`. In the change shown in the preceding table, if the value of `student_id` is between 100001 and 100050 (inclusive), requests are sent to the servers in `ORA_GRP1`. Other requests are sent to `ORA_GRP2`.

Note: Dynamic changes that you make to a routing parameter with `tmconfig` take effect on subsequent invocations and do not affect outstanding invocations.

You can also dynamically change the `TA_FACTORYROUTING` assignment in the `INTERFACES` section. For example:

1. Start a `tmconfig` session.
2. Select the `INTERFACES` section of the configuration file (choice #12 on the menu of configuration file sections).
3. Using the `FIRST` and `NEXT` operations, select the interface entry for which you want to change the FBR. For example, if you defined a new factory-based routing criterion named `CAMPUS` in the `ROUTING` section, you could reassign a Registrar interface to this criterion.

4. Select the UPDATE operation.
5. Enter y (for yes) when prompted to say whether you want to start editing.

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

Changing the Data-dependent Routing (DDR) for the Application

To change the data-dependent routing for an application., complete the following steps:

1. Start a tmconfig session.
2. Select the ROUTING section of the configuration file (choice #7 in the list).
3. Using the FIRST and NEXT operations, select the entry for which you want to change the DDR.
4. Select the UPDATE operation.
5. Enter y (for yes) when prompted to say whether you want to start editing.

   ```
   Do you want to edit(n)? y
   ```
6. Change the relevant fields to values such as those shown in the middle column of the following table.

Field	Sample Value	Meaning
TA_ROUTINGNAME	account_routing	Name of the routing section.
TA_BUFTYPE	FML	Buffer type.
TA_FIELD	account_ID	The value of this field is subject to the criterion (specified in the TA_RANGES field); that is, the value of this field determines the routing result.
TA_RANGES	1-10:group1,*:*	The routing criterion being used.

The value of the `TA_RANGES` field is the routing criterion. If the value of `account_ID` is between 1 and 10 (inclusive), requests are sent to the servers in group 1. Otherwise, requests are sent to any other server in the configuration.

Note: For details, see the `tmconfig`(1) reference page in the *BEA Tuxedo Reference Manual*.

Changing Application-wide Parameters

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

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

1. Start a `tmconfig` session.
2. Select the `RESOURCES` section of the configuration file. (The `RESOURCES` section, choice #1 on the menu of configuration file sections, is the default selection.)
3. Using the `FIRST` and `NEXT` operations, select the entry that you want to display. (Because the first entry is the default selection, in this case you can simply accept the default.)
4. Select the `FIRST` operation (the default selection).
5. Respond no (by accepting the default) when asked whether you want to edit.

   ```
   Do you want to edit(n)?
   ```
6. Respond yes (by accepting the default) when asked whether you want the specified operation (`FIRST`) to be performed.

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

Listing 20-5 illustrates a `tmconfig` session in which the first entry in the `RESOURCES` section is displayed.

Listing 20-5   Displaying the First Entry in the RESOURCES Section

```
Section: 1) RESOURCES, 2) MACHINES, 3) GROUPS 4) SERVERS
 5)SERVICES  6) NETWORK 7) ROUTING q) QUIT 9) WSL
 10) NETGROUPS 11) NETMAPS 12) INTERFACES [1]:
Operation: 1) FIRST 2) NEXT 3) RETRIEVE 4) ADD 5) UPDATE
 6) CLEAR BUFFER 7) QUIT [1]: 1
Enter editor to add/modify fields [n]?
Perform operation [y]?
Return value TAOK
Buffer contents:
TA_OPERATION       1
TA_SECTION         0
TA_STATUS          Operation completed successfully
TA_OCCURS          1
TA_PERM            432
TA_BBLQUERY        30
TA_BLOCKTIME       6
TA_DBBLWAIT        2
TA_GID             10
TA_IPCKEY          80997
TA_LICMAXUSERS     1000000
TA_MAXACCESSERS    100
TA_MAXBUFSTYPE     32
TA_MAXBUFTYPE      16
TA_MAXCONV         10
TA_MAXDRT          0
TA_MAXGROUPS       100
TA_MAXGTT          25
TA_MAXMACHINES     256
TA_MAXQUEUES       36
TA_MAXRFT          0
TA_MAXRTDATA       8
TA_MAXSERVERS      36
TA_MAXSERVICES     100
TA_MIBMASK         0
TA_SANITYSCAN      12
TA_SCANUNIT        10
TA_UID             5469
TA_MAXACLGROUPS    16384
TA_MAXNETGROUPS    8
TA_MAXINTERFACES   150
TA_MAXOBJECTS      1000
TA_STATE           ACTIVE
TA_AUTHSVC
TA_CMTRET          COMPLETE
TA_DOMAINID
TA_LDBAL           Y
```

```
TA_LICEXPIRE       1998-09-15
TA_LICSERIAL       1234567890
TA_MASTER          SITE1
TA_MODEL           SHM
TA_NOTIFY          DIPIN
TA_OPTIONS
TA_SECURITY        NONE
TA_SYSTEM_ACCESS   FASTPATH
TA_USIGNAL         SIGUSR2
TA_PREFERENCES
TA_COMPONENTS      TRANSACTIONS,QUEUE,TDOMAINS,TxRPC,
EVENTS,WEBGUI,WSCOMPRESSION,TDOMCOMPRESSION
```

Changing an Application Password

To change an application password, complete the following steps:

1. Start a `tmconfig` session.
2. Select the `RESOURCES` section (#1, the default choice on the menu of sections).
3. Clear the buffer.
4. Enter (in the buffer):

   ```
   TA_PASSWORD   new_password
   ```

   ```
   wq!
   ```

Listing 20-6 illustrates a `tmconfig` session in which an application password is changed.

Listing 20-6 Changing an Application Password

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

```
   6) CLEAR BUFFER 7) QUIT [6]: 5
Enter editor to add/modify fields [n]? y
1
c
TA_PASSWORD          neptune
.
w
49
q
Perform operation [y]?
Return value TAUPDATED
Buffer contents:
TA_OPERATION       1
TA_SECTION         0
TA_STATUS          Operation completed successfully
TA_OCCURS          1
TA_PERM            432
TA_BBLQUERY        30
TA_BLOCKTIME       6
TA_DBBLWAIT        2
TA_GID             10
TA_IPCKEY          80997
TA_LICMAXUSERS     1000000
TA_MAXACCESSERS    100
TA_MAXBUFSTYPE     32
TA_MAXBUFTYPE      16
TA_MAXCONV         10
TA_MAXDRT          0
TA_MAXGROUPS       100
TA_MAXGTT          25
TA_MAXMACHINES     256
TA_MAXQUEUES       36
TA_MAXRFT          0
TA_MAXRTDATA       8
TA_MAXSERVERS      36
TA_MAXSERVICES     100
TA_MIBMASK         0
TA_SANITYSCAN      12
TA_SCANUNIT        10
TA_UID             5469
TA_MAXACLGROUPS    16384
TA_MAXNETGROUPS    8
TA_MAXINTERFACES   150
TA_MAXOBJECTS      1000
TA_PASSWORD        neptune
TA_STATE           ACTIVE
TA_AUTHSVC
TA_CMTRET          COMPLETE
TA_DOMAINID
```

```
TA_LDBAL            Y
TA_LICEXPIRE        1998-09-15
TA_LICSERIAL        1234567890
TA_MASTER           SITE1
TA_MODEL            SHM
TA_NOTIFY           DIPIN
TA_OPTIONS
TA_SECURITY         NONE
TA_SYSTEM_ACCESS    FASTPATH
TA_USIGNAL          SIGUSR2
TA_PREFERENCES
TA_COMPONENTS       TRANSACTIONS,QUEUE,TDOMAINS,TxRPC,EVENTS,WEBGUI,
                      WSCOMPRESSION,TDOMCOMPRESSION
```

Final Advice About Dynamic Reconfiguration

Keep in mind the following restrictions. Be careful about setting parameters that cannot be changed easily.

- Associated with each section is a set of key fields that are used to identify the record upon which to operate. (For details see the `tmconfig`(1) reference page in the *BEA Tuxedo Reference Manual*.) Key field values cannot be changed while an application is running. Normally, it is sufficient to add a new entry (with a new key field value) and use it instead of the old entry. In this case, the old entry in the configuration is not booted by the administrator; the new entry is used, instead.
- Generally speaking, you cannot update a parameter while the configuration component associated with it is booted. (For example, you cannot change an entry in the `MACHINES` or `NETWORK` section while the machine associated with that entry is booted.) Specifically:
  - If any server in a group is booted, you cannot change the entry for that group.
  - If a server is booted, you cannot change its name, type (conversational or not), or parameters related to its message queue. (You can change other

server parameters at any time but your changes will not take effect until the next time the server is booted.)

- You can change a `SERVICES` entry at any time but your changes will not take effect until the next time the service is advertised.
- Updates to the `RESOURCES` section are restricted by the following conditions. The `UID`, `GID`, `PERM`, `MAXACCESSERS`, `MAXGTT`, and `MAXCONV` parameters cannot be updated in the `RESOURCES` section but can be updated on a per-machine basis. The `IPCKEY`, `MASTER`, `MODEL`, `OPTIONS`, `USIGNAL`, `MAXSERVERS`, `MAXSERVICES`, `MAXBUFTYPE`, and `MAXBUFSTYPE` parameters cannot be changed.

Note: Before shutting down the `MASTER` machine, make sure to migrate it to the `acting backup` machine.

- Be sure to keep track of the section of the configuration file in which you are working; `tmconfig` does not warn you if you try to perform an operation that is wrong for the section currently available in the buffer. For example, if you try to update the `ENVFILE` parameter (in the `MACHINES` section) while you are working in the `RESOURCES` section, the operation will appear to succeed (that is, `tmconfig` will return `TAOK`), but the change will not appear in your unloaded `UBBCONFIG` file. The only way you can be sure that an update has been done is by seeing the `TAUPDATED` status message displayed.
- With regard to interoperability, updates and additions are not allowed to any site in an application if a Release 4.1 (R4.1) site is booted. You must shut down the R4.1 site before updates can be done. When the updates are complete, you can reboot the R4.1 site; the updated `TUXCONFIG` will be propagated to the R4.1 node automatically.

In a multimachine configuration, always do the following:

- Specify a backup for the `MASTER` machine, along with the `MIGRATE` option (even if application server migration is not anticipated).
- Set `MAXSERVERS`, `MAXSERVICES`, and other “`MAX`” parameters high enough to allow for sufficient growth. If your application is, initially, a single-machine configuration but is expected to grow to a multimachine configuration, use the MP model, specifying the `LAN` option and a network entry for the initial machine.

- Set the parameters in the `MACHINES` section carefully since updating them requires shutting down the machine (and switching the `MASTER` to the backup in the case of the `MASTER` machine).

CHAPTER

21 Event Broker/Monitor (BEA Tuxedo Systems)

The BEA Tuxedo Event Broker/Monitor is a tool that enhances the tracking of events in a running application.

This topic includes the following topics:

- Events
- Setting Up Event Detection
- Subscribing to Events
- Application-specific Event Broker/Monitors
- How an Event Broker/Monitor Might Be Deployed
- How the Event Broker/Monitor Works

Note: This chapter is specific to the BEA Tuxedo system. However, BEA WebLogic Enterprise administrators should know that each BEA WebLogic Enterprise application relies on the BEA Tuxedo System Event Broker. This event broker must be started before any servers providing the NameManager service in a BEA WebLogic Enterprise application's `UBBCONFIG` file are started. For details, see the section "Required Order in Which to Boot Servers (BEA WebLogic Enterprise Servers)" on page 3-49 in Chapter 3, "Creating a Configuration File."

The BEA Tuxedo Event Broker/Monitor extends the usefulness of the `USERLOG` (in which the BEA Tuxedo system records system events) by providing the following:

- A system-wide summary of events

- A tool that lets you set up various types of automatic notification when certain events occur

The BEA Tuxedo Event Broker/Monitor is built on the AdminAPI, the administrative programming interface to the BEA Tuxedo system. It is an example of administration through programming.

Note: This chapter demonstrates how you can use the BEA Tuxedo AdminAPI to enhance your application. For an actual example that you can run as a demo and copy from, see the `bankapp` application (distributed with the BEA Tuxedo system) and the *BEA Tuxedo Application Development Guide*.

Events

An event is a change in a component of a running application. This change may be harmless or it may cause a problem that requires work by the operator or administrator (and, in some cases, particular software) to be resolved.

Event Classifications

The BEA Tuxedo Event Monitor keeps track of events in a running application and classifies them on the basis of severity. The Event Monitor uses the same three severity classifications used by the BEA Tuxedo system to sort system messages sent to the `USERLOG`: information (`INFO`), warnings (`WARN`), and errors (`ERROR`).

- An `INFO` event is one of the following:
  - A state change of a process
  - The detection of a configuration change
- A `WARN` event is a configuration change that threatens the performance of the application.
- An `ERROR` event is an abnormal occurrence, such as:
  - A server dying

- A network connection being dropped

List of Events

Events affecting objects in the classes defined in `TM_MIB`(5) are tracked. The list is published in the `EVENTS`(5) reference page in the *BEA Tuxedo Reference Manual* on the online documenatation CD.

The designers of an Event Broker/Monitor need to decide which events to track. Users of the system need to know the list of events being tracked.

Setting Up Event Detection

You can set the BEA Tuxedo system event detection logic to do two things:

- Post messages to a UNIX error message log (`syslogd`)
- Post events to the BEA Tuxedo event server

To activate event detection logic, set and export `TMSYSLOGD_FACILITY` to a numeric value from 0 to 7.

For details, see `syslogd`(3c) in a UNIX system reference manual.

Subscribing to Events

Clients subscribe to events by calls to `tpsubscribe`(3c). A call to `tpsubscribe` has a required argument, *`eventexpr`*, that points to a wildcard string. This string, in turn, identifies the events about which the user wants to know. The wildcard string makes use of the syntax described in `recomp`(3c) to apply the subscription to more than one type of event. The wildcard string is used to match the message distributed when the event is detected.

In the BEA Tuxedo System Monitor, the message includes the severity level, so a user can subscribe accordingly. For example:

- A user who wants to be notified of all events related to BEA Tuxedo networking sets the value of `eventexpr` to the following:

  ```
  \.SysNetwork.*
  ```

- A user who wants to subscribe to all events with a severity level of `ERROR` sets the value of `eventexpr` to the following:

  ```
  \.*(ERR|err)\.*
  ```

When a client leaves an application (by calling `tpterm`) all of its subscriptions are "canceled." If the client later rejoins the application and wants those subscriptions, it must subscribe again. A well-behaved client unsubscribes before calling `tpterm`. A client that accepts notification via unsolicited messages should issue a `tpunsubscribe`(3c) call before leaving the application.

Another argument of the `tpsubscribe` call (in addition to `eventexpr`) is a pointer to a structure of type `TPEVCTL` (defined in `atmi.h`). Through the use of the `TPEVCTL` structure (or non-use, if the argument is NULL), the user can select the notification method to be used for sending information about subscribed events. If the argument is NULL, the Event Broker sends an unsolicited message to the subscriber. The subscriber can alternatively elect to have the notification sent to a service or to a queue in stable storage. If a client wants to enter such a subscription, it must invoke a service routine to subscribe on its behalf.

As a BEA Tuxedo system administrator, you can enter subscription requests on behalf of a client or server process through calls to the `EVENT_MIB`(5). You may also use two notification methods that are specified in entries in the `EVENT_MIB` (besides the three available in `tpsubscribe`):

- A command can be invoked via the UNIX `system`(2) command.
- A message can be sent to the `userlog`.

Application-specific Event Broker/Monitors

By “application-specific Event Broker/Monitor” we mean a monitor customized to recognize events generated by application code. For example, a stock brokerage system could be programmed to post an event when a stock trades at or above a certain price. A banking application might be programmed to post an event when a withdrawal or deposit above a specified amount is detected.

The function of an application-specific Event Broker/Monitor is similar to that of the BEA Tuxedo System Event Broker/Monitor: when an event is posted, subscribers are notified (or an action specified by the subscriber is initiated). This section describes the same three areas that were described above, pointing out how the customized monitor resembles and differs from the BEA Tuxedo system monitor.

Events

The real distinction between a BEA Tuxedo System Event Broker/Monitor and an Event Broker/Monitor for a specific application is the way events are defined. System events are defined in advance by the BEA Tuxedo system code. For an application, designers must select application events to monitor. Application programs must be written to a) detect when an event of interest has occurred, and b) post the event to the Event Monitor via `tppost`.

Event List

There is no difference between the Event Lists generated and used on an application-specific Event Broker/Monitor and a BEA Tuxedo System Event Broker/Monitor. The BEA Tuxedo System Event Broker/Monitor makes a list of monitored events available to interested users. (For details, see the `EVENTS`(5) reference page in the *BEA Tuxedo Reference Manual*.) In the same way, when an application-specific Event Broker/Monitor is being used, interested users should have access to a list of monitored events. The names of system events begin with a dot (`.`); application-specific event names may not begin with a dot (`.`).

Subscriptions

The process of subscribing to an event in an application-specific Event Monitor is the same as that of subscribing with the BEA Tuxedo system Event Monitor. Subscriptions are made by calls to `tpsubscribe` using the published list of events, so the application can identify the events to which you are subscribing.

Note: For the BEA Tuxedo System Event Monitor, `EVENTS`(5) lists the notification message generated by an event, as well as the event name. The event name is used as an argument when `tppost` is called. Subscribers, on the other hand, can take advantage of the wildcard capability of regular expressions to make a single call to `tpsubscribe` to cover a whole category of events. We strongly recommend using the same format for the published event list for an application-specific Event Monitor/Broker.

How an Event Broker/Monitor Might Be Deployed

The client interfaces with the Event Broker/Monitor through either of two servers provided by the BEA Tuxedo system:

- `TMSYSEVT`(5)
- `TMUSREVT`(5)

These servers introduce the concept of a principal server and zero or more secondary servers. Both types (principal and secondary) process events and trigger notification actions.

To install the BEA Tuxedo system Event Broker/Monitor, configure:

- The principal server on the `MASTER` site
- Whatever secondary servers your installation might need on other machines on your network

With an application-specific Event Broker/Monitor, the primary server may be on any machine other than the `MASTER`; secondary servers may be located around your network.

The reason for locating secondary servers on other nodes of your network is to reduce the amount of network traffic caused by posting events and by distributing event notifications to subscribers. The secondary server periodically polls the primary server to get the latest version of the subscription list, which stores filtering and notification rules.

You can configure the polling interval as needed. There may be a perception that event messages are lost during this period between the time at which subscriptions are initially added and the time at which all secondary servers are updated. If the application cannot “lose” messages, the programs must wait, at least until the end of the polling period, before `tppost` is called for the new event.

How the Event Broker/Monitor Works

The BEA Tuxedo Event Broker/Monitor is built with the following AdminAPI components:

- ATMI Extensions—the Event Monitor uses three function calls in the ATMI library:
  - `tppost`
  - `tpsubscribe`
  - `tpunsubscribe`

  These three functions appear in both the C library and the COBOL library. (For details, see Sections (3c) and (3cbl) in the *BEA Tuxedo Reference Manual*.
- MIB component—the `EVENT_MIB` management information base is the control file in which you can store subscription information and filtering rules. In your own application, you cannot define new events for the BEA Tuxedo system Event Broker/Monitor, but you can customize the Event Broker/Monitor to do the following:
  - Track events
  - Distribute notifications of special interest to the application

22 Troubleshooting Applications

Other chapters of this document discuss many diagnostic tools provided by your BEA WebLogic Enterprise or BEA Tuxedo system: commands and log files that help you monitor a running system, identify potential problems while there is still time to prevent them, and detect error conditions once they have occurred. This chapter provides additional information to help you identify and recover from various system errors.

This topic includes the following sections:

- Distinguishing Between Types of Failures
- Broadcasting Unsolicited Messages (BEA Tuxedo System)
- Performing System File Maintenance
- Repairing Partitioned Networks
- Restoring Failed Machines
- Replacing System Components (BEA Tuxedo System)
- Replacing Application Components
- Cleaning Up and Restarting Servers Manually
- Checking the Order in Which Servers Are Booted (BEA WebLogic Enterprise Servers)
- Checking Hostname Format and Capitalization (BEA WebLogic Enterprise Servers)

- Some Clients Fail to Boot (BEA WebLogic Enterprise Servers)
- Checking the Order in Which Servers Are Booted (BEA WebLogic Enterprise Servers)
- Recovering from Failures When Transactions Are Used

Distinguishing Between Types of Failures

The first step in troubleshooting is to determine the area in which the problem has occurred. In most applications, you must consider six possible sources of trouble:

- Application
- The BEA WebLogic Enterprise or BEA Tuxedo system
- Database management software
- Network
- Operating system
- Hardware

To resolve the trouble in most of these areas, you must work with the appropriate administrator. If, for example, you determine that the trouble is being caused by a networking problem, you must work with the network administrator.

Determining the Cause of an Application Failure

To detect the source of an application failure, complete the following steps:

1. Check any BEA WebLogic Enterprise or BEA Tuxedo system warnings and error messages in the user log (`ULOG`).
2. Select the messages you think are most likely to reflect the current problem. Note the catalog name and the message number of each of those messages and look them up in the *BEA WebLogic Enterprise System Messages* or *BEA Tuxedo System Message Manual*. The document entry provides:

   - Details about the error condition flagged by the message
   - Recommendations for actions you can take to recover

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

Determining the Cause of a BEA WebLogic Enterprise or BEA Tuxedo System Failure

To detect the source of a system failure, complete the following steps:

1. Check any BEA WebLogic Enterprise or BEA Tuxedo system warnings and error messages in the user log (`ULOG`):
   - `TPEOS` messages indicate errors in the operating system.
   - `TPESYSTEM` messages indicate errors in the BEA WebLogic Enterprise or BEA Tuxedo system.

2. Select the messages you think are most likely to reflect the current problem. Note the catalog name and message number of each of those messages and locate the messages in the *BEA WebLogic Enterprise System Messages* or the *BEA Tuxedo System Message Manual*. The message manual provides the following information about each system message:
   - Details about the error condition flagged by the message
   - Recommendations for actions you can take to recover

Broadcasting Unsolicited Messages (BEA Tuxedo System)

To send an unsolicited message, enter the following command:

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

By default, the message is sent to all clients. You have the choice, however, of limiting distribution to one of the following recipients:

- One machine (`-m` *`machine`*)
- One client group (`-c` *`client_group`*)
- One user (`-u` *`user`*)

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

Performing System File Maintenance

This section provides instructions for the following tasks that you may need to perform in the course of maintaining your file system:

- Creating a device list
- Destroying a device list
- Reinitializing a device
- Printing the Universal Device List
- Printing VTOC information

Creating a Device List

Complete the following steps to create a device list.

1. Start a `tmadmin` session.
2. Enter the following command:

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

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

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

Destroying a Device List

To destroy a device list with index *devindx*, enter the following command:

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

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

Reinitializing a Device

To reinitialize a device on a device list, enter the following command:

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

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

Printing the Universal Device List (UDL)

To print a UDL, enter the following command:

```
lidl
```

To specify the device from which you want to obtain the UDL, you have a choice of two methods:

- Specify the following on the `lidl` command line:

  `-z` *`device name`* `[`*`devindx`*`]`

- Set the environment variable `FSCONFIG` to the name of the desired device.

Printing VTOC Information

To get information about all VTOC table entries, enter the following command:

```
livtoc
```

To specify the device from which you want to obtain the VTOC, you have a choice of two methods:

- Specify the following on the `lidl` command line:

  `-z` *`device name`* `[`*`devindx`*`]`

- Set the environment variable `FSCONFIG` to the name of the desired device.

Repairing Partitioned Networks

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

A network partition may be caused by the following:

- A network failure—one of two types:
  - Transient failure, which corrects itself in minutes
  - Severe failure, which requires you to take the partitioned machine out of the network
- A machine failure on either:
  - The master machine
  - The nonmaster machine
- A `BRIDGE` failure

The procedure you follow to recover from a partitioned network depends on the cause of the partition. Recovery procedures for these situations are provided in this section.

Detecting Partitioned Networks

There are several ways to detect a network partition:

- You can check the user log (`ULOG`) for messages that may shed light on the origin of the problem.
- You can gather information about the network, server, and service by running the `tmadmin` commands provided for this purpose.

Checking the ULOG

When things go wrong with the network, BEA WebLogic Enterprise or BEA Tuxedo system administrative servers start sending messages to the `ULOG`. If the `ULOG` is set up over a remote file system, all messages are written to the same log. In such a case you can run the `tail`(1) command on one file and check the failure messages displayed on the screen.

If, however, the remote file system is using the same network, the remote file system may no longer be available.

Example

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

Gathering Information about the Network, Server, and Service

Listing 22-1 provides an example of a `tmadmin` session in which information is being collected about a partitioned network, and a server and a service on that network. Three `tmadmin` commands are run:

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

Listing 22-1 Example of a tmadmin Session

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

> psr -m SITE1
a.out Name     Queue Name   Grp Name   ID   Rq Done  Load Done  Current Service
BBL            30002.00000  SITE1       0      -         -        (- )
DBBL           123456       SITE1       0     121       6050      MASTERBB
simpserv       00001.00001  GROUP1      1      -         -        ( - )
BRIDGE         16900672     SITE1       0      -         -        ( DEAD )


>psc -m SITE1
Service Name   Routine Name  a.out     Grp Name ID  Machine   # Done Status
------------   ------------  --------  -------- --  -------   ------------
ADJUNCTADMIN   ADJUNCTADMIN  BBL       SITE1     0   SITE1      - PART
ADJUNCTBB      ADJUNCTBB     BBL       SITE1     0   SITE1      - PART
TOUPPER        TOUPPER       simpserv  GROUP1    1   SITE1      - PART
BRIDGESVCNM    BRIDGESVCNM   BRIDGE    SITE1     1   SITE1      - PART
```

Restoring a Network Connection

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

Recovering from Transient Network Failures

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

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

   ```
   rco non-partioned_node1 partioned_node2
   ```

Recovering from Severe Network Failures

To recover from severe network failure, complete the following steps:

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

   ```
   pcl partioned_machine
   ```

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

Restoring Failed Machines

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

Restoring a Failed Master Machine

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

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

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

   ```
   boot -B SITE1
   ```

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

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

Restoring a Failed Nonmaster Machine

To restore a failed nonmaster machine, complete the following steps:

1. On the master machine, start a `tmadmin` session.
2. Run `pclean`, specifying the partitioned machine on the command line.
3. Fix the machine problem.

4. Restore the failed machine by booting the Bulletin Board Listener (BBL) for it from the master machine.
5. If you have migrated application servers and data off the failed machine, boot them or migrate them back.

In Listing 22-2, `SITE2`, a nonmaster machine, is restored.

Listing 22-2 Example of Restoring a Failed Nonmaster Machine

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

> pclean SITE2
Cleaning the DBBL.

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

> boot -B SITE2
Booting admin processes ...

Exec BBL -A :

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

Replacing System Components (BEA Tuxedo System)

To replace BEA Tuxedo system components, complete the following steps:

1. Install the BEA Tuxedo system software that is being replaced.
2. Shut down those parts of the application that will be affected by the changes:
   - The BEA Tuxedo system servers may need to be shut down if libraries are being updated.

- Application clients and servers must be shut down and rebuilt if relevant BEA Tuxedo system header files or static libraries are being replaced. (Application clients and servers do not need to be rebuilt if the BEA Tuxedo system message catalogs, system commands, administrative servers, or shared objects are being replaced.)

3. If relevant BEA Tuxedo system header files and static libraries have been replaced, rebuild your application clients and servers.
4. Reboot the parts of the application that you shut down.

Replacing Application Components

To replace components of your application, complete the following steps:

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

Cleaning Up and Restarting Servers Manually

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

Cleaning Up Resources

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

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

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

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

To clean up other resources, complete the following steps:

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

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

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

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

Checking the Order in Which Servers Are Booted (BEA WebLogic Enterprise Servers)

If a BEA WebLogic Enterprise application fails to boot, open the application's `UBBCONFIG` file with a text editor and check whether the servers are booted in the correct order in the `SERVERS` section. The following is the correct order in which to boot the servers on a BEA WebLogic Enterprise system. A BEA WebLogic Enterprise application will not boot if this order is not adhered to.

Boot the servers in the following order:

1. The system Event Broker, `TMSYSEVT.`
2. The `TMFFNAME` server with the `-N` option and the `-M` option, which starts the NameManager service (as a master). This service maintains a mapping of application-supplied names to object references.
3. The `TMFFNAME` server with the `-N` option only, to start a slave NameManager service.
4. The `TMFFNAME` server with the `-F` option, to start the FactoryFinder.
5. The application servers that are advertising factories.

For a detailed example, see the section "Required Order in Which to Boot Servers (BEA WebLogic Enterprise Servers)" on page 3-49 in Chapter 3, "Creating a Configuration File."

Checking Hostname Format and Capitalization (BEA WebLogic Enterprise Servers)

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

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

For example, if the network address is specified as `//TRIXIE:3500` in the `ISL` command-line option string (in the server application's `UBBCONFIG` file), specifying either `//192.12.4.6:3500` or `//trixie:3500` in the Bootstrap object constructor or in `TOBJADDR` will cause the connection attempt to fail.

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

Some Clients Fail to Boot (BEA WebLogic Enterprise Servers)

You may want to perform the following steps on a Windows NT server that is running a BEA WebLogic Enterprise application, if the following problem occurs: some Internet Inter-ORB Protocol (IIOP) clients boot, but some clients fail to create a Bootstrap object and return an `InvalidDomain` message, even though the `//host:port` address is correctly specified. (For related information, see the section "Checking Hostname Format and Capitalization (BEA WebLogic Enterprise Servers)" on page 22-16.)

1. Start `regedt32`, the Registry Editor.
2. Go to the `HKEY_LOCAL_MACHINE on Local Machine` window.
3. Select:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Afd\Parameters
```

4. Add the following values by using the Edit —> Add Value menu option:

```
DynamicBacklogGrowthDelta: REG_DWORD : 0xa
EnableDynamicBacklog: REG_DWORD: 0x1
MaximumDynamicBacklog: REG_DWORD: 0x3e8
MinimumDynamicBacklog: REG_DWORD: 0x14
```

5. Restart the Windows NT system for the changes to take effect.

These values replace the static connection queue (that is, the backlog) of five pending connections with a dynamic connection backlog, that will have at least 20 entries (minimum 0x14), at most 1000 entries (maximum 0x3e8), and will increase from the minimum to the maximum by steps of 10 (growth delta 0xa).

These settings only apply to connections that have been received by the system, but are not accepted by an IIOP Listener. The minimum value of 20 and the delta of 10 are recommended by Microsoft. The maximum value depends on the machine. However, Microsoft recommends that the maximum value not exceed 5000 on a Windows NT server.

Aborting or Committing Transactions

This section provides instructions for aborting and committing transactions.

Aborting a Transaction

To abort a transaction, enter the following command:

`aborttrans (abort) [-yes] [-g `*`groupname`*`] `*`tranindex`*

- To determine the value of *`tranindex`*, run the `printtrans` command (a `tmadmin` command).
- If *`groupname`* is specified, a message is sent to the TMS of that group to mark as "aborted" the transaction for that group. If a group is not specified, a message is sent, instead, to the coordinating TMS, requesting an abort of the transaction. You must send abort messages to all groups in the transaction to control the abort.

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

Committing a Transaction

To commit a transaction, enter the following command:

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

- Both *`groupname`* and *`tranindex`* are required arguments.
- The operation fails if the transaction is not precommitted or has been marked aborted.
- This message should be sent to all groups to fully commit the transaction.

Cautions

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

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

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

Recovering from Failures When Transactions Are Used

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

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

1. Shut down the BEA WebLogic Enterprise or BEA Tuxedo application. For instructions, see Chapter 4, "Starting and Shutting Down Applications."
2. Get the last full backup of the database and restore the file. For example, restore the full backup version of the database from last Sunday at 12:01 A.M.
3. Apply the incremental backup files, such as the incrementals from Monday and Tuesday. For example, assume that this step restores the database up until 11:00 P.M. on Tuesday.
4. Apply the AIJ, or transaction journal file, that contains the transactions from 11:15 P.M. on Tuesday up to 2:50 P.M. on Wednesday.
5. Open the database again.
6. Restart the BEA WebLogic Enterprise or BEA Tuxedo applications.

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

Index