BEA WebLogic Enterprise

Administration Guide

BEA WebLogic Enterprise 4.2
Document Edition 4.2
July 1999

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Administration Guide

Document Edition	Date	Software Version
4.2	July 1999	BEA WebLogic Enterprise 4.2

Contents

4. Starting and Shutting Down Applications

5. Distributing Applications

6. Building Networked Applications

7. Configuring Transactions

8. Managing Interface Repositories (WLE System)

11. Managing Workstation Clients (BEA TUXEDO System)

12. Managing Remote Client Applications (WLE Systems)

13. Managing Queued Messages (BEA TUXEDO System)

14. Securing Applications

15. Monitoring a Running System

16. Monitoring Log Files

17. Tuning Applications

18. Migrating Applications

19. Dynamically Modifying Systems

20. Dynamically Reconfiguring Applications

21. Event Broker/Monitor (BEA TUXEDO System)

22. Administrative Reference (WLE System)

23. Troubleshooting Applications

Index

Preface

Purpose of This Document

This document describes how to administer BEA WebLogic Enterprise (sometimes referred to as WLE) and BEA TUXEDO systems.

Note: Effective February 1999, the BEA M3 product is renamed. The new name of the product is BEA WebLogic Enterprise (WLE).

Who Should Read This Document

This document is intended for administrators who configure operational parameters that support mission-critical WLE and BEA TUXEDO systems.

How This Document Is Organized

The *Administration Guide* is organized as follows:

- Chapter 1, "Introduction to Administration," introduces the administration tasks.
- Chapter 2, "Administration Tools," identifies the administration tools that are part of the WLE and TUXEDO systems.
- Chapter 3, "Creating a Configuration File," details the application, machine, group, server, services, interfaces, routing, and network parameters in an application's `UBBCONFIG` configuration file.

- Chapter 4, “Starting and Shutting Down Applications,” explains how to start and shut down applications.
- Chapter 5, “Distributing Applications,” explains how to distribute applications.
- Chapter 6, “Building Networked Applications,” explains how to build networked applications.
- Chapter 7, “Configuring Transactions,” explains how to configure transactions.
- Chapter 8, “Managing Interface Repositories (WLE System),” explains how to manage Interface Repositories. This chapter is specific to the WLE system.
- Chapter 9, “Configuring Multiple Domains (WLE System),” explains how to configure multiple domains. This chapter is specific to WLE.
- Chapter 10, “Working with Multiple Domains (BEA TUXEDO System),” explains how to manage multiple domains. This chapter is specific to the BEA TUXEDO System.
- Chapter 11, “Managing Workstation Clients (BEA TUXEDO System),” explains how to manage workstation clients. This chapter is specific to the BEA TUXEDO System.
- Chapter 12, “Managing Remote Client Applications (WLE Systems),” explains how to manage remote WLE client applications. This chapter is specific to the WLE system.
- Chapter 13, “Managing Queued Messages (BEA TUXEDO System),” explains how to manage queued messages. This chapter is specific to the BEA TUXEDO system.
- Chapter 14, “Securing Applications,” explains how to implement application security. The access control list (ACL) mechanism used in the BEA TUXEDO system is not present in the WLE 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 WLE administrator.
- Chapter 15, “Monitoring a Running System,” explains how to monitor a running system.
- Chapter 16, “Monitoring Log Files,” explains how to monitor log files.
- Chapter 17, “Tuning Applications,” explains how to tune applications.

- Chapter 18, “Migrating Applications,” explains how to migrate applications.
- Chapter 19, “Dynamically Modifying Systems,” explains how to modify systems dynamically.
- Chapter 20, “Dynamically Reconfiguring Applications,” explains how to reconfigure applications dynamically.
- Chapter 21, “Event Broker/Monitor (BEA TUXEDO System),”explains how to use Event Broker. This chapter is specific to the BEA TUXEDO system.
- Chapter 22, “Administrative Reference (WLE System),” describe administrative commands and server processes.
- Chapter 23, “Troubleshooting Applications,” explains how to troubleshoot problems.

How to Use This Document

This document, the *Administration Guide*, is designed primarily as an online, hypertext document. If you are reading this as a paper publication, note that to get full use from this document you should access it as an online document via the Online Documentation CD for the BEA WebLogic Enterprise 4.2 release.

The following sections explain how to view this document online, and how to print a copy of this document.

Opening the Document in a Web Browser

To access the online version of this document, open the following file:

```
\doc\wle\v42\index.htm
```

Note: The online documentation requires Netscape Communicator version 4.0 or later, or Microsoft Internet Explorer version 4.0 or later.

Printing from a Web Browser

You can print a copy of this document, one file at a time, from the Web browser. Before you print, make sure that the chapter or appendix you want is displayed and *selected* in your browser. To select a chapter or appendix, click anywhere inside the chapter or appendix you want to print.

The Online Documentation CD includes Adobe Acrobat PDF files of all of the online documents. You can use the Adobe Acrobat Reader to print all or a portion of each document. On the CD Home Page, click the PDF Files button and scroll to the entry for the document you want to print.

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.
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names 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 *`
	`.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`

Convention	Item
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 firstfile-syntax] [-l lastfile-syntax]`
|	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*: `genicf [options] idl-filename...`
. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

Related Documentation

The following sections list the documentation provided with the BEA WebLogic Enterprise software, related BEA publications, and other publications related to the technology.

BEA WebLogic Enterprise Documentation

The BEA WebLogic Enterprise information set consists of the following documents:

Installation Guide

C++ Release Notes

Java Release Notes

Getting Started

Guide to the University Sample Applications

Guide to the Java Sample Applications

Creating Client Applications

Creating C++ Server Applications

Creating Java Server Applications

Administration Guide (this document)

Using Server-to-Server Communication

C++ Programming Reference

Java Programming Reference

Java API Reference

JDBC Driver Programming Reference

System Messages

Glossary

Technical Articles

Note: The Online Documentation CD also includes Adobe Acrobat PDF files of all of the online documents. You can use the Adobe Acrobat Reader to print all or a portion of each document.

BEA Publications

Selected BEA TUXEDO Release 6.5 for BEA WebLogic Enterprise version 4.2 documents are available on the Online Documentation CD.

To access these documents:

1. Click the Other Reference button from the main menu.
2. Click the TUXEDO Documents option.

Other Publications

For more information about CORBA, Java, and related technologies, refer to the following books and specifications:

Cobb, E. 1997. *The Impact of Object Technology on Commercial Transaction Processing*. VLDB Journal, Volume 6. 173-190.

Edwards, J. with DeVoe, D. 1997. *3-Tier Client/Server At Work*. Wiley Computer Publishing.

Edwards, J., Harkey, D., and Orfali, R. 1996. *The Essential Client/Server Survival Guide*. Wiley Computer Publishing.

Flanagan, David. May 1997. *Java in a Nutshell*, 2nd Edition. O'Reilly & Associates, Incorporated.

Flanagan, David. September 1997. *Java Examples in a Nutshell*. O'Reilly & Associates, Incorporated.

Fowler, M. with Scott, K. 1997. *UML Distilled, Applying the Standard Object Modeling Language*. Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series.

Jacobson, I. 1994. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley.

Mowbray, Thomas J. and Malveau, Raphael C. (Contributor). 1997. *CORBA Design Patterns*, Paper Back and CD-ROM Edition. John Wiley & Sons, Inc.

Orfali, R., Harkey, D., and Edwards, J. 1997. *Instant Corba*. Wiley Computer Publishing.

Orfali, R., Harkey, D. February 1998. *Client/Server Programming with Java and CORBA*, 2nd Edition. John Wiley & Sons, Inc.

Otte, R., Patrick, P., and Roy, M. 1996. *Understanding CORBA*. Prentice Hall PTR.

Rosen, M. and Curtis, D. 1998. *Integrating CORBA and COM Applications*. Wiley Computer Publishing.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Loresen, W. 1991. *Object-Oriented Modeling and Design*. Prentice Hall.

The Common Object Request Broker: Architecture and Specification. Revision 2.2, February 1998. Published by the Object Management Group (OMG).

CORBAservices: Common Object Services Specification. Revised Edition. Updated: November 1997. Published by the Object Management Group (OMG).

Contact Information

The following sections provide information about how to obtain support for the documentation and the software.

Documentation Support

If you have questions or comments on the documentation, you can contact the BEA Information Engineering Group by e-mail at **docsupport@beasys.com**. (For information about how to contact Customer Support, refer to the following section.)

Customer Support

If you have any questions about this version of the BEA WebLogic Enterprise product, or if you have problems installing and running the BEA WebLogic Enterprise software, contact BEA Customer Support through BEA WebSupport at `www.beasys.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

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 WLE or BEA TUXEDO system, and then make it happen.

This chapter discusses the following topics:

- 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 WLE 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 WLE 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 runtime 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 following 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 WLE system and the application.
Configure	Your system	Set the parameters provided by the WLE 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 WLE system for protecting your application and data.

Depending on your application, you may also need to set up the following:

Distributed applications	Create distributed applications with the routing tools: factory-based routing in WLE applications and data-dependent routing in BEA TUXEDO applications.
Networked applications	Set up any networked applications.

WLE remote client applications	To support WLE 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 shown in this table, except installation. For installation instructions, see the *Installation Guide*.

The Operational Phase

During the this phase, you must do the following 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:

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.
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 WLE and BEA TUXEDO Systems

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

The UBBCONFIG configuration file for WLE systems includes the following enhancements to support the configuration of WLE 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 WLE 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 WLE systems.

  However, WLE systems use a different communications protocol to connect remote and foreign clients to WLE 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 WLE system calls its gateway processes the IIOP Server Handler (ISH) and the IIOP Server 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 WLE 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 WLE 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 WLE systems, the system designer must personally communicate to you the routing criteria of CORBA interfaces. For WLE systems, there is no service request message data or associated buffer information available for routing. This occurs because WLE 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 WLE 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 WLE.

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."

Also see 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. Those MIB classes were enhanced for WLE.

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 WLE 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 WLE application?
- Will data-dependent routing be used in your BEA TUXEDO application?
- In what order of priority should interfaces in WLE applications or BEA TUXEDO services be available?
- For WLE 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 WLE 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 WLE or BEA TUXEDO system gives you a choice of several methods for performing the same set of administrative tasks. Whether you are more comfortable using a graphical user interface or entering commands at a shell prompt, you will be able to find a comfortable method of doing your job as the administrator of a domain. This chapter describes the menu of administration tools.

This chapter discusses the following topics:

- 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 WLE 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 WLE 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 WLE 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 WLE 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 WLE 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 WLE and BEA TUXEDO documentation, as `UBBCONFIG`. The `UBBCONFIG` file may, in fact, be given any file name. 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 the section "Create TUXCONFIG" in Chapter 4, "Starting and Shutting Down Applications." For specific 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 System)."

Tools for Run-time Administration

With your WLE 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. For more information about designating the MASTER machine, see "Identifying the Master Machine" on page 3-12.

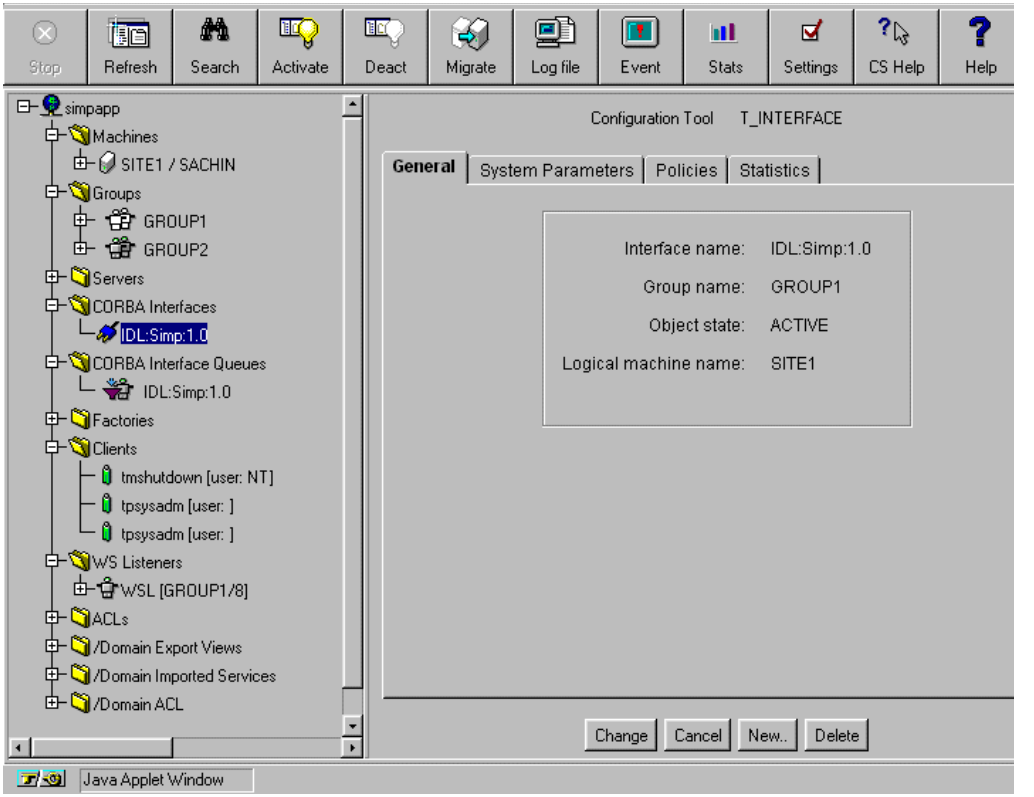
As the processing and resource usage requirements of your application evolve, you may need to add machines, servers, clients, interfaces, services, and so on, to your existing system.

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 WLE and BEA TUXEDO applications. Figure 2-1 shows a sample Administration Console screen.

Figure 2-1 Sample BEA Administration Console Screen



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 WLE 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 WLE or BEA TUXEDO servers, or removes a set of WLE 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 crucially important 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. Those MIB classes were added for WLE.

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."

Also see 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. Those MIB classes were enhanced for WLE.

3 Creating a Configuration File

This chapter discusses the following topics:

- About the Configuration File
- Administrative Requirements and Hints
- Configuring RESOURCES
- Configuring Machines
- Configuring Groups
- Configuring Servers
- Configuring Services (BEA TUXEDO System)
- Configuring Interfaces (WLE Servers)
- Configuring Routing
- Configuring Network Information

About the Configuration File

The configuration file is the primary means of defining the configuration of WLE applications. It consists of parameters that the WLE software interprets to create an executable application.

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

Build Environment

In addition to the configuration file, building a WLE application requires three other basic components:

- 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 formats:

`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 WLE 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) in its monitoring activity. 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 path name of the device or system file where UXCONFIG is to be loaded.

Configuration File Content

The configuration file can contain up to 9 specification sections and 80 different parameters. Lines beginning with an asterisk (*) indicate the beginning of a specification section. Each such line contains the name of the section immediately following the asterisk.

Section Names and Functions

Allowable 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. (This section is required.)
- MACHINES—specifies the logical names for physical machines for the configuration and parameters specific to a given machine. (This section is required.)
- GROUPS—defines all application server groups by group name, logical machine, and group number. (This section is required.)
- SERVERS—specifies server processes to be booted in the application. (This section is 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. (This section is required.)

- INTERFACES—defines application-wide, default parameters for CORBA interfaces used by the application. (This section is optional.)
- ROUTING—defines the routing criteria named in the INTERFACES section for WLE factory-based routing, or in the SERVICES section for BEA TUXEDO data-dependant routing. (This section is optional.)
- NETGROUPS —specifies the network groups available to an application in a LAN environment. (This section is optional.)
- NETWORK—describes the network configuration for a LAN environment. (This section is optional.)

Each of these sections 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*.

Formatting Requirements and Guidelines

The sections must be arranged in the file as follows:

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

Additionally, the following guidelines apply to all sections except the RESOURCES section:

- You can specify multiple entries, each with its own selection of parameters.
- You can use 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 WLE software. Examine the content of this sample file.

Notice that 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. Notice that the UBBCONFIG file also contains the required SERVICES section, but that section contains no information. Also, the syntax used for entries in the file is described in detail in 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 WLE 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 Hints

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

Administrative Requirements

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

1. 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.

2. 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, since the application must be rebooted, the NameManagers are rebooted as part of the application startup. 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 userlog.

3. 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.

4. 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 userlog. 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.

5. 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.

6. 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

Adhering to the following requirements will improve system reliability:

1. The FactoryFinder gives out object references for factories that are no longer active. This occurs because the servers containing those factories have become unavailable and 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 current active server.

2. 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.

3. 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:

- Running FactoryFinder and NameManager services in separate servers on the same machine may provide a quicker response than running these services on different machines, because there is no network inter-machine communication.

Configuring RESOURCES

This section explains how to set RESOURCES parameters that control the application as a whole. This is a required section and must appear as the first section in the configuration file. Some of the parameters in this 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 WLE 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	The IPC limit for the number of server processes	20	Allows up to 20 active server processes for this application.
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 WLE system.
MAXSERVICES	The IPC limit for the number of services offered	25	Allows up to 25 services to be advertised. On WLE systems, each CORBA interface is mapped to a BEA TUXEDO service.

Parameter	Description	Sample Value	Meaning of Sample Value
MAXOBJECTS	The IPC limit for the number of CORBA objects	`800`	Allows up to 800 WLE active CORBA objects in the Active Object Map tables in the Bulletin Board. The `MAXOBJECTS` parameter is specific to the WLE 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 that this is 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.
`AUTHSVC`	The name of an application authentication service that is 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.

Parameter	Description	Sample Value	Meaning of Sample Value
LDBAL	Server load balancing enabled	Y	Indicates that load balancing is on. This value is always Y in WLE systems; that is, setting LDBAL to N is ignored in the WLE system.

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

Setting the Address of Shared Memory

You set the address of shared memory using the IPCKEY parameter. This parameter is used by the WLE system to allocate application IPC resources such that they may be located easily by new processes joining the application. This key and its variations are used internally to allocate the Bulletin Board, message queues, and semaphores that must be available to new application processes. In 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 has the following characteristics:

- It is required and must appear in the configuration file.
- It is used to access the Bulletin Board and other IPC resources.
- Its value must be an integer in the range 32,769 to 262,143.
- No other application on the system may use this specific value for its IPCKEY.

Identifying 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 as a Logical Machine Identifier (LMID). This is an alphanumeric name chosen by the administrator. (LMIDs are discussed further in the section "Configuring Machines" on page 3-24.)

Two LMIDs 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 has the following characteristics:

- It is required and it controls booting and administration.
- Two LMIDs are required for migration to back up the master machine.

Setting the Application Type

Among the architectural decisions needed for a WLE 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 server migration be supported?

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

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

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.

Parameter	Characteristics
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. ♦ In the sample RESOURCES section, model is MP; options is set to LAN and MIGRATE.

Note: No OPTIONS are specified for the SHM model.

Defining Access Control (BEA TUXEDO Servers)

The WLE system provides security features, but does not support access control lists (ACLs) at this time. This section is specific 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.

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 WLE or BEA TUXEDO system at one site. It 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.)

  For multithreaded WLE 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 WLE Java software, as opposed to threads started and managed by an application program. Internally, WLE 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 done, the worker thread is returned to the pool of available threads.

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

  A single threaded server counts as one accessor.

  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 WLE 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 WLE system, the maximum number of CORBA interfaces to be accommodated in the interface table of the Bulletin Board. Valid values are from 0 to 32,766. If not specified, and if the WLE system is licensed for the domain, the default is 100. If the WLE 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 WLE 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 32,766. If not specified, and if the WLE 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 WLE 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 WLE systems, each CORBA interface is mapped to a BEA TUXEDO service. Make sure you account for the number of services generated.

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 more important to scrutinize 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 3-3 lists the characteristics for the MAXACCESSERS, MAXSERVERS, MAXINTERFACES, MAXOBJECTS, and MAXSERVICES parameters.

Table 3-3 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 (WLE 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 (WLE 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.

Parameter	Characteristics
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 WLE 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 WLE 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 3-60.

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 current 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.

The MAXBUFTYPE and MAXBUFSTYPES parameters have the following characteristics:

Parameter	Characteristics
MAXBUFTYPE	Maximum number of buffer types allowed in the system. Default is 16. Example: MAXBUFTYPE 20

Parameter	Characteristics
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. It must not be set so that SANITYSCAN * SCANUNIT exceeds 300; its current default is set so that 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

The SCANUNIT, SANITYSCAN, and BLOCKTIME parameters have the following characteristics:

Parameter	Characteristics
SCANUNIT	Establishes granularity of check intervals and timeouts. Value must be multiples of 5 seconds. Example: SCANUNIT 10 The default is 10.
SANITYSCAN	Frequency that the BBL checks the server (in SCANUNIT intervals). SCANUNIT * SANITYSCAN must not exceed 300. Default so that SCANUNIT * SANITYSCAN is approximately 120 seconds. Example: SANITYSCAN 3
BLOCKTIME	Timeout for blocking messages. SCANUNIT * BLOCKTIME must not exceed 300. Defaults so that 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 is specific 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 32,766.
- 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 the following three levels of security:

- 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 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: The WLE CORBA API does not support access control lists (ACLs) at this time. The SECURITY MANDATORY_ACL parameter is ignored in WLE systems. For details about the supported security parameters, see Chapter 14, "Securing Applications."

- 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.

The SECURITY and AUTHSVC parameters have the following 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 WLE 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 is specific 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 choice in the TPINIT structure when tpinit() is called.

Following are three possible methods:

- IGNORE—clients should ignore unsolicited messages.
- DIPIN—clients should receive unsolicited messages only when they call tpchkunsol() or when they make an ATMI call.
- SIGNAL—clients should 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.

Characteristics of the NOTIFY and USIGNAL Parameters

The NOTIFY and USIGNAL parameters have the following 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

Parameter	Characteristics
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 and/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. Following are the options for this parameter:

- PROTECTED—WLE or BEA TUXEDO libraries compiled with application code will not attach to shared memory while executing system code.
- FASTPATH—WLE 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.

The PROTECTED, FASTPATH, and NO_OVERRIDE parameters have the following 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.
NO_OVERRIDE	The specified option cannot be changed.

Note: An 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).
- The location of the configuration file (TUXCONFIG).
- The location of the installed WLE or BEA TUXEDO software (TUXDIR). Note that the TUXDIR parameter is used on WLE systems. Use it to identify the top-level location where you installed the WLE system, such as c:\wledir.
- The location of the application servers (APPDIR).

- The location of the application log file (ULOGPFX).
- The location of the environment file (ENVFILE).
- The maximum number of active CORBA objects to be accommodated in the Active Object Table for this processor (MAXOBJECTS).

The only required parameters for the MACHINES section are LMID, TUXCONFIG, TUXDIR and APPDIR.

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
```

Description of Parameters in a Sample MACHINES Section

The following table provides a sample of parameters and their values in the MACHINES section of the configuration file.

Parameter	Meaning
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*.

Parameter	Meaning
TUXDIR	The double quoted string of the full path to the installed WLE or BEA TUXEDO software.
APPDIR	The double quoted string of the full path to the application directory.
TUXCONFIG	The double quoted string of the full path name of the configuration file.
ENVFILE	The double quoted string of the full path name of a file containing environment information.
MAXOBJECTS	Override the system wide value defined in RESOURCES with 700.
ULOGPFX	The double quoted string of the full path name prefix of the log file.
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:

  <address> LMID=*<logical_machine_name>*

- The address identifies the physical processor name.

- 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 path name 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 file name (though it should remain TUXCONFIG for convention purposes) for the machine.
- The full path name for TUXCONFIG can be up to 64 characters.
- The value of TUXCONFIG must match the TUXCONFIG environment variable.

Identifying the Locations of the System Software and Application Server Machines

Each machine in an application must have a copy of the WLE 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 WLE or BEA TUXEDO software looks in the TUXDIR/bin and APPDIR for executables.

The TUXDIR and APPDIR parameters have the following characteristics:

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

Identifying the Location of the User Log File

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 2-digit year. By default, the file is written into the APPDIR.

You can override the default directory and prefix by specifying the ULOGPFX parameter that is the absolute path name of the application log file, without the date. For example, it 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 application log contains all explanations of TPESYSTEM and TPEOS errors.
- You can use the application to log application errors.
- 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 WLE 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 because they affect an application's operation. Most of these settings are specific 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 WLE or BEA TUXEDO system.
- Set FIELDTBLS, FLDTBLDIR, and so on, 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 3-4 lists the system-wide parameters you can override for a specific machine.

Table 3-4 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, 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.
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.
MAXOBJECTS (WLE 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.

Parameter	Characteristics
MAXGTT	Maximum number of simultaneous global transactions in which a particular machine can be involved.

Note: You can override values on remote machines.

Configuring Groups

You can use GROUPS to group servers together logically, which 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 application servers 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. Additional flexibility is available to support distributed transactional systems.

Specifying a Group Name, Number, and LMID

The group name is the basis for a GROUPS section entry and is an alphanumeric name by which the group is identified. It is given a mandatory, unique group number (GRPNO). Each group must reside wholly 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 WLE 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 need to be configured, and the machines on which they run on need to 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 in the University Production Sample Application (WLE Servers)" on page 3-63 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 WLE 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.

Configuring Servers

This section explains 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 especially the section "Starting the Java Server" on page 3-38.

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)

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

Table 3-5 provides a sample of parameters and their values in the SERVERS section of the configuration file.

Table 3-5 SERVERS Section Parameters

Parameter	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 in the GRACE parameter. The default is 1. If the server is to be restartable, MAXGEN must be >= 2. The number of restarts is at most *number* - 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 - 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.

Parameter	Meaning
ENVFILE="/usr/home/envfile"	Read environment settings from the file ENVFILE.
Java-options	Java Virtual Machine (JVM) options, similar to the options passed to the Java interpreter command. These options are outlined in the section "Starting the Java Server" on page 3-38.
archive [options]	For the JavaServer, *archive* is the name of the Java Archive (JAR) file that was created for the application by the buildjavaserver command. The options that appear after the archive file name are passed to the application via the com.beasys.Tobj.Server.initialize operation. JavaServer looks for the application's JAR file in the value for the APPDIR environment variable. For more information, see the section "Starting the Java Server" on page 3-38.
SYSTEM_ACCESS= FASTPATH	Specifies the default mode used by system libraries within application processes to gain access to the WLE 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, which is the name of an executable file built with**:**

- buildserver(1), on BEA TUXEDO systems
- buildobjserver, for WLE C++ server applications
- buildjavaserver, for WLE 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.

The Server Name, SRVGRP, and SRVID parameters have the following characteristics:

Parameter	Characteristics
Server name	It identifies the executable to be booted. It is built with buildserver(1) on BEA TUXEDO systems, or with buildobjserver (C++) or buildjavaserver (Java) on WLE systems. It is required, but may not be unique.
SRVGRP (Server Group)	It identifies the group affiliation. The group name from a GROUPS section entry. It is required.
SRVID (Server ID)	It is numeric. It is required and unique within a server group.

Using Server Command-Line Options

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

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 WLE 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 WLE C++ servers, or the `Server.initialize` method on WLE Java servers.

The following table provides a partial list of the available options.

Option	Purpose
`-o` *`filename`*	Redirects standard output to file *`filename.`*
`-e` *`filename`*	Redirects standard error to file *`filename.`*
`-f` *filename* (WLE Interface Repository servers)	Specifies a nondefault location, name, or both of an Interface Repository. This option can only be used for WLE Interface Repository servers.
`-s` *`services`* (BEA TUXEDO servers)	Advertises services (not supported in WLE servers).
`-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.

Option	Purpose
-A	The default for CLOPT is -A, which tells the server to advertise all the services built into it with buildserver(1) or buildobjserver.

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 syntax is CLOPT="*servopts -- application_opts*".
- This is an optional parameter with a default of -A.
- Both main() and tpsvrinit() use server command-line options (BEA TUXEDO servers only).
- The servopts(5) options are passed to main() (BEA TUXEDO servers only).
- The application options are passed to tpsvrinit() (BEA TUXEDO servers only).
- A BANKAPP example is CLOPT="-A -- -T 10".

Starting the Java Server

In the WLE 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 WLE server to be integrated in a WLE application. By default, the server that loads the JVM is called JavaServer.

You include the options to start JavaServer in the SERVERS section of the application's UBBCONFIG file.

See the section "Required Order in Which to Boot Servers (WLE Servers)" on page 3-45 for important information about starting the WLE servers in the correct order.

Threading Options

The WLE Java system supports the ability to configure multithreaded WLE applications written in Java. A multithreaded WLE Java server can service multiple object requests simultaneously, while a single-threaded WLE 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 option` to the `JavaServer` parameter in the `SERVERS` section. The `-M` options are described in the section "WLE System Options" on page 3-41.

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

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

Programs written to WLE Java run without modification in both modes.

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 multi-processor machine

If the application is running on a single-processor machine and the application is CPU-intensive only, without any I/O or 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 multi-processor 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 an XA-enabled version of JavaServer is built using `buildXAJS`, the server supports only the single-threaded mode; in this case, the WLE system ignores the `-M number` command line argument for multithreading.

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 Command Line Options

When you start `JavaServer`, the options are:

```
JavaServer
       SRVGRP=group
       SRVID=number
       CLOPT="-A [java_options] archive-file [options]"
```

The `JavaServer` options are as follows:

- *`group`* is the name of the WLE group in the `GROUPS` section of the `UBBCONFIG` file.
- *`number`* is a numeric identifier of the server in the group.
- *`java_options`* 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 Options,” “WLE System Options,” and “Nonstandard Java Options.”
- *`archive-file`* is the Java Archive (JAR) file that was created for the application by the `buildjavaserver` command. You can specify a fully qualified path to the location of the JAR file; or, `JavaServer` looks for the application’s JAR file in the value for the `APPDIR` environment variable.
- The *`[options]`* after the archive file name are passed to the application via the `com.beasys.Tobj.Server.initialize` operation. These options are described in the sections “Standard Java Options,” “WLE System Options,” and “Nonstandard Java Options.”

For example:

```
JavaServer
       SRVGRP=BANK_GROUP1
       SRVID=8
```

```
CLOPT="-A -- -M 10 Bankapp.jar TellerFactory_1"
SYSTEM_ACCESS=FASTPATH
```

In this example, the JavaServer for Bankapp's TellerFactory interface is 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.

A worker thread is a thread that is started and managed by the WLE Java software, as opposed to threads started and managed by an application program. Internally, WLE 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 done, the worker thread is returned to the pool of available threads.

Standard Java Options

The standard Java 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.

WLE System Options

The following options are provided by the WLE system:

`-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 the number after `-M` is a negative decimal, the server will revert to single-threaded mode. If the number after `-M` is larger than `500`, the server will use a maximum of 500 worker threads. In all cases, if the number specified is invalid, the WLE 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`).

Nonstandard Java Options

The Java Virtual Machine (JVM) in JDK 1.2 supports the nonstandard options in the following list. To display the nonstandard Java 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. `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. `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 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.

Setting the Order in Which Servers Are Booted

You can specify the sequence of servers to be booted with the SEQUENCE parameter, which is 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 way they were booted.

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 WLE system, there is a strict order in which the WLE system Event Broker, the WLE FactoryFinder object, and the application factories must be booted. A WLE application program will not boot if the order is changed. See the section "Required Order in Which to Boot Servers (WLE Servers)" on page 3-45 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 WLE 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

The SEQUENCE, MIN, and MAX parameters have the following characteristics:

Parameter	Characteristics
SEQUENCE	It is an optional parameter with a numeric range of 1 - 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	It 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 are specific to the BEA TUXEDO system. All instances have the same server options. SRVIDs are SRVID + n - 1. The default is 1.
MAX	It represents the maximum number of servers to boot. It defaults to MIN.

Required Order in Which to Boot Servers (WLE Servers)

The following is the correct order in which to boot the servers on a WLE system. A WLE 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 WLE University Basic application, which is one of the sample applications included with the WLE software. This SERVERS section is 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 WLE provided server that
    # runs the NameManager and FactoryFinder services.

    # The NameManager service is a BEA WLE-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 (WLE System)."
- 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 (WLE Systems)."

Identifying the Location of the Server Environment File

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; the semantics are the same. If both the MACHINES section ENVFILE and the SERVERS section ENVFILE are specified, both go into effect. For any overlapping variable defined in both the MACHINES and SERVERS sections, the setting in the SERVERS section prevails.

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 and access of 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 WLE system.

MSSQ Example (BEA TUXEDO Servers)

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, along the lines of 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

The RQADDR, RQPERM, REPLYQ, and RPPERM parameters have the following 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.
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 allows you to 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

The RESTART, RCMD, MAXGEN, and GRACE parameters have the following 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. Allows you to 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.

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

Specifying a Server as Conversational (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.

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 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 Services (BEA TUXEDO System)

This section is specific to BEA TUXEO systems. For information relevant to WLE systems, see the section "Configuring Interfaces (WLE Servers)" on page 3-55.

Note: Although each WLE 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 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 exert significant control over 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

With the BUFTYPE parameter, you can tune a service to check buffer types independently of the actual service code. If you set this parameter, it 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.

A service can have BUFTYPE set to ALL, which means that this service accepts all buffer types. If this parameter is not specified, the default is ALL.

Examples of the BUFTYPE Parameter

The BUFTYPE parameter has the following 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 (WLE Servers)

This section is specific to the WLE system.

The WLE software has an INTERFACES section in the UBBCONFIG file. In this section, you define application-wide default parameters for CORBA 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 WLE system should apply to each interface. Factory-based routing is a feature that allows you to distribute processing to specific server groups.

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)
- The routing criteria to be used for factory-based routing for this CORBA interface (FACTORYROUTING)
- 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 interface (TIMEOUT)

The AUTOTRAN, FACTORYROUTING, LOAD, PRIO, SRVGRP, TRANTIME, and TIMEOUT parameters have the following 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.
FACTORYROUTING = criterion-name	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.
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 WLE 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.
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.)

Parameter	Characteristic
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 timeout. 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 a FACTORYROUTING Criteria

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 3-4). You can find the UBBCONFIG files (ubb_p.nt or ubb_p.mk) for this sample in the directory where the WLE software is installed. Look in the \samples\corba\university\production subdirectory.

Listing 3-4 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 in the University Production Sample Application (WLE Servers)" on page 3-63.

Bankapp Sample

Listing 3-5 shows how factory-based routing is specified in the Bankapp sample application.

Listing 3-5 Bankapp Sample Factory-base 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 WLE 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 WLE 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 WLE 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

The following table identifies the information required for an entry in the ROUTING section.

Parameter	Characteristics
criterion_name	This is a string value with a maximum length of 15 characters. For BEA TUXEDO data-dependent routing, it is the routing criteria name that you specified as the ROUTING parameter in the SERVICES section. For WLE factory-based routing, it is 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 WLE interface.
FIELD	The name of the buffer field on which the routing is to be done. In BEA TUXEDO data-dependent routing, this value is 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 WLE 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.

Parameter	Characteristics
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 quote-enclosed (‘) string, 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 in the University Production Sample Application (WLE 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 WLE 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 WLE 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 following example:

```
*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.

The following example shows the ROUTING section of the UBBCONFIG file used in the Production sample application:

```
*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.

The following example 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 WLE software is configured to run entirely on one machine. However, you can easily configure this application to run on multiple machines.)

```
*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 (WLE Servers)

The following example extends the Bankapp sample application to use factory-based routing. The sample included with the WLE software does not contain these parameter settings.

```
*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.

The NETGRPNO, NETPRIO, NETGROUP, MAXNETGROUPS, and MAXPENDINGBYTES parameters have the following 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-v6.4 releases of the BEA TUXEDO system use only DEFAULTNET.
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 v6.4 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.

Parameter	Required/Optional	Description
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 v6.3 of the BEA TUXEDO system, situation (1) above did not apply.

Sample NETGROUPS Configuration

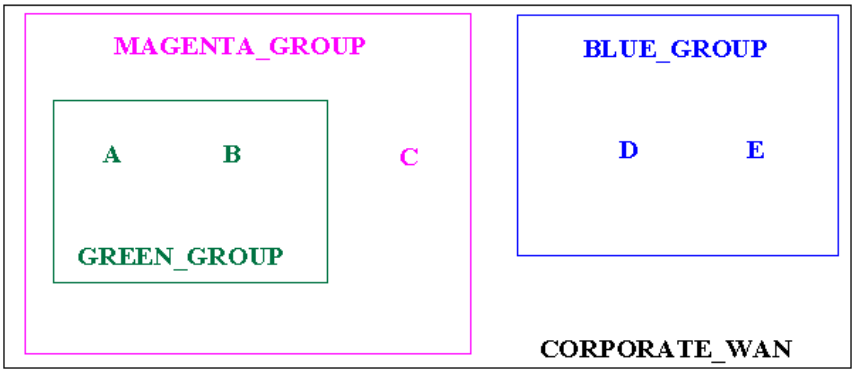
You can associate network addresses with a network group. The following example illustrates how this capability may be useful.

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



The following table shows you which machines have addressees for which 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 3-6.

Listing 3-6 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 WLE or BEA TUXEDO applications.

Topics covered in this chapter are:

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

Starting Applications

Before you issue the command to 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:

Table 4-1 Preliminary Tasks

Task	Procedure
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 WLE 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 NT, no variable for shared libraries is required.

For WLE Java, verify that the following environment variables were defined by the WLE Java installation procedure:

- JAVA_HOME, the directory where the JDK is installed
- CLASSPATH, which must point to:
  - The location of the WLE Java ARchive (JAR) file, which contains all the class files
  - The location of the WLE message catalogs
- TUXDIR, the directory where the WLE 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.

For example, on Windows NT systems:

```
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%
```

For example, on Solaris systems:

```
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 Calculate minimum IPC resources of the configuration.
- -n Do 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 *BEA TUXEDO Installation Guide* to judge whether you need to make some changes. If you do want to change IPC resources, check the administration guide for your platform.

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 Limit the size of the TUXCONFIG file.
- -y Overwrite the existing TUXCONFIG file without asking.

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

Propagate the Software

TUXCONFIG is automatically propagated by the WLE 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 4-2 is a list of files and directories needed for a networked application.

Table 4-2 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 path name 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, WLE 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 WLE 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. Refer to the section "Step 2: Create a UDL Entry" on page 16-10 for details.

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

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

   where `-z` *`config`* specifies the full path name for the device where the UDL should be created (that is, where the TLOG will reside), and `-b` *`blocks`* specifies the number of blocks to be allocated on the device. The value of *`config`* should match the value of the TLOGDEVICE parameter in the MACHINES section. If *`config`* is not specified, it defaults to the value of the variable FSCONFIG (which points to the application's databases).
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 path name of the network device. For the WLE system and BEA TUXEDO system version 6.4 or above, this option is not required. For earlier versions of the BEA TUXEDO system (v6.3 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, require 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 WLE 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 4-3 describes the tmboot options shown above.

Table 4-3 tmboot Options

Option	Meaning
-g *grpname*	Boot all TMS and application servers in groups using this *grpname* parameter.
-o *sequence*	Boot all servers in the order shown in their SEQUENCE parameter.
-s server-name	Boot individual servers.
-S	Boot all servers listed in the SERVERS section.
-A	Boot 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 leaves 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 provided for shutting down an application.

The rules for use of this command are very similar to those of tmboot(1).

Administrators face several problems when shutting down an application. The two most common situations are discussed in the last section of this chapter.

The tmshutdown(1) command is the inverse of the tmboot(1) command. It shuts down part or all of the WLE or BEA TUXEDO application.

When the entire application is shut down, tmshutdown(1) removes the IPC resources associated with the WLE 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 shuts the servers down 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 needed 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 WLE 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 WLE system. This section lists and discusses some of the common startup and shutdown problems.

Common Startup Problems

This section describes a few problems you may encounter when starting your first WLE application. Evidence that a problem exists comes 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 WLE 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.

Problems of this kind can be avoided if you check the syntax of the TLOG parameters in the MACHINES section of the UBBCONFIG file (see ubbconfig(5)).

Following are other reasons why the TLOG might not get created:

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

Server Not Built Correctly

Following are two reasons a server may not start correctly:

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

BUILDOBJSERVER FAILURE

An error in this area should be noticed before you attempt to boot a WLE 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 carries 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 DBMS; the information is stored in the WLE 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 Message Manual* 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 WLE 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
```

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 and/or the user ID are not the same on both machines
- Access (permissions) problems

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

Table 4-4 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.

Table 4-4 Possible Solutions to Propagation Failure

Problem	Solution
Group ID and/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.
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 4-5.

Table 4-5 Two Common Shutdown Problems and Their Solutions

Problem	Solution
Shutting down administrative servers before application servers	The WLE 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, the application servers must be shut down ahead of 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 WLE 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.

5 Distributing Applications

This chapter discusses the following topics:

- Why distribute an application?
- Using Factory-based Routing (WLE Servers)
- Using Data-dependent Routing (BEA TUXEDO Servers)
- Modifying and Creating the UBBCONFIG Sections for a Distributed Application
- Example of UBBCONFIG Sections in a Distributed Application
- Modifying the Domain Gateway Configuration File to Support Routing (BEA TUXEDO Servers)

Why distribute an application?

Distributing an application enables you to select which parts of an application should be grouped together logically and where these groups should run. You distribute an application by creating more than one entry in the `GROUPS` section of the `UBBCONFIG` file, and by dividing application resources or tasks among the groups. Creating groups of servers enables you to partition a very large application into its component business applications, and in turn each of these into logical components of manageable size and optimal location.

Benefits of a Distributed Application

- Scalability—The load an application can sustain can be increased by placing extra server processes in a group; adding machines to the application and redistributing the groups across the machines; replicating a group onto other machines within the application and using load balancing; segmenting a database and using data-dependent routing to reach the groups dealing with these separate database segments (the BEA TUXEDO system).
  With the WLE system, you can use factory-based routing to distribute processing of a particular CORBA interface across multiple server groups, and (if desired) across multiple machines. This feature allows you to distribute the processing load, which can prevent the processing bottlenecks that occur when concurrent, resource-intensive applications compete for the available CPU, memory, and disk I/O resources.
- Ease of development/maintainability—The separation of the business application logic into services or components that communicate through well-defined messages or interfaces allows both development and maintenance to be similarly separated and so simplified.
- Resilience—When multiple machines are in use and one fails, the remainder can continue operation. Similarly, when multiple server processes are within a group and one fails, the others are present to perform work. Finally, if a machine should break, but there are multiple machines within the application, these other machines can be used to perform the work of the application.
- Coordination of autonomous actions—If you have separate applications, you can coordinate autonomous actions among the applications. You can coordinate *autonomous actions* as a single logical *unit of work*. *Autonomous actions* are actions that involve multiple server groups and/or multiple resource manager interfaces.

Characteristics of Distributing an Application

A distributed application has the following characteristics:

- Enlarges the client and/or server model
- Establishes multiple server groups

- Enables transparent access to BEA TUXEDO services or WLE interfaces
- In the BEA TUXEDO system, allows data-dependent partitioning of data
- In the WLE system, allows partitioning of CORBA objects in multiple groups across multiple machines, or the WLE distribution of application factory interfaces and application interfaces
- Enables management of multiple resources
- Supports a networked model

Using Factory-based Routing (WLE Servers)

Factory-based routing is a WLE feature that you can use to distribute processing of CORBA interface invocations across specific server groups. Routing is done when a factory creates an object reference. The factory specifies field information in its call to the WLE TP Framework to create an object reference.

The TP Framework executes the routing algorithm based on the routing criteria that you define in the `ROUTING` section of an application's `UBBCONFIG` file. The resulting object reference has as its target an appropriate server group for the handling of method invocations on the object reference. Any server that implements the interface in that server group is eligible to activate the servant for the object reference.

The activation of CORBA objects can be distributed by server group based on a criteria defined by you, in cooperation with a system designer. Different implementations of CORBA interfaces can be supplied in different groups. This feature enables you to replicate the same CORBA interface across multiple server groups, based on group-specific differences that you define.

The factory-based routing criteria must be personally communicated to you by the system designer of the application. In the BEA TUXEDO system, an `FML` field used for a service invocation can be used for routing. This information can be discovered by you independently. For WLE client and server applications, because the routing is done at the factory level and not on a method invocation on the target CORBA object, there is no service request message data or associated buffer information available for routing.

Characteristics of Factory-based Routing

Factory-based routing has the following characteristics:

- You must meet with the system designer to understand the fields and values to be used as the basis for routing.
- The routing criteria identifier for a CORBA interface is specified in the `INTERFACES` section of the `UBBCONFIG` file.
- In the `GROUPS` section, you should define as many server groups as are required for distributing the system.
- The routing criteria is defined in the `ROUTING` section of the `UBBCONFIG` file.
- Factory-based routing is done once per CORBA object, when the object reference is created.
- An implementation of a particular CORBA interface can exist in more than one server process. (See the following section, “Example: Factory-based Routing”.)
- Multiple CORBA interfaces can reside in a single server group.
- Server processes in a particular server group do not have to all use the same CORBA interfaces.
- The factory object implementation can indirectly control the location of the created CORBA object by supplying application-specific routing information.
- Routing uses the Bulletin Board criteria and occurs in a server call.

Example: Factory-based Routing

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 WLE 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 WLE 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 following example:

```
*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 application 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 application, the type of routing is factory-based routing. Therefore, this parameter is defined as `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 application, 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.

The following example shows the ROUTING section of the UBBCONFIG file used in the Production sample application:

```
*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 application 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.

The following example 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 application packaged with the WLE software is configured to run entirely on one machine. However, you can easily configure this application to run on multiple machines.)

```
*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"
```

Using Data-dependent Routing (BEA TUXEDO Servers)

Data-dependent routing is a mechanism whereby a service request is routed by a client (or a server acting as a client) to a server within a specific group based on a data value contained within the buffer that is sent. Within the internal code of a service call, the BEA TUXEDO system chooses a destination server by comparing a data field with the routing criteria it finds in the Bulletin Board shared memory.

For any given service, a routing criteria identifier can be specified in the `SERVICES` section of the `UBBCONFIG` file. The routing criteria identifier, in particular, the mapping of data ranges to server groups, is specified in the `ROUTING` section.

Characteristics of Data-dependent Routing

Data-dependent routing has the following characteristics:

- The service request assigned to a server in the group is based on a data value.
- Routing uses the Bulletin Board criteria and occurs in a server call.
- The routing criteria identifier for a service is specified in the SERVICES section of the UBBCONFIG file.
- The routing criteria identifier is defined in the ROUTING section of the UBBCONFIG file.

Example: A Distributed Application

The following table illustrates how client requests are routed to servers. In this example, a banking application called bankapp uses data-dependent routing. For bankapp, there are three groups (BANKB1, BANKB2, and BANKB3), and two routing criteria (Account ID and Branch ID). The services WITHDRAW, DEPOSIT, and INQUIRY are routed using the Account_ID field; the services OPEN and CLOSE are routed using the Branch_ID field.

Server Group	Routing Criteria Used	For These Services
BANKB1	Account_ID: 10000 - 49999	WITHDRAW, DEPOSIT, and INQUIRY
	Branch_ID: 1 - 4	OPEN and CLOSE
BANKB2	Account_ID: 50000 - 79999	WITHDRAW, DEPOSIT, and INQUIRY
	Branch_ID: 5 - 7	OPEN and CLOSE
BANKB3	Account_ID: 80000 - 109999	WITHDRAW, DEPOSIT, and INQUIRY
	Branch_ID: 8 - 10	OPEN and CLOSE

Modifying and Creating the UBBCONFIG Sections for a Distributed Application

Data-dependent routing (BEA TUXEDO system) or factory-based routing (WLE system) is described in the UBBCONFIG file, as follows:

- The GROUPS section is populated with as many server groups as are required for distributing the system. This allows the system to route a request to a server in a specific group. These groups can all reside on the same site (SHM mode) or, if there is networking, the groups can reside on different sites (MP mode).
- For data-dependent routing in the BEA TUXEDO system, the SERVICES section must list the routing criteria for each service that uses the ROUTING parameter.

  Note: If a service has multiple entries, each with a different SRVGRP parameter, all such entries must set ROUTING the same way. Otherwise, routing cannot be done consistently for that service. A service can route only on one field, which must be the same for all the same services.
- For factory-based routing in the WLE system, the INTERFACES section must list the name of the routing criteria for each CORBA interface that uses the FACTORYROUTING parameter. This parameter is set to the name of a routing criteria defined in the ROUTING section.
- You must add a ROUTING section to the configuration file to show mappings between data ranges and groups. This allows the system to send the request to a server in a specific group. Each ROUTING section item contains an identifier that is used in the INTERFACES section (for the WLE system) or in the SERVICES section (for the BEA TUXEDO system).

Modifying the GROUPS Section

Parameters in the GROUPS section implement two important aspects of distributed transaction processing. They associate a group of servers with a particular LMID and a particular instance of a resource manager. In addition, by allowing a second LMID to

be associated with the server group, they name an alternate machine to which a group of servers can be migrated if the MIGRATE option is specified. Table 5-1 describes the parameters in the GROUPS section.

Table 5-1 Description of the GROUPS Section Parameters

Parameter	Meaning
LMID	LMID must be assigned in the MACHINES section. It indicates that this server group runs on this particular machine. A second LMID value can be specified (separated from the first by a comma) to name an alternate machine to which this server group can be migrated if the MIGRATE option has been specified. Servers in the group must specify RESTART=Y to migrate.
GRPNO	GRPNO is a required parameter that associates a numeric group number with this server group. The number must be greater than 0 and less than 30000. It must be unique among entries in the GROUPS section in this configuration file.
TMSNAME	Specifies which transaction management server (TMS) should be associated with this server group.
TMSCOUNT	An optional parameter that can be used to specify how many copies of TMSNAME should be started for this server group. The minimum value is 2. If not specified, the default is 3. All TMSNAME servers started for a server group are automatically set up in an MSSQ set.

Parameter	Meaning
OPENINFO	Specifies information needed to open a particular instance of a particular resource manager, or it indicates that such information is not required for this server group. When a resource manager is named in the OPENINFO parameter, information such as the name of the database and the access mode is included. The entire value string must be enclosed in double quotes and must not be more than 256 characters. The format of the OPENINFO string is dependent on the requirements of the vendor providing the underlying resource manager. The string required by the vendor must be prefixed with rm_name:, which is the published name of the vendor's transaction (XA) interface followed immediately by a colon (:). The OPENINFO parameter is ignored if TMSNAME is not set or is set to TMS. If TMSNAME is set but the OPENINFO string is set to the null string ("") or if this parameter does not appear on the entry, it means that a resource manager exists for the group but does not require any information for executing an open operation.
CLOSEINFO	Specifies information the resource manager needs when closing a database. The parameter can be omitted or the null string can be specified. The default is the null string.

Modifying the SERVICES Section

The SERVICES section contains parameters that control the way application services are handled. An entry line in this section is associated with a service by its identifier name. Because the same service can be link edited with more than one server, the SRVGRP parameter is provided to tie the parameters for an instance of a service to a particular group of servers. Three parameters in the SERVICES section are particularly related to DTP: ROUTING, AUTOTRAN, and TRANTIME. Table 5-2 describes the parameters in the SERVICES section.

Table 5-2 Description of the SERVICES Section Parameters

Parameter	Meaning
ROUTING	The ROUTING parameter in the SERVICES section points to an entry in the ROUTING section where data-dependent routing is specified for transactions that request this service.
AUTOTRAN	Setting the AUTOTRAN parameter to Y or N determines whether a transaction should be started automatically if a message received by this service is not already in transaction mode.The default is N. Use of the parameter should be coordinated with the programmers coding the services for your application.
TRANTIME	The TRANTIME parameter sets a timeout value in seconds for transactions automatically started in this service. The default is 30 seconds. The TRANTIME parameter is needed only if AUTOTRAN =Y, and not even then if the default is acceptable.

Sample SERVICES Section

The following listing shows a sample SERVICES section:

```
*SERVICES

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

Creating the ROUTING Section

For information about ROUTING parameters that support the BEA TUXEDO system data-dependent routing or the WLE factory-based routing, see Chapter 3, "Creating a Configuration File."

Example of UBBCONFIG Sections in a Distributed Application

The following UBBCONFIG file contains the GROUPS, SERVICES, and ROUTING sections of a configuration file to accomplish data-dependent routing in the BEA TUXEDO system.

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

Modifying the Domain Gateway Configuration File to Support Routing (BEA TUXEDO Servers)

This section is specific to the BEA TUXEDO system and explains how and why you need to modify the domain gateway configuration to support routing. For more information about the domain gateway configuration file, see Chapter 14, ‘‘Working with Multiple Domains (BEA TUXEDO System).’’

What is the Domains gateway configuration file?

All Domains gateway configuration information is stored in a binary file, the `BDMCONFIG` file. The `DMCONFIG` file (ASCII) is created and edited with any text editor. The compiled `BDMCONFIG` file can be updated while the system is running by using the `dmadmin`(1) command.

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

Note: For more information about the `DMCONFIG` file, refer to the `dmconfig`(5) reference page in the *BEA TUXEDO Reference Manual.*

Description of Parameters in the ROUTING Section of the DMCONFIG File

The DM_ROUTING section provides information for data-dependent routing of service requests using FML, VIEW, X_C_TYPE, and X_COMMON typed buffers. Lines within the DM_ROUTING section have the form CRITERION_NAME, where CRITERION_NAME is the (identifier) name of the routing entry specified in the SERVICES section. The CRITERION_NAME entry may contain no more than 15 characters. The following table describes the parameters in the DM_ROUTING section:

Parameter	Meaning
FIELD = identifier	Specifies the name of the routing field. It must contain 30 characters or fewer. This field is assumed to be a field name identified in an FML field table (for FML buffers) or an FML VIEW table (for VIEW, X_C_TYPE, or X_COMMON buffers). The FLDTBLDIR and FIELDTBLS environment variables are used to locate FML field tables; the VIEWDIR and VIEWFILES environment variables are used to locate FML VIEW tables. If a field in an FML32 buffer is used for routing, it must have a field number less than or equal to 8191.

Parameter	Meaning
RANGES ="range1:rdom1[,range2:rdom2 ...]"	Specifies the ranges and associated remote domain names (RDOM) for the routing field. The string must be enclosed in double quotes, with the format of a comma-separated ordered list of range/RDOM pairs. A range is either a single value (signed numeric value or character string in single quotes), or a range of the form *lower - upper* (where lower and upper are both signed numeric values or character strings in single quotes). The value of *lower* must be less than or equal to *upper*. A single quote embedded in a character string value (as in "O'Brien," for example), must be preceded by two backslashes ("O\\'Brien"). Use MIN 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. Use MAX 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 6 - MAX is all numbers greater than or equal to 6. The metacharacter * (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).
BUFTYPE = "type1[:subtype1[,subtype2 . . .]][;type2[:subtype3[, . . .]]] . . ."	A list of types and subtypes of data buffers for which this routing entry is valid. The types are restricted to FML, VIEW, X_C_TYPE, and X_COMMON. No subtype can be specified for type FML, and subtypes are required for the other types (* is not allowed). Duplicate type/subtype pairs cannot be specified for the same routing criteria name; more than one routing entry can have the same criteria name as long as the type/subtype pairs are unique. This parameter is required. If multiple buffer types are specified for a single routing entry, the data types of the routing field for each buffer type must be the same. (If the field value is not set (for FML buffers), or does not match any specific range, and a wildcard range has not been specified, an error is returned to the application process that requested the execution of the remote service.)

Routing Field Description

The routing field can be of any data type supported in FML or VIEW. 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(): an optional sign, then a string of digits optionally containing a decimal point, then an optional e or E followed by an optional sign or space, followed by an integer.
When a field value matches a range, the associated RDOM value specifies the remote domain to which the request should be routed. An RDOM value of * indicates that the request can go to any remote domain known by the gateway group. Within a range/RDOM pair, the range is separated from the RDOM by a : (colon).

Example of a Five-Site Domain Configuration Using Routing

The following configuration file defines a five-site domain configuration. Listing 5-1 shows four bank branch domains communicating with a Central Bank Branch. Three of the bank branches run within other BEA TUXEDO system domains. The fourth branch runs under the control of another TP domain, and OSI-TP is used in the communication with that domain. The example shows the BEA TUXEDO system Domains gateway configuration file from the Central Bank point of view. In the DM_TDOMAIN section, this example shows a mirrored gateway for b01.

Listing 5-1 A Five-Site Domains Configuration

```
# BEA TUXEDO DOMAIN CONFIGURATION FILE FOR THE CENTRAL BANK
#
#
*DM_LOCAL_DOMAINS
# <local domain name> <Gateway Group name> <domain type> <domain id> <log device>
#                     [<audit log>] [<blocktime>]
#                     [<log name>] [<log offset>] [<log size>]
#                     [<maxrdom>] [<maxrdtran>] [<maxtran>]
#                     [<maxdatalen>] [<security>]
#                     [<tuxconfig>] [<tuxoffset>]

#
```

```
#
DEFAULT: SECURITY = NONE
c01     GWGRP = bankg1
        TYPE = TDOMAIN
        DOMAINID = "BA.CENTRAL01"
        DMTLOGDEV = "/usr/apps/bank/DMTLOG"
        DMTLOGNAME = "DMTLG_C01"
c02     GWGRP = bankg2
        TYPE = OSITP
        DOMAINID = "BA.CENTRAL01"
        DMTLOGDEV = "/usr/apps/bank/DMTLOG"
        DMTLOGNAME = "DMTLG_C02"
        NWDEVICE = "OSITP"
        URCH = "ABCD"
#
*DM_REMOTE_DOMAINS
#<remote domain name>  <domain type> <domain id>
#
b01     TYPE = TDOMAIN
        DOMAINID = "BA.BANK01"
b02     TYPE = TDOMAIN
        DOMAINID = "BA.BANK02"
b03     TYPE = TDOMAIN
        DOMAINID = "BA.BANK03"
b04     TYPE = OSITP
        DOMAINID = "BA.BANK04"
        URCH = "ABCD"
#
*DM_TDOMAIN
#
#     <local or remote domainname> <network address> [nwdevice]
#
# Local network addresses
c01     NWADDR = "//newyork.acme.com:65432"     NWDEVICE ="/dev/tcp"
c02     NWADDR = "//192.76.7.47:65433"     NWDEVICE ="/dev/tcp"
# Remote network addresses: second b01 specifies a mirrored gateway
b01     NWADDR = "//192.11.109.5:1025" NWDEVICE = "/dev/tcp"
b01     NWADDR = "//194.12.110.5:1025" NWDEVICE = "/dev/tcp"
b02     NWADDR = "//dallas.acme.com:65432"  NWDEVICE = "/dev/tcp"
b03     NWADDR = "//192.11.109.156:4244" NWDEVICE = "/dev/tcp"
#
*DM_OSITP
#
#<local or remote domain name> <apt> <aeq>
#                              [<aet>] [<acn>] [<apid>] [<aeid>]
#                              [<profile>]
#
c02     APT = "BA.CENTRAL01"
        AEQ = "TUXEDO.R.4.2.1"
```

```
        AET = "{1.3.15.0.3},{1}"
        ACN = "XATMI"
b04     APT = "BA.BANK04"
        AEQ = "TUXEDO.R.4.2.1"
        AET = "{1.3.15.0.4},{1}"
        ACN = "XATMI"
*DM_LOCAL_SERVICES
#<service_name>  [<Local Domain name>] [<access control>] [<exported svcname>]
#                [<inbuftype>] [<outbuftype>]
#
open_act        ACL = branch
close_act       ACL = branch
credit
debit
balance
loan            LDOM = c02      ACL = loans
*DM_REMOTE_SERVICES
#<service_name>   [<Remote domain name>] [<local domain name>]
#                 [<remote svcname>] [<routing>] [<conv>]
#                 [<trantime>] [<inbuftype>] [<outbuftype>]
#
tlr_add   LDOM = c01  ROUTING = ACCOUNT
tlr_bal   LDOM = c01  ROUTING = ACCOUNT
tlr_add   RDOM = b04  LDOM = c02 RNAME ="TPSU002"
tlr_bal   RDOM = b04  LDOM = c02 RNAME ="TPSU003"
*DM_ROUTING
# <routing criteria>    <field> <typed buffer> <ranges>
#
ACCOUNT FIELD = branchid  BUFTYPE ="VIEW:account"
        RANGES ="MIN - 1000:b01, 1001-3000:b02, *:b03"
*DM_ACCESS_CONTROL
#<acl name>   <Remote domain list>
#
branch  ACLIST = b01, b02, b03
loans   ACLIST = b04
```

6 Building Networked Applications

This chapter covers the following topics:

- 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 WLE 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 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.

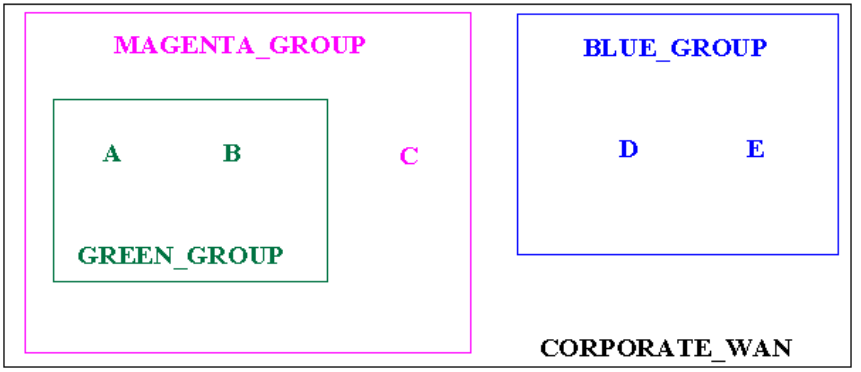
The four groups in the First State Bank example are as follows:

- `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"
```

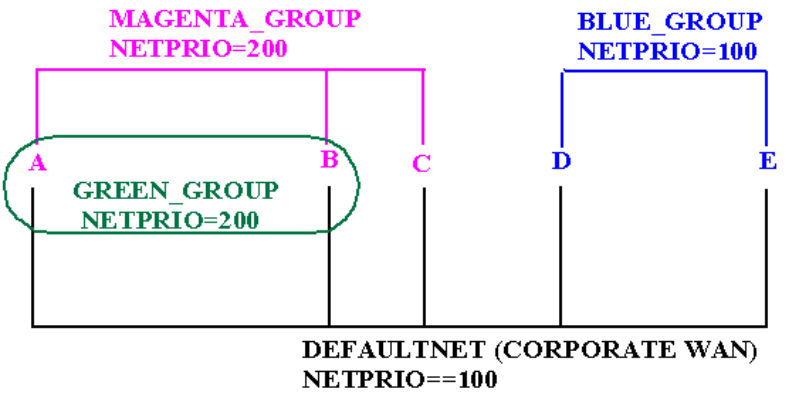
Assigning Priorities for Each Network Group

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 WLE 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

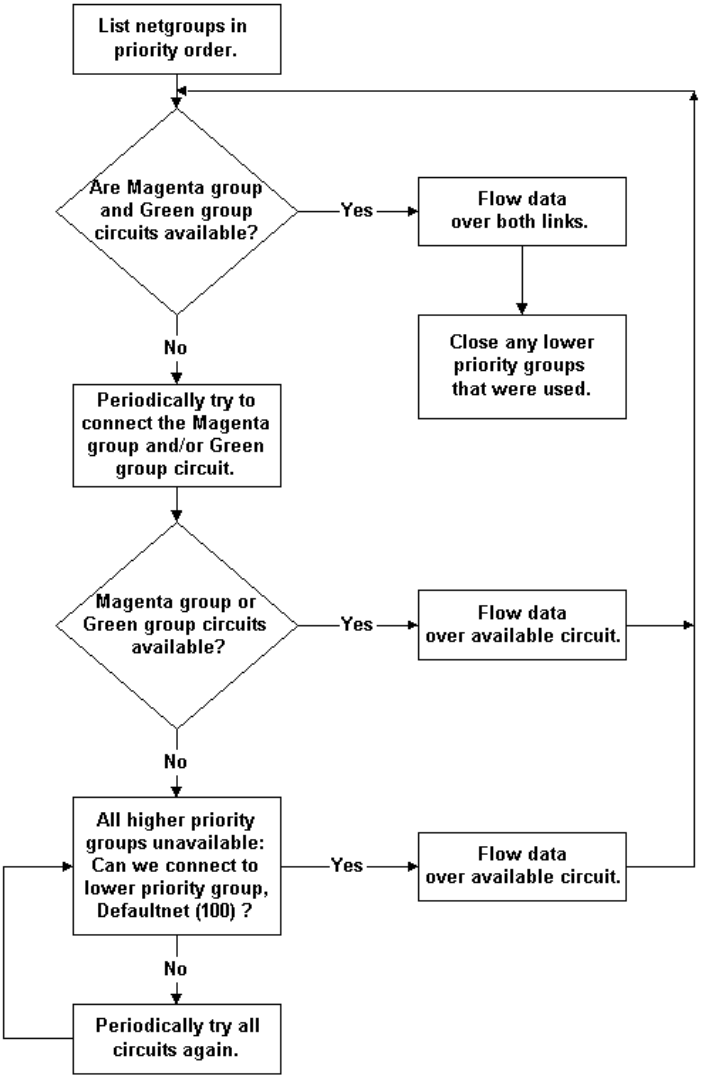


Figure 6-3 illustrates the flow of data when machine A attempts to contact machine B. First, the BRIDGE determines which network groups are common to both machine A and machine B. They are the MAGENTA_GROUP, the GREEN_GROUP, and the DEFAULTNET.

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 network. Set remote data compression to the lowest limit for WLE 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 network. 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 WLE 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 WLE 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: US/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 US/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 WLE-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 (40-bit capability only)

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 US/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 US/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*.

7 Configuring Transactions

This chapter discusses the following topics:

- Understanding Transactions
- Modifying the UBBCONFIG File to Accommodate Transactions
- Modifying the Domain Configuration File to Support Transactions (BEA TUXEDO Servers)
- Example: A Distributed Application Using Transactions

Understanding Transactions

Transactions greatly simplify the writing of distributed applications. They allow your application to cope more easily with a large set of problems that occur in a distributed environment, such as machine, program, or network failures. One of the greatest strengths of the WLE or BEA TUXEDO system is that the transaction semantic was built into the software and into the BEA TUXEDO ATMI. Global transactions were woven into the fabric of the system and into its communication APIs and protocols.

The ability to define a global transaction around communication calls makes it an indispensable tool for writing distributed applications. A global transaction allows not only the effects of your communications to be committed as a single unit, but it also gives you a simple, programmatic way to undo work if errors occur.

To illustrate the power of global transactions, consider the following example. A Place Order service performs two operations: it updates the Order database and enqueues the order to the shipping department. The business intends that both these actions occur together as a unit or that neither action should occur if one action fails.

To accomplish this, the client application invoking the Place Order service associates the call with a global transaction. You do this by using the ATMI begin-transaction function before issuing the service request, and issuing the commit-transaction function after it. Because the service is invoked as part of a global transaction, its work is done on its behalf. The server is propagated with the client's transaction when the Place Order service is invoked. Both the database access and the enqueue operation to the shipping application queue become part of the client's transaction.

Should either operation fail because of an application or system error, the work done in the transaction is rolled back to its state at the outset of the transaction. If both succeed, however, the client's call to commit the transaction causes the effects of the database update and the enqueued message to become permanent records of this transaction.

Modifying the UBBCONFIG File to Accommodate Transactions

You must modify the `RESOURCES`, `MACHINES`, `GROUPS`, and the `INTERFACES` or `SERVICES` sections of the application's `UBBCONFIG` file in the following way to accommodate transactions:

- In the `RESOURCES` section, specify the application-wide number of allowed transactions, and the value of the commit control flag.
- In the `MACHINES` section, create the `TLOG` information for each machine.
- In the `GROUPS` section, indicate information about each resource manager, and about the transaction manager server.
- In the `INTERFACES` section (WLE System) or the `SERVICES` section (BEA TUXEDO System), enable the automatic transaction option.

Specifying Application-wide Transactions in the RESOURCES Section

The following table provides a description of transaction-related parameters in the RESOURCES section of the configuration file.

Parameter	Meaning
MAXGTT	Limits the total number of global transaction identifiers (GTRIDs) allowed on one machine at one time. The maximum value allowed is 2048, minimum 0, and default 100. You can override this value on a per-machine basis in the MACHINES section. Entries remain in the table only while the global transaction is active, so this parameter has the effect of setting a limit on the number of simultaneous transactions.
CMTRET	Specifies the initial setting of the TP_COMMIT_CONTROL characteristic. The default is COMPLETE. Following are its two settings: ♦ LOGGED—The TP_COMMIT_CONTROL characteristic is set to TP_CMT_LOGGED, which means that tpcommit() returns when all the participants have successfully precommitted. ♦ COMPLETE—The TP_COMMIT_CONTROL characteristic is set to TP_CMT_COMPLETE, which means that tpcommit() will not return until all the participants have successfully committed. **Note:** You should consult with the RM vendors to determine the appropriate setting. If any RM in the application uses the *late commit* implementation of the XA standard, the setting should be COMPLETE. If all the resource managers use the *early commit* implementation, the setting should be LOGGED for performance reasons. (You can override this setting with tpscmt().)

Creating a Transaction Log (TLOG)

This section discusses creating a TLOG.

Creating the UDL

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

To create the UDL, enter a command using the following format, before the application has been booted:

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

In the preceding format statement, specify in the `-z config` argument the full path name for the device where you should create the UDL. Specify in the `-b blocks` argument the number of blocks to be allocated on the device. *config* should match the value of the TLOGDEVICE parameter in the MACHINES section of the UBBCONFIG file.

Note: In general, the value that you supply for blocks should not be less than the value for TLOGSIZE. For example, if TLOGSIZE is specified as 200 blocks, specifying `-b 500` would not cause a degradation.

For more information about storing the TLOG, see the *Installation Guide*.

Defining Transaction-related Parameters in the MACHINES Section

You can define a global transaction log (TLOG) using several parameters in the MACHINES section of the UBBCONFIG file. You must manually create the device list entry for the TLOGDEVICE on each machine where a TLOG is needed. You can do this either before or after TUXCONFIG has been loaded, but it must be done before the system is booted.

Note: If you are not using transactions, the TLOG parameters are not required.

The following table provides a description of transaction-related parameters in the MACHINES section of the configuration file.

Parameter	Meaning
TLOGNAME	The name of the DTP transaction log for this machine.
TLOGDEVICE	Specifies the WLE or BEA TUXEDO file system that contains the DTP transaction log (TLOG) for this machine. If this parameter is not specified, the machine is assumed not to have a TLOG. The maximum string value length is 64 characters.
TLOGSIZE	The size of the TLOG file in physical pages. Its value must be between 1 and 2048, and its default is 100. The value should be large enough to hold the number of outstanding transactions on the machine at a given time. One transaction is logged per page. The default should be enough for most applications.
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 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, you can use TLOGOFFSET to indicate a starting address relative to the address of the device.

Creating the Domains Transaction Log (BEA TUXEDO Servers)

This section is specific to the BEA TUXEDO system.

You can create the Domains transaction log before starting the Domains gateway group by using dmadmin(1) crdmlog (crdlog) -d *local_domain_name*. Create the Domains transaction log for the named local domain on the current machine (that is, the machine where dmadmin is running). The command uses the parameters specified in the DMCONFIG file. This command fails if the named local domain is active on the current machine or if the log already exists. If the transaction log has not been created, the Domains gateway group creates the log when Domains gateway group starts.

Defining Each Resource Manager (RM) and the Transaction Manager Server in the GROUPS Section

Additions to the GROUPS section fall into two categories:

- Defining the transaction manager servers that perform most of the work that controls global transactions. The TMSNAME parameter is the name of the server executable; TMSCOUNT is the number of such servers to boot (minimum 2, maximum 10, default 3).
  A null transactional manager server does not communicate with any resource manager. It is used to exercise an application's use of the transactional primitives before actually testing the application in a recoverable, *real* environment. This server is named TMS and it simply begins, commits, or terminates without talking to any RM.
- Defining opening and closing information for each resource manager. OPENINFO is a string with information used to open a resource manager, and CLOSEINFO is used to close a resource manager.

Sample of the GROUPS Section

The following is an example from the GROUPS section in the banking application, called bankapp.

```
BANKB1 GRPNO=1 TMSNAME=TMS_SQL TMSCOUNT=2
OPENINFO="TUXEDO/SQL:<APPDIR>/bankdl1:bankdb:readwrite"
```

Description of Transaction Values in the Sample GROUPS Section

The following table describes the transaction values shown in the sample GROUPS section:

Transaction Value	Meaning
BANKB1 GRPNO=1 TMSNAME=TMS_SQL\ TMSCOUNT=2	Contains the name of the transaction manager server (TMS_SQL), and the number (2) of these servers to be booted in the group BANKB1
TUXEDO/SQL	Published name of the resource manager

Transaction Value	Meaning
<APPDIR>/bankdl1	Includes a device name
bankdb	Database name
readwrite	Access mode

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

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

Parameter	Characteristics
TMSNAME	Name of the transaction manager server executable. Required parameter for transactional configurations. TMS is a null transactional manager server.
TMSCOUNT	Number of transaction manager servers (must be between 2 and 10). Default is 3.
OPENINFO, CLOSEINFO	Represents information to open or close a resource manager. Content depends on the specific resource manager. Starts with the name of the resource manager. Omission means the RM needs no information to open.

Enabling an Interface to Begin a Transaction in the INTERFACES Section (WLE Servers)

The INTERFACES section has been added to the UBBCONFIG file to support WLE CORBA interfaces. There are two transaction-related additions in the INTERFACES section:

- 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 depends on the transactional policy specified by the developer in the Implementation Configuration File (ICF), or the 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 affects the behavior of the application is if the developer specified a transaction policy of optional.

  Note: To work properly, this feature depends on personal communication between the system designer and the administrator. If the administrator sets this value to Y without prior knowledge of the transaction policy defined by the developer in the interface's ICF or XML file, the actual run time effect of the parameter might be unknown.

- If AUTOTRAN is set to Y, you must set the TRANTIME parameter, which is the transaction timeout in seconds for the transactions to be created. 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 zero implies there is no timeout for the transaction. (The default is 30 seconds.)

Characteristics of the AUTOTRAN and TRANTIME Parameters

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

Parameter	Characteristics
AUTOTRAN	Makes an interface the initiator of a transaction. To work properly, it is dependent on personal communication between the system designer and the system administrator. If the administrator sets this value to Y without prior knowledge of the ICF or XML transaction policy set by the developer, the actual run-time effort of the parameter might be unknown. The only time this value affects the behavior of the application is if the developer specified a transaction policy of optional. If a transaction already exists, a new one is not started. Default is N.
TRANTIME	Represents the timeout for the AUTOTRAN transactions. Valid values are between 0 and 2^{31} - 1, inclusive. 0 represents no timeout. Default is 30 seconds.

Enabling a Service to Begin a Transaction in the SERVICES Section (BEA TUXEDO Servers)

The following are three transaction-related additions in the SERVICES section:

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

  Note: Generally, clients are the best initiators of transactions because a service has the potential of participating in a larger transaction.

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

Characteristics of the AUTOTRAN, TRANTIME, and ROUTING Parameters

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

Parameter	Characteristics
AUTOTRAN	Makes a service the initiator of a transaction. Relieves the client of the transactional burden. If a transaction already exists, a new one is not started. Default is N.
TRANTIME	Represents the timeout for the AUTOTRAN transactions. Valid values are between 0 and 2^{31} - 1, inclusive. 0 represents no timeout. Default is 30 seconds.
ROUTING	Points to an entry in the ROUTING section where data-dependent routing is specified for transactions that request this service.

Modifying the Domain Configuration File to Support Transactions (BEA TUXEDO Servers)

This section is specific to BEA TUXEDO servers.

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

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

The DM_LOCAL_DOMAINS section of the Domains configuration file identifies local domains and their associated gateway groups. This section must have an entry for each gateway group (Local Domain). Each entry specifies the parameters required for the Domains gateway processes running in that group.

The following table provides a description of the five transaction-related parameters in this section: DMTLOGDEV, DMTLOGNAME, DMTLOGSIZE, MAXRDTRAN, and MAXTRAN.

Parameter	Characteristics
DMTLOGDEV	Specifies the BEA TUXEDO file system that contains the Domains transaction log (DMTLOG) for this machine. The DMTLOG is stored as a BEA TUXEDO VTOC table on the device. If this parameter is not specified, the Domains gateway group is not allowed to process requests in transaction mode. Local domains running on the same machine can share the same DMTLOGDEV file system, but each local domain must have its own log (a table in the DMTLOGDEV) named as specified by the DMTLOGNAME keyword.

Parameter	Characteristics
DMTLOGNAME	Specifies the name of the Domains transaction log for this domain. This name must be unique when the same DMTLOGDEV is used for several local domains. If a value is not specified, the value defaults to the string DMTLOG. The name must contain 30 characters or less.
DMTLOGSIZE	Specifies the numeric size of the Domains transaction log for this machine (in pages). It must be greater than zero and less than the amount of available space on the BEA TUXEDO file system. If a value is not specified, the value defaults to 100 pages. **Note:** The number of domains in a transaction determine the number of pages you must specify in the DMTLOGSIZE parameter. One transaction does not necessarily equal one log page.
MAXRDTRAN	Specifies the maximum number of domains that can be involved in a transaction. It must be greater than zero and less than 32,768. If a value is not specified, the value defaults to 16.
MAXTRAN	Specifies the maximum number of simultaneous global transactions allowed on this local domain. It must be greater than or equal to zero, and less than or equal to the MAXGTT parameter specified in the TUXCONFIG file. If not specified, the default is the value of MAXGTT.

Characteristics of the AUTOTRAN and TRANTIME Parameters

The DM_REMOTE_SERVICES section of the Domains configuration file identifies information on services *imported* and available on remote domains. Remote services are associated with a particular remote domain.

The following table describes the two transaction-related parameters in this section: AUTOTRAN and TRANTIME.

Parameter	Characteristics
AUTOTRAN	Used by gateways to automatically start/terminate transactions for remote services. This capability is required if you want to enforce reliable network communication with remote services. You specify this capability by setting the AUTOTRAN parameter to Y in the corresponding remote service definition.
TRANTIME	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 zero, and less than 2147483648. The default is 30 seconds. A value of zero implies the maximum timeout value for the machine.

Example: A Distributed Application Using Transactions

The following configuration file shows bankapp as an application that is distributed over three sites and that uses transactions. The application includes the following:

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

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

The RESOURCES Section

The RESOURCES section shown in Listing 7-1 specifies the following parameters:

- MAXSERVERS, MAXSERVICES, and MAXGTT are less than the defaults. This makes the Bulletin Board smaller.
- MASTER is SITE3 and the backup master is SITE1.
- MODEL is set to MP and OPTIONS is set to LAN, MIGRATE. This allows a networked configuration with migration.
- BBLQUERY is set to 180 and SCANUNIT is set to 10. This means that DBBL checks of the remote BBLs are done every 1800 seconds (one half hour).

Listing 7-1 Sample RESOURCES Section

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

The MACHINES Section

The MACHINES section shown in Listing 7-2 specifies the following parameters:

- TLOGDEVICE and TLOGNAME are specified, which indicate that transactions will be done.
- The TYPE parameters are all different, which indicates that encode/decode will be done on all messages sent between machines.

Listing 7-2 Sample MACHINES Section

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

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

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

The GROUPS and NETWORK Sections

The GROUPS and NETWORK sections shown in Listing 7-3 specify the following parameters:

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

Listing 7-3 Sample GROUPS and NETWORK Sections

```
*GROUPS
DEFAULT:         TMSNAME=TMS_SQL       TMSCOUNT=2
BANKB1           LMID=SITE1            GRPNO=1
 OPENINFO="TUXEDO/SQL:/usr/home/bankdl1:bankdb:readwrite"
BANKB2           LMID=SITE2            GRPNO=2
 OPENINFO="TUXEDO/SQL:/usr/home/bankdl2:bankdb:readwrite"
BANKB3            LMID=SITE3            GRPNO=3
 OPENINFO="TUXEDO/SQL:/usr/home/bankdl3:bankdb:readwrite"

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

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

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

The SERVERS, SERVICES, and ROUTING Sections

The SERVERS, SERVICES, and ROUTING sections shown in Listing 7-4 specify the following parameters:

- The TLR servers have a -T number passed to their *tpsrvrinit()* functions.
- All requests for the services are routed on the ACCOUNT_ID field.
- None of the services will be performed in AUTOTRAN mode.

Listing 7-4 Sample SERVERS, SERVICES, and ROUTING Sections

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

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

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

8 Managing Interface Repositories (WLE System)

This chapter, which is specific to WLE systems, describes the following topics:

- 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 WLE domain. Administration of the Interface Repository is done using tools specific to WLE 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 *Java Programming Reference* or the *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 WLE domain:

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

Prerequisites

Before executing a WLE 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 "idl2ir" on page 22-2.

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 "ir2idl" on page 22-4.

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 “irdel” on page 22-5.

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 WLE 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 WLE applications is critical. A WLE application will not boot if the order is changed. For more information, see the section “Required Order in Which to Boot Servers (WLE Servers)” 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 "TMIFRSVR" on page 22-16. 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 3-33 for details about parameters such as `SRVGRP, SRVID, RESTART, MAXGEN`, and `GRACE`.

9 Configuring Multiple Domains (WLE System)

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.

Benefits of Multiple Domains

In the versions 4.0 and 4.1 releases of the WLE 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 WLE version 4.0 and 4.1 systems consisted of only one “local domain.”

Since WLE 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).

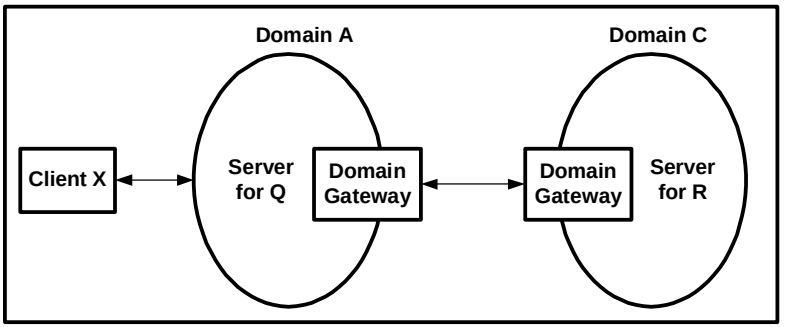
Eventually, an enterprise wants the different applications in those domains to be able to cooperate. It is often impossible to expand a single domain to encompass the enterprise. However, the size of an expanded domain in terms of the number of machines and services would be impractical. Since a single 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 “WLE 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.

- Invocation of an object in a remote domain

  The whole point of the "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 for each domain

and on the coordination of two of those configuration 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.

- 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.

Configuring Multiple Domains

You use 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
```

This is the name that will be referenced in a DMCONFIG file. In that file, a WLE domain name will be referenced as:

```
"//<domain-name>"
```

The quotes are part of the reference and the slashes (//) mean that the name applies to WLE 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. The service name is GWTDOMAIN and must be associated, like every other group, with a server group and a server ID. The GROUPS section might contain, for example:

```
LGWGRP      GRPNO=4      LMID=LDOM
```

In this line, 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 you specify it in the SERVERS section associated with the server group name chosen. For example:

```
GWTDOMAIN SRVGRP=LGWGRP SRVID=1
```

This tells the 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 WLE domains.

The sections below concentrate on the information that applies to WLE 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 WLE. For WLE, the "service" name is the name of another WLE domain that can service WLE requests.

The DMCONFIG file consists of up to eight parts, but one part, DM_ROUTING, does not apply to WLE domains. The other seven parts refer to WLE 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

```
#
# WLE 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 UBBCONFIG-defined gateway group 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 (it is not the domain ID) and it is not the same name as the gateway group. It 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.

`DOMAINID = string`

This 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 DOMAINID found in 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).

`TYPE = TDOMAIN`

This parameter is required to specify the use of domains for WLE.

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 *BEA TUXEDO Reference*.

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 (it is not the domain ID), and it is not the same name as the gateway group. It 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 WLE.

`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. It has no required relationship to the DOMAINID found in 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 WLE 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.*

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 in a `UBBCONFIG` file `RESOURCES` section 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 WLE 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 WLE 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 a UBBCONFIG file RESOURCES section as DOMAIN_NAME. Most likely this is the name of the domain in which the gateway resides, meaning that this (local) domain accepts WLE 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 Factory Finder domain configuration file (`factory_finder.ini`). This is an ASCII file that can be created and updated using a text editor.

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"
    RNAME=”remote_factory_id.factory_kind”
    DOMAINID=”domain_id”
    ...
[*DM_LOCAL_FACTORIES]
  ["factory_id.factory_kind"]
  ...
```

The Master NameManager reads the FactoryFinder domain configuration 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 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 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, one that specifies those factories that the local domain can find in a remote domain (DM_REMOTE_FACTORIES), and another that specifies which of the factories in the local domain are available to other domains (DM_LOCAL_FACTORIES). Either section can be absent or contain nothing.

For more information about the factory_finder.ini file, see "factory_finder.ini" on page 22-17.

DM_REMOTE_FACTORIES

The DM_REMOTE_FACTORIES section provides information about the factories that are imported and are available in remote domains. It lists identifiers for factories in a remote domain that can be used by applications in the local domain. The identifier, under which the factory 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 in domain "Norwest" would be specified as:

```
*DM_REMOTE_FACTORIES
 "Teller.FactoryInterface"
    RNAME="BankTeller.FactoryInterface"
    DOMAINID="Norwest"
```

If the RNAME is not specific, the factory_kind must be specified in the factory name; 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".

An entry for an imported factory must specify the identity of the factory in the remote domain using the keyword RNAME. The identifier of the remote factory must be enclosed within double quotation marks.

Because it is likely that the identities of factories in a remote domain may collide with the identity of factories in the local domain, the syntax supports the ability to specify alternative identities or "aliases" in the local domain for a factory. Specifying an alias can be accomplished by providing multiple entries for factories that have the same DOMAINID and RNAME values.

For example, the entry for a factory object to be registered by the TP Framework with the identifier `Teller` in domain `"Norwest"`, but made available in local domain as `"TellerFactory"`, would be specified as:

```
*DM_REMOTE_FACTORIES
  "Teller.FactoryInterface" or Teller
     DOMAINID="Norwest"
     RNAME="TellerFactory.FactoryInterface"
```

DM_LOCAL_FACTORIES

The `DM_LOCAL_FACTORIES` section provides information about the factories that are exported by the local domain. Specification of factories can also be used to restrict access to local factories from remote domains; that is, only factories specified are available to the remote domain.

This section contains the list of identifiers for factories that can be used by applications in a remote domain. The identifier, under which the factory 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. An entry that does not contain a `factory_kind` value is not defaulted with a value of `"FactoryInterface"`. This allows for future expansion when a Naming Service is available.

If the `DM_LOCAL_FACTORIES` section does not exist in a `factory_finder.ini`, or exists, but is empty, all factories in the local domain are available to remote domains. This allows administrators an easy means to provide this indication without having to provide an entry for every factory object 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 factories in the local domain are available to remote domains.

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
```

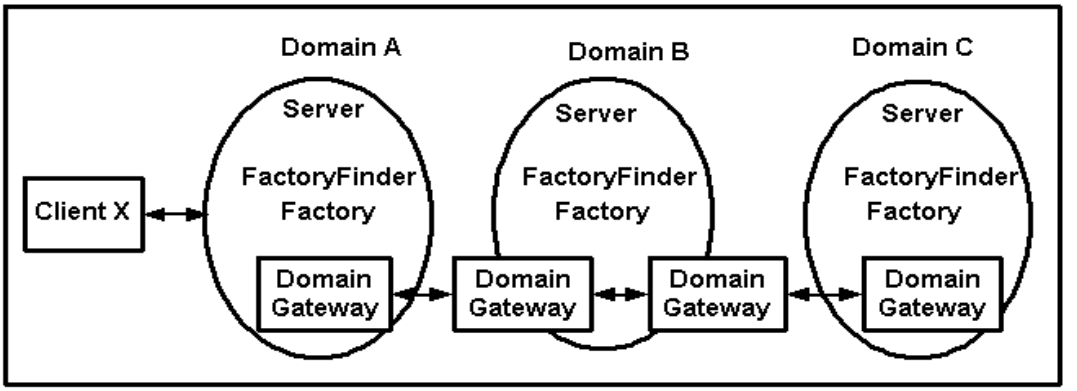
Domain A has a gateway process, GAB (the Gateway from A to B), that connects to domain B. The domain A DMCONFIG file states that GAB acts as a gateway to two domains, domains B and C. The DMCONFIG file for domain C has a similar

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, WLE knows that it can send the request to gateway GAB. The WLE 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

The listings in this section show the UBBCONFIG files for three directly connected domains: Here, There, and Yonder.

Listing 9-2 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 9-3 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 9-4 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 9-5 shows the DMCONFIG file for three directly connected domains: Here, There, and Yonder.

Listing 9-5 DMCONFIG File for 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
```

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 9-6 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 9-7 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"
```

10 Working with Multiple Domains (BEA TUXEDO System)

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 (WLE System)."

This chapter discusses the following topics:

- 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.

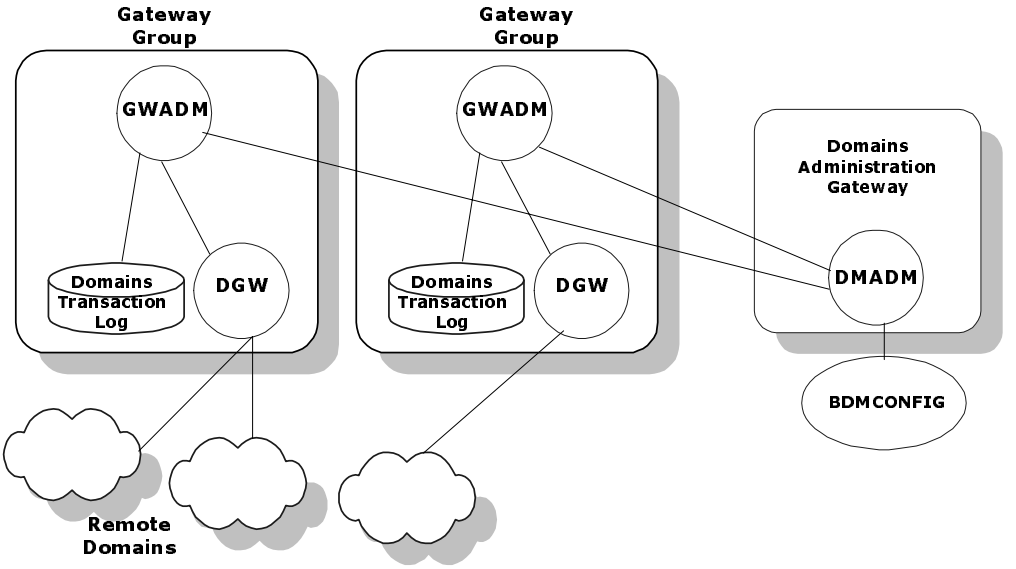
What is the domains gateway configuration file?

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

The following table describes the sections of the `DMCONFIG` file.

Section of the DMCONFIG File	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.

Section of the DMCONFIG File	Purpose
DM_REMOTE_SERVICES	Describes the set of services provided by remote domains that are accessible from this domain.
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 NT, no variable for shared libraries is required.

Example

```
$ 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
```

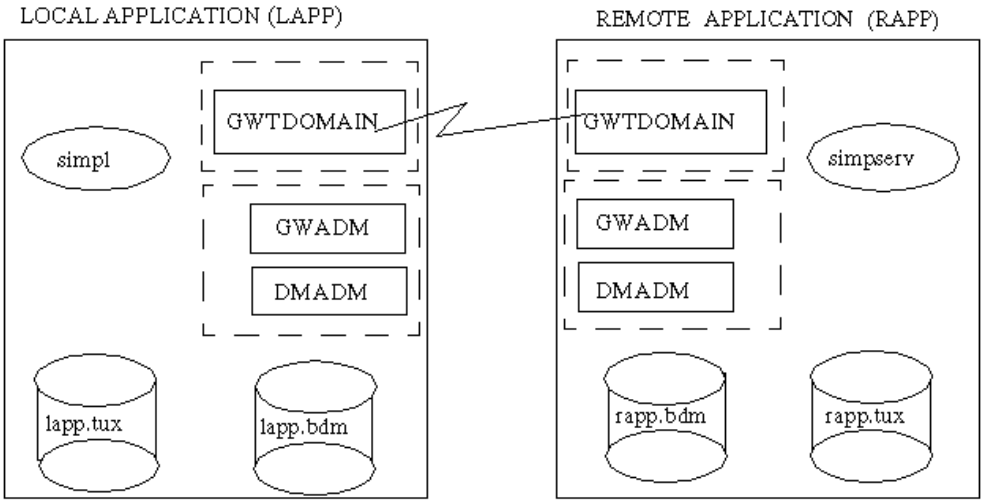
Once you create both the local and remote domains, you can then boot the application using `tmboot`(1). The order in which the two domains are booted does not matter. Monitor the applications with `dmadmin`(1).

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 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`.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. The following chart describes these two fields.

Domain ACL Field	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. The following table describes these fields.

Routing Table 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 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.

Routing Table 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 System)

This chapter is specific to the BEA TUXEDO system. If you are using the WLE system and you need to configure remote clients or the Internet Inter-ORB Protocol (IIOP) Listener/Handler, see Chapter 12, "Managing Remote Client Applications (WLE Systems)" for more information.

This chapter discusses the following BEA TUXEDO topics:

- 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

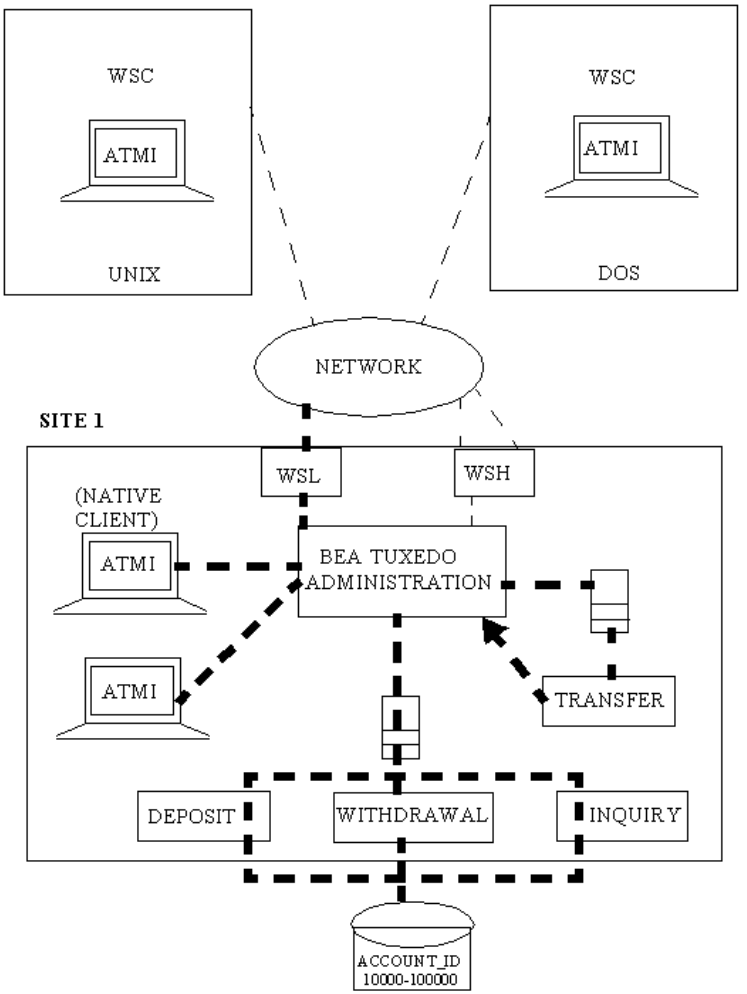
Figure 11-1 shows an example of an application with two WSCs connected. The client on the left is running on a UNIX system workstation, while the client on the right is running on an MS-DOS workstation. Both WSCs are communicating with the application through the WSH process. Initially, both joined by 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).

12 Managing Remote Client Applications (WLE 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 WLE servers.

This chapter discusses the following topics:

- 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 Server Handler. This is a client process running on an application site that acts as a surrogate on behalf of the remote client.

ISL

IIOP Server 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 WLE domain. A server is built with the WLE `buildobjserver` command. Servers implement WLE functionality, such as security, transactions, and object state management.

Note: In WLE V4.0, servers could only make invocations on other servers inside the WLE domain. In WLE V4.2, the servers can make invocations on any server, inside or outside a WLE domain.

native client

A client located within a WLE domain, using the WLE ORB to make invocations on objects either inside or outside the WLE domain. A native client's host contains the WLE administrative and infrastructure components, such as tmadmin, FactoryFinder, and ISL/ISH . Native clients use the environmental objects to access WLE objects. You build native clients with either the `buildobjclient` command or Java client commands.

Note: In WLE V4.0, a native client could not make invocations on objects outside the WLE domain.

remote client

A client not located within a WLE domain. A remote client can use the WLE ORB to make invocations on objects either inside or outside the WLE domain. A remote client's host does not contain WLE administrative and infrastructure components, such as tmadmin, FactoryFinder, and ISL/ISH; it does contain supporting software (the WLE ORB) that allows remote clients to invoke objects. Remote clients use the environmental objects to access WLE 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) execute code acting as the starter for some business actions and 2) execute method code for invocations on objects. A joint client/server located within a WLE domain. You build C++ native joint client/servers with the `buildobjclient` command. Java native joint client servers are not supported.

Note: In WLE V4.0 and V4.1, 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 WLE administrative and infrastructure components, such as tmadmin, FactoryFinder, and ISL/ISH (hence none of WLE's scalability and reliability attributes), it does not use the WLE 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 WLE domain. The joint client/server does not use the WLE 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 WLE V4.0, 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 WLE

administrative and infrastructure components, such as tmadmin, FactoryFinder, and ISL/ISH (hence, none of WLE's scalability and reliability attributes).

WLE object

A CORBA object that is implemented using TP Framework and that implements security, transactions, and object state management. WLE objects are implemented in servers; that is, it is in a WLE domain and uses the WLE 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 WLE domain.

Note: In WLE V4.0 and V4.1, callback objects existed but were not named as such; they could have implementations only in the WLE 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 WLE client applications that you deployed on systems that do not have the full WLE 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

A client process can be running UNIX or Microsoft Windows. The client has access to the CORBA ORB interface. 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:

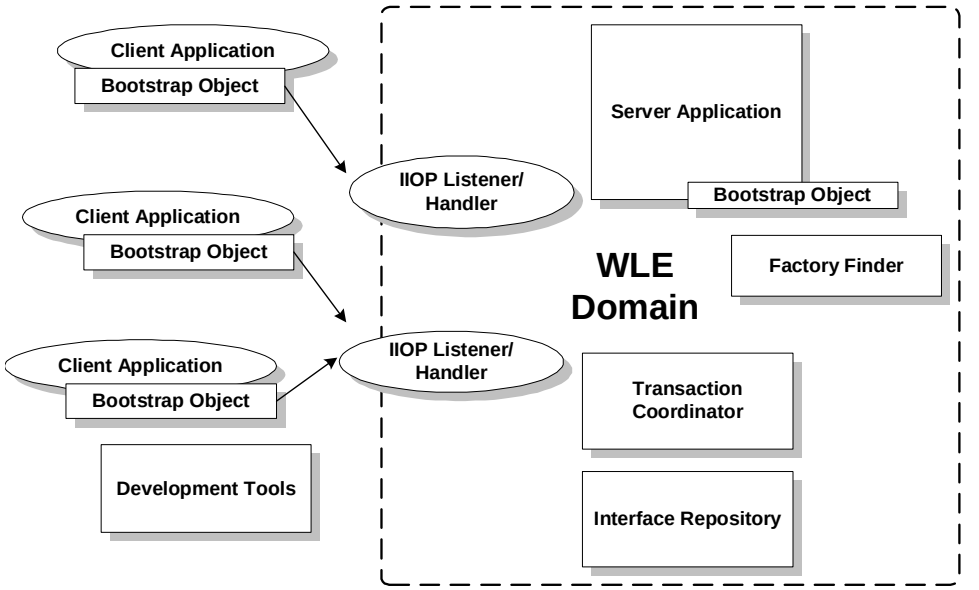
- Invoke methods on remote CORBA objects
- Begin, roll back, or commit transactions
- Be required to pass application security

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 WLE 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 WLE 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 WLE 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 CLOPT command line options, see "ISL" on page 22-6.

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 WLE 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 the version4.0 and version 4.1 releases of the WLE 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:

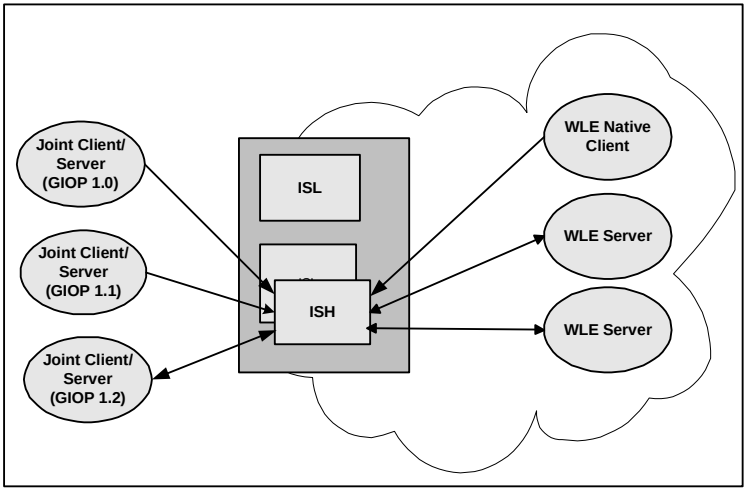
- Bidirectional—Outbound IIOP reusing the same connection (supported only for WLE V4.2 C++ GIOP 1.2 servers, clients, and joint client/servers)
- Asymmetric—Outbound IIOP via a second connection (supported for GIOP 1.0, GIOP 1.1, and GIOP 1.2 servers, clients, and joint client/server applications)
- Dual-paired connection—Outbound IIOP (supported for GIOP 1.0, GIOP 1.1, and GIOP 1.2 servers, clients, and joint client/server applications)

  Note: GIOP 1.2 is supported only by WLE V4.2 C++ clients, servers, and joint client/servers; GIOP 1.2 is not supported by WLE V4.0 or V4.1 clients and servers. Java clients, servers, and joint client/servers only support GIOP 1.0. WLE V4.0 and V4.1 C++ clients and servers support GIOP 1.0 and 1.1.

Bi-directional 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 WLE 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



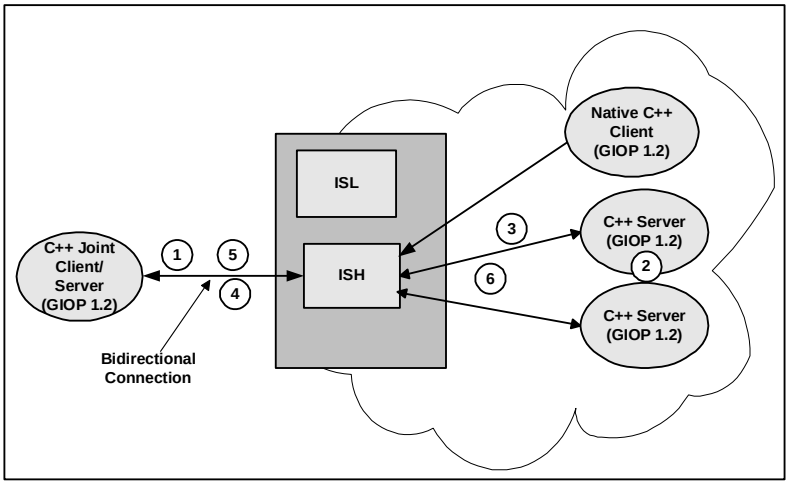
Bidirectional Outbound IIOP

With bidirectional outbound IIOP, the following operations are executed (see Figure 12-3):

1. A client creates an object reference and invokes on a WLE server. The client ORB identifies the connection as being bidirectional using the service context. The service context travels with the message to the WLE server.
2. When unmarshaling the object reference, the WLE 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 WLE servers.
3. At some point in time, a WLE 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 WLE 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 WLE 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.

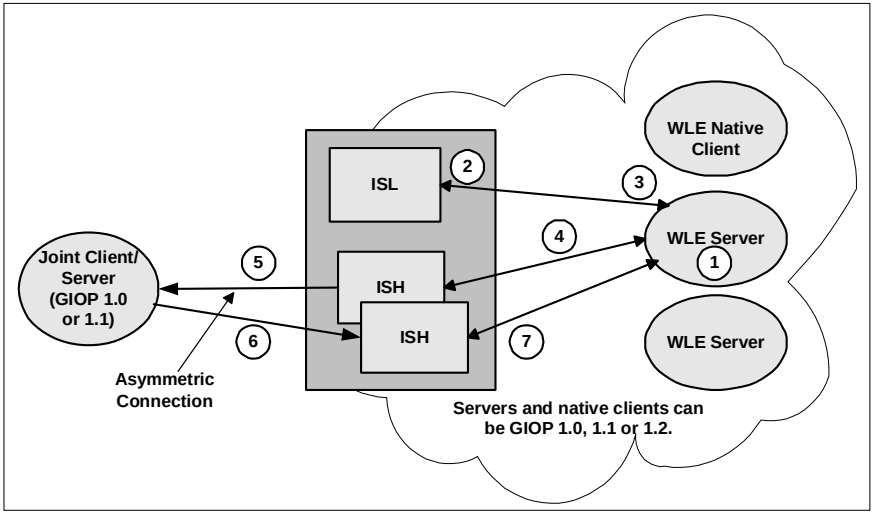
3. The ISL selects an ISH to handle the outbound invoke and returns the ISH information to the WLE server.

4. The WLE 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 WLE server.

Figure 12-4 Asymmetric Outbound IIOP



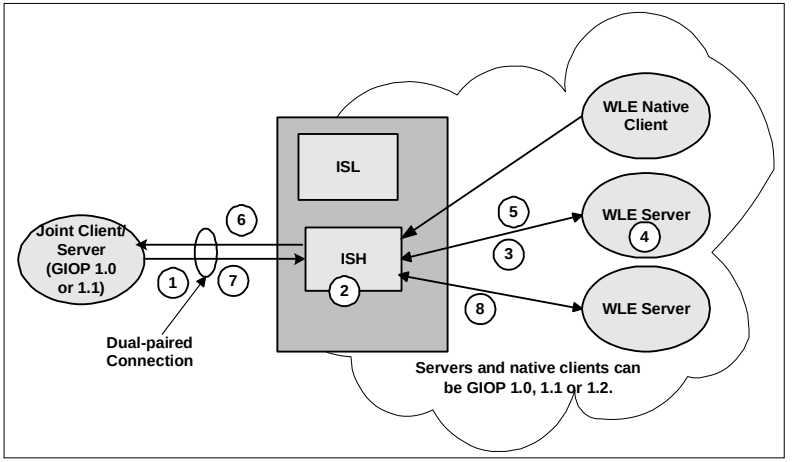
Dual-paired Connection Outbound IIOP

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 WLE 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 WLE server.
4. When unmarshaling the object reference, the WLE 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 WLE servers.

5. At some point in time, a WLE 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 WLE server. If the client disconnects from the ISH, the second connection is also disconnected.

Figure 12-5 Dual-paired Connections Outbound IIOP



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 non-WLE ORB or from a remote WLE joint client/server.

Types of Object References

There are two kinds of remote object references:

- Object references created by WLE remote joint client/servers outside of the WLE 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 commandline interface for booting the ISL process(es). New command-line options to configure the outbound IIOP processing were added to the ISL command in this release of the WLE software. These options enable support for asymmetric IIOP to object references not located in clients connected to an ISH.

The ISL command syntax listed below shows the new options for outbound IIOP support:

```
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]
#NEW options for outbound IIOP
           [-O]
           [-o outbound-max-connections]
           [-s Server-timeout]
           [-u out-mpx-users] "
```

For a detailed description of the `CLOPT` command-line options, see “ISL” on page 22-6.

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.

The following topics are presented:

- Terms and Definitions
- Overview of the BEA TUXEDO Queued Message Facility
- Administrative Tasks
- Setting the QMCONFIG Environment Variable
- Using qmadmin, the /Q Administrative Interface
- Creating an Application Queue Space and Queues
- Modifying the Configuration File

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 Facility allows 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)

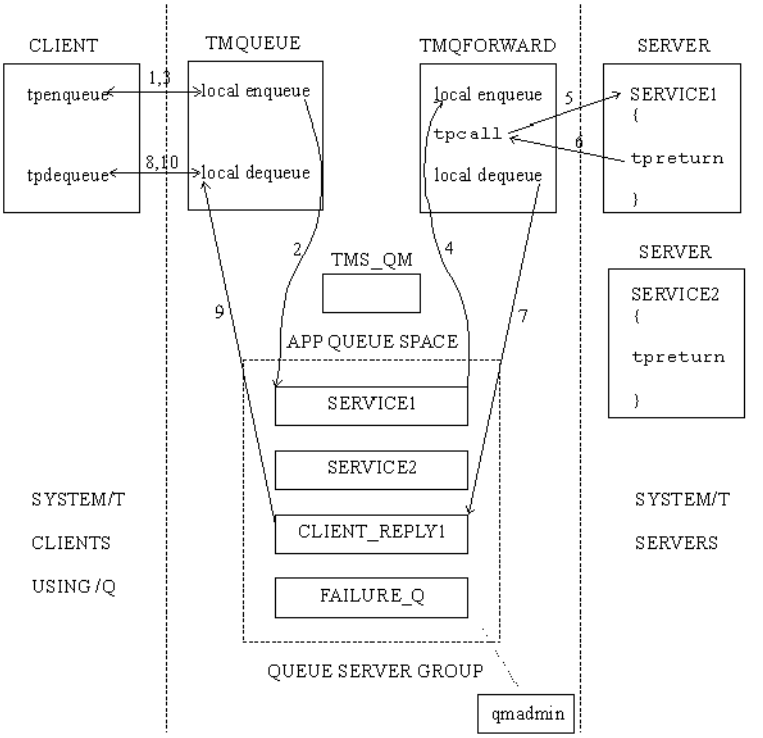
Also, the administrator must create a queue space using the queue administration program, `qmadmin`(1). The queue space contains a collection of queues. In Figure 13-1, for example, four queues are present within the queue space named `APP`. There is a one-to-one mapping of queue space to queue server 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 path name 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 Facility application 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 Applications

This chapter discusses the levels of security that are available to WLE and BEA TUXEDO system applications, and describes how to implement the level of security your designers decide best serves the requirements of your application.

The following topics are presented:

- Security Strategy
- Configuring the RESOURCES SECURITY Parameter
- Implementing Operating System Security
- Implementing Application Password-level Security
- Implementing Security via an Authentication Server
- Implementing Security via Access Control Lists (BEA TUXEDO System)

Security Strategy

This section covers the levels of security provided by the WLE system or the BEA TUXEDO system. Application designers need to decide the appropriate level for their applications.

Operating System

For platforms that have underlying security mechanisms, this is the first line of defense. The security level is configured to "`NONE`" (configuration is discussed below). This implies that there are no additional mechanisms (for example, WLE or BEA TUXEDO system password) beyond what the platform provides.

Most WLE and BEA TUXEDO applications are managed by a system administrator who configures the application, starts up the application (servers run with the permissions of this administrator), and monitors the running application, making dynamic changes as necessary. This arrangement implies that server programs are "trusted," since they run with the administrator's permissions. This working method is supported by the login mechanism and read/write permissions on files, directories, and system resources provided by the underlying operating system.

Client programs are run directly by users with their own permissions. Normally, however, users have access to the administrative configuration file and interprocess communication mechanisms, such as the Bulletin Board (in shared memory), as part of normal processing. This is true regardless of whether additional WLE or BEA TUXEDO system security is configured.

For some applications running on platforms that support greater security, a more secure approach is to limit access to the files and IPC mechanisms to the application administrator and to have "trusted" client programs run with the permissions of the administrator (using a `setuid` mechanism). Combining these practices with WLE or BEA TUXEDO system security allows the application to "know" who is making the request.

For the most secure environment, only workstation clients should be allowed to access the application; client programs should not be allowed to run on the same machines on which application server and administrative programs run.

The WLE or BEA TUXEDO system security mechanisms can be used in addition to operating system security to prevent unauthorized access. The additional security can be used to avoid simple violations, such as someone accessing an unattended terminal. In addition, it can protect the boundaries of the administrative domain from interdomain or workstation client access over the network by unauthorized users.

Application Password

This security level requires that every client provide an application password as one step in the process of joining the application. The security level is configured to `APP_PW`. The administrator must provide an application password when this level is configured. (The password can be changed by the administrator.) It is the responsibility of the administrator to inform authorized users of the application about the password.

If this level of security is used, WLE or BEA TUXEDO system-supplied client programs, `ud`(1) for example, prompt for the application password.

Application-written client programs must include code to obtain the password from a user. The password should not be echoed to the user's screen.

Note: For BEA TUXEDO systems, the password is placed in clear text in the `TPINIT` buffer and is evaluated when the client calls `tpinit()` to join the application.

See "Writing Client Programs" in the *BEA TUXEDO Programmer's Guide* for examples of code for handling a password.

User Authentication

The third level of WLE or BEA TUXEDO system security is based on authenticating each individual user in addition to providing the application password. The security level is configured to `USER_AUTH`.

This level involves passing user-specific data to an authentication service. Often, the data is a per-user password. This data is automatically encrypted when it is sent over the network from workstation clients. The default authentication service, `AUTHSVC`, is provided by a BEA TUXEDO system-supplied server, `AUTHSVR`. The operation of an authentication service is described in "Writing Service Routines" in the *BEA TUXEDO Programmer's Guide*. This server can be replaced with an application authentication server that has logic specific to the application. (For example, it might access the widely used Kerberos mechanism for authentication.)

With this level of security, authentication is provided, but access control is not provided. That is, the user is checked when joining the application, but then is free to execute any services, to post events, and to access application queues. It is possible for servers to do application-specific authorization within the logic of the service routines, but there are no hooks for authorization that check for access to events or application queues. The alternative is to use the built-in access control checking.

Access Control (BEA TUXEDO system)

Note: This feature is not supported on WLE systems.

With the use of access control lists (ACLs), not only is a user authenticated when joining the application, but in addition, permissions are checked automatically when attempts are made to access application entities (such as services). ACL security also includes the user-authentication security equivalent to `USER_AUTH`.

Optional Access Control Lists (BEA TUXEDO system)

There are two levels of ACL checking. The first ACL security level is simply called `ACL`. If `ACL` is configured, the access control lists are checked whenever a user attempts to access a service name, queue name, or event name within the application. If there is no ACL associated with the user's name, the assumption is that permission is granted. For this reason, this level is considered "optional." It allows the administrator to configure access for only those resources that need more security; ACLs need not be configured for services, queues, or events that are open to everyone.

For some applications, it may be necessary to use both system-level and application authorization. An ACL can be used to control who can request a service, and application logic can control data-dependent access (for example, who can handle transactions for more than one million dollars).

Note that ACL checking is not done for administrative services, queues, and events with names that begin with a dot (.). For example, anyone can subscribe to administrative events such as, `.SysMachineBroadcast`, `.SysNetworkConfig`, `.SysServerCleaning`.

Mandatory Access Control Lists (BEA TUXEDO system)

The second ACL security level is `MANDATORY_ACL`. This level is similar to `ACL`, but an access control list must be configured for every entity (such as a service, queue, or event) that users can access. If `MANDATORY_ACL` is specified and there is no ACL for a particular entity, permission for that entity is denied.

Note: The WLE CORBA API does not support access control lists (ACLs). The WLE system ignores the `SECURITY MANDATORY_ACL` parameter in an application's `UBBCONFIG` configuration file on machines using the WLE CORBA API. You can use the `SECURITY MANDATORY_ACL` parameter on machines with WLE installed if the applications use only the ATMI API. Also, specifying the `SECURITY ACL` parameter does not cause access control checks to be performed on WLE servers that were built using the CORBA API.

Link-Level Encryption (BEA TUXEDO system)

Users of the BEA TUXEDO Security Add-On Package (US/Canada or International) can establish data privacy for messages moving over the network links that connect the machines in a BEA TUXEDO application. For a detailed description of this feature, see Chapter 6, "Building Networked Applications."

Configuring the RESOURCES SECURITY Parameter

You can designate a security scheme by setting the value of one parameter in the RESOURCES section of the configuration file: SECURITY. (The parameter AUTHSVC also comes into play if SECURITY is set to USER_AUTH, ACL or MANDATORY_ACL.)

To set the SECURITY parameter, perform the following steps.

1. Open the UBBCONFIG file in a text editor.
2. Set the SECURITY parameter as follows.

   SECURITY=*METHOD* where the value of *METHOD* is one of the following:

   - NONE
   - APP_PW
   - USER_AUTH
   - ACL
   - MANDATORY_ACL

   The default is NONE.

The value APP_PW indicates that application password security will be enforced (that is, clients will be required to provide the application password during initialization). Setting APP_PW causes tmloadcf to prompt for an application password.

The value USER_AUTH is similar to APP_PW but, in addition, indicates that per-user authentication will be done during client initialization.

The value ACL is similar to USER_AUTH but, in addition, indicates that access control checks will be done on service names, queue names, and event names. If an associated ACL is not found for a name, it is assumed that permission is granted.

The value MANDATORY_ACL is similar to ACL, but permission is denied if an associated ACL is not found for the name.

Implementing Operating System Security

Implementing operating system security is one of the easier tasks you will have as an administrator. It consists entirely of not implementing any higher level of security.

In the RESOURCES section, set SECURITY to NONE. If you leave SECURITY blank, NONE is the default.

Operating system security depends on the underlying password and the read/write permission structure of the operating system. You need to make sure that your users are able to connect to the application and have access to application files and programs. Consult the administrator's guide for your operating system to learn how to do this.

Implementing Application Password-level Security

Application password-level security requires all users to enter the same password to be allowed access to the application.

It is implemented as follows:

- The application programmer writes code that prompts the user for the password. The password must be put into the passwd field of the TPINIT buffer before tpinit(3c) is called to join the application.
- The administrator sets the SECURITY parameter (in the RESOURCES section of the UBBCONFIG file) to APP_PW.
- When the configuration is loaded via tmloadcf(1), the administrator is prompted for a password. The password entered at that time becomes the password for the application and remains in effect until it is changed by the administrator via the passwd command of tmadmin(1).

At runtime, all clients need to provide this password to access the application.

Implementing Security via an Authentication Server

User authentication-level security requires a server that can authenticate users by checking individual passwords against a file of legal users.

The authentication server shipped with the WLE or BEA TUXEDO system, AUTHSVR, provides two levels of security checks:

- User level security—determines whether or not a particular user can log on to the system. When the tpinit(3c) function is called to join the application, the user's ID, client name, and user name are verified with a password.
- Group level security—determines which application programs users can use once they have logged on. Once users have logged on, the application knows which users belong to which groups.

The Authentication Server

The WLE or BEA TUXEDO system user authentication is provided by AUTHSVR(5).

AUTHSVR provides per-user authentication. When a client process calls tpinit(3c) to join the application, AUTHSVR validates the user name, client name, and password. If tpinit fails for security reasons, a security violation is logged both in the userlog, and as a system event. On success, AUTHSVR provides the client with an application key that cannot be forged. The client presents the application key in the appkey field of the TPSVCINFO structure on each service invocation.(See "Writing Client Programs" in the *BEA TUXEDO Programmer's Guide*.)

Currently, authentication is not provided by a standard authentication mechanism, such as Kerberos, DCE, or public key encryption. When enhancements are provided to use such mechanisms, authentication is based on the user name. The client name is used for application logic only (for example, filtering of broadcast messages). If you are planning to use an alternate authentication scheme, we recommend that you do not associate client names with users. In this case, administrators should use only wildcard values for the *client* name in the user file; they should not use wildcard values for the *user* name.

Configuring the Authentication Server

To add AUTHSVR to an application, you must define AUTHSVR as a server in the TUXCONFIG file. To do so, add the following lines to the UBBCONFIG file.

```
RESOURCES
SECURITY   "USER_AUTH"
AUTHSVC    "AUTHSVC"

SERVERS
AUTHSVR SRVGRP="groupname" SRVID=1 RESTART=Y GRACE=0 MAXGEN=2
CLOPT="-A"
```

Adding, Modifying, and Deleting User Accounts

The shell-level commands tpusradd(1) and tpgrpadd(1) allow you to create files containing lists of authorized users and groups. The tpusrdel(1), tpusrmod(1), tpgrpdel(1), and tpgrpmod(1) commands enable you to maintain your user and group files. The following parameters are used in one or more of these six commands:

- *usrname*—a character string that represents the name of a user.
- *client_name* or *cltname*—a character string that represents the name of a client.
- *UID*—an integer between 0 and 128K used internally by the application to refer to a particular user. This is not the same as the UID parameter in the RESOURCES section of the configuration file, which designates the owner of the application.
- *group_name* or *grpname*—a character string that represents the name of a group.
- *GID*—an integer between 0 and 16K used internally by the application to refer to an application group. This is not the same as the GID parameter in the RESOURCES section of the configuration file, which designates the group of the owner of the application.

Two files are used for user and group administration:

- $APPDIR/tpusr
- $APPDIR/tpgrp

The files are colon-delimited, flat ASCII files, readable only by the application's administrator.

The files are kept in the application directory, specified by the environment variable `$APPDIR`. The format of the files is irrelevant, since they are fully administered with shell-level commands.

The commands `tpusradd`(1), `tpusrdel`(1), and `tpusrmod`(1) are available for modifying the files `tpusr` and `tpgrp`. For all of these commands, the environment variable `$APPDIR` must be set to the path name of the BEA TUXEDO system application that will be modified. In addition, only the application owner, as specified in `$TUXCONFIG`, is allowed to use these commands.

Following is the syntax of the commands.

```
tpusradd [-u UID] [-g GID] [-c client_name] usrname

tpusrdel [-c client_name] usrname

tpusrmod [-u UID] [-g GID] [-c client_name] [-l new_login] [-n\
new_client_name] [-p] usrname
```

Section (1) of the *BEA TUXEDO Reference Manual* also includes the commands `tpaddusr`(1), `tpdelusr`(1), and `tpmodusr`(1), which are functionally similar to the three commands described here. `tpaddusr`(1), `tpdelusr`(1), and `tpmodusr`(1) are provided for compatibility with releases prior to Release 6.0; if you are running Release 6.0 or later, we recommend you use the commands described in this section.

Adding, Modifying, and Deleting Groups

The commands `tpgrpadd`(1), `tpgrpdel`(1), and `tpgrpmod`(1) enable you to modify the files `tpusr` and `tpgrp`. For all of these commands, the environment variable `$APPDIR` must be set to the path name of the WLE or BEA TUXEDO system application that will be modified. In addition, only the application owner, as specified in `$TUXCONFIG`, is allowed to use these commands.

Following is the syntax of the commands.

```
tpgrpadd [-g GID] grpname
tpgrpdel grpname
tpgrpmod [-g GID] [-n new_grpname] grpname
```

Implementing Security via Access Control Lists (BEA TUXEDO System)

Note: The WLE CORBA API does not support access control lists (ACLs). The WLE system ignores the `SECURITY MANDATORY_ACL` parameter in an application's `UBBCONFIG` configuration file on machines using the WLE CORBA API.

You can use the `SECURITY MANDATORY_ACL` parameter on machines with WLE installed if the applications only use the ATMI API. Also, specifying the `SECURITY ACL` parameter does not cause access control checks to be performed on WLE servers that were built using the CORBA API.

Access control lists (ACLs) enhance the security features of BEA TUXEDO systems. ACLs provide group-based access control to application entities (services, events, and /Q queues). By looking at the client's application key, these entities can identify the group to which the user belongs; by looking at the ACL, the entity can determine whether the client's group has access permission.

Access control is done at the group level for the following reasons:

- System administration is simplified. It is easier to give a group of people access to a new service than it is to give each individual user access to the service.
- Performance is improved. Because access permission needs to be checked for each invocation of an entity, permission should be resolved quickly. Because there are fewer groups than users, it is quicker to search through the list of privileged groups than it is to search through a list of privileged users.

If user-level ACLs are needed, they may be implemented by creating a group for each user, and then setting up the group to have the desired permissions for its single member.

Limitations of ACLs

Access control lists have the following limitations:

- A user can be associated with only one group at a time. To be a member of more than one group, a user must have multiple entries in the file `$APPDIR/tpusr`.
- ACLs are name based. They do not distinguish between services, events, or queues; they look only at the name. Because of this, all entities must be named uniquely. It is not valid to have a queue and a service with the same name, unless access to both entities is always either granted or denied.
- User identification aging is not supported. If a user is removed from the system, it is up to the administrator to decide when it is appropriate to add another user with the same ID to the application.

Administering ACLs

ACLs are stored in the file `$APPDIR/tpacl`, an ASCII file that is readable and writable only by the application administrator. The file is administered with the following commands:

- `tpacladd`

  `tpacladd [-g` *`GID`* `|` *`group_name`*`] [,`*`GID`* `|` *`group_name`*`...] `*`entity_name`*

  `entity_name` is the name of the service, event, or /Q queue for which to create an ACL.

- `tpacldel`

  `tpacldel` *`entity_name`*

  `entity_name` is the name of the ACL entry that is to be deleted.

- `tpaclmod`

  `tpaclmod [-g` *`GID`* `|` *`group_name`*`][,`*`GID`* `|` *`group_name`*`...] `*`entity_name`*

  The `-g` option allows the specification of a group or list of groups that can access the feature provided by *`entity_name`*.

CHAPTER

15 Monitoring a Running System

As an administrator, you must ensure that once your application is up and running, it meets (and continues to meet) the performance, availability, and security requirements your company has set for it. 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 WLE and BEA TUXEDO systems provides 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, well, and securely.

Specifically, this chapter discusses the following topics:

- 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 WLE or BEA TUXEDO system maintains parameter settings and generates statistics for the following system components:

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

Where the Data Resides

To ensure that you have the information necessary for monitoring your system, the WLE or BEA TUXEDO system provides the following three 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 13, "Monitoring Log Files."

How You Can Use the Data

The administrative data provided by your WLE or BEA TUXEDO system allows you to monitor a multitude of potential trouble areas on your system. For example, this data allows you to:

- Tune the running system based on actual loads
- Detect security breaches

Moreover, you can 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 your help. With proper configuration, your system may be able to do 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.

Types of Data

Two types of administrative data are available on every running WLE and BEA TUXEDO system: static and dynamic.

Static Data

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

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). You can look up the maximum number of machines allowed by checking the current values of the system-wide parameters for your configuration (one of which is `MAXMACHINES`).
- Suppose you think you may be able to improve the performance of your application by tuning your system. To determine whether tuning is required, you need to check on the amount of local IPC resources currently available.

Dynamic Data

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

You will need to check the dynamic data about your configuration frequently. For example:

- Suppose throughput is suffering 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

  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. The WLE and BEA TUXEDO system provides the following methods of doing both tasks, as shown in this 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 System)" *BEA TUXEDO Reference* manual, section 5
BEA TUXEDO Web-based GUI	Using a graphical interface	The Help accessed directly from the GUI

Which method is best for you? The answer depends on your answers to several questions.

- How much experience do you have as a WLE or a BEA TUXEDO system administrator?

  If you have a lot of experience as an administrator (and shell programming expertise), you may prefer to write programs that automate your most frequently run commands.

- Are you an experienced UNIX system user?

  If not, you may be most comfortable using the Web-based GUI.

- What information do you want to view?

  If you examine the `RESOURCES` section of the `UBBCONFIG` file through the `tmadmin` command, you see only the current values; the defaults are not displayed.

If you decide to monitor your system at run time through the `tmadmin` command interpreter, continue reading; this chapter describes `tmadmin` and explains how to use it.

Using the tmadmin Command Interpreter

This section provides the following information:

- A step-by-step description of what happens during a typical `tmadmin` session, including:
  - 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
  - Descriptions of the `tmadmin` meta-commands: commands that help you make the best—and most efficient—use of the `tmadmin` commands
- A step-by-step procedure that you can follow to run `tmadmin` for most tasks

  Instructions for individual tasks are provided in later sections of this chapter.

What is tmadmin?

The `tmadmin` command is an interpreter for 50 commands that let you view and modify a Bulletin Board and its associated entities.

Note: `tmadmin` is supported on UNIX and Windows NT platforms.

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.

How a tmadmin Session Works

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 and TUXOFFSET environment variables to get the location and offset at which the configuration file has been loaded. (Be sure you have defined these variables 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 WLE 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*.

Summary of 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. In addition, you can generate a report of the WLE 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:

1. Have administrator privileges (that is, your effective `UID` and `GID` must be those of the administrator).
2. Invoke the command interpreter without any options:

```
$ tmadmin
```

Version number and license number report

To find out which version of the WLE 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 Meta-commands

The `tmadmin` command interpreter is equipped with a set of meta-commands, commands that help you use `tmadmin`. Table 15-1 lists the `tmadmin` meta-commands.

Note: The tables and examples in this chapter include the abbreviated forms of the `tmadmin` command names.

Table 15-1 tmadmin Meta-commands

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
`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` meta-command (`d`) lets you set and unset defaults for the following frequently used parameters for most `tmadmin` commands: group name, server ID, machine, user name, client name, queue address, service name, device blocks, device offset, 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.

Once 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` meta-command 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)
         Blocks: 1000
         Offset: 0
           Path: /home/apps/bank/bankdll # Path defaults to value of FSCONFIG
>
```

- 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 versus 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*.

Perform the following steps to run the tmadmin commands.

1. Make sure the TUXCONFIG and TUXOFFSET environment variables have 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: bbparms -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.

Table 15-2 Commands for Monitoring Tasks

To Determine Whether . . .	Run this Command . . .	If . . .	Then . . .
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.
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.

Table 15-2 Commands for Monitoring Tasks

To Determine Whether . . .	Run this Command . . .	If . . .	Then . . .
Your operating system resources (such as shared memory and semaphores) on a local machine are sufficient	`$tmadmin -r` `> bbsread`	You do not have sufficient resources in the operating system	Increase the IPC resources (semaphores, shared memory segments, and so on) in the operating system.
You want to keep the current values for system-wide parameters (in the `RESOURCES` section of your `UBBCONFIG` file)	`$ 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. Re-boot the application.

Example: Output from tmadmin Commands

This section provides examples of output from the following `tmadmin` monitoring commands:

- `printqueue`
- `printconn`
- `printnet`
- `printtrans`

Note: For a list of all 50 `tmadmin` commands, see the `tmadmin`(1) reference page in the *BEA TUXEDO Reference Manual*.

printqueue Output

The following output from the `printqueue` command lets you check the distribution of work in the `bankapp` application.

```
printqueue [qaddress]

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

>pq
```

a.out Name	Queue Nam	# Svrs	Wk Q'd	# 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 . . .
`a.out 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)
`# Svrs`	The number of servers connected to the queue
`Wk Q'd`	The load factor of all requests currently queued
`# Queued`	The actual number of requests queued
`Ave. Len`	The average queue length **Note:** Not available in MP mode.
`Machine`	The LMID of the machine on which the queue is located

printconn Data

The following (verbose) output from the `printconn` command shows that the client process has:

- Begun two conversations
- Maintained control of both lines
- Not yet sent any requests

```
printconn [-m machine]

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

> 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

This section 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`.)

printnet [-m *machine_list*]

```
tmadmin - Copyright © 1987-1990 AT&T; 1991-1993 USL. 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.

This section shows the output produced by running the `printtrans` command in 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.

printtrans [-m *machine*] [-g *groupname*]

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

> pt
>> 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

>
```

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 15-3.

Listing 15-3 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 15-4.

Listing 15-4 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 15-5.

Listing 15-5 Sample bbparms Output

```
> bbparms
Bulleti
Board Parameters:
        MAXSERVERS: 35
       MAXSERVICES: 75
      MAXACCESSERS: 40
            MAXGTT: 20
           MAXCONV: 10
        MAXBUFTYPE: 16
       MAXBUFSTYPE: 32
             IPCKEY: 35384
             MASTER: SITE1,SITE2
              MODEL: MP
              LDBAL: Y
            OPTIONS: LAN,MIGRATE
          SCANUNIT: 10
        SANITYSCAN: 12
          DBBLWAIT: 6
          BBLQUERY: 180
           BLOCKTIME: 30
```

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 WLE 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 chapter discusses the following topics:

- What is the ULOG?
- What is tlisten?
- 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 WLE or BEA TUXEDO system—error messages, warning messages, information messages, and debugging messages—are written to this log. Application clients and servers can also write to the user log.

A new log is created every day and there can be a different log on each machine. However, a ULOG can be shared across machines when a remote file system is being used.

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 WLE or BEA TUXEDO system and application failures.

How is the ULOG created?

The ULOG is created by the WLE or BEA TUXEDO system whenever one of the following events occurs:

- A new configuration file is loaded
- An application is booted

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 WLE 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
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 *WebLogic Enterprise 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 tlisten?

`tlisten` is the section of the `ULOG` in which error messages for the `tlisten` process are recorded. (The `tlisten` process provides remote service connections for other machines.) As part of the `ULOG` file, a `tlisten` log can be viewed with any text editor.

Each machine, including the master machine, should have a `tlisten` process running on it. Separate `tlisten` logs are maintained in the `ULOG` on each machine. However, they can be shared across remote file systems.

Purpose

The `tlisten` log is a record of `tlisten` process failures. It is used, during the boot process, by `tmboot` and, while an application is running, by `tmadmin`.

How is the tlisten log created?

The `tlisten` log is created by the `tlisten` process as soon as that process is started. Whenever a `tlisten` process failure occurs, an appropriate message is recorded in the `tlisten` log.

Message Format

Each `tlisten` log message includes the date and time at which the message was added to the log.

Location

By default, the `tlisten` log resides in `$TUXDIR/udataobj`. You can store it in another location, however, by entering the following command.

```
tlisten -L new_pathname
```

For example, if you want your `TLOG` file to be named `TLLOG` and to reside in the `/home/apps/logs` directory, enter the following command.

```
tlisten -L /home/apps/logs/TLOG
```

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)" in this chapter.

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)" in this chapter.)

Creating and Maintaining Logs

The ULOG is generated by various WLE 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 the following instructions:

- How to assign a location for the ULOG
- How to create 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 file name.

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 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 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 WLE 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 WLE 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 TLOGs on any machine other than the master machine. The WLE or BEA TUXEDO system creates TLOGs on nonmaster machines (as long as a UDL exists on those machines) when the application is booted.

2. Enter the following 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 WLE 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 WLE 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 the following.

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

where *new_device* specifies the full path name for the new device on which space is to be allocated. This command creates a new entry on the UDL and makes the space 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 WLE 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 WLE 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. 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.) If 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 the following command.

```
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

Perform the following steps to create the log.

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 the following command.

   ```
   tmadmin -c
   ```

3. At the tmadmin command prompt (>), enter the following.

   ```
   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

TLOGs require little maintenance. This section provides instructions for two common maintenance tasks: reinitializing a TLOG and removing a TLOG:

- To reinitialize a TLOG, enter the following.

  ```
  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 TLOGs, possibly causing transactions to abort.

- To delete a TLOG, enter the following.

  ```
  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.

Consider the example shown in Listing 16-3. Notice how `LIBTUX_CAT` message 358 identifies the exact nature of the problem causing problems reported in subsequent messages, 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 *WebLogic Enterprise System Messages or BEA TUXEDO System Message Manual* for complete descriptions of user log messages and recommendations for any actions that should be taken to resolve the problems indicated.

Analyzing the tlisten Log

Keep the following guidelines in mind as you check the `tlisten` messages in your `ULOG`:

- A message is placed in the `tlisten` 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.

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 message in a `tlisten` log.

```
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

This chapter discusses the following topics:

- Maximizing Your Application Resources
- When to Use MSSQ Sets (BEA TUXEDO Servers)
- Enabling Load Balancing
- Using Multithreaded JavaServers
- Bundling Services into Servers (BEA TUXEDO Servers)
- Enhancing Efficiency with Application Parameters
- Setting Application Parameters
- Determining IPC Requirements
- Measuring System Traffic

Maximizing Your Application Resources

Making correct decisions in response to the following questions can improve the functioning of your WLE or BEA TUXEDO application:

- When should I use MSSQ sets? (BEA TUXEDO System)
- How should I assign load factors?
- How should I package interfaces and/or services into servers?
- How should I set my application parameters?

- How should I tune my operating system IPC parameters?
- How should I hunt for and eliminate bottlenecks?

When to Use MSSQ Sets (BEA TUXEDO Servers)

Note: MSSQ sets are not supported in the WLE system.

When is it beneficial to use MSSQ sets?

When to Use MSSQ Sets	When Not to Use MSSQ Sets
There are several, but not too many servers.	There are a large number of servers. (A compromise is to use many MSSQ sets.)
Buffer sizes are not too large.	Buffer sizes are large enough to exhaust one queue.
The servers offer identical sets of services.	Services are different for each server.
The messages involved are reasonably sized.	Long messages are being passed to the services causing the queue to be exhausted. This causes nonblocking sends to fail, or blocking sends to block.
Optimization and consistency of service turnaround time is paramount.	

Two analogies from everyday life may help to show why using MSSQ sets is sometimes, but not always, beneficial:

- An application in which MSSQ sets are used appropriately is similar to a bank, where all the tellers offer the same services and customers wait in line for the first available teller. This efficient arrangement ensures the best use of available services.
- An application in which it is better to avoid using MSSQ sets is similar to a supermarket, where each cashier offers a different set of services: some accept

cash only; some accept credit cards; and still others serve only customers buying fewer than ten items.

Enabling Load Balancing

On BEA TUXEDO systems, you can control whether a load balancing algorithm is used on the system as a whole. With load balancing, a load factor is applied to each service within the system, and you can track the total load on every server. Every service request is sent to the qualified server that is least loaded.

Note: On WLE systems, load balancing is always enabled. In other words, while you can specify `LDBAL=N`, the parameter is ignored for WLE systems.

This algorithm, although effective, is expensive and should be used only when necessary, that is, only when a service is offered by servers that use more than one queue. Services offered by only one server, or by multiple servers all in an MSSQ (multiple server single queue) do not need load balancing. The `LDBAL` parameter for these services should be set to `N`. In other cases, you may want to set `LDBAL` to `Y`.

To figure out how to assign load factors (located in the `SERVICES` section), run an application for a long period of time. Note the average time it has taken for each service to be performed. Assign a `LOAD` value of 50 (`LOAD=50`) to any service that takes roughly the average amount of time. Any service taking longer than the average amount of time to execute should have a `LOAD>50`; any service taking less than the average amount of "code" time to execute should have a `LOAD<50`.

Two Ways to Measure Service Performance Time

You can measure service performance time in one of the following ways:

- Enter `servopts -r` in the configuration file. The `-r` option causes a log of services performed to be written to standard error. You can then use the `txrpt`(1) command to analyze this information. (For details about `servopts`(5) and `txrpt`(1), see the *BEA TUXEDO Reference Manual*.)

- Insert calls to `time`(2) at the beginning and end of a service routine. Services that take the longest time receive the highest load; those that take the shortest time receive the lowest load. (For details about `time`(2), see a UNIX System Reference Manual.)

Using Multithreaded JavaServers

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

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

Programs written to WLE Java run without modification in both modes.

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 multi-processor machine

If the application is running on a single-processor machine and the application is CPU-intensive only, without any I/O or 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 multi-processor machine.

You can establish the number of threads for a Java server application by using the `-M option` to the `JavaServer` parameter. This parameter is used in the `SERVERS` section of the application's `UBBCONFIG` file. The `-M` options are described in the section "JavaServer Command Line Options" on page 3-40.

For multithreaded WLE 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 WLE Java software, as opposed to threads started and managed by an application program. Internally, WLE 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 done, the worker thread is returned to the pool of available threads.

The `MAXACCESSERS` parameter in the application's `UBBCONFIG` file sets the maximum number of concurrent accessers of a WLE system. Accessers include native and remote clients, servers, and administration processes.

A single threaded server counts as one accesser.

For a multithreaded JavaServer, the number of accessers 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 WLE system running multithreaded servers, **do not** simply double the existing `MAXACCESSERS` value of the whole system. Instead, you add up the accessers 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 accesser requirement of these servers is calculated by using the following formula:

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

Assigning Priorities to Interfaces or Services

You can exert significant control over the flow of data in an application by assigning priorities to BEA TUXEDO services using the `PRIO` parameter.

For an application running on a BEA TUXEDO system, you can specify the `PRIO` parameter for each service named in the `SERVICES` section of the application's `UBBCONFIG` file.

For example, 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 is always dequeued before a request for A or B. Requests for A and B are dequeued equally with respect to one another. The system dequeues every tenth request in first-in, first-out (FIFO) order to prevent a message from waiting indefinitely on the queue.

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

Characteristics of the PRIO Parameter

The `PRIO` parameter should be used cautiously. Depending on the order of messages on the queue (for example, A, B, and C), some (such as A and B) will be dequeued only one in ten times. This means reduced performance and potential slow turnaround time on the service.

The characteristics of the `PRIO` parameter are as follows:

- It determines the priority of an interface or a service on the server's queue.
- The highest assigned priority gets first preference. This interface or service should occur less frequently.
- A lower priority message does not remain forever enqueued, because every tenth message is retrieved on a FIFO basis. Response time should not be a concern of the lower priority interface or service.

Assigning priorities enables you to provide faster service to the most important requests and slower service to the less important requests. You can also give priority to specific users or in specific circumstances.

Bundling Services into Servers (BEA TUXEDO Servers)

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

When to Bundle Services

You should bundle services for the following reasons:

- Functional similarity—If some services are similar in their role in the application, you can bundle them in the same server. The application can offer all or none of them at a given time. An example is the `bankapp` application, in which the `WITHDRAW`, `DEPOSIT`, and `INQUIRY` services are all teller operations. Administration of services becomes simpler.
- Similar libraries—For example, if you have three services that use the same 100K library and three services that use different 100K libraries, bundling the first three services saves 200K. Often, functionally equivalent services have similar libraries.
- Filling the queue—Bundle only as many services into a server as the queue can handle. Each service added to an unfilled MSSQ set may add relatively little to the size of an executable, and nothing to the number of queues in the system. Once the queue is filled, however, the system performance degrades and you must create more executables to compensate.
- Placement of call-dependent services—Avoid placing in the same server two (or more) services that call each other. If you do so, the server will issue a call to itself, causing a deadlock.

Enhancing Efficiency with Application Parameters

You can set the following application parameters to enhance the efficiency of your system:

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

Setting the MAXACCESSERS, MAXSERVERS, MAXINTERFACES, and MAXSERVICES Parameters

The MAXACCESSERS, MAXSERVERS, MAXINTERFACES, and MAXSERVICES parameters increase semaphore and shared memory costs, so you should choose the minimum value that satisfies the needs of the system. You should also allow for the variation in the number of clients accessing the system at the same time. Defaults may be appropriate for a generous allocation of IPC resources; however, it is prudent to set these parameters to the lowest appropriate values for the application.

For multithreaded WLE JavaServers, you must account for the number of worker threads that each server is configured to run. The MAXACCESSERS parameter sets the maximum number of concurrent accessers of a WLE system. Accessers include native and remote clients, servers, and administration processes.

A single threaded server counts as one accesser.

For a multithreaded JavaServer, the number of accessers 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 WLE system

running multithreaded servers, **do not** simply double the existing MAXACCESSERS value of the whole system. Instead, you add up the accessers 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 accesser requirement of these servers is calculated by using the following formula:

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

Setting the MAXGTT, MAXBUFTYPE, and MAXBUFSTYPE Parameters

You should increase the value of the MAXGTT parameter if the product of multiplying the number of clients in the system times the percentage of time they are committing a transaction is close to 100. This may require a great number of clients, depending on the speed of commit. If you increase MAXGTT, you should also increase TLOGSIZE accordingly for every machine. You should set MAXGTT to 0 for applications that do not use distributed transactions.

You can limit the number of buffer types and subtypes allowed in the application with the MAXBUFTYPE and MAXBUFSTYPE parameters, respectively. The current 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.

Setting the SANITYSCAN, BLOCKTIME, BBLQUERY, and DBBLWAIT Parameters

If a system is running on slow processors (for example, due to heavy usage), you can increase the timing parameters: SANITYCAN, BLOCKTIME, and individual transaction timeouts. If networking is slow, you can increase the value of the BLOCKTIME, BBLQUERY, and DBBLWAIT parameters.

Setting Application Parameters

The following table describes the system parameters available for tuning an application.

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

Determining IPC Requirements

The values of different system parameters determine IPC requirements. You can use the tmboot -c command to test a configuration's IPC needs. The values of the following parameters affect the IPC needs of an application:

- MAXACCESSERS
- REPLYQ
- RQADDR (that allows MSSQ sets to be formed)

- MAXSERVERS
- MAXSERVICES
- MAXGTT

Table 17-1 describes the system parameters that affect the IPC needs of an application.

Table 17-1 Tuning IPC Parameters

Parameter(s)	Action
MAXACCESSSERS	Equals the number of semaphores. Number of message queues is almost equal to MAXACCESSERS + number of servers with reply queues (number of servers in MSSQ set + number of MSSQ sets).
MAXSERVERS, MAXSERVICES, and MAXGTT	While MAXSERVERS, MAXSERVICES, MAXGTT, and the overall size of the ROUTING, GROUP, and NETWORK sections affect the size of shared memory, an attempt to devise formulas that correlate these parameters can become complex. Instead, simply run `tmboot -c` or `tmloadcf -c` to calculate the minimum IPC resource requirements for your application.
Queue-related kernel parameters	Need to be tuned to manage the flow of buffer traffic between clients and servers. The maximum total size of a queue in bytes must be large enough to handle the largest message in the application, and to typically be 75 to 85 percent full. A smaller percentage is wasteful; a larger percentage causes message sends to block too frequently. Set the maximum size for a message to handle the largest buffer that the application sends. Maximum queue length (the largest number of messages that are allowed to sit on a queue at once) must be adequate for the application's operations. Simulate or run the application to measure the average fullness of a queue or its average length. This may be a trial and error process in which tunables are estimated before the application is run and are adjusted after running under performance analysis. For a large system, analyze the effect of parameter settings on the size of the operating system kernel. If unacceptable, reduce the number of application processes or distribute the application to more machines to reduce MAXACCESSERS.

Measuring System Traffic

As on any road in which traffic exists and runs at finite speed, bottlenecks can occur in your system. On a highway, cars can be counted with a cable strung across the road, that causes a counter to be incremented each time a car drives over it. Similarly, you can measure service traffic. For example, at boot time (that is, when `tpsvrinit()` is invoked), you can initialize a global counter and record a starting time. Subsequently, each time a particular service is called, the counter is incremented. When the server is shut down (by invoking the `tpsvrdone`() function, the final count and the ending time are recorded. This mechanism allows you to determine how busy a particular service is over a specified period of time.

In the BEA TUXEDO system, bottlenecks can originate from data flow patterns. The quickest way to detect bottlenecks is to begin with the client and measure the amount of time required by relevant services.

Example: Detecting a System Bottleneck

Client 1 requires 4 seconds to print to the screen. Calls to `time`(2) determine that the `tpcall` to service A is the culprit with a 3.7 second delay. Service A is monitored at the top and bottom and takes 0.5 seconds. This implies that a queue may be clogged, which was determined by using the `pq` command.

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

Using `time`(2), you can measure the duration of the following:

- The entire client program
- A client service request only
- The entire service function
- The service function making a service request (if any)

Detecting Bottlenecks on UNIX Platforms

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

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

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

Use This Option	To
`-u`	Gather CPU utilization numbers, including the portion of the time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle.
`-b`	Report buffer activity, including transfers per second of data between system buffers and disk, or other block devices.
`-c`	Report system call activity. This includes system calls of all types, as well as specific system calls such as `fork`(2) and `exec`(2).
`-w`	Monitor system swapping switching activity. This includes the number of transfers for swapins and swapouts.
`-q`	Report average queue lengths while occupied and the percent of time occupied.
`-m`	Report message and system semaphore activities, including the number of primitives per second.
`-p`	Report paging activity, including the address translation page faults, page faults and protection errors, and the valid pages reclaimed for free lists.

Use This Option	To
`-r`	Report unused memory pages and disk blocks, including the average number of pages available to user processes and the disk blocks available for process swapping.

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

Detecting Bottlenecks on Windows NT Platforms

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

```
Start ––> Programs ––> Administration Tools ––> NT Performance Monitor
```

CHAPTER

18 Migrating Applications

This chapter discusses the following topics:

- 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 WLE 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 WLE or BEA TUXEDO system, and the application all need to be maintained. The WLE and BEA TUXEDO systems provide a way to migrate the applications so that they can be serviced.

The WLE and BEA TUXEDO systems offer migration tools that can also be used to recover from a machine crash, network partitions, database corruptions, WLE 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 WLE 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 chapter discusses the following topics:

- 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 simpler 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
```

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
```

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 (WLE 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.

CHAPTER

20 Dynamically Reconfiguring Applications

This chapter presents 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 WLE 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 WLE 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 WLE 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 WLE 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
  ```

  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
  ```

  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 WLE 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 path name 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 WLE 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 [n].

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

   field_name<*tabs*>*field_value*

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

   In most cases, the field name is the same as the 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 [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). Here are a few examples:

- 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

Complete the following steps to add a new machine.

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

Complete the following steps to add a server.

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

Complete the following steps to add a group.

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:

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

Complete the following steps to change the data-dependent routing for an application.

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 procedure.

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

Complete the following steps to change an application password.

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).

21 Event Broker/Monitor (BEA TUXEDO System)

The BEA TUXEDO Event Broker/Monitor is a tool that enhances the tracking of events in a running application.

Note: This chapter is specific to the BEA TUXEDO system. However, WLE administrators should know that each WLE application relies on the BEA TUXEDO System Event Broker. This event broker must be started before any servers providing the NameManager service in a WLE application's `UBBCONFIG` file are started. For details, see the section "Required Order in Which to Boot Servers (WLE Servers)" 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.

The chapter discusses 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 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 EVENTS(5).

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. Here are two examples:

- 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 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 `EVENTS`(5) 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. (See Sections (3c) and (3cbl) in the *BEA TUXEDO Reference Manual* for details.)
- 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

CHAPTER

22 Administrative Reference (WLE System)

The WLE system supports the following administrative commands and server processes:

- `idl2ir`
- `ir2idl`
- `irdel`
- `ISL`
- `TMFFNAME`
- `TMIFRSVR`
- `factory_finder.ini`

This chapter describes these commands and processes.

idl2ir

Synopsis Creates the Interface Repository and loads interface definitions into it.

Syntax

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

Options The options are as follows:

```
[-f repository-name] [-c]
[-D identifier[=definition]]
[-I pathname [-I pathname] [...]] [-N{i|e}]
```

Description Use this command to create the Interface Repository and to load it with interface definitions. If no repository file exists, this command creates it. If a repository file does exist, this command loads the specified interface definitions into it and, in effect, updates the file.

One of the side effects of doing this is that a new Interface Repository database file is created.

Parameters

`definition-filename-list`
: A list of file specifications containing the repository definitions. These files are treated as one logical file and are loaded in one operation.

`-f repository-name`
: The filename of the Interface Repository file. If you do not specify the `-f` option, the `idl2ir` command creates `repository.ifr as the` Interface Repository file on UNIX systems and `repository_1.ifr` on Microsoft Windows NT systems.

-c
: Indicates that a new repository is to be created. If a repository exists and this option is specified, the `idl2ir` command ignores the existing repository and replaces it with a new one. If a repository exists and this option is not specified, the `idl2ir` command updates the existing repository.

-D identifier[=definition]
: Performs the same function as the `#define` preprocessor directive; that is, the `-D` option defines a token string or a macro to be substituted for every occurrence of a given identifier in the definition file. If a definition is not specified, the identifier is defined as 1.You can specify multiple `-D` options.

`-I pathname`

Specifies a directory within which to search for include files, in addition to any directories specified with the `#include` OMG IDL preprocessor directive.

There are two types of `#include` OMG IDL preprocessor directives: system (for example, `<a.idl>`) and user (for example, `"a.idl"`). The path for system `#include` directives is `/usr/include` for UNIX systems, and any directories specified with the `-I` option. The path for system `#include` directives is the local directory for Windows NT systems, and any directories specified with the -I option.

The path for user `#include` directives is the current directory and any directories specified with the `-I` option. Multiple `-I` options can be specified.

ir2idl

Synopsis Shows the contents of an Interface Repository.

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

Options The options are as follows:

```
[-f repository-name] [-n]
[-t interface-type] [-o filename]
```

Description This command shows the contents of an Interface Repository. By directing the output to a file with the `-o` option, you can extract the OMG IDL file from the repository. By default, the repository file is `repository.ifr`.

Parameters

`interface-name`
The name of the interface whose contents are to be shown. If you do not specify an interface name, all interfaces in the repository are shown.

`-f repository-name`
The name of the repository to search for the interface definitions. If you do not specify the `-f` option, `repository.ifr` is used.

`-n`
Specifies that the output should not include those objects that were inherited.

`-t interface-type`
Indicates the type of objects to display. The object type must be one of the following keywords:

```
Attribute
Constant
Exception
Interface
Method
Module
Operation
Typedef
```

If you do not specify this option, the default is to display all of the types.

`-o filename`
The file specification for the file in which to write the retrieved OMG IDL statements. The default is standard output.

irdel

Synopsis Deletes the specified object from an Interface Repository.

Syntax

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

Description This command deletes the specified interface from the repository. Only interfaces not referenced from another interface can be deleted. By default, the repository file is `repository.ifr`.

Parameters

`-f repository-name`
: An optional parameter that specifies an Interface Repository. The `repository-name` value is the file specification of an Interface Repository. If this option is not specified, the `repository.ifr` is used as the default.

`-i id`
: The repository `id` for the specified object. The `id` is used as a secondary level of lookup. If the `id` does not match the `id` of the named object, the object is not deleted.

`object-name`
: The name of the interface to delete from the repository. The name can be a simple object name or a scoped name, for example, `MOD1::INTERF2::OP3` (operation `OP3` is within interface `INTERF2`, which is in application `MOD1`).

ISL

Synopsis Enables access to WLE objects by remote WLE clients using IIOP.

Syntax

```
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]
#NEW options for Outbound IIOP
           [-O]
           [-o outbound-max-connections]
           [-s Server-timeout]
           [-u out-mpx-users]"
```

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

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

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

For joint client/servers, if the remote joint client/server ORB supports bidirectional IIOP connections, the ISL can use the same inbound connection for outbound invokes to the remote joint client/server. The ISL also allows outbound invokes (outbound IIOP) to objects located in a joint client/server that is not connected to an ISH. This

capability is enabled when the `-O` option is specified. The associated command-line options (those shown above in **boldface text**) allow configuration of outbound IIOP support:

Parameters `-A`

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

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

`-n netaddr`

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

TCP/IP addresses must be specified in one of the following two formats:
`"//hostname:port_number"`
`"//#.#.#.#:port_number"`

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

Note: The hostname must begin with a letter character.

In the second format, the `"#.#.#.#"` is the dotted decimal format. In dotted decimal format, each # must be a number from 0 to 255. This dotted decimal number represents the IP address of the local machine.
In both of the above formats, `port_number` is the TCP port number at which the domain process listens for incoming requests. `port_number` can be a number between 0 and 65535 or a name. If `port_number` is a name, it must be found in the network services database on your local machine.

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

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

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

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

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

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

`[-C detect|warn|none]`

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

The official way for the CORBA client to connect to the IIOP Listener/Handler is via a Bootstrap object. The unofficial connection is established directly from an IOR. For example, a client could connect to one IIOP Listener/Handler via a Bootstrap object and then, perhaps inadvertently, connect to a second IIOP Listener/Handler by using an IOR that contains the host and port of the second IIOP Listener/Handler. Typically, this is not the case. Usually, the client uses IORs that contain the host and port of the IIOP Listener/Handler that the client connected to via a Bootstrap object. Use of such IORs does not cause an additional connection to be made.

Caution: The use of unofficial connections can cause problems for remote client applications that use transactions. The application may have the notion that invocations on both the official and unofficial connections within the same

transaction have succeeded; however, in reality, only invocations on the official connection are ACID (Atomicity, Consistency, Isolation, and Durability).

A value of `detect` causes the ISL/ISH to raise a `NO_PERMISSION` exception when an unofficial connection is detected. A value of `warn` causes the ISL/ISH to log a message to the user log exception when an unofficial connection is detected; no exception will be raised. A value of `none` causes the ISL/ISH to ignore unofficial connections.

`[-d device]`

Specifies the device filename used for network access by the server listener and its server handlers. This parameter is optional because some transport providers (for example, sockets) do not require a device name. However, other providers (for example, TLI) do require a device name. In the case of TLI, this option is mandatory. There is no default for this parameter. (This does not apply to Windows NT systems.)

`[-K {client|handler|both|none}]`

Directs the client, or the ISH process, or both, to activate the network provider's `KEEPALIVE` option. This option improves the speed and reliability of network failure detection by actively testing an idle connection's state at the protocol stack level. The availability and timeout thresholds for this feature are determined by operating system tunable parameters.

A value of `client` configures this option for the client; a value of `handler` configures this option for the ISL; and a value of `both` will configure both sides of the connection. The default value is `none`, in which case neither side has the `KEEPALIVE` option configured.

Note: The `KEEPALIVE` interval is an operating system parameter, so changing the value affects any other applications that enable `KEEPALIVE`. Many platforms have a two-hour default value that may be longer than desired.

This option is not available on all platforms. A userlog warning message is generated if the `KEEPALIVE` option is specified but is not available on the ISH's machine. If `KEEPALIVE` is requested but is not available on the client's machine, the setting is ignored.

`[-m minh]`

Specifies the minimum number of handlers that should be available in conjunction with this ISL at any given time. The default is 0. The The ISL will start this many ISHs immediately upon being booted and will not deplete

the supply of ISHs below this number until the administrator issues a shutdown to the ISL. The default value for this parameter is 0. The legal range is between 0 and 255.

`[-M maxh]`

Specifies the maximum number of handlers that should be available in conjunction with this ISL at any given time. The Handlers are started as necessary to meet the demand of remote WLE clients attempting to access the system. The default value for this parameter is equal to the setting for `MAXWSCLIENTS` on the logical machine, divided by the multiplexing factor for this ISL (see `-x` option below), rounded up by one. The legal range for this parameter is between 1 and 4096. The value must be equal to or greater than `minh`.

`[-T Client-timeout]`

Allows more time for a client to join an application 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 a secure application. For information about `SCANUNIT`, see the section "Characteristics of the SCANUNIT, SANITYSCAN, and BLOCKTIME Parameters" on page 3-19.

`[-x` mpx-factor]

This is an optional parameter used to control the degree of multiplexing desired within each ISH. The value for this parameter indicates the number of remote WLE clients that can be supported simultaneously by each ISH. The ISH ensures that new handlers are started as necessary to handle new remote WLE clients. This value must be greater than or equal to 1 and less than or equal to 4096. The default value for this parameter is 10.

`[-H external netadder]`

Specifies the external network address to be set as the host and port in interoperable object references returned to clients of the ISL. It has the same format as the ISL CLOPT `-n netaddr` option. This feature is useful when an IIOP, or remote, client needs to connect to an ISL through a firewall.

`[-O]`

This option (uppercase letter O) enables outbound IIOP to objects that are not located in a client that is connected to an ISH. Since the `–O` option requires a small amount of extra resources, the default is to not allow outbound IIOP.

`[-o outbound-max-connections]`

This option (lowercase letter o) specifies the maximum number of outbound connections that each ISH may have. In effect, it limits the number of simultaneous Outbound IIOP sockets that any single ISH under the control of this ISL will have active at one time.

This option requires that the `–O` (uppercase letter O) option is also specified. The value of this option must be greater than 0, but not more than 4096. An additional requirement is that the value of this option, (`outbound-max-connections`) times the maximum number of handlers, must be less than 32767. The default for this option is 20.

`[-s Server-timeout]`

Server-timeout is the time, in minutes, allowed for a remote server to remain idle. If a remote server does not receive any requests within this time period, the ISL disconnects the outbound IIOP connection to the server. The ISH reconnects to the remote server on subsequent requests. This option can be used for server platforms that are unstable. Note that this is a best-attempt value in that the ISL does not disconnect the connection before this time is up, but does not guarantee to disconnect the connection once the exact time has elapsed. This option requires that the `–O` (uppercase letter O) option is also specified. The value must be greater than or equal to 1. If this option is not specified, the default is 60 (one hour).

`[-u out-mpx-users]`

An optional parameter used to control the degree of outbound multiplexing desired within each ISH. The value for this option indicates the number of outbound IIOP users (native clients or servers) that can be supported simultaneously by each outbound IIOP connection in the ISH. The ISL ensures that new ISHs are started, as necessary, to handle new users up to the value of this option (`out-mpx-users`). This option requires that the –O (uppercase letter O) option is also specified. This option must be greater than 0 (zero), but not more than 1024; the default value is 10.

Portability The IIOP Server Listener is supported as a WLE-supplied server on UNIX and Microsoft Windows NT operating systems.

Interoperability The ISL works with any IIOP compliant ORB.

Network Addresses

Suppose the local machine on which the ISL is being run is using TCP/IP addressing and is named `backus.company.com`, with address `155.2.193.18`. Further suppose that the port number at which the ISL should accept requests is 2334. The address specified by the `-l` option could be:

```
//155.2.193.18:2334

//backus.company.com:2334
```

Examples

```
*SERVERS

ISL SRVGRP="ISLGRP" SRVID=1002 RESTART=Y GRACE=0

        CLOPT="-A -- -n //piglet:1900 -d /dev/tcp"
```

See Also

`UBBCONFIG`(5)

TMFFNAME

Synopsis Server that runs the FactoryFinder and supporting NameManager services.

Syntax

```
TMFFNAME SRVGRP="identifier" SRVID="number"
     [CLOPT="[-A] [servopts options]
[-- [-F ] [-N | -N –M [-f filename]]]"]
```

Description `TMFFNAME` is a server provided by WLE that runs the FactoryFinder and supporting NameManager services which maintain a mapping of application-supplied names to object references.

Parameters

`-A`
: Advertise all services built into the server

`-F`
: FactoryFinder service

`-N`
: Slave NameManager service
This is the default.

`–M`
: Master NameManager service

`-f filename`
: Location of FactoryFinder import file

The FactoryFinder service is a CORBA-derived service that provides client applications with the ability to find application factories that correspond to application-specified search criteria. Consult the *C++ Programming Reference* for a complete description on the FactoryFinder API and *Creating C++ Server Applications* for a description of registering and unregistering factories. The FactoryFinder service is the "default" service if no services are specified in the CLOPT.

The NameManager service is a WLE-specific service that maintains a mapping of application-supplied names to object references. One usage of this service is to maintain the application factory name-to-object reference list. The NameManager service can be booted with an `-M` option that designates a Master role. If the `-M` option is not specified, the NameManager is assumed to be a Slave. Slave NameManagers obtain updates from the Master. Only one Master NameManager can be specified in an application.

The master NameManager can be configured to support the location of factory objects that reside in a remote domain through the use of an initialization file (for example `import_factories.ini`). The location of the initialization file is specified with the `-f` command-line option. If the `-f` option is not specified, the Master NameManager will search the directory specified by the `APPDIR` environment variable for a file with the name `factory_finder.ini`. If the file is not found, the Master NameManager will continue initialization, but will be unable to locate factory objects in domains other than the local one.

Note: It is possible to boot one or more `TMFFNAME` processes running the same service. To provide increased reliability, at least two NameManager services must be configured, preferably on different machines.

Interoperability

The TMFFNAME servers run on WLE version 4.0 software and later.

Notes

If there are less than two NameManager services configured in the application's `UBBCONFIG` (`TMFFNAME -N`), the server terminates itself during boot and writes an error message to the user log.

If a Master NameManager service is not configured in the application's `UBBCONFIG` file and is running when a Slave NameManager service starts, the server terminates itself during boot and writes an error message to the user log. Additionally, if the Master is down, registration and unregistration of factories is disabled until the Master restarts.

If a `TMSYSEVT` server is not configured in the application's `UBBCONFIG` file and is not running when a NameManager service is being started, the server terminates itself during boot and writes an error message to the user log.

If a NameManager service is not configured in the application's `UBBCONFIG` file and a FactoryFinder service is being started, the server terminates itself during boot and writes an error message to the user log.

Example

```
*SERVERS
TMSYSEVT SRVGRP=ADMIN1 SRVID=44 RESTART=Y
          CLOPT="-A"

TMFFNAME SRVGRP=ADMIN1 SRVID=45 RESTART=Y
CLOPT="-A -- -F"

TMFFNAME SRVGRP=ADMIN1 SRVID=46 RESTART=Y
CLOPT="-A -- -N –M –f c:\appdir\import_factories.ini"
TMFFNAME SRVGRP=ADMIN2 SRVID=47 RESTART=Y
CLOPT="-A -- -N"
TMFFNAME SRVGRP=ADMIN3 SRVID=48 RESTART=Y
```

```
CLOPT="-A -- -F"
TMFFNAME SRVGRP=ADMIN4 SRVID=49 RESTART=Y
CLOPT="-A -- -F"
```

See Also `TMSYSEVT`(5), `userlog`(3), `UBBCONFIG`(5), and the TP Framework chapter in the *C++ Programming Reference*.

TMIFRSVR

Synopsis The Interface Repository server.

Syntax

```
TMIFRSVR SRVGRP="identifier" SRVID="number" RESTART=Y GRACE=0
CLOPT="[ servopts options  ] -- [ -f repository_file_name ]"
```

Description The `TMIFRSVR` server is a server provided by BEA for accessing the Interface Repository. The API is a subset of the CORBA-defined Interface Repository API. For a description of the Interface Repository API, see the *C++ Programming Reference*.

Parameter

`[-f repository_file_name]`

Interface Repository file name. This file must have been generated previously using the `idl2ir` command. If this parameter is not specified, the default repository file name `repository.ifr` located in the application directory (`APPDIR`) for the machine is used. If the repository file cannot be read, the server fails to boot.

Examples

```
*SERVERS

#This server uses the default repository TMIFRSVR
SRVGRP="IFRGRP" SRVID=1000 RESTART=Y GRACE=0

#This server uses a non-default repository TMIFRSVR
SRVGRP="IFRGRP" SRVID=1001 RESTART=Y GRACE=0
CLOPT="-- -f /nfs/repository.ifr"
```

See Also `idl2ir`, `UBBCONFIG`(5), and `servopts`(5)

factory_finder.ini

Name ASCII FactoryFinder domain configuration file

Description `factory_finder.ini` is the FactoryFinder configuration file for domains. This file is parsed by the `TMFFNAME` service when it is started as a Master NameManager. The file contains information used by NameManagers to control the import and the export of object references for factory objects with other domains.

Definitions A WLE system domain application is defined as the environment described in a single `TUXCONFIG` file. A WLE system application can communicate with another WLE system application or with another TP application via a domain gateway group. In "WLE system domain" terms, an application is the same as a TP domain.

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

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

File Format The file is made up of two specification sections. Allowable section names are: `DM_REMOTE_FACTORIES` and `DM_LOCAL_FACTORIES`.

- Formatting Guidelines

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

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

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

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

  "#" introduces a comment. A newline ends a comment.

  Blank lines and comments are ignored.

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

- `DM_ACCESS_CONTROL` section

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

  Lines within this section have the form:

  ```
  ACL_NAME ACLIST = identifier [,identifier]
  ```

  where `ACL_NAME` is the name (`identifier`) of an access control list and it must be 15 characters or less in length.

  Required parameters are:

  ```
  ACLIST = identifier [, identifier]
  ```

  This parameter specifies one or more remote domain names (`RDOM`), separated by commas. The wildcard character (*) can be used to specify all remote domains that can retrieve factory objects in the local domain.

- `DM_LOCAL_FACTORIES` section

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

  Lines within this section have the form:

  ```
  factory_id.factory_kind
  ```

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

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

- `DM_REMOTE_FACTORIES` section

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

  ```
  factory_id.factory_kind required parameters
  ```

  where `factory_id.factory_kind` is the name (`identifier`) of the factory object used by the local WLE system domain application for a particular remote

factory object. Remote factory objects are associated with a particular remote domain.

The required parameter is:

```
DOMAINID = domain_id
```

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

The optional parameter is:

```
RNAME = string
```

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

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

Examples

♦ **Example 1**

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

```
# BEA WebLogic Enterprise FactoryFinder Domain
# Configuration File
#
*DM_REMOTE_FACTORIES
  Teller.FactoryIdentity
    DOMAINID="Northwest"
    RNAME=Teller.FactoryType
  Teller.FactoryIdentity
     DOMAINID="Southwest"
```

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

In the second entry, a factory object is to be imported from the remote domain with an identity of "Southwest" that has been registered with a factory identity

of `Teller.FactoryIdentity`. Note that because no `RNAME` parameter was specified, the name of the factory object in the remote domain is assumed to be the same as the factory's name in the local domain.

- **Example 2**

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

```
# BEA WebLogic Enterprise FactoryFinder Domain
# Configuration File
#
*DM_LOCAL_FACTORIES
  Teller.FactoryInterface
```

- **Example 3**

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

```
# BEA WebLogic Enterprise FactoryFinder Domain
# Configuration File
#
*DM_LOCAL_FACTORIES
   NONE
```

See Also `TMFFNAME`, *BEA TUXEDO /Domain Guide*, *C++ Programming Reference*, and *Java Programming Reference*

CHAPTER

23 Troubleshooting Applications

Other chapters of this document discuss many diagnostic tools provided by your WLE 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 chapter discusses the following topics:

- 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 (WLE Servers)
- Checking Hostname Format and Capitalization (WLE Servers)

- Some Clients Fail to Boot (WLE Servers)
- Checking the Order in Which Servers Are Booted (WLE 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 WLE 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 WLE 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 WLE or BEA TUXEDO System Failure

To detect the source of a system failure, complete the following steps:

1. Check any WLE 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 WLE 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, WLE 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 23-1 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 23-1 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 procedure.

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

Perform the following steps to recover from severe network failure.

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 procedure.

1. On the master machine, start a `tmadmin` session.
2. Run `pclean`, specifying the partitioned machine on the command line.
3. Fix the machine problem.

4. Restore the failed machine by booting the Bulletin Board 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 23-2, `SITE2`, a nonmaster machine, is restored.

Listing 23-2 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 procedure.

1. Install the BEA TUXEDO system software that is being replaced.
2. Shut down those parts of the application that will be affected by the changes:
   - The BEA TUXEDO system servers may need to be shut down if libraries are being updated.

- Application clients and servers must be shut down and rebuilt if relevant BEA TUXEDO system header files or static libraries are being replaced. (Application clients and servers do not need to be rebuilt if the BEA TUXEDO system message catalogs, system commands, administrative servers, or shared objects are being replaced.)

3. If relevant BEA TUXEDO system header files and static libraries have been replaced, rebuild your application clients and servers.
4. Reboot the parts of the application that you shut down.

Replacing Application Components

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

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

Cleaning Up and Restarting Servers Manually

By default, the WLE 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 procedure.

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

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

If the Specified Machine Is . . .	Then . . .
Not partitioned	`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 (WLE Servers)

If a WLE 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 WLE system. A WLE 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 (WLE Servers)" in Chapter 3, "Creating a Configuration File."

Checking Hostname Format and Capitalization (WLE 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 (WLE Servers)

You may want to perform the following steps on a Windows NT server that is running a WLE 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 (WLE Servers)" in this chapter.)

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 WLE 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 WLE or BEA TUXEDO applications.

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

Index

Symbols

A

B

C

D

E

F

G

I

J

K

L

M

N

O

T

U

V

W