



BEA Tuxedo®

Setting Up a BEA Tuxedo Application

Copyright

Copyright © 1995-2006 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software is protected by copyright, and may be protected by patent laws. No copying or other use of this software is permitted unless you have entered into a license agreement with BEA authorizing such use. This document is protected by copyright and may not be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form, in whole or in part, without prior consent, in writing, from BEA Systems, Inc.

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

Trademarks and Service Marks

Copyright © 1995-2006 BEA Systems, Inc. All Rights Reserved. BEA, BEA JRockit, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop, Built on BEA, Jolt, JoltBeans, SteelThread, Top End, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA AquaLogic, BEA AquaLogic Data Services Platform, BEA AquaLogic Enterprise Security, BEA AquaLogic Interaction, BEA AquaLogic Interaction Analytics, BEA AquaLogic Interaction Collaboration, BEA AquaLogic Interaction Content Services, BEA AquaLogic Interaction Data Services, BEA AquaLogic Interaction Integration Services, BEA AquaLogic Interaction Process, BEA AquaLogic Interaction Publisher, BEA AquaLogic Interaction Studio, BEA AquaLogic Service Bus, BEA AquaLogic Service Registry, BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Kodo, BEA Liquid Data for WebLogic, BEA Manager, BEA MessageQ, BEA SALT, BEA Service Architecture Leveraging Tuxedo, BEA WebLogic Commerce Server, BEA WebLogic Communications Platform, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Enterprise Security, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Java Adapter for Mainframe, BEA WebLogic JDriver, BEA WebLogic Log Central, BEA WebLogic Mobility Server, BEA WebLogic Network Gatekeeper, BEA WebLogic Personalization Server, BEA WebLogic Personal Messaging API, BEA WebLogic Platform, BEA WebLogic Portlets for Groupware Integration, BEA WebLogic Real Time, BEA WebLogic RFID Compliance Express, BEA WebLogic RFID Edge Server, BEA WebLogic RFID Enterprise Server, BEA WebLogic Server Process Edition, BEA WebLogic SIP Server, BEA WebLogic WorkGroup Edition, BEA Workshop for WebLogic Platform, BEA Workshop JSP, BEA Workshop JSP Editor, BEA Workshop Struts, BEA Workshop Studio, Dev2Dev, Liquid Computing, and Think Liquid are trademarks of BEA Systems, Inc. Accelerated Knowledge Transfer, AKT, BEA Mission Critical Support, BEA Mission Critical Support Continuum, and BEA SOA Self Assessment are service marks of BEA Systems, Inc.

All other names and marks are property of their respective owners.

Contents

About This Document

What You Need to Know	xvi
e-docs Web Site	xvi
How to Print the Document	xvi
Related Information	xvii
Contact Us!	xvii
Documentation Conventions	xix

Administrative Tasks and Tools

Tasks an Administrator Performs	1-1
Setup Tasks	1-2
Run-time Tasks	1-3
Differences Between the BEA Tuxedo ATMI and CORBA Environments	1-4
Planning the Design of Your Application	1-6
Tools to Help You Administer Your Application	1-8

About the Configuration File

What Is the Configuration File?	2-1
Text and Binary Versions of the Configuration File	2-1
Contents of the Configuration File.	2-2
CORBA Administrative Requirements and Performance	2-4
Configuring NameManager	2-4
Reliability Requirements	2-5

Performance Hint	2-5
----------------------------	-----

Creating the Configuration File

How to Create a Configuration File	3-2
How to Create the Configuration File for a Single-machine Application.	3-2
How to Create the Configuration File for a Multiple-machine (Distributed) Application	3-3
How to Create the Configuration File for a Multiple-domain Application.	3-4
How to Create the RESOURCES Section of the Configuration File	3-7
Sample RESOURCES Section.	3-9
Defining the Application Type	3-9
Characteristics of the MODEL and OPTIONS Parameters	3-10
Example Settings	3-10
Controlling the Number of Buffer Types and Subtypes	3-10
Characteristics of the MAXBUFTYPE and MAXBUFSTYPES Parameters	3-11
Example Settings	3-11
Controlling the Number of Conversations	3-11
Characteristics of the MAXCONV Parameter.	3-11
Example Setting	3-12
Defining IPC Limits	3-12
Characteristics of MAXACCESSERS, MAXSERVERS, MAXSERVICES, MAXINTERFACES, and MAXOBJECTS Parameters	3-13
Example Settings	3-15
Enabling Load Balancing	3-15
Characteristics of the LDBAL Parameter	3-15
Example Settings	3-16
Identifying the Master Machine	3-16
Characteristics of the MASTER Parameter	3-16
Example Settings	3-17

Specifying the Maximum Number of Network Groups	3-17
Specifying the Number of Sanity Checks and Blocking Timeouts	3-17
Characteristics of the SCANUNIT, SANITYSCAN, and BLOCKTIME Parameters	3-18
Timeouts for Blocking ATMI Operations	3-18
Example Settings	3-19
Establishing Operating System-level Security	3-19
Characteristics of the UID, GID, and PERM Parameters	3-20
Specifying the Security Level	3-20
Characteristics of the SECURITY and AUTHSVC Parameters	3-21
Defining the Security Attributes of a Server	3-22
Protecting Shared Memory	3-23
Characteristics of the PROTECTED, FASTPATH, and NO_OVERRIDE Parameters ..	3-24
Example Settings	3-24
Setting the Address of the System Resources for an Application	3-24
Characteristics of the IPCKEY Parameter	3-24
Example Settings	3-25
Specifying How Clients Receive Unsolicited Notification	3-25
Characteristics of the NOTIFY and USIGNAL Parameters	3-26
How to Create the MACHINES Section of the Configuration File	3-26
Sample MACHINES Section	3-30
Specifying the Maximum Number of ACL Entries in the Cache	3-32
Defining an Additional Service Request Load	3-32
Reserving the Physical Address and Machine ID	3-32
Characteristics of the Address and the LMID Parameter	3-33
Setting the Number of Lock Spins	3-33
Characteristics of the SPINCOUNT Parameter	3-33
Specifying Machines as Types	3-34

Characteristics of the TYPE Parameter	3-34
Identifying the Location of the Configuration File	3-34
Characteristics of the TUXCONFIG Parameter	3-34
Indicating the Size of the DTP Transaction Log	3-35
Defining the DTP Transaction Log Name	3-35
Specifying Environment Variable Settings	3-35
Characteristics of the ENVFILE Parameter	3-36
Defining the BEA Tuxedo Filesystem Containing the TLOG	3-36
Specifying a Machine's Maximum Number of Simultaneous Global Transactions	3-36
Defining the Number of Accesser Entries on a Workstation Client	3-36
Defining Space Limits for Messages Transmitted by the BRIDGE	3-37
Indicating the Offset for the DTP Transaction Log	3-37
Defining the Offset for TUXCONFIG	3-38
Characteristics of the TUXOFFSET Parameter	3-38
Identifying the Locations of the System Software and Application Server Software	3-38
Characteristics of the APPDIR and TUXDIR Parameters	3-39
Indicating a Threshold Message Size for Compression	3-39
Example	3-39
Specifying the Pathname for the ULOG	3-39
Characteristics of the ULOGPFX Parameter	3-40
How to Create the GROUPS Section of the Configuration File	3-41
Sample GROUPS Section for ATMI	3-42
Sample GROUPS Section for CORBA	3-42
Specifying a Group Name, Number, and LMID	3-44
Characteristics of the Group Name, Group Number, and LMID	3-44
Indicating a Transaction Manager Server Name and Numbers per Group	3-45
Identifying the Environment File Location for Servers in a Group	3-45
Defining Information Needed When Opening and Closing the Resource Manager	3-46

How to Create the NETWORK Section of the Configuration File	3-48
Sample NETWORK Section	3-48
Specifying a Device Name for the BRIDGE Process	3-49
Assigning a BRIDGE Network Address	3-49
Assigning Encryption Levels.	3-50
Example	3-50
Assigning a tlisten Network Address	3-50
How to Create the NETGROUPS Section of the Configuration File	3-52
Sample Network Groups Configuration.	3-52
Configuring a Sample UBBCONFIG File with Netgroups	3-54
Assigning a Name to a Network Group.	3-55
Assigning a Network Group Number	3-55
Assigning a Priority to the Network Group.	3-56
How to Create the SERVERS Section of the Configuration File	3-56
Sample SERVERS Section.	3-60
Specifying a Server as Conversational	3-62
Characteristics of the CONV Parameter.	3-62
Setting the Order in Which Servers Are Booted	3-63
Required Order in Which to Boot CORBA C++ Servers	3-63
Characteristics of the SEQUENCE, MIN, and MAX Parameters.	3-66
Specifying Server Command-line Options	3-67
Characteristics of the CLOPT Parameter	3-68
Identifying the Location of the Server Environment File	3-68
Characteristics of the Server Environment File	3-69
Defining Server Name, Group, and ID	3-69
Characteristics of the Server Name, SRVGRP, and SRVID Parameters	3-70
Identifying Server Queue Information.	3-70
MSSQ Example	3-70

Characteristics of the RQADDR, RQPERM, REPLYQ, and RPPERM Parameters	3-71
Defining Server Restart Information	3-72
Characteristics of the RESTART, RCMD, MAXGEN, and GRACE Parameters . .	3-72
Defining Server Access to Shared Memory	3-73
Characteristics of the SYSTEM_ACCESS Parameter	3-73
Defining the Server Dispatch Threads	3-73
Setting Security Parameters for ISL Servers	3-75
How to Create the SERVICES Section of the Configuration File	3-75
Sample SERVICES Section	3-77
Specifying Automatic Starts and Timeout Intervals for Transactions	3-77
Specifying a List of Allowable Buffer Types for a Service	3-78
Examples of the BUFTYPE Parameter	3-78
Designating How Much Time to Process a Request	3-78
What Happens When a Timeout Occurs	3-79
How a Service Timeout Is Reported	3-79
Specifying Nontransactional Service-Level Blocktime	3-80
Enabling Load Balancing	3-81
Characteristics of the LDBAL Parameter	3-81
Defining the Name of the Routing Criteria	3-82
Specifying Service Parameters for Different Server Groups	3-82
Controlling the Flow of Data by Service Priority	3-82
Characteristics of the PRIO Parameter	3-82
Sample SERVICES Section Using Different Priorities	3-83
Indicating Service Processing Time	3-83
How to Create the INTERFACES Section of the Configuration File	3-83
Specifying CORBA Interfaces in the INTERFACES Section	3-84
Specifying FACTORYROUTING Criteria	3-86
Enabling Load Balancing	3-88

Controlling the Flow of Data by Interface Priority	3-88
Specifying Different Interface Parameters for Different Server Groups	3-89
How to Create the ROUTING Section of the Configuration File	3-89
ROUTING Section Example	3-90
Defining the Routing Buffer Field and Field Type	3-90
Specifying Range Criteria	3-92
Defining Buffer Types	3-92
CORBA Factory-based Routing in the University Production Sample Application	3-92
CORBA Factory-based Routing in the Bankapp Sample Application	3-96
How to Configure the BEA Tuxedo System to Take Advantage of Threads	3-97
How to Compile a Configuration File	3-100

About Transactions

What Is a Transaction?	4-1
What Are the ACID Properties?	4-2
How a Transaction Succeeds or Fails	4-4
Benefits of Using Transactions	4-4
Example of a Global Transaction	4-5
What Is the BEA Tuxedo Transaction Manager (TM)?	4-5
How the System Tracks Distributed Transaction Processing	4-7
How the System Uses Global Transaction Identifiers (GTRIDs) for Tracking	4-7
How the System Uses a Transaction Log (TLOG) for Tracking	4-8
How the System Uses a Two-Phase Commit to Commit Transactions	4-8
How the System Handles Transaction Infection	4-9
How the ATMI Protects a Transaction's Integrity Before a Two-Phase Commit	4-10
See Also	4-11

Configuring Your ATMI Application to Use Transactions

Modifying the UBBCONFIG File to Accommodate ATMI Transactions	5-1
Specifying Global Transaction Parameters in the RESOURCES Section	5-3
Creating a Transaction Log (TLOG) in the MACHINES Section	5-4
Creating the UDL	5-4
Defining Transaction-related Parameters in the MACHINES Section	5-4
Creating the Domains Transaction Log	5-5
See Also	5-6
Defining Resource Managers and the Transaction Manager Server in the GROUPS Section	5-6
Sample of the GROUPS Section	5-6
Enabling a Service to Begin a Transaction in the SERVICES Section	5-8
Characteristics of the AUTOTRAN, TRANTIME, and ROUTING Parameters.	5-8
Modifying the Domains Configuration File to Support Transactions.	5-9
Characteristics of the DMTLOGDEV, DMTLOGNAME, DMTLOGSIZE, MAXRAPTRAN, and MAXTRAN Parameters.	5-9
Characteristics of the AUTOTRAN and TRANTIME Parameters	5-10
Example: A Distributed Application with Transactions	5-12
Sample RESOURCES Section.	5-13
Sample MACHINES Section.	5-14
Sample GROUPS and NETWORK Sections.	5-15
Sample SERVERS, SERVICES, and ROUTING Sections	5-17
See Also	5-18

Using Tuxedo with Oracle Real Application Clusters (RAC)

Overview	6-1
Limitations	6-2
Prerequisites	6-2

Software Requirements	6-2
Configuring Tuxedo for Oracle RAC	6-3
Configuring Transaction Propagation	6-3
Configuring Transaction Recovery	6-13
See Also	6-17

Managing The Tuxedo Service Metadata Repository

Tuxedo Service Metadata Repository	7-1
JOLT Repository Similarities and Differences.	7-2
MIB(5) Similarities and Differences	7-3
Creating The Tuxedo Service Metadata Repository	7-3
The Tuxedo Service Metadata Repository Input File.	7-4
Configuring The Tuxedo Service Metadata Repository Server	7-12
Configuring Multiple Tuxedo Service Metadata Repository Servers	7-12
Accessing The Tuxedo Service Metadata Repository File	7-13

Managing CORBA Interface Repositories

Administration Considerations	8-3
Using Administration Commands to Manage Interface Repositories	8-3
Prerequisites	8-4
Creating and Populating an Interface Repository	8-5
Displaying or Extracting the Content of an Interface Repository	8-5
Deleting an Object from an Interface Repository.	8-5
Configuring the UBBCONFIG File to Start One or More Interface Repository Servers	8-6

Distributing ATMI Applications Across a Network

What Is a Distributed ATMI Application?	9-1
Example of a Distributed Application	9-3

Implementing a Distributed Application	9-4
Why Distribute an ATMI Application Across a Network?	9-5
Features of a Distributed Application	9-6

Creating the Configuration File for a Distributed ATMI Application

Configuration File Requirements for a Distributed BEA Tuxedo ATMI Application . .	10-1
Creating the RESOURCES Section	10-3
Creating the MACHINES Section	10-5
Creating the GROUPS Section	10-7
Creating the SERVICES Section	10-8
Creating the ROUTING Section	10-10
Example Configuration File for a Distributed Application	10-11
Modifying the Domain Gateway Configuration File to Support Routing	10-12
Description of ROUTING Section Parameters in DMCONFIG	10-13

Setting Up the Network for a Distributed Application

Configuring the Network for a Distributed Application	11-1
How Data Moves Over a Network	11-5
How Data Moves Over Parallel Networks	11-6
Example of a Network Configuration for a Simple Distributed Application	11-8
How Failover and Failback Work in Scheduling Network Data	11-8
Example Configuration of Multiple Netgroups	11-8
Configuration File for the Sample Network	11-10
Assigning Priorities for Each Network Group	11-11

About Workstation Clients

What Is the Workstation Component?	12-1
Sample Application with Four Workstation Clients	12-2

How the Workstation Client Connects to an Application	12-4
---	------

Setting Up Workstation Clients

Defining Workstation Clients	13-1
Specifying the Maximum Number of Workstation Clients.	13-4
Defining a Workstation Listener (WSL) as a Server.	13-4
Passing Information to a WSL Process	13-5
Using Command-line Options Set with CLOPT	13-5
Detecting Network Failures.	13-7
Using the Keep-alive Option	13-8
Limitations When Using the Keep-alive Option	13-9
Using the Network Timeout Option	13-10
How Network Timeout Works	13-10
Limitations When Using the Network Timeout Option	13-10
Setting the Network Timeout Option	13-11
Sample Configuration File that Supports Workstation Clients	13-11
Modifying the MACHINES and SERVERS Sections	13-11

Managing Remote BEA Tuxedo CORBA Client Applications

CORBA Object Terminology	14-2
Remote CORBA Client Overview.	14-4
Illustration of an Application with Remote CORBA Clients.	14-4
How the Remote Client Connects to an Application	14-6
Setting Environment Variables for Remote CORBA Clients	14-6
Setting the Maximum Number of Remote CORBA Clients	14-7
Configuring a Listener for a Remote CORBA Client	14-8
Format of the CLOPT Parameter	14-8
Modifying the Configuration File to Support Remote CORBA Clients	14-8

Configuring Outbound IIOP for Remote Joint Client/Servers	14-10
Functional Description	14-10
Using the ISL Command to Configure Outbound IIOP Support	14-15
Types of Object References	14-16
User Interface	14-16

About This Document

This document explains how to plan, design, and configure the BEA Tuxedo® system, for either an data-dependent ATMI environment or an object-oriented CORBA environment.

This document includes the following topics:

- [Chapter 1, “Administrative Tasks and Tools,”](#) introduces the BEA Tuxedo administrator’s tasks.
- [Chapter 2, “About the Configuration File,”](#) provides an overview of the BEA Tuxedo configuration file, a repository that contains all the information necessary to boot and run an application, as well as an overview of the NameManager for CORBA environments.
- [Chapter 3, “Creating the Configuration File,”](#) explains how to create a configuration file for single-machine running in an ATMI or CORBA environment.
- [Chapter 4, “About Transactions,”](#) provides an overview of BEA Tuxedo transactions.
- [Chapter 5, “Configuring Your ATMI Application to Use Transactions,”](#) describes how to configure transactions in a BEA Tuxedo ATMI environment.
- [Chapter 7, “Managing The Tuxedo Service Metadata Repository,”](#) describes how to create a Service Metadata Repository file, configure the server to use the Service Metadata Repository, and access the repository to update, add, or delete service parameter information in the repository.
- [Chapter 8, “Managing CORBA Interface Repositories,”](#) describes how to create an interface repository for CORBA objects implemented within the BEA Tuxedo domain.

- [Chapter 9, “Distributing ATMI Applications Across a Network,”](#) describes how to distribute a Tuxedo ATMI application for local or remote clients over one or more servers on several machines through a single BEA Tuxedo configuration file.
- [Chapter 10, “Creating the Configuration File for a Distributed ATMI Application,”](#) describes how to create a configuration file for multiple machines in an ATMI environment.
- [Chapter 11, “Setting Up the Network for a Distributed Application,”](#) describes how to configure your network environment in order to support a distributed application within the BEA Tuxedo system.
- [Chapter 12, “About Workstation Clients,”](#) provides an explanation of the BEA Tuxedo Workstation component of the BEA Tuxedo system.
- [Chapter 13, “Setting Up Workstation Clients,”](#) describes how to configure the BEA Tuxedo environment so that a Tuxedo Workstation client can join an application.
- [Chapter 14, “Managing Remote BEA Tuxedo CORBA Client Applications,”](#) explains how to configure connections from remote CORBA client applications to CORBA objects via the standard Internet Inter-ORB Protocol (IIOP).

What You Need to Know

This document is intended mainly for administrators who configure operational parameters that support mission-critical the BEA Tuxedo systems. It assumes a familiarity with the BEA Tuxedo platform.

e-docs Web Site

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

How to Print the Document

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

A PDF version of this document is available on the BEA Tuxedo documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the

PDFs, open the BEA Tuxedo documentation Home page, click the PDF files button and select the document you want to print.

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

Related Information

The following documents provide related information about BEA Tuxedo software.

- *Installing the BEA Tuxedo System*—paper copy distributed with the CD
- *BEA Tuxedo Release Notes*—paper copy distributed with the CD
- *Administering a BEA Tuxedo Application at Run Time*—available through the BEA Tuxedo Online Documentation CD, this guide describes the command-line interface access to BEA Tuxedo administration tasks.
- *Using the BEA Tuxedo Domains Component*—available through the BEA Tuxedo Online Documentation CD, this guide describes how to configure and manage BEA Tuxedo domains.
- *Scaling, Distributing, and Tuning CORBA Applications*—available through the BEA Tuxedo Online Documentation CD, this guide describes how to tune and scale CORBA applications that run in the BEA Tuxedo environment.
- *Using CORBA Transactions*—available through the BEA Tuxedo Online Documentation CD, this guide describes how to configure CORBA transactions in a BEA Tuxedo environment.

For more information about configuring and administering BEA Tuxedo ATMI and BEA Tuxedo CORBA environments, see the *CORBA Bibliography* in the BEA Tuxedo online documentation.

Contact Us!

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

In your e-mail message, please indicate that you are using the documentation for the BEA Tuxedo 8.0 release.

If you have any questions about this version of BEA Tuxedo, or if you have problems installing and running BEA Tuxedo, contact BEA Customer Support through BEA WebSupport at

www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

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

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

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
<i>italics</i>	Indicates emphasis or book titles.
monospace text	<p>Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames and their extensions. Monospace text also indicates text that you must enter from the keyboard.</p> <p><i>Examples:</i></p> <pre>#include <iostream.h> void main () the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float</pre>
monospace boldface text	<p>Identifies significant words in code.</p> <p><i>Example:</i></p> <pre>void commit ()</pre>
<i>monospace italic text</i>	<p>Identifies variables in code.</p> <p><i>Example:</i></p> <pre>String <i>expr</i></pre>
UPPERCASE TEXT	<p>Indicates device names, environment variables, and logical operators.</p> <p><i>Examples:</i></p> <pre>LPT1 SIGNON OR</pre>

Convention	Item
{ }	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. <i>Example:</i> <pre>buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...</pre>
	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.
...	Indicates one of the following in a command line: <ul style="list-style-type: none"> • 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. <i>Example:</i> <pre>buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...</pre>
.	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

Administrative Tasks and Tools

This topic includes the following sections:

- [Tasks an Administrator Performs](#)
- [Planning the Design of Your Application](#)
- [Tools to Help You Administer Your Application](#)

Tasks an Administrator Performs

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

- [Setup tasks](#)—all the tasks required to prepare your system before booting your application.
- [Run-time administration](#)—any tasks performed on an application that has been booted.

Setup Tasks

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

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

Run-time Tasks

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

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

During run time, you may need to respond quickly to potential problems or evolving requirements of an application. To help you perform these functions, you have a choice of three

tools: the BEA Tuxedo Administration Console, the command-line interface, and the AdminAPI. The following chart describes some of the circumstances in which your intervention may be needed.

To...	You May Want to...
Maximize performance	Adds load balancing or set priorities for interfaces and services.
Fix problems that may develop on the MASTER machine	Replaces it with a designated BACKUP machine.
Change processing and resource usage requirements	Adds machines, servers, clients, interfaces, services, and so on.

See Also

- [“Planning the Design of Your Application” on page 1-6](#)
- [“Tools to Help You Administer Your Application” on page 1-8](#)

Differences Between the BEA Tuxedo ATMI and CORBA Environments

For the BEA Tuxedo CORBA environment, the BEA Tuxedo administration facilities support the administration of applications running within the context of the Object Request Broker (ORB) and the TP Framework.

The `UBBCONFIG` configuration file for BEA Tuxedo CORBA environments supports the configuration of client and server applications, as follows:

- The `RESOURCES` section provides application-wide defaults for the sizing of bulletin board tables.
- The `MACHINES` section allows the specification of processor-specific values for sizing of those tables.
- The `INTERFACES`, section allows the specification of information about CORBA interfaces used by the application.

- The `ROUTING` section provides support for a different type of routing criteria used with Tuxedo CORBA environments. Also, existing `ROUTING` sections that specify BEA Tuxedo ATMI data-dependent routing parameters continue to work without modification.
- In the BEA Tuxedo ATMI environment, you configure workstation handlers and listeners for connections from client applications to server applications. From an administrative viewpoint, this task is similar in BEA Tuxedo CORBA environments.

However, the BEA Tuxedo CORBA environment uses a different communications protocol to connect remote and foreign clients to BEA Tuxedo 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 CORBA environment calls its gateway processes the IIOP Handler (ISH) and the IIOP Listener (ISL). This results in a slight syntax difference, `ISL` instead of `WSL`, in the `SERVERS` section of each application's `UBBCONFIG` configuration file.

Overall, the administration tasks for the BEA Tuxedo CORBA and ATMI environments are similar. There are a few principal differences between the environments, however, as follows:

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

In the BEA Tuxedo ATMI environment, you can examine any FML field used for a service invocation to determine the data-dependent routing criteria. In BEA Tuxedo CORBA environments, the system designer must personally communicate the routing criteria of CORBA interfaces. For BEA Tuxedo CORBA environments, there is no service request message data or associated buffer information available for routing. This occurs because CORBA 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.
- The LDAP single security administration feature is not supported by the CORBA interface.

Note: The Management Information Base (MIB) defines the set of classes through which the fundamental aspects of an application can be configured and managed. The MIB classes

provide an administrative programming interface to the BEA Tuxedo CORBA and ATMI environments.

Planning the Design of Your Application

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

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

1. How many machines will be used? _____
2. Will client applications reside on machines that are remote from the server applications?

3. For ATMI, which services will your application offer?

4. For CORBA, which interfaces will your client or server application use?

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

6. What "open" strings will the resource managers need?

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

8. Will transactions be distributed? _____
9. Will the application use global transactions? _____

10. What buffer types will be used?

11. Will data be distributed across machines?

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

13. Will factory-based or data-dependent routing be used in your application?

14. What are the names of the CORBA interfaces or ATMI services?

15. In what order of priority should the interfaces or services be available?

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

17. For CORBA environments, 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?

18. Are there any conversational services? What resource managers do they access? What buffer types do they use?

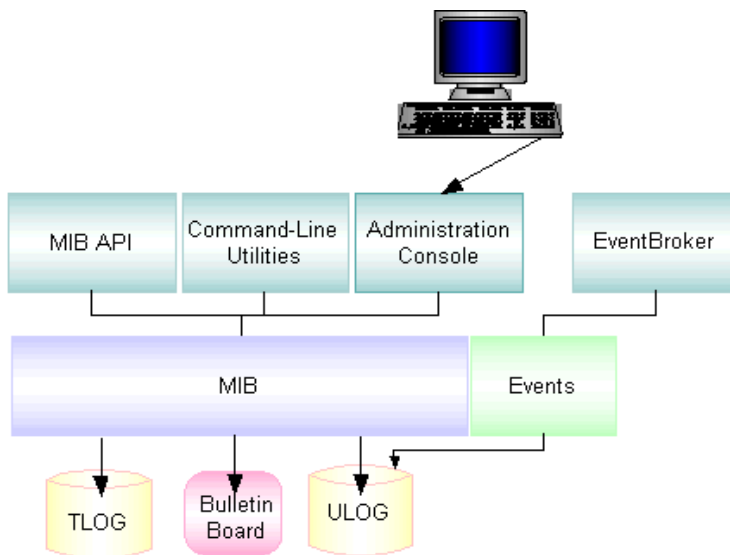
See Also

- [“Tools to Help You Administer Your Application” on page 1-8](#)

Tools to Help You Administer Your Application

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

Figure 1-1 Administration Tools



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

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

See Also

- “Management Operations Using the BEA Tuxedo Administration Console” in *Introducing BEA Tuxedo ATMI*
- “Managing Operations Using the MIB” in *Introducing BEA Tuxedo ATMI*
- “Managing Operations Using Command-Line Utilities” in *Introducing BEA Tuxedo ATMI*
- “Tasks an Administrator Performs” on page 1-1
- “BEA Tuxedo ATMI Architecture” in *Introducing BEA Tuxedo ATMI*
- “The Tuxedo CORBA Programming Environment,” in *Getting Started with BEA Tuxedo CORBA Applications*
- [ACL_MIB\(5\)](#), [APPQ_MIB\(5\)](#), [EVENT_MIB\(5\)](#), [MIB\(5\)](#), [TM_MIB\(5\)](#), [WS_MIB\(5\)](#), and [UBBCONFIG\(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*
- [tmshutdown\(1\)](#) in the *BEA Tuxedo Command Reference*

About the Configuration File

This topic includes the following sections:

- [What Is the Configuration File?](#)
- [Contents of the Configuration File](#)

What Is the Configuration File?

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

Text and Binary Versions of the Configuration File

The configuration file exists in two versions:

- The `UBBCONFIG` file is a text version of the configuration file, created and edited with any text editor. Except for sample configuration files distributed with BEA Tuxedo sample applications, no `UBBCONFIG` file is provided. You must create a `UBBCONFIG` file for each new application. The syntax used for entries in the file is described in [UBBCONFIG \(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*.

Note: The BEA Tuxedo software provides three sample `UBBCONFIG` files—`ubbsbm`, `ubbbmp`, and `ubbsimple`—as part of the `bankapp` and `simpapp` applications. (See *Tutorials for Developing BEA Tuxedo ATMI Applications*.)

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

Contents of the Configuration File

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

Section	Required or Optional	Purpose
RESOURCES	Required	Defines all system parameters.
MACHINES	Required	Specifies all the machines in your application.
GROUPS	Required	Defines all groups, group names, and group IDs for your application.
SERVERS	Optional	Specifies the initial conditions for servers started in the system.
SERVICES	Optional	Provides information on services used by the application.
INTERFACES	Optional	For CORBA environments, provides information on application-wide, default parameters for interfaces used by the application.
NETWORK	Optional	Describes the network configuration for a LAN environment.
NETGROUPS	Optional	Describes the network groups available to the application in the LAN environment.
ROUTING	Optional	Provides information for data-dependent routing of service requests using FML buffers and views.

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

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

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

CORBA Administrative Requirements and Performance

This section provides information to assist you in administering your CORBA environment in the BEA Tuxedo system.

Configuring NameManager

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

- NameManagers should coordinate their activities with each other using the BEA Tuxedo EventBroker without administrative or operations intervention. The EventBroker must be started before any servers provide the NameManager service. If the EventBroker is not configured into the application and is not running when the NameManager service is booted, the NameManager aborts its startup and writes an error message to the user log.
- At least two servers must be configured to run the NameManager service as part of any application. This requirement is to ensure that a working copy of the “name-to-IOR” mapping is always available. If the servers are on different machines, and one machine crashes, when the machine and application are restarted, the new NameManager obtains the mapping from the other NameManager. If an application is solely contained on one machine and the machine crashes, the NameManagers are rebooted as part of the application startup because the application must be rebooted. If two NameManagers are not configured in the application when a NameManager service is booted, the NameManager aborts its startup and writes an error message to the user log.
- NameManagers can be designated as either *master* or *slave*, the default being slave. If a master NameManager server is not configured in the application and is not running when a slave NameManager server starts, the server terminates itself during boot and writes an error message to the user log.
- If a NameManager service is not configured in the application when a FactoryFinder service is booted, the FactoryFinder aborts its startup and writes an error message to the user log. It is not necessary for the NameManager service to start before a FactoryFinder service because the FactoryFinder only communicates with a NameManager when a “find” request is received from an application. NameManagers, on the other hand, attempt to communicate with each other when they boot. FactoryFinders do not communicate with each other except when a request is received to find a factory that is in a remote domain.
- BEA Tuxedo EventBroker, NameManager, and FactoryFinder services must be started before any of the application-specific servers. However, if more than one EventBroker is to be configured in the application, all secondary EventBrokers 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 EventBrokers after the application servers.

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

Reliability Requirements

This section contains information that will improve CORBA reliability.

Managing Factory Entries

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

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

Configuring Multiple NameManagers and FactoryFinders

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

Designating a Master NameManager

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

Performance Hint

You can optimize FactoryFinder and NameManager performance by running these services on separate servers within the same machine rather than running these services on different

machines. This provides a quicker response because it eliminates the need for machine-to-machine communication.

See Also

- [“How to Create a Configuration File” on page 3-2](#)
- [“How to Create the Configuration File for a Multiple-machine \(Distributed\) Application” on page 3-3.](#)
- [“BEA Tuxedo Domains \(Multiple-Domain\) Servers”](#) in *Introducing BEA Tuxedo ATMI*
- [“How to Create the TUXCONFIG File” on page 1-4](#) in *Administering a BEA Tuxedo Application at Run Time*
- For distributed BEA Tuxedo CORBA applications, refer to the *Scaling, Distributing, and Tuning CORBA Applications* guide.

Creating the Configuration File

This topic includes the following sections:

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

How to Create a Configuration File

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

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

See Also

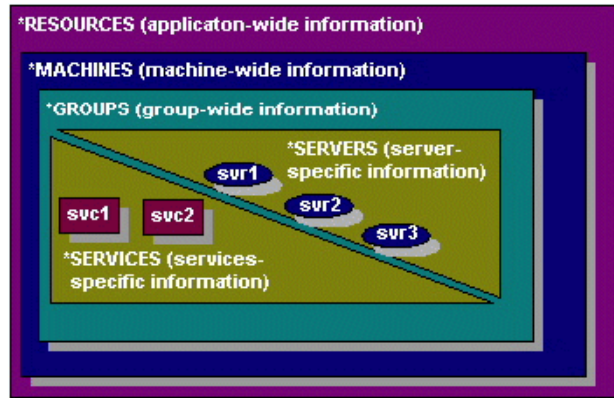
- [“About the Configuration File” on page 2-1](#)
- [UBBCONFIG \(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*

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

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

1. [Create the RESOURCES section of the configuration file](#)
2. [Create the MACHINES section of the configuration file](#)
3. [Create the GROUPS section of the configuration file](#)
4. [Create the SERVERS section of the configuration file](#)
5. [Create the SERVICES section of the configuration file](#)
6. [Create the INTERFACES section of the configuration file \(CORBA only\)](#)
7. [Create the ROUTING section of the configuration file](#)

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



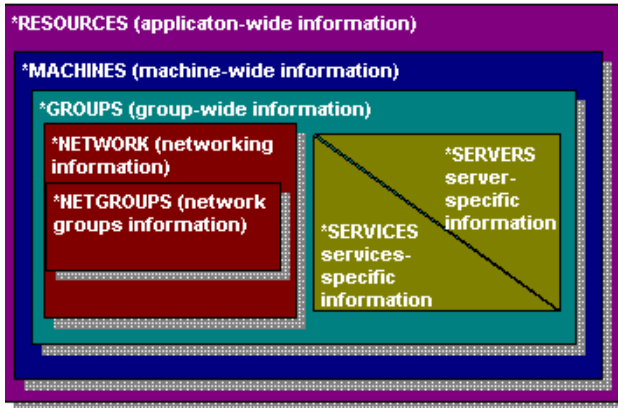
How to Create the Configuration File for a Multiple-machine (Distributed) Application

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

1. [Create the `RESOURCES` section of the configuration file](#)
2. [Create the `MACHINES` section of the configuration file](#)
3. [Create the `GROUPS` section of the configuration file](#)
4. [Create the `NETWORK` section of the configuration file](#)
5. [Create the `NETGROUPS` section of the configuration file](#)
6. [Create the `SERVERS` section of the configuration file](#)
7. [Create the `SERVICES` section of the configuration file](#)
8. [Create the `ROUTING` section of the configuration file \(optional\)](#)

Note: For detailed information about creating a configuration file for a distributed CORBA application in the BEA Tuxedo system, refer to the *Scaling, Distributing, and Tuning CORBA Applications* guide.

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



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

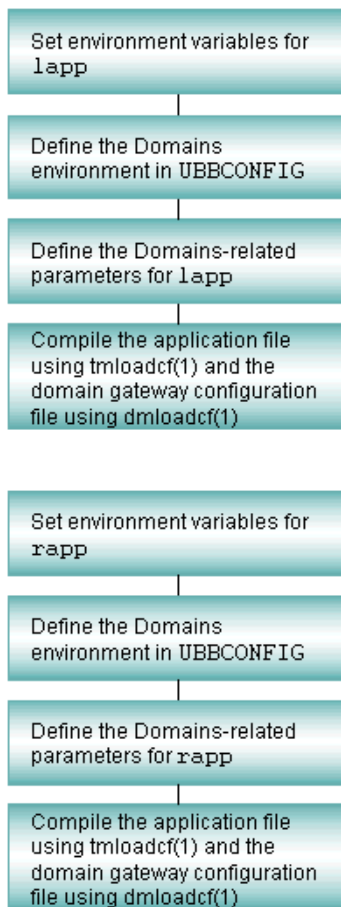
For a multiple-domain configuration, you need to create two configuration files for each participating domain:

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

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

Click on each task for instructions on completing that task.

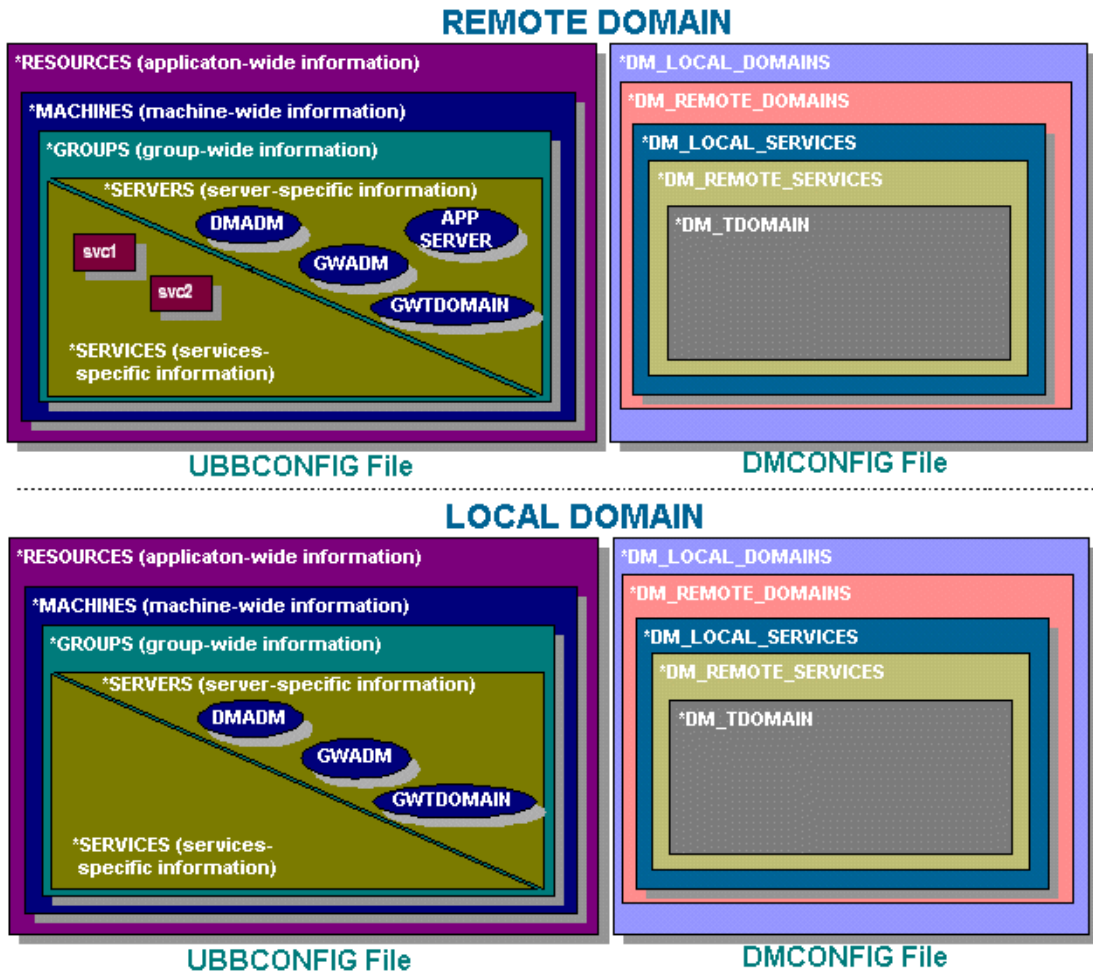
Figure 3-1 Configuration Tasks for a Sample Multiple-domain Application



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

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

Figure 3-2 Configuring a Multiple-domain Application



See Also

- “About Domains” on page 1-1 in *Using the BEA Tuxedo Domains Component*

- [“Planning and Configuring ATMI Domains”](#) in *Using the BEA Tuxedo Domains Component*
- [DMCONFIG\(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*

How to Create the RESOURCES Section of the Configuration File

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

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

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

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

Sample RESOURCES Section

The following is a sample `RESOURCES` section of a configuration file.

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

See Also

- [UBBCONFIG \(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*
- [“How to Create the MACHINES Section of the Configuration File”](#) on page 3-26

Defining the Application Type

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

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

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

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

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

Characteristics of the `MODEL` and `OPTIONS` Parameters

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

Example Settings

The following is a sample setting in the `RESOURCES` section of a configuration file.

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

Controlling the Number of Buffer Types and Subtypes

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

Characteristics of the MAXBUFTYPE and MAXBUFSTYPES Parameters

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

Example Settings

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

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

Controlling the Number of Conversations

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

Characteristics of the MAXCONV Parameter

The MAXCONV parameter has the following characteristics:

- It defines the maximum number of simultaneous conversations allowed on each machine.
- The default for an application that has conversational servers listed in the SERVERS section is 10; otherwise, the default is 1.

- You can overwrite this parameter for any machine by specifying a different value in the `MACHINES` section.

Example Setting

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

```
*RESOURCES
      MAXCONV      15
```

Defining IPC Limits

Because most interprocess communication (IPC) and shared memory bulletin board tables are statically allocated for speedy processing, it is important to tune them correctly. If they are sized too generously, memory and IPC resources are wasted; if too small, processes fail when the limits are exceeded. You can use the `tmloadcf -c` command to find out the maximum IPC resources required by a specific application. (See [tmloadcf\(1\)](#) in the *BEA Tuxedo Command Reference*.)

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

Characteristics of MAXACCESSERS, MAXSERVERS, MAXSERVICES, MAXINTERFACES, and MAXOBJECTS Parameters

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

Parameter	Characteristics
MAXINTERFACES	<p>For CORBA environments, the maximum number of CORBA interfaces that can be advertised in the application. The value of MAXINTERFACES must be greater than 0 and less than 32,766. If not specified, the default is 100.</p> <p>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.</p>
MAXOBJECTS	<p>For CORBA environments, the maximum number of active CORBA objects in the application. The value of MAXOBJECTS must be greater than 0 and less than 32,766. If not specified, the default is 100.</p>

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

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

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

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

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

Example Settings

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

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

Enabling Load Balancing

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

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

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

Characteristics of the LDBAL Parameter

The `LDBAL` parameter has the following characteristics:

- If `LDBAL` is set to `Y`, then load balancing is used.
- If `LDBAL` is set to `Y` and the application is networked, you can use `TMNETLOAD` for local preference.
- If `LDBAL` is set to `N`, the server assigned is the first available server.

- The default is N.
- Because LDBAL incurs overhead, use it only when necessary.
- Do not use load balancing if every BEA Tuxedo service is offered by only one server.
- Do not use load balancing if every BEA Tuxedo service is offered by one MSSQ server set.

Example Settings

In this example, load balancing is enabled for the application.

```
*RESOURCES
    LDBAL    Y
```

See Also

- [“What Is Load Balancing?”](#) in *Introducing BEA Tuxedo ATMI*

Identifying the Master Machine

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

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

Characteristics of the MASTER Parameter

The MASTER parameter has the following characteristics:

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

Example Settings

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

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

Specifying the Maximum Number of Network Groups

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

Specifying the Number of Sanity Checks and Blocking Timeouts

Periodically (every 120 seconds, by default) the Bulletin Board Liaison (BBL) checks the sanity of the servers on its machine. You can change the frequency of these checks, however, by setting the `SCANUNIT` and `SANITYSCAN` parameters.

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

In addition, you can specify the number of timeout periods for blocking messages, transactions, and other system activities by setting the `BLOCKTIME` parameter.

Note: *Nontransactional* blocking time values can be set on a per service, per ATMI call, and per context basis. These blocktime values override the system-wide default `BLOCKTIME` values set in the `RESOURCES` section of the `UBBCONFIG` file. For further information see [Specifying Nontransactional Service-Level Blocktime](#).

Characteristics of the SCANUNIT, SANITYSCAN, and BLOCKTIME Parameters

Parameter	Characteristics
SCANUNIT	Controls the granularity of check intervals and timeouts. SCANUNIT must be a multiple of 2 or 5 between 0 and 60 seconds. Example: SCANUNIT 10 The default is 10.
SANITYSCAN	Specifies how many scan units elapse between sanity checks of the servers. SANITYSCAN may be any number up to 32,767. The default is such that SCANUNIT * SANITYSCAN is approximately 120 seconds.
BLOCKTIME	Controls how long a message can block before it times out. SCANUNIT * BLOCKTIME must not exceed 32,767. The default is such that SCANUNIT * BLOCKTIME is approximately 60 seconds.

Timeouts for Blocking ATMI Operations

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

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

The term *blocking timeout* refers to the amount of time spent by a client request waiting for a blocking condition to clear up. Block timeouts for asynchronous service requests and conversations apply to individual send and receive operations. When a process sends a message using `tpacall (3c)`, `tpconnect (3c)`, or `tpsend (3c)`, the timeout applies only to the period during which the request waits to get on the queue if the queue is full. When a client process

issues a `tpgetrply(3c)` or `tprecv(3c)` call to receive a message, the timeout specifies how long the client may wait for the incoming message if its queue is empty.

Example Settings

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

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

Establishing Operating System-level Security

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

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

Characteristics of the UID, GID, and PERM Parameters

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

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

Specifying the Security Level

You can set the following three levels of security:

- `PERM` parameter—provides minimal security by restricting, through permissions, the ability to write to the application queues.
- `SECURITY` parameter—provides greater security. When this parameter is set, a client must supply a password when joining the application. This password is checked against the

password supplied by the administrator when the `TUXCONFIG` file is generated from the `UBBCONFIG` file.

- **AUTHSVC** parameter—sets the maximum level of security. When this parameter is set, any client request to join the application is sent to an authentication service. The authentication service may be the default service supplied by the BEA Tuxedo system or a third-party vendor service, such as a Kerberos service. This level of security cannot be used unless the **SECURITY** parameter is set.
- **LAUTHSVR** must be set in the **SERVERS** section of the `UBBCONFIG` file to enable LDAP single security administration.

Characteristics of the SECURITY and AUTHSVC Parameters

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

See Also

- [“Introducing ATMI Security”](#) in *Using Security in ATMI Applications*
- *Using Security in CORBA Applications*
- *File Formats, Data Descriptions, MIBs, and System Processes Reference*
- *BEA Tuxedo Command Reference*

Defining the Security Attributes of a Server

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

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

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

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

See Also

- [“Introducing ATMI Security”](#) in *Using Security in ATMI Applications*
- [“Administering Security”](#) in *Using Security in CORBA Applications*

Protecting Shared Memory

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

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

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

Characteristics of the PROTECTED, FASTPATH, and NO_OVERRIDE Parameters

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

Example Settings

```
SYSTEM_ACCESS PROTECTED, NO_OVERRIDE
```

Setting the Address of the System Resources for an Application

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

Characteristics of the IPCKEY Parameter

The `IPCKEY` parameter has the following characteristics:

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

Example Settings

*RESOURCES

IPCKEY 39211

Specifying How Clients Receive Unsolicited Notification

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

Following are four possible methods:

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

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

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

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

Note: This method is not available on all platforms.

Characteristics of the NOTIFY and USIGNAL Parameters

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

How to Create the MACHINES Section of the Configuration File

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

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

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

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

To Specify This Information in the <code>MACHINES</code> Section	Set This Parameter (Required/Optional)	For More Information, Click the Following
The number of entries in the cache used for ACL entries when <code>SECURITY</code> is set to <code>ACL</code> or <code>MANDATORY_ACL</code> .	<code>MAXACLCACHE</code> (Optional)	ACL entries in the cache
The additional load to be added when computing the cost of sending a service request from this machine to another machine.	<code>NETLOAD</code> (Optional)	Additional loads
The address is the name of the physical processor, which all other entries describe. The <code>LMID</code> parameter specifies the logical name of the computer.	<code>LMID</code> (Required)	Address and machine ID
The number of attempts that should be made at user level to lock the bulletin board before blocking processes on a UNIX semaphore.	<code>SPINCOUNT</code> (Optional)	Bulletin board locking limit
A value used for grouping machines into classes.	<code>TYPE</code> (Optional)	Class grouping value
The absolute pathname of the file or device where the binary <code>TUXCONFIG</code> file is found on this machine. Note: The pathname specified for this parameter must match exactly (including case) the pathname specified for the <code>TUXCONFIG</code> environment variable. Otherwise, <code>tmloadcf(1)</code> cannot be run successfully.	<code>TUXCONFIG</code> (Required)	Configuration file location
The maximum number of simultaneous conversations in which processes on a particular machine can be involved.	<code>MAXCONV</code> (Optional)	Conversation limits
The numeric size, in pages, of the DTP transaction log for this machine.	<code>TLOGSIZE</code> (Optional)	DTP TLOG size
The name of the DTP transaction log for this machine.	<code>TLOGNAME</code> (Optional)	DTP transaction log name

To Specify This Information in the MACHINES Section	Set This Parameter (Required/Optional)	For More Information, Click the Following
A value that specifies that all clients and servers on the machine are to be executed with the environment specified in the named file.	ENVFILE (Optional)	Environment variable settings
The BEA Tuxedo filesystem that contains the DTP transaction log (TLOG) for this machine.	TLOGDEVICE (Optional)	Filesystem containing the TLOG
The maximum number of processes that can have access to the bulletin board on this processor at any one time.	MAXACCESSERS (Optional)	IPC limits
For CORBA environments, the maximum number of CORBA objects that can be accommodated in the Active Object Table on this processor at any one time.	MAXOBJECTS (Optional)	IPC limits
The maximum number of simultaneous global transactions in which a particular machine can be involved.	MAXGTT (Optional)	Limit of simultaneous global transactions
The number of accesser entries on this processor to be reserved for Workstation clients. The parameter is only used when the BEA Tuxedo system Workstation component is used.	MAXWSCLIENTS (Optional)	Limit of workstation accesser entries
A limit for the amount of space that can be allocated for messages waiting to be transmitted by the bridge process.	MAXPENDINGBYTES (Optional)	Message space limits
The numeric offset in pages (from the beginning of the device) to the start of the BEA Tuxedo filesystem that contains the DTP transaction log for this machine.	TLOGOFFSET (Optional)	Numeric offset containing the DTP TLOG
The numeric offset in pages (from the beginning of the device) to the start of the BEA Tuxedo filesystem that contains the TUXCONFIG file for this machine.	TUXOFFSET (Optional)	Numeric offset containing the TUXCONFIG

To Specify This Information in the MACHINES Section	Set This Parameter (Required/Optional)	For More Information, Click the Following
The numeric group ID to be associated with the IPC structures created for the bulletin board. The valid range is 0-2147483647. If not specified, the default is the value specified in the RESOURCES section.	GID (Optional)	Security access
The numeric permissions associated with the IPC structures that implement the bulletin board. This parameter is used to specify the read/write permissions for processes in the usual UNIX system fashion (that is, with an octal number such as 0600). The value can be between 0001 and 0777, inclusive. If not specified, the default is the value specified in the RESOURCES section.	PERM (Optional)	Security access
The numeric user ID to be associated with the IPC structures created for the bulletin board. The valid range is 0-2147483647. If not specified, the default is the value specified in the RESOURCES section.	UID (Optional)	Security access
Principal name of the process used for identification, location of private key of principal user, and the environment variable containing the password	SEC_PRINCIPAL_NAME, SEC_PRINCIPAL_LOCA TION, SEC_PRINCIPAL_PASS VAR	Security attributes
The absolute pathname of the application directory (APPDIR), which is the current directory for all application and administrative servers booted on this machine; and the absolute pathname of the directory where the BEA Tuxedo system software is found on this machine.	TUXDIR (Required)	System and application software locations
The threshold message size for messages—bound to remote processes (<i>string_value1</i>) and local processes (<i>string_value2</i>), respectively—on which automatic data compression will be performed.	CMPLIMIT (Optional)	Threshold message size
The full pathname to be used as the prefix of the name of the userlog(3c) message file on this machine.	ULOGPFX (Optional)	ULOG pathname

Sample MACHINES Section

Following is a sample `MACHINES` section of a configuration file in an ATMI environment.

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

Following is a sample `MACHINES` section of a configuration file in a CORBA environment.

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

Sample MACHINES Parameters

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

Parameter	Meaning
gumby	The machine name obtained with the command <code>uname -n</code> on UNIX systems. On a Windows system, the value can be set using the Computer Name value in the Network Control Panel and must be specified in uppercase.
LMID=SITE1	The logical machine identifier of the machine <i>gumby</i> .
TUXDIR	The full path to the installed BEA Tuxedo software (shown in double quotation marks).

Parameter	Meaning
APPDIR	The full path to the application directory (shown in double quotation marks).
TUXCONFIG	<p>The full pathname of the configuration file (shown in double quotation marks).</p> <p>Note: The pathname specified for this parameter must match exactly (including case) the pathname specified for the <code>TUXCONFIG</code> environment variable. Otherwise, <code>tmloadcf(1)</code> cannot be run successfully.</p>
ENVFILE	The full pathname of a file containing environment information (shown in double quotation marks).
ULOGPFX	The full pathname to be used as the prefix of the name of the log file (shown in double quotation marks).
MAXACCESSERS	For this machine, override the system-wide value (defined in the <code>RESOURCES</code> section) with 100.
MAXOBJECTS	(For the CORBA example.) For this machine, override the system-wide value (defined in the <code>RESOURCES</code> section) with 700.
MAXCONV	For this machine, override the system-wide value (defined in the <code>RESOURCES</code> section) with 15.

How to Customize the Sample MACHINES Section

You can customize the `MACHINES` section by indicating the following:

- Your machine name for *gumby*

Note: On a Windows system, the machine name must be specified in UPPERCASE.

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

See Also

- [UBBCONFIG \(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*
- “How to Create the GROUPS Section of the Configuration File” on page 3-41

Specifying the Maximum Number of ACL Entries in the Cache

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

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

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

Defining an Additional Service Request Load

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

See Also

- “What Is Load Balancing?” in *Introducing BEA Tuxedo ATMI*

Reserving the Physical Address and Machine ID

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

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

Characteristics of the Address and the LMID Parameter

The address and machine ID have the following characteristics:

- The address and machine ID are specified as follows:

```
address LMID=logical_machine_name
```

The address identifies the physical processor name.

- The LMID is specified as follows:

```
LMID=logical_machine_name
```

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

Setting the Number of Lock Spins

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

Characteristics of the SPINCOUNT Parameter

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

- A process on a uniprocessor system should not spin. If the bulletin board is locked when a uniprocessor process tries to access it, then the process with the lock should be allowed to run as quickly as possible. This is possible only if the *newcomer* process gives up immediately.
- A `SPINCOUNT` value of 1 is appropriate for uniprocessors.
- On multiprocessors, a good starting value is 5,000, but some customers have benefited from a `SPINCOUNT` value as high as 100,000.
- Set the `SPINCOUNT` value and observe your application throughput. Because you can tune the `SPINCOUNT` value using the `TMIB`, you can adjust it while the system is running.

Specifying Machines as Types

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

Characteristics of the TYPE Parameter

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

Identifying the Location of the Configuration File

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

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

Characteristics of the TUXCONFIG Parameter

The `TUXCONFIG` parameter has the following characteristics:

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

Indicating the Size of the DTP Transaction Log

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

Defining the DTP Transaction Log Name

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

Specifying Environment Variable Settings

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

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

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

Characteristics of the ENVFILE Parameter

ENVFILE is an optional parameter with the following characteristics:

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

For more information about setting environment variables, refer to [tuxenv\(5\)](#) in *File Formats, Data Descriptions, MIBs, and System Processes Reference*.

Defining the BEA Tuxedo Filesystem Containing the TLOG

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

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

Specifying a Machine's Maximum Number of Simultaneous Global Transactions

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

Defining the Number of Accesser Entries on a Workstation Client

Use the MAXWSCLIENTS parameter to define the number of entries on a machine to be reserved for Workstation clients. Set the number of accesser slots reserved for MAXWSCLIENTS cautiously,

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

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

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

Defining Space Limits for Messages Transmitted by the BRIDGE

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

There are two situations when `MAXPENDINGBYTES` is significant:

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

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

Indicating the Offset for the DTP Transaction Log

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

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

Defining the Offset for TUXCONFIG

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

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

Characteristics of the TUXOFFSET Parameter

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

Identifying the Locations of the System Software and Application Server Software

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

Characteristics of the APPDIR and TUXDIR Parameters

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

Indicating a Threshold Message Size for Compression

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

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

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

Example

```
CMPLIMIT=string_value1,string_value2
```

Specifying the Pathname for the ULOG

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

environment variable is used. *mmddyy* (month, day, year) is appended to the prefix to form the full name of the log file.

Characteristics of the ULOGPFX Parameter

The ULOGPFX parameter has the following characteristics:

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

See Also

- [“How to Create the GROUPS Section of the Configuration File” on page 3-41](#)

How to Create the GROUPS Section of the Configuration File

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

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

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

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

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

Sample GROUPS Section for ATMI

Following is a sample GROUPS section of a configuration file in an ATMI environment.

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

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

Sample GROUPS Section for CORBA

The following sample GROUPS section is from the UBBCONFIG file in the Tuxedo CORBA 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 must be configured, and the machines on which they run on must be identified.

```

*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 [“CORBA Factory-based Routing in the University Production Sample Application” on page 3-92](#) to understand how the names in the GROUPS section match the group names specified in the ROUTING section. This match is critical for the routing function to work correctly. Also, any change in the way groups are configured in an application must be reflected in the ROUTING section.

Note: The Production sample application packaged with the BEA Tuxedo 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.

See Also

- [“How to Create the SERVERS Section of the Configuration File” on page 3-56](#)

Specifying a Group Name, Number, and LMID

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

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

Characteristics of the Group Name, Group Number, and LMID

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

See Also

- [UBBCONFIG \(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*
- “How to Create the NETWORK Section of the Configuration File” on page 3-48

Indicating a Transaction Manager Server Name and Numbers per Group

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

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

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

Identifying the Environment File Location for Servers in a Group

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

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

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

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

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

Defining Information Needed When Opening and Closing the Resource Manager

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

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

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

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

- On UNIX
`OPENINFO = "TUXEDO/QM:qmconfig:qspace"`
- On Windows
`OPENINFO = "TUXEDO/QM:qmconfig;qspace"`

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

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

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

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

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

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

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

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

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

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

How to Create the NETWORK Section of the Configuration File

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

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

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

Sample NETWORK Section

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

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

See Also

- [UBBCONFIG \(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*
- “How to Create the NETGROUPS Section of the Configuration File” on page 3-52

Specifying a Device Name for the BRIDGE Process

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

```
BRIDGE=string_value
```

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

The pathname for the network transport endpoint file has the following form:

```
/dev/provider_name
```

Assigning a BRIDGE Network Address

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

```
NADDR = string_value
```

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

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

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

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

In the second format, the string *#. #. #. #* represents four decimal numbers (each of which is between 0 and 255), separated by periods. The value of *port_number* is a decimal number in the

range 0 to 65,535 (the hexadecimal representations of the string specified). The value of `port_number` can be a symbolic name or a decimal number.

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

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

Assigning Encryption Levels

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

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

Example

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

See Also

- [“Link-Level Encryption”](#) in *Using Security in CORBA Applications*

Assigning a tlisten Network Address

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

```
NLSADDR=string_value
```

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

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

- `//host.name:port_number`
- `//#. #. #. #:port_number`

- *0xhex-digits* or *\\xhex-digits*

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

In the second format, the string *#.#.#.#* represents four decimal numbers (each of which is between 0 and 255), separated by periods. The value of *port_number* is a decimal number in the range 0 to 65,535 (the hexadecimal representations of the string specified). The value of *port_number* can be a symbolic name or a decimal number.

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

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

How to Create the NETGROUPS Section of the Configuration File

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

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

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

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

Sample Network Groups Configuration

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

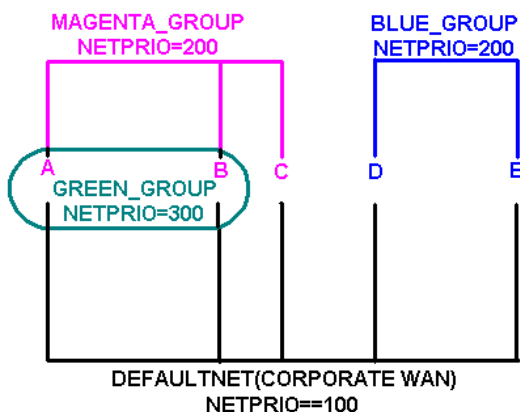
First State Bank has a network of five machines (A-E). Each machine belongs to two or three of four netgroups that you have defined in the following way:

- `DEFAULTNET` (the default network, which is the corporate WAN)

- MAGENTA_GROUP (a LAN)
- BLUE_GROUP (a LAN)
- GREEN_GROUP (a private LAN that provides high-speed, fiber, point-to-point links between member machines)

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

Figure 3-3 Example of a Network Grouping



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

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

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

Configuring a Sample UBBCONFIG File with Netgroups

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

Listing 3-1 Sample NETGROUPS and NETWORK Sections

```
*NETGROUPS

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

*NETWORK

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

B	NETGROUP=DEFAULTNET	NADDR="//B_CORPORATE:5723"
B	NETGROUP=MAGENTA_GROUP	NADDR="//B_MAGENTA:5724"
B	NETGROUP=GREEN_GROUP	NADDR="//B_GREEN:5725"
C	NETGROUP=DEFAULTNET	NADDR="//C_CORPORATE:5723"
C	NETGROUP=MAGENTA_GROUP	NADDR="//C_MAGENTA:5724"
D	NETGROUP=DEFAULTNET	NADDR="//D_CORPORATE:5723"
D	NETGROUP=BLUE_GROUP	NADDR="//D_BLUE:5726"
E	NETGROUP=DEFAULTNET	NADDR="//E_CORPORATE:5723"
E	NETGROUP=BLUE_GROUP	NADDR="//E_BLUE:5726"

See Also

- [UBBCONFIG \(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*
- “How to Create the SERVERS Section of the Configuration File” on page 3-56
- “Setting Up the Network for a Distributed Application” on page 11-1

Assigning a Name to a Network Group

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

```
NETGROUP required_parameters [optional_parameters]
```

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

Assigning a Network Group Number

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

```
NETGRPNO=numeric_value
```

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

Assigning a Priority to the Network Group

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

How to Create the SERVERS Section of the Configuration File

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

- The name, group, and numeric identifier for a server (`SRVGRP`, `SRVID`)
- Server command-line options defined by `servopts` (`CLOPT`)
- Parameters to determine the booting order and number of servers to boot (`SEQUENCE`, `MIN`, `MAX`)
- A server-specific environment file (`ENVFILE`)
- Server queue-related information (`RQADDR`, `RQPERM`, `REPLYQ`, `RPPERM`)
- Restart information (`RESTART`, `RCMD`, `MAXGEN`, `GRACE`)
- Designation as a conversational server (`CONV`)
- Overriding of system-wide shared memory access (`SYSTEM_ACCESS`)
- Setting security parameters for IIOP Listener (ISL) servers

Note: Command-line options supported by the BEA Tuxedo system are described in [servopts\(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*.

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

To Specify This Information in the SERVERS Section (Optional)	Set This Parameter (Required/Optional)	For More Information, Click the Following
Whether the server is a conversational server. Connections can be made only to conversational servers, and <code>rpc</code> requests (via <code>tpacall(3c)</code> or <code>tpcall(3c)</code>) can be made only to non-conversational servers.	<code>CONV</code> (optional run-time parameter)	Conversational server
Principal name of the process used for identification, location of the principal user's private key, and the environment variable containing the password	<code>SEC_PRINCIPAL_NAME</code> , <code>SEC_PRINCIPAL_LOCATION</code> , <code>SEC_PRINCIPAL_PASSVAR</code>	Security attributes
When this server should be booted or shut down relative to other servers.	<code>SEQUENCE</code> (Optional boot parameter)	Server boot order
The minimum number of occurrences of the server to be booted by <code>tmboot</code> .	<code>MIN</code> (Optional boot parameter)	Server boot order
The maximum number of occurrences of the server that can be booted.	<code>MAX</code> (Optional boot parameter)	Server boot order
A list of <code>servopts(5)</code> options to be passed to a server process at boot time. If none are specified, the default is <code>-A</code> . <i>string_value</i> may contain up to 256 characters.	<code>CLOPT</code> (Optional boot parameter)	Server command-line options
A request for the addition of the values in this file to the environment of the server during its initialization. If a server is associated with a server group that can be migrated to a second machine, the <code>ENVFILE</code> must be in the same location on both machines.	<code>ENVFILE</code> (Optional run-time parameter)	Server environment file
The name of the group in which the server is to run. <i>string_value</i> must be the logical name associated with a server group in the <code>GROUPS</code> section.	<code>SRVGRP</code> (Required)	Server group
An integer that uniquely identifies a server within a group. Identifiers must be between 1 and 30,000 inclusive.	<code>SRVID</code> (required)	Server ID

To Specify This Information in the SERVERS Section (Optional)	Set This Parameter (Required/Optional)	For More Information, Click the Following
The symbolic name of the request queue for the process.	RQADDR (Optional run-time parameter)	Server queue information
The numeric permissions on the request queue.	RQPERM (Optional run-time parameter)	Server queue information
Whether a reply queue should be established for the process.	REPLYQ (Optional run-time parameter)	Server queue information
The numeric permissions on the reply queue.	RPPERM (Optional run-time parameter)	Server queue information
The command that should be executed when the process abnormally terminates, if the process is restartable.	RCMD (Optional run-time parameter)	Server restart information
The maximum number minus one time that the process can be restarted within the period specified by GRACE, if the process is restartable.	MAXGEN (Optional run-time parameter)	Server restart information
A parameter that specifies that the process can have up to MAXGEN lives within the specified number of seconds, if the process is restartable.	GRACE (Optional run-time parameter)	Server restart information
Whether the process is restartable. Default is N. If server migration is specified, RESTART must be set to Y. (A server terminated with a SIGTERM signal must be rebooted.)	RESTART (Optional run-time parameter)	Server restart information
The default mode used by BEA Tuxedo system libraries within application processes to gain access to BEA Tuxedo system internal tables.	SYSTEM_ACCESS (Optional run-time parameter)	System access to servers

To Specify This Information in the SERVERS Section (Optional)	Set This Parameter (Required/Optional)	For More Information, Click the Following
<p>The minimum number of server dispatch threads started on initial server boot. The separate dispatched thread that is used when <code>MAXDISPATCHTHREADS>1</code> is not counted as part of the <code>MAXDISPATCHTHREADS</code> value. It is required that <code>MINDISPATCHTHREADS<=MAXDISPATCHTHREADS</code>. The default for this parameter is 0.</p>	MINDISPATCHTHREADS	Threads
<p>The maximum number of concurrently dispatched threads that each server process may spawn. If <code>MAXDISPATCHTHREADS>1</code>, then a separate dispatcher thread is used and does not count against this limit. It is required that <code>MINDISPATCHTHREADS<=MAXDISPATCHTHREADS</code>. The default for this parameter is 1.</p>	MAXDISPATCHTHREADS	Threads

To Specify This Information in the SERVERS Section (Optional)	Set This Parameter (Required/Optional)	For More Information, Click the Following
The stack size in bytes for each server thread after the initial thread. If not specified or specified as 0, the operating system default is used. This option has an affect on the server only when a value greater than 1 is specified for MAXDISPATCHTHREADS.	THREADSTACKSIZE	threads
<p>The WebLogic Server embedded LDAP-based authentication server. It is a System/T provided server that offers the authentication service while the user security information is located in WebLogic Server. This server may be used in a secure application to provide per-user authentication when clients join the application.</p> <p>SECURITY USER_AUTH or higher must be specified.</p> <p>Default uses the file \$TUXDIR/udataobj/tpldap to get LDAP configuration information.</p> <p>Example: LAUTHSVR SRVGRP="AUTH" SRVID=100</p> <p>CLOPT="-A--</p> <p>-f/usr/tuxedo/udataobj/tpldap"</p>	LAUTHSVR (Optional)	LAUTHSVR(5)

Sample SERVERS Section

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

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



```

RINGUP1          SRVGRP=GROUP1 SRVID=1 MIN=3
                  RQADDR="ring1 "
RINGUP2          SRVGRP=GROUP1 SRVID=4 MIN =3
                  RQADDR="ring2 "

```

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

Sample SERVERS Section Parameters

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

Parameter	Meaning
<code>RESTART=Y</code> (default)	Restart the servers.
<code>MAXGEN=5</code> (default)	The <code>MAXGEN</code> parameter specifies a number greater than 0 and less than 256 that controls the number of times a server can be started within the period specified by the <code>GRACE</code> parameter. The default is 1. If the server is to be restartable, <code>MAXGEN</code> must be ≥ 2 . The number of restarts is at most $number - 1$ times. <code>RESTART</code> must be <code>Y</code> or <code>MAXGEN</code> is ignored.
<code>GRACE=3600</code> (default)	If <code>RESTART</code> is <code>Y</code> , the <code>GRACE</code> 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 <code>GRACE</code> is not specified, the default is 86,400 seconds (24 hours). As soon as one <code>GRACE</code> 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.
<code>REPLYQ=N</code> (default)	There is no reply queue.
<code>CLOPT="-A"</code> (default)	Specify <code>-A</code> on the command line of each server.
<code>ENVFILE="/usr/home/envfile"</code> (default)	Read environment settings from the file <code>ENVFILE</code> .
<code>SYSTEM_ACCESS=PROTECTED</code> (default)	Deny access to internal tables outside system code.
<code>RINGUP1</code>	Sample name of the first server to be booted.

Parameter	Meaning
SRVGRP=GROUP1 SRVID=1 MIN=3 RQADDR="ring1"	Three instances of the sample server will be booted in group GROUP1 with server IDs of 1, 2, and 3, respectively. The three servers will form an MSSQ set and will read requests from queue <i>ring1</i> . Note: RQADDR assigns a symbolic name to the request queue of this server. MSSQ sets are established by using the same symbolic queue name for more than one server, as well as same executable name for all the servers (and by specifying a value greater than 1 for MIN).
RINGUP2	Name of the second sample server to be booted.

See Also

- [UBBCONFIG \(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*
- “How to Create the SERVICES Section of the Configuration File” on page 3-75

Specifying a Server as Conversational

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

Characteristics of the CONV Parameter

The `CONV` parameter has the following characteristics:

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

Setting the Order in Which Servers Are Booted

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

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

Warning: In CORBA environments, there is a strict order in which the system EventBroker, the FactoryFinder object, and the application factories must be booted. A CORBA application program will not boot if the order is changed. See the section [“Required Order in Which to Boot CORBA C++ Servers” on page 3-63](#) for details.

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

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

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

Required Order in Which to Boot CORBA C++ Servers

The following is the correct order in which to boot the servers in a BEA Tuxedo CORBA environment. A CORBA application program will not boot if the order is changed.

1. The system EventBroker, `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 C++ servers that are advertising factories.

[Listing 3-2](#) shows the order in which servers are booted for the BEA Tuxedo CORBA University Basic application, which is one of the sample applications included with the BEA Tuxedo software. This `SERVERS` section is excerpted from an edited version of the `ubb_b.nt` configuration file.

Listing 3-2 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 EventBroker. This event broker
# must be started before any servers providing the
# NameManager Service
#
TMSYSEVT
    SRVGRP = SYS_GRP
    SRVID  = 1

# TMFFNAME is a BEA Tuxedo CORBA provided server that
# runs the NameManager and FactoryFinder services.

# The NameManager service is a BEA Tuxedo CORBA-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 CORBA Interface Repositories.”](#)
- The `univb_server`, for the University Basic sample application. For details about the sample applications, see the [Guide to the CORBA 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, refer to [Chapter 14, “Managing Remote BEA Tuxedo CORBA Client Applications.”](#)

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

Characteristics of the `SEQUENCE`, `MIN`, and `MAX` Parameters

Parameter	Characteristics
<code>SEQUENCE</code>	<p>It is an optional parameter with a numeric range of 1 - 10,000.</p> <p>Smaller values are booted before larger values.</p> <p>Servers for which this parameter is not set are booted in the order in which they are listed in the <code>SERVERS</code> section.</p> <p>All sequenced servers are booted before any unsequenced servers.</p>

Parameter	Characteristics
MIN	<p>It represents the minimum number of servers to boot during run time.</p> <p>If RQADDR is specified and MIN>1, an MSSQ set is created.</p> <p>All instances have the same server options.</p> <p>The range of values is 0 to 1000.</p> <p>The default is 1.</p>
MAX	<p>It represents the maximum number of servers to boot.</p> <p>The range of values for MAX is 0 to 1000. If MAX is not specified, the default is the value of MIN.</p>

Specifying Server Command-line Options

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

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

Use This Option	To
<code>-o filename</code>	Redirect standard output to file <i>filename</i> .
<code>-e filename</code>	Redirect standard error to file <i>filename</i> .
<code>-s services</code>	Advertise services. For example, <code>-s x,y,z</code> to advertise services <i>x</i> , <i>y</i> , and <i>z</i> .
<code>-s x,y,z:funcname</code>	Advertise services <i>x</i> , <i>y</i> , and <i>z</i> , but process requests for those services with function <i>funcname</i> . This is called <i>aliasing</i> a function name.

Use This Option	To
-r	Specify that the server should log the services performed.
-v	Print out the list of the service name/function name to standard output. This option cannot be used in the CLOPT in the UBBCONFIG. It must be used when manually invoking the server.

Note: You can find other standard `main()` options listed on [servopts\(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*.

Characteristics of the CLOPT Parameter

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

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

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

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

See Also

- [servopts\(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*

Identifying the Location of the Server Environment File

Use the `ENVFILE` parameter in the `MACHINES` section to specify environment settings. You can also specify the same parameter for a specific server process; the semantics are the same. If both

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

Characteristics of the Server Environment File

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

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

For more information about setting environment variables, refer to [tuxenv\(5\)](#) in *File Formats, Data Descriptions, MIBs, and System Processes Reference*.

Defining Server Name, Group, and ID

You initially assign a name to a server in the `SERVERS` section. The name you specify must be the name of an executable file built with one of the following commands:

- `buildserver(1)` for ATMI applications
- `buildobjserver(1)` for CORBA C++ server applications

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

Characteristics of the Server Name, SRVGRP, and SRVID Parameters

Parameter	Characteristics
<i>Server_name</i>	It identifies the executable to be booted. It is built with <code>buildserver(1)</code> for ATMI. It is built with <code>buildobjserver(1)</code> for CORBA. It is required, but may not be unique within a server group.
SRVGRP (Server Group)	It identifies the group affiliation. The group name begins with a GROUPS section entry. It is required.
SRVID (Server ID)	It is numeric. It is required and unique within a server group.

Identifying Server Queue Information

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

MSSQ Example

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

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

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

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

Parameter	Characteristics
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 and the servers in an MSSQ set should have the same executable name. In order to boot multiple servers, set the value greater than 1 for <code>Min</code> parameter.
RQPERM	Represents the permissions on a request queue. If no parameter is specified, the permissions of the bulletin board, as specified by <code>PERM</code> in the <code>RESOURCES</code> section, are used. If no value is specified there, the default of 0666 is used. When the default is used, your application is available to anyone with a login on the system.
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, <code>REPLYQ</code> should be set to <code>Y</code> so that an individual reply queue is created for this server. If not, the reply is sent to the request queue shared by all servers of the MSSQ set, and there is no way of assuring that it will be picked up by the server that is waiting for it. Multithreaded servers automatically create <code>REPLYQs</code> even if this parameter is not set.
RPPERM	Assigns permissions to the reply queue. This parameter is useful only when <code>REPLYQ=Y</code> . If requests and replies are read from the same queue, only <code>RQPERM</code> is needed; <code>RPPERM</code> is ignored.

Defining Server Restart Information

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

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

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

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

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

Parameter	Characteristics
RESTART	A setting of <code>Y</code> enables a server to restart. The default is <code>N</code> .
RCMD	Specifies an executable file to be run at restart time. Allows you to take an action when a server is restarted.
MAXGEN	Represents the maximum number of server lives in a specific interval. The default is 1; the maximum is 256.
GRACE	Represents the interval used by <code>MAXGEN</code> . Zero represents unlimited restart. It must be between 0 and 2147,483,647 ($2^{31} - 1$). The default is 24 hours.

Defining Server Access to Shared Memory

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

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

Characteristics of the `SYSTEM_ACCESS` Parameter

The `SYSTEM_ACCESS` parameter has the following characteristics:

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

Defining the Server Dispatch Threads

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

`MINDISPATCHTHREADS`<=`MAXDISPATCHTHREADS`. If not specified, the default for this parameter is 1.

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

`MINDISPATCHTHREADS`<=`MAXDISPATCHTHREADS`. The default for this parameter is 0.

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

Setting Security Parameters for ISL Servers

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

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

Table 3-1 lists the SSL parameters characteristics.

Table 3-1 ISL and SSL Parameters Characteristics

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

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

How to Create the SERVICES Section of the Configuration File

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

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

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

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

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

Sample SERVICES Section

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

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

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

See Also

- [UBBCONFIG \(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*
- [“How to Create the ROUTING Section of the Configuration File”](#) on page 3-89

Specifying Automatic Starts and Timeout Intervals for Transactions

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

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

```
TRANTIME=number
```

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

An additional transaction timeout property named `MAXTRANTIME` is available from the `RESOURCES` section of the `UBBCONFIG` file. If the `MAXTRANTIME` timeout value is less than the `TRANTIME` timeout value or the timeout value passed in a `tpbegin(3c)` call to start a transaction, the timeout for a transaction is reduced to the `MAXTRANTIME` value.

Note: `MAXTRANTIME` has no effect on a transaction started on a machine running BEA Tuxedo 8.0 or earlier, except that when a machine running BEA Tuxedo 8.1 or later is infected by the transaction, the transaction timeout value is capped—reduced if necessary—to the `MAXTRANTIME` value configured for that node.

See Also

- *Using the BEA Tuxedo Domains Component*
- For more information about MAXTRANTIME, see MAXTRANTIME in the RESOURCES section in UBBCONFIG (5) or TA_MAXTRANTIME in the T_DOMAIN class in TM_MIB (5).

Specifying a List of Allowable Buffer Types for a Service

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

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

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

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

Examples of the BUFTYPE Parameter

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

Designating How Much Time to Process a Request

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

The SVCTIMEOUT parameter allows you to designate an amount of time (in seconds) in which a service should be able to process a request. If the interval defined by this parameter elapses and

a service has not finished processing a request, the process for that request is killed. In essence, the service timeout mechanism acts like a scavenger for frozen or out of control application servers. By default, the BEA Tuxedo system does not terminate any service process; you must set the `SVCTIMEOUT` parameter to activate this feature.

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

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

What Happens When a Timeout Occurs

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

How a Service Timeout Is Reported

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

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

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

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

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

How to Control a Service Timeout

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

Specifying Nontransactional Service-Level Blocktime

Different services take different amounts of time and need individual `BLOCKTIME` values. Sometimes, an application needs or desires to override the default blocktime value for an individual client or for an individual service call.

The `UBBCONFIG` file `SERVICES` section `BLOCKTIME` parameter allows you to designate the blocking time value, per second, for individual *nontransactional* services. It overrides the default `RESOURCES` section `BLOCKTIME` parameter value for the designated service. Per service `BLOCKTIME` parameter values can also be set for remote services using the `DMCONFIG` file. For more information, see [UBBCONFIG \(5\)](#), `SERVICES` section and [DMCONFIG \(5\)](#), `DM_IMPORT` section.

Unlike the `SVCTIMEOUT` parameter, the `BLOCKTIME` parameter *does not* terminate a service application. Instead, it lets the client know that (after a specified time in seconds), no reply has been received by the server while the service request is still processing.

Note: Application programmers can also set nontransactional blocktime requests and retrieve blocktime values by using the [tpsblktime\(3c\)](#) and [tpgblktime\(3c\)](#) functions.

Enabling Load Balancing

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

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

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

Characteristics of the LDBAL Parameter

The `LDBAL` parameter has the following characteristics:

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

Defining the Name of the Routing Criteria

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

```
ROUTING=string_value
```

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

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

Specifying Service Parameters for Different Server Groups

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

Controlling the Flow of Data by Service Priority

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

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

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

Characteristics of the PRIO Parameter

The `PRIO` parameter has the following characteristics:

- It determines the priority of a service on the server's queue.

- The highest assigned priority gets first preference.
- Every tenth request is dequeued FIFO.

Sample SERVICES Section Using Different Priorities

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

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

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

Indicating Service Processing Time

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

```
SVCTIMEOUT=number
```

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

How to Create the INTERFACES Section of the Configuration File

Note: This section applies only to the CORBA environments in BEA Tuxedo.

The `INTERFACES` section in the configuration file is used to define parameters for CORBA environments in the BEA Tuxedo system. 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 Tuxedo CORBA environment should apply to each interface. Factory-based routing is a feature that lets you 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 [“How to Create the ROUTING Section of the Configuration File”](#) in this chapter.

Specifying CORBA Interfaces in the INTERFACES Section

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

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

[Table 3-2](#) lists the `AUTOTRAN`, `FACTORYROUTING`, `LOAD`, `PRIO`, `SRVGRP`, `TRANTIME`, and `TIMEOUT` parameters characteristics.

Table 3-2 INTERFACES Section Parameters Characteristics

Parameter	Characteristic
<code>AUTOTRAN = {Y N }</code>	<p>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.</p> <p>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 <code>T_IFQUEUE</code> 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 <i>optional</i>.</p> <p>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.</p>
<code>FACTORYROUTING = criterion-name</code>	Specify the name of the routing criteria to be used for factory-based routing for this CORBA interface. You must specify a <code>FACTORYROUTING</code> parameter for interfaces requesting factory-based routing.
<code>LOAD = number</code>	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 <code>LOAD</code> numbers assigned to other CORBA interfaces used by this application. The default is 50. The number is used by the BEA Tuxedo system to select the best server to route the request.
<code>PRIO = number</code>	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.
<code>SRVGRP = server-group-name</code>	Use <code>SRVGRP</code> to indicate that any parameter defined in this portion of the <code>INTERFACES</code> section applies to the interface within the specified server group. For a given CORBA interface, this feature lets you define different parameter values in different server groups.

Table 3-2 INTERFACES Section Parameters Characteristics (Continued)

Parameter	Characteristic
TRANTIME = number	If AUTOTRAN is set to Y, you must set the TRANTIME parameter, which is the transaction timeout in seconds, for the transactions to be computed. The value must be greater than or equal to zero and must not exceed 2,147,483,647 (231 - 1), or about 70 years. A value of 0 (zero) implies there is no timeout for the transaction. (The default is 30 seconds.)
TIMEOUT=number	The amount of time, in seconds, to allow for processing of a method for this CORBA interface. The values must be greater than or equal to 0. A value of 0 indicates that the interface cannot time out. A timed-out method causes the server processing the method for the interface to terminate with a SIGKILL event. You should consider specifying a timeout value for the longest-running method for the interface.

Specifying FACTORYROUTING Criteria

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

Listing 3-3 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 [“CORBA Factory-based Routing in the University Production Sample Application” on page 3-92](#).

Bankapp Sample

[Listing 3-4](#) shows how factory-based routing is specified in the Bankapp sample application.

Listing 3-4 Bankapp Sample Factory-based Routing

```
*INTERFACES

    "IDL:BankApp/Teller:1.0"
        FACTORYROUTING=atmID

*ROUTING

    atmID

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

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

Enabling Load Balancing

In BEA Tuxedo CORBA environments, 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.

For more information about load balancing in BEA Tuxedo CORBA environments, refer to “[Enabling System-controlled Load Balancing](#),” in the *Scaling, Distributing, and Tuning CORBA Applications* manual.

Support for parallel objects in CORBA environments has been added for release 8.0 of BEA Tuxedo, which introduces load balancing across multiple servers in a local domain. For more information about parallel objects in BEA Tuxedo CORBA environments, refer to the “[Using Parallel Objects](#)” section in *Scaling, Distributing, and Tuning CORBA Applications*.

Controlling the Flow of Data by Interface Priority

You can control the flow of data in a BEA Tuxedo client or server application by assigning interface priorities using the `PRIOR` 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 `PRIOR` 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.

How to Create the ROUTING Section of the Configuration File

The `ROUTING` section of `UBBCONFIG` allows you to provide a full definition of the routing criteria named in the `SERVICES` section (for ATMI data-dependent routing) or in the `INTERFACES` section (for CORBA factory-based routing).

Note: For more information about configuring factory-based routing for CORBA environments, refer to the *Scaling, Distributing, and Tuning CORBA Applications* guide.

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

To Specify This Information in the <code>ROUTING</code> Section (Optional)	Set This Parameter (Required/Optional)	For More Information, Click the Following
Ranges and associated server groups for the routing field.	<code>RANGES</code> (Required)	Range criteria
<p>The value must be a string with a maximum length of 15 characters.</p> <p>For ATMI, the routing criteria name specified as the value of the <code>ROUTING</code> parameter in the <code>SERVICES</code> section for data-dependent routing.</p> <p>For CORBA, the routing criteria name specified in the <code>INTERFACES</code> section as the <code>FACTORYROUTING</code> parameter factory-based routing.</p>	<code>criterion_name</code> (required)	
<p>Specifies the routing type.</p> <p>For ATMI, the default is <code>TYPE=SERVICE</code> to ensure that existing <code>UBBCONFIG</code> files used in Tuxedo ATMI environments continue to work properly.</p> <p>For CORBA, use <code>TYPE=FACTORY</code> when implementing factory-based routing for a CORBA interface.</p>	<code>TYPE</code>	

To Specify This Information in the ROUTING Section (Optional)	Set This Parameter (Required/Optional)	For More Information, Click the Following
<p>Name of the routing field, which is assumed to be an FML buffer, XML element or element attribute, view field name identified in an FML field table (using FLDTBLEDIR and FLDTBLIS environment variables), or an FML view table (using the VIEWDIR and VIEWFILES environment variables), respectively. This information is used to obtain the associated field value for data-dependent routing when sending a message.</p> <p>In CORBA 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.</p>	FIELD (Required)	Routing buffer field and type
<p>A list of types and subtypes of data buffers for which this routing entry is valid. This parameter may contain up to 256 characters with a maximum of 32 type/subtype combinations.</p>	BUFTYPE (required)	Buffer types and subtypes

ROUTING Section Example

The following is a sample ROUTING section from a configuration file:

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

Defining the Routing Buffer Field and Field Type

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

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

Specifying Range Criteria

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

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

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

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

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

Note: Overlapping ranges are allowed, but values that belong to both ranges map to the first group. For example, if `RANGES` is specified as `RANGES="0-5:Group1,3-5:Group2"`, then a range value of 4 routes to `Group1`.

Defining Buffer Types

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

CORBA Factory-based Routing in the University Production Sample Application

The CORBA University Production sample application demonstrates how to implement factory-based routing in BEA Tuxedo. You can find the `ubb_p.nt` or `ubb_p.mk` `UBBCONFIG` files

for this sample in the directory where the BEA Tuxedo 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 CORBA application in BEA Tuxedo.

The `INTERFACES` section lists the names of the interfaces for which you want to enable factory-based routing. For each interface, this section specifies what kinds of criteria the interface routes on. This section specifies the routing criteria via an identifier, `FACTORYROUTING`, as in the example in [Listing 3-5](#).

Listing 3-5 Production Sample INTERFACES Section

```
*INTERFACES
```

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

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

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

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

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

- The `RANGES` parameter, which associates a server group with a subset of the valid ranges for each routing value.

[Listing 3-6](#) shows the `ROUTING` section of the `UBBCONFIG` file used in the Production sample application.

Listing 3-6 Production Sample `ROUTING` Section

```
*ROUTING
```

```
STU_ID
```

```
    FIELD      = "student_id"
```

```
    TYPE       = FACTORY
```

```
    FIELDTYPE  = LONG
```

```
    RANGES     = "100001-100005:ORA_GRP1,100006-100010:ORA_GRP2"
```

```
ACT_NUM
```

```
    FIELD      = "account_number"
```

```
    TYPE       = FACTORY
```

```
    FIELDTYPE  = LONG
```

```
    RANGES     = "200010-200014:APP_GRP1,200015-200019:APP_GRP2"
```

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

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

[Listing 3-7](#) shows the `GROUPS` section of the Production sample `UBBCONFIG` file. Notice how the names in the `GROUPS` section match the group names specified in the `ROUTING` section; this is critical for factory-based routing to work correctly. Furthermore, any change in the way groups are configured in an application must be reflected in the `ROUTING` section. (Note that the

Production sample packaged with the BEA Tuxedo software is configured to run entirely on one machine. However, you can easily configure this application to run on multiple machines.)

Listing 3-7 Production Sample GROUPS Section

```
*GROUPS

APP_GRP1
    LMID = SITE1
    GRPNO = 2
    TMSNAME = TMS

APP_GRP2
    LMID = SITE1
    GRPNO = 3
    TMSNAME = TMS

ORA_GRP1
    LMID = SITE1
    GRPNO = 4

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

ORA_GRP2
    LMID = SITE1
    GRPNO = 5

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

CORBA Factory-based Routing in the Bankapp Sample Application

[Listing 3-8](#) shows how the `INTERFACES` section extends the Bankapp sample application to use factory-based routing. The sample included with the BEA Tuxedo software does not contain these parameter settings.

Listing 3-8 Bankapp Sample INTERFACES Section

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

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

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

In this example, the `IDL:Bankapp/Teller` interface employs a factory-based routing scheme called `atmID`, as defined in the `ROUTING` section. 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.

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

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

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

In This Section	Set These Parameters	With These Considerations.
RESOURCES	<code>MAXACCESSERS</code>	Optional parameter, but you must assign a value to it you want more than 50 accessers (the default number). Each context of a multicontexted client is counted separately for licensing purposes.
	<code>NOTIFY</code>	Optional parameter that defines the default method to be used for unsolicited notification. Valid values for multicontexted applications are: <ul style="list-style-type: none"> • <code>DIPIN</code> • <code>THREAD</code> • <code>IGNORE</code>

Table 3-3 Setting Parameters in the Configuration File to Use Threads (Continued)

In This Section	Set These Parameters	With These Considerations.
MACHINES	MAXACCESSERS	Optional parameter, but you must assign a value to it you want more than 50 accessers (the default number). Each context of a multicontexted client is counted separately for licensing purposes.
	MAXWSCLIENTS	Optional parameter. Each context of a multicontexted Workstation client is counted separately for licensing purposes. Because the default is 0, this parameter must be set if any Workstation clients are to access the system via the machine being defined.

Table 3-3 Setting Parameters in the Configuration File to Use Threads (Continued)

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

How to Compile a Configuration File

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

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

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

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

About Transactions

This topic includes the following sections:

- [What Is a Transaction?](#)
- [Benefits of Using Transactions](#)
- [Example of a Global Transaction](#)
- [What Is the BEA Tuxedo Transaction Manager \(TM\)?](#)
- [How the System Tracks Distributed Transaction Processing](#)
- [How the System Uses a Two-Phase Commit to Commit Transactions](#)

Note: For information about using transactions in a BEA Tuxedo CORBA environment, refer to *Using CORBA Transactions*.

What Is a Transaction?

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

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

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

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

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

What Are the ACID Properties?

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

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

Table 4-1 ACID Properties of BEA Tuxedo Transactions

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

How a Transaction Succeeds or Fails

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

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

Benefits of Using Transactions

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

By using transactions you can:

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

Example of a Global Transaction

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

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

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

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

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

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

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

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

What Is the BEA Tuxedo Transaction Manager (TM)?

A *resource manager* (RM) is a data repository, such as a database management system or the Application Queuing Manager, with tools for accessing the data. The BEA Tuxedo system uses one or more RMs to maintain the state of an application. For example, bank records in which account balances are maintained are kept in an RM. When the state of the application changes

through a service that allows a customer to withdraw money from an account, the new balance in the account is recorded in the appropriate RM.

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

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

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

Table 4-2 Actions Performed by the Transaction Manager

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

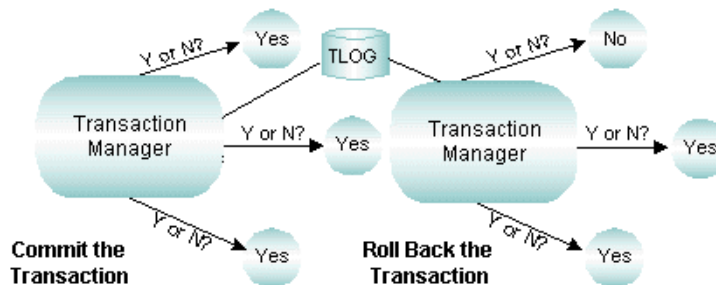
How the System Tracks Distributed Transaction Processing

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

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

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

Figure 4-1 Transaction Management



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

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

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

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

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

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

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

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

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

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

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

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

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

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

2. In Phase 2 of the project, the organizer notifies each member of the group that the vacation will take place as planned.

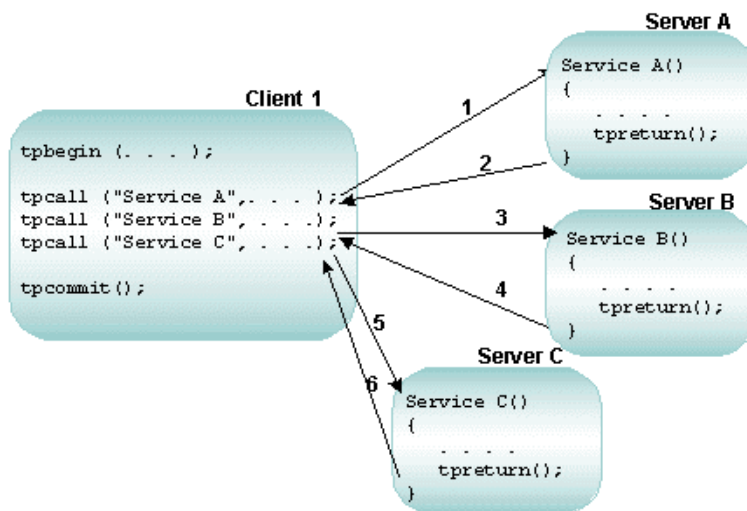
A two-phase transaction commit works in much the same way as the vacation planning project.

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

How the System Handles Transaction Infection

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

Figure 4-2 Transactional Infection



In the preceding figure, Client 1 begins the transaction and calls three services: A, B, and C. Because they have been called into the transaction, Services A, B, and C are transactionally

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

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

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

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

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

- c. Calls `tpreturn()`
 - d. Completes its work for the transaction
5. Client_1 initiates (with `tpcommit()`) the commitment process.

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

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

See Also

- [“Modifying the UBBCONFIG File to Accommodate ATMI Transactions” on page 5-1](#)
- [“Modifying the Domains Configuration File to Support Transactions” on page 5-9](#)
- [“Example: A Distributed Application with Transactions” on page 5-12](#)
- [“Writing Global Transactions” in *Programming BEA Tuxedo ATMI Applications Using C*](#)
- [“What You Can Do Using the ATMI” in *Introducing BEA Tuxedo ATMI*](#)
- For more information about using transactions in a BEA Tuxedo CORBA environment, refer to *Using CORBA Transactions*

Configuring Your ATMI Application to Use Transactions

This topic includes the following sections:

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

Note: For information about using transactions in a BEA Tuxedo CORBA environment, refer to *Using CORBA Transactions*.

Modifying the UBBCONFIG File to Accommodate ATMI Transactions

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

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

Specifying Global Transaction Parameters in the RESOURCES Section

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

Set This Parameter ...	To ...
MAXGTT	<p>Limit the total number of global transaction identifiers (GTRIDs) allowed on one machine at one time. The maximum value allowed is 2048; the minimum, 0; and the default, 100. You can override the value of MAXGTT on a per-machine basis in the <code>MACHINES</code> section.</p> <p>Entries remain in the table only while a global transaction is active, so this parameter has the effect of setting a limit on the number of simultaneous transactions.</p>
CMTRET	<p>Indicate the initial setting of the <code>TP_COMMIT_CONTROL</code> characteristic as one of the following:</p> <ul style="list-style-type: none"> LOGGED—the <code>TP_COMMIT_CONTROL</code> characteristic is set to <code>TP_CMT_LOGGED</code>, which means that <code>tpcommit()</code> returns when all the participants have successfully pre-committed. COMPLETE—the <code>TP_COMMIT_CONTROL</code> characteristic is set to <code>TP_CMT_COMPLETE</code>, which means that <code>tpcommit()</code> does not return until all the participants have successfully committed. <p>The default is <code>COMPLETE</code>.</p> <p>To determine the appropriate setting, consult your resource manager (RM) vendors. If any RM in the application uses the <i>late commit</i> implementation of the XA standard, the setting should be <code>COMPLETE</code>. If all RMs use the <i>early commit</i> implementation, the setting should be <code>LOGGED</code> for performance reasons. (You can override this setting with <code>tpscmt()</code>.)</p>
MAXTRANTIME	<p>Specify the maximum length of the timeout for the transactions.</p> <p>Valid values are between 0 and 2,147,483,647 inclusive.</p> <p>0 represents no limitation on transaction timeout value occurs.</p> <p>Default is 0.</p> <p>Note: For more information about MAXTRANTIME, see MAXTRANTIME in the <code>RESOURCES</code> section in <code>UBBCONFIG(5)</code> or <code>TA_MAXTRANTIME</code> in the <code>T_DOMAIN</code> class in <code>TM_MIB(5)</code>.</p>

Creating a Transaction Log (TLOG) in the MACHINES Section

To create a TLOG, complete the following tasks:

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

Creating the UDL

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

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

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

Note: The command fails if the device already exists.

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

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

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

Defining Transaction-related Parameters in the MACHINES Section

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

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

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

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

Creating the Domains Transaction Log

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

```
dmadmin crdmllog crdlog -d local_domain_name
```

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

See Also

- [“What Is the Transaction Log \(TLOG\)?”](#) on page 2-15 in *Administering a BEA Tuxedo Application at Run Time*

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

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

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

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

Sample of the GROUPS Section

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

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

Description of Transaction Values in the Sample GROUPS Section

This table describes the transaction values shown in the sample GROUPS entry.

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

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

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

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

Enabling a Service to Begin a Transaction in the SERVICES Section

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

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

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

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

Characteristics of the AUTOTRAN, TRANTIME, and ROUTING Parameters

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

Set This Parameter . . .	To . . .
AUTOTRAN	<p>Make a service the initiator of a transaction.</p> <p>To work properly, may be dependent on personal communication between the application designer and the application administrator. If the administrator sets this value to Y without prior knowledge of the ICF parameters set by the developer, the wrong application behavior, or failure of the application might be observed.</p> <p>If a transaction already exists, a new one is not started.</p> <p>Default is N.</p>
TRANTIME	<p>Specify the length of the timeout for the AUTOTRAN transactions.</p> <p>Valid values are between 0 and 2,147,483,647 inclusive.</p> <p>0 represents no timeout.</p> <p>Default is 30 seconds.</p>
ROUTING	<p>Point to an entry in the ROUTING section where data-dependent routing is specified for transactions that request this service.</p>

Modifying the Domains Configuration File to Support Transactions

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

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

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

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

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

Characteristics of the AUTOTRAN and TRANTIME Parameters

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

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

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

An additional transaction-timeout property named `MAXTRANTIME` from the `RESOURCES` section of the `UBBCONFIG` file is also available. If the `MAXTRANTIME` timeout value is less than the `TRANTIME` timeout value or the timeout value passed in a `tpbegin(3c)` call to start a transaction, the timeout for a transaction is reduced to the `MAXTRANTIME` value. `MAXTRANTIME` has no effect on a transaction started on a machine running BEA Tuxedo 8.0 or earlier, except that when a machine running BEA Tuxedo 8.1 or later is infected by the transaction, the transaction timeout value is capped—reduced if necessary—to the `MAXTRANTIME` value configured for that node.

For a Domains configuration, the following transaction-handling scenarios are possible:

- If an interdomain transaction infects a node that does not understand the `MAXTRANSTIME` parameter, or the node understands the `MAXTRANSTIME` parameter but the parameter is not set, the timeout value for the transaction is determined by `TRANSTIME` or by the timeout value passed in the `tpbegin()` call that started the transaction. If the `TRANSTIME` or `tpbegin()` timeout value is exceeded, all BEA Tuxedo nodes infected with the transaction—including the node that started the transaction—generate a TMS timeout message.
- If an interdomain transaction infects a node that understands the `MAXTRANSTIME` parameter and the parameter is set for that node, the timeout value for the transaction is reduced to no greater than the `MAXTRANSTIME` value on that node.

If the `TRANSTIME` or `tpbegin()` timeout value is less than or equal to `MAXTRANSTIME`, the transaction-handling scenario becomes the one previously described.

If the `TRANSTIME` or `tpbegin()` timeout value is greater than `MAXTRANSTIME`, the infected node reduces the timeout value for the transaction to `MAXTRANSTIME`. If the `MAXTRANSTIME` timeout value is exceeded, the infected node generates a TMS timeout message.

For more information about `MAXTRANSTIME`, see `MAXTRANSTIME` in the `RESOURCES` section in `UBBCONFIG(5)` or `TA_MAXTRANSTIME` in the `T_DOMAIN` class in `TM_MIB(5)`.

Example: A Distributed Application with Transactions

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

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

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

Sample RESOURCES Section

The following listing shows a sample `RESOURCES` section.

Listing 5-1 Sample RESOURCES Section

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

In the preceding listing, note the following:

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

Sample MACHINES Section

The following listing shows a sample MACHINES section.

Listing 5-2 Sample MACHINES Section

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

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

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

In the preceding listing, note the following:

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

Sample GROUPS and NETWORK Sections

The following listing shows sample GROUPS and NETWORK sections.

Listing 5-3 Sample GROUPS and NETWORK Sections

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

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

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

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

In the preceding listing, note the following:

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

Sample SERVERS, SERVICES, and ROUTING Sections

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

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

```
*SERVERS
DEFAULT:  RESTART=Y  MAXGEN=5  REPLYQ=N  CLOPT="-A"

TLR      SRVGRP=BANKB1    SRVID=1    CLOPT="-A -- -T 100"
TLR      SRVGRP=BANKB2    SRVID=3    CLOPT="-A -- -T 400"
TLR      SRVGRP=BANKB3    SRVID=4    CLOPT="-A -- -T 700"
XFER     SRVGRP=BANKB1    SRVID=5    REPLYQ=Y
XFER     SRVGRP=BANKB2    SRVID=6    REPLYQ=Y
XFER     SRVGRP=BANKB3    SRVID=7    REPLYQ=Y

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

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

In the preceding listing, note the following:

- Calls to the `tpsvrinit()` function by `TLR` servers will include a number (100, 400, or 700) specified with the `-T` option.
- All service requests are routed on the `ACCOUNT_ID` field.

- No services are performed in AUTOTRAN mode.

See Also

- [“What Is a Transaction?” on page 4-1](#)
- [“Using Tuxedo with Oracle Real Application Clusters \(RAC\)” on page 6-1](#)
- [“Writing Global Transactions” on page 9-1](#) in *Programming BEA Tuxedo ATMI Applications Using C*
- [“What You Can Do Using the ATMI”](#) in *Introducing BEA Tuxedo ATMI*
- For more information about using transactions in a BEA Tuxedo CORBA environment, refer to *Using CORBA Transactions*

Using Tuxedo with Oracle Real Application Clusters (RAC)

This topic includes the following sections:

- [Overview](#)
- [Limitations](#)
- [Prerequisites](#)
- [Configuring Tuxedo for Oracle RAC](#)

Overview

The Oracle Real Application Clusters (RAC) feature supports clustering of machines that utilize replicated Oracle database services accessing the same Oracle database. Oracle RAC provides the ability to concurrently access the same Oracle database from instances physically located on multiple Oracle server machines, and offers the ability to failover unsuccessful database instances to alternate locations.

However, specific support for Oracle RAC is required by the Transaction Monitor in order to take advantage of these replication and failover features in an XA transaction environment and to obtain optimal RAC performance. This is because Oracle RAC does not allow the *same* database to be accessed from *multiple* RAC instances within the *same* XA transaction.

In addition, Oracle 10gR1 requires Transaction Monitor involvement when prepared transactions failover from one RAC instance to another.

Tuxedo 9.1 provides Transaction Monitor support for Oracle RAC by allowing an administrator to specify lists of groups associated with different RAC instances. This allows Tuxedo to ensure

that groups associated with different instances of the same RAC database do not participate in the same transaction. The Tuxedo Oracle RAC support feature also provides a way for Tuxedo transaction manager server (TMS) processes to be notified of RAC failover events which is required when using Oracle 10gR1.

Consequently, this allows the TMS to re-obtain a list of Oracle 10gR1 prepared transactions from Oracle as required for RAC failover recovery.

When using Oracle 10gR2, administrators should use an Oracle **DTP Service** to access the Oracle RAC system, and this DTP service name should be specified in the OPENINFO string for the associated Tuxedo groups. Oracle will ensure that , and will take care of migrating the service an alternate instance if required.

Limitations

- Tuxedo 9.1 supports Oracle RAC only when using Oracle 10g, and does not support Oracle RAC when using Oracle 9i. For Oracle 10gR1, patch set 10.1.0.3 or later is required, and for Oracle 10gR2, patch set 10.2.0.2 or above is required. Use of Oracle 10gR2 is encouraged, due to the significant improvements in RAC capabilities.
- In some instances, using Oracle RAC with the *Dynamic XA* switch enabled may generate a core dump and cause a system crash. Please contact Oracle Support directly if you encounter this issue and provide the following information:
 - BUG 4644880 - Oracle bug fix identification number
 - the patch set version for the 10g release you are using

The reason that patchset 10.2.0.2 or above is required for Oracle 10gR2 is due to the bug described at

https://metalink.oracle.com/metalink/plsql/f?p=130:14:3193163745563425327:::p14_database_id

Prerequisites

Software Requirements

For specific platform software requirements, refer to the “[BEA Tuxedo 9.1 Platform Data Sheets](#)” in *Installing the BEA Tuxedo System*.

Configuring Tuxedo for Oracle RAC

Tuxedo support for Oracle RAC requires two steps:

- [Configuring Transaction Propagation](#)
- [Configuring Transaction Recovery](#)

The following command and environment variables are used to exclusively configure Tuxedo 9.1 for Oracle RAC support:

- Three environment variables
 - [TUXRACGROUPS](#) (required for all releases of Oracle)
 - [XARETRYDURATIONSECONDS](#) (required only for Oracle 10gR1)
 - [XARETRYINTERVAL](#) (required only for Oracle 10gR1)
- One Command
 - `TMS_rac_refresh(1)` (required only for Oracle 10gR1)

Configuring Transaction Propagation

Oracle 10gR1 does not allow the *same* database to be accessed from *multiple* RAC instances within the *same* XA transaction. In addition, Oracle 10gR1 requires Transaction Monitor involvement when prepared transactions failover from one RAC instance to another.

Oracle 10gR2 permits different RAC instances to operate on different transaction branches in RAC, but if transaction branches are on different instances, then they are loosely coupled and do not share locks. Also, for optimum commit performance, it is important to use only a single RAC instance within a given XA transaction.

For this reason, it is still important to associate an XA transaction with a single RAC instance in Oracle 10gR2. For more information, see the "Developing Applications with Oracle XA" chapter in the *Oracle Database Application Developer's Guide - Fundamentals*.

The `TUXRACGROUPS` environment variable is used to associate Tuxedo groups with specific instances of Oracle RAC configurations so that Tuxedo does not include groups from multiple instances of the same RAC configuration within the same XA transaction.

A single transaction should not span multiple Oracle RAC instances. The groups that participate in a particular transaction are determined at the time the transaction is started. Each transaction is assigned to *one particular instance* of each RAC configuration such that the groups in each instance of a particular RAC configuration are assigned to an equal number of transactions.

TUXRACGROUPS

The TUXRACGROUPS environment variable specifies the groups that are associated with a particular RAC configuration, and will disallow sending service calls in the same transaction to two or more groups identified as different instances of the same RAC configuration.

Warning: The TUXRACGROUPS environment variable *must* be set on all machines in a configuration, and *must* have the same sets of groups specified in the same order on all machines.

If this restriction is not followed, then inconsistent sets of groups can be included within a transaction. The coordinating group will notice the inconsistency at commit time, roll back the transaction, and send an error message to the userlog.

TUXRACGROUPS Syntax

The TUXRACGROUPS environment variable is used to define Oracle RAC group configurations. Its syntax is as follows:

```
TUXRACGROUPS="G1,G2,...,Gm;H1,H2,...,Hn[:I1,I2,...,Io;J1,J2,...,Jp[:...][:...]"
```

Comma (,) separated list

Used to specify groups in the same instance of an Oracle RAC configuration. Multiple groups from a comma separated list can be used together in the same transaction.

Note: Typically, most users place all of the services associated with one database instance in a single group, therefore commas are not needed in the TUXRACGROUPS value.

Semicolon (;) separated list

Used to specify sets of groups in different instances of an oracle RAC configuration. Groups from different RAC instances from the same RAC database configuration cannot be used together in the same transaction.

Since the purpose of the TUXRACGROUPS environment variable is to specify groups associated with different instances of the same Oracle RAC configuration, all applications using the TUXRACGROUPS variable should have at least one semicolon in the environment variable value.

Colon (:) separated list

Used to separate information about one Oracle RAC configuration from information about a different Oracle RAC configuration. The colon indicates that multiple Oracle RAC database configurations are totally independent of each other.

Note: Typically, most users specify only one RAC database configuration, therefore colons are not needed in the `TUXRACGROUPS` value.

TUXRACGROUPS Examples

This section describes four different examples for defining Oracle RAC group configurations:

- [Example 1: Simple Configuration](#)
- [Example 2: Oracle RAC Single Instance with Multiple Groups](#)
- [Example 3: Multiple Oracle RAC Instances with Multiple Groups](#)
- [Example 4: Routing Transactional/Non-transactional Requests](#)

Example 1: Simple Configuration

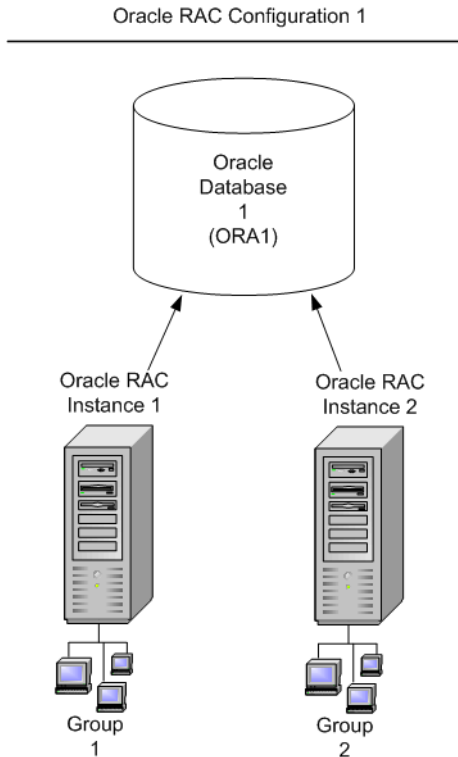
```
TUXRACGROUPS="G1;G2"
```

[Figure 6-1](#) shows a simple Oracle RAC configuration.

In this example, there is one Oracle database, (ORA1), two Oracle RAC instances with 1 group per each instance.

The same transaction request to both `GROUP1` and `GROUP2` cannot be sent because they access database services through different instances that map to the same Oracle RAC database configuration.

Figure 6-1 (ORA1) Simple Configuration



Example 2: Oracle RAC Single Instance with Multiple Groups

`TUXRACGROUPS="GROUP1;GROUP2:GROUP3;GROUP4, GROUP5"`

Figure 6-2 shows an example of adding multiple groups to a single instance.

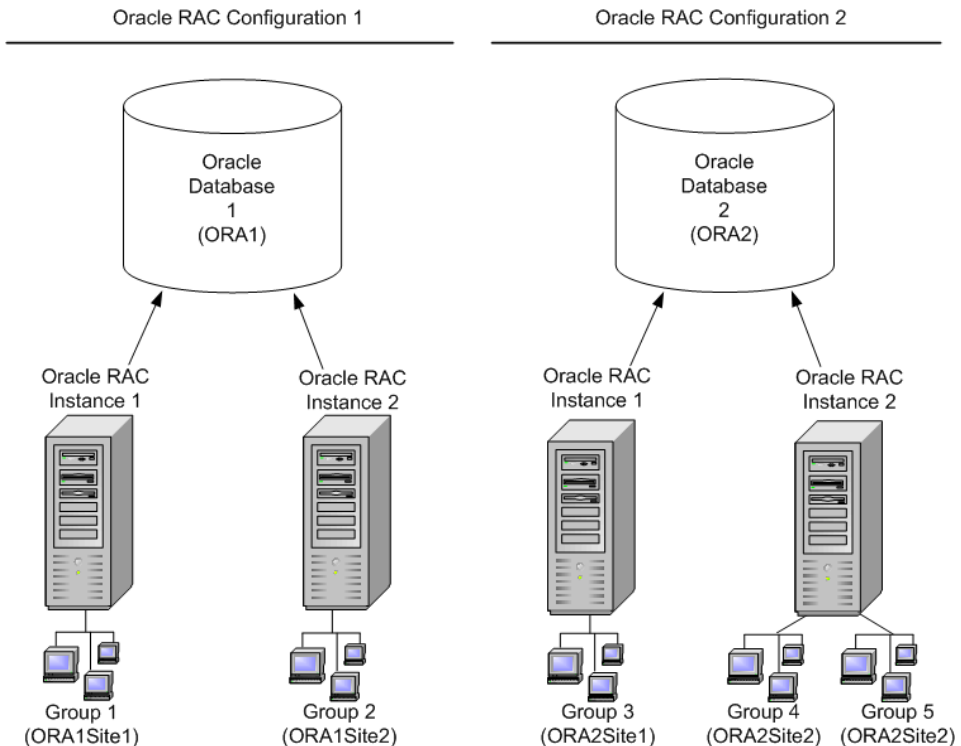
In this example, there are two Oracle databases: ORA1 and ORA2. ORA1 offers machine-specific services ORA1SITE1 and ORA1SITE2, and ORA2 offers machine-specific services ORA2SITE1 and ORA2SITE2. The objective is to assign an approximately equal number of transactions and configure the same services to the groups associated with each instance of an Oracle RAC configuration.

The same transaction request to both GROUP1 and GROUP2 cannot be sent because they access database services through different instances that map to the same Oracle RAC database configuration. The same applies to GROUP3 and GROUP4 or GROUP3 GROUP5, the same transaction cannot be sent to both these groups.

GROUP4 and GROUP5 both access the same database service of the same Oracle RAC database configuration, so these groups would be permitted together. GROUP1 and GROUP4 would be permitted together, because they access different RAC database configurations. If there is also a GROUP6 in this configuration, it would be permitted with any other group, because GROUP6 is not an Oracle RAC group.

Note: The number of groups in each Oracle RAC instance *does not* have to be the same.

Figure 6-2 (ORA2) Single Oracle RAC Instance with Multiple Groups



The `*GROUPS` and `*SERVERS` sections of the `UBBCONFIG` file for this configuration might look as follows:

Listing 6-1 UBBCONFIG File `*GROUPS` and `*SERVERS` Sections Example

`*GROUPS`

Using Tuxedo with Oracle Real Application Clusters (RAC)

```
DEFAULT: TMSNAME=TMS_ORA TMSCOUNT=2

GROUP1      LMID=SITE1      GRPNO=1
OPENINFO="ORACLE_XA:Oracle_XA+Acc=P/scott/tiger+SqlNet=ORA1SITE1+SesTm=100
+LogDir=..+MaxCur=5 "

GROUP2      LMID=SITE2      GRPNO=2
OPENINFO="ORACLE_XA:Oracle_XA+Acc=P/scott/tiger+SqlNet=ORA1SITE2+SesTm=100
+LogDir=..+MaxCur=5 "

GROUP3      LMID=SITE1      GRPNO=3
OPENINFO="ORACLE_XA:Oracle_XA+Acc=P/scott/tiger+SqlNet=ORA2SITE1+SesTm=100
+LogDir=..+MaxCur=5 "

GROUP4      LMID=SITE2      GRPNO=4
OPENINFO="ORACLE_XA:Oracle_XA+Acc=P/scott/tiger+SqlNet=ORA2SITE2+SesTm=100
+LogDir=..+MaxCur=5 "

GROUP5      LMID=SITE2      GRPNO=5
OPENINFO="ORACLE_XA:Oracle_XA+Acc=P/scott/tiger+SqlNet=ORA2SITE2+SesTm=100
+LogDir=..+MaxCur=5 "

GROUP6      LMID=SITE1      GRPNO=6 TMSNAME=TMS_QM
OPENINFO="TUXEDO/QM:/home/myapplication/QUE:QSPACE"

*SERVERS

DEFAULT: RESTART=Y MAXGEN=5 REPLYQ=Y CLOPT="-A "
EMPLOYEE_SVR SRVGRP=GROUP1 SRVID=1
EMPLOYEE_SVR SRVGRP=GROUP2 SRVID=2
BANKING_SVR  SRVGRP=GROUP3 SRVID=3
BANKING_SVR  SRVGRP=GROUP4 SRVID=4
BANKING_SVR  SRVGRP=GROUP5 SRVID=5
```

Note: GROUP4 and GROUP5 have the same OPENINFO strings, because they both use the same database service from the same database.

The specification of the OPENINFO string for Oracle groups in the *GROUPS section is the same as when using Oracle without RAC. For information on how to specify an OPENINFO string for an Oracle group, refer to the *Developing Applications with Oracle XA* chapter in the *Oracle Database Application Developer's Guide - Fundamentals*.

Example 3: Multiple Oracle RAC Instances with Multiple Groups

```
TUXRACGROUPS="GROUP11, GROUP12, GROUP13; GROUP21, GROUP22: GROUP3; GROUP4,  
GROUP5"
```

Figure 6-3 shows an example of adding multiple groups to multiple instances.

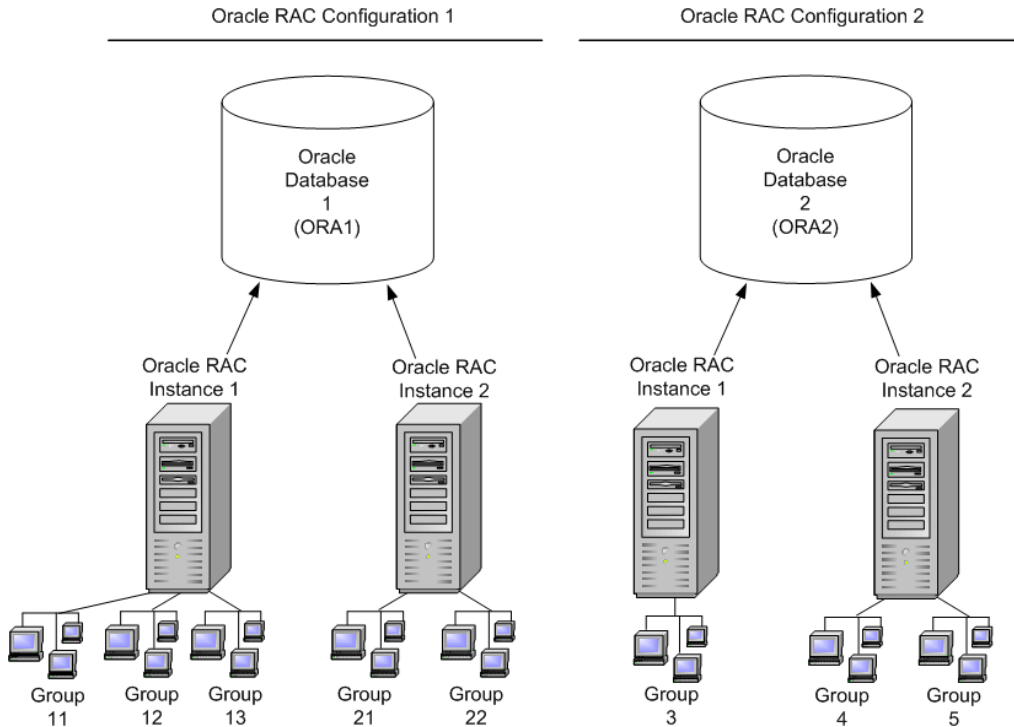
This example is similar to the previous example — except that GROUP11, GROUP12, and GROUP13 are all associated with the first RAC instance of the first RAC configuration, and GROUP21 and GROUP22 are both associated with the second RAC instance.

If the first service call in a transaction in this configuration goes to GROUP12, then it would be possible to send other service calls in this transaction to GROUP11, GROUP12, or GROUP13, but not to GROUP21 or GROUP22.

If a transactional service call is made to a service that is not advertised in any permitted groups but is available in one or more prohibited groups, the result is:

- the call fails
- `tapering` is set to `TPENOENT`
- `tperrordetail` is set to the new value `TPED_GROUP_FORBIDDEN`

Figure 6-3 Multiple Oracle RAC Instances with Multiple Groups



For each RAC configuration defined as part of the `TUXRACGROUPS` environment variable, Tuxedo determines which RAC group(s) in that configuration participate in a particular transaction when that transaction is started.

Example 4: Routing Transactional/Non-transactional Requests

```
TUXRACGROUPS="GROUP1A, GROUP2A; GROUP1B, GROUP2B"
```

Figure 6-4 shows an example of routing transactional and non-transactional requests in an Oracle RAC configuration.

`GROUP1A` and `GROUP2A` are in RAC instance 1 and `GROUP1B` and `GROUP2B` are in RAC instance 2.

Data dependent routing for transactional services offered in RAC groups is likely to achieve the desired result only if:

- each RAC configuration instance offers a service instance that can process each data value.

Since all but one of the instances in a RAC configuration are disallowed in a particular transaction, each data value must be specified for a service in each RAC instance. Otherwise, that data value will not be processed by any service in the RAC configuration for some transactions.

- different RAC service instances process different data values.

If all data values are processed by the same set of service instances, then there is no need to use data dependent routing.

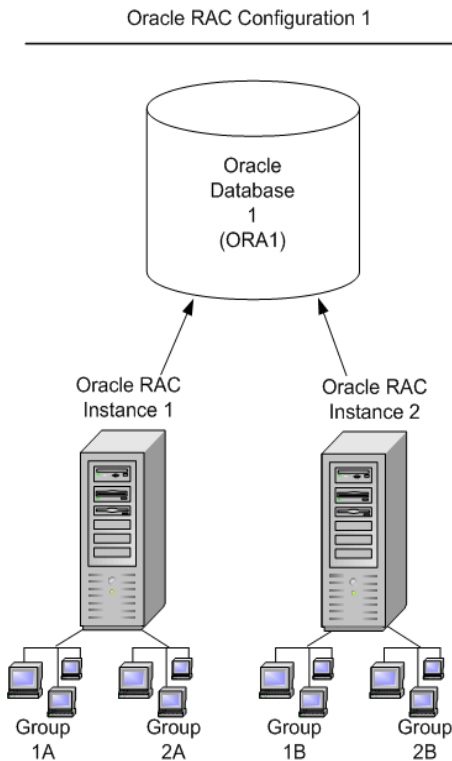
When routing occurs, any groups that are not permitted for the current transaction are ignored. The routing decision only considers:

- groups associated with the allowable RAC instance.
- groups not associated with a RAC configuration.

If routing is performed for a non-transactional request, all groups can participate. The service is routed to the first group matching the data value listed in the `UBBCONFIG` file `*ROUTING` section `RANGES` field.

If routing is performed for a mixture of transactional and non-transactional requests, some applications may not require non-transaction request load balancing. You can vary the RAC instances listed first in your application for different data values so that non-transactional requests are balanced accordingly among services offered by different RAC instances.

Figure 6-4 Routing Transactional/Non-Transactional Requests



The `*SERVICES` and `*ROUTING` sections of the `UBBCONFIG` file for this configuration might look as follows:

Listing 6-2 UBBCONFIG File `*SERVICES` and `*ROUTING` Sections Example

```
*SERVICES

DEPOSIT SRVGRP=GROUP1A ROUTING=MYROUTE
DEPOSIT SRVGRP=GROUP2A ROUTING=MYROUTE
DEPOSIT SRVGRP=GROUP1B ROUTING=MYROUTE
DEPOSIT SRVGRP=GROUP2B ROUTING=MYROUTE

*ROUTING
```

```
MYROUTE FIELD="BRANCH_ID"
RANGES="1-5:GROUP1A, 1-5:GROUP1B, 6-10:GROUP2B, 6-10:GROUP2A, *.*"
BUFTYPE="FML32"
```

Requests with a `BRANCH_ID` of 1 through 5 must be handled by `GROUP1A` or `GROUP1B`. Requests with a `BRANCH_ID` of 6 through 10 must be handled by `GROUP2A` or `GROUP2B`.

Note: The `RANGES` value should duplicate each routing data value. It should specify a service associated with each RAC configuration instance in the `RANGES` field.

For transactional requests, the first half, branches 1-5 map to `GROUP1A` and branches 6-10 map to `GROUP2A`; the other half branches 1-5 map to `GROUP1B` and branches 6-10 map to `GROUP2B`.

For non-transactional requests, branches 1-5 map to `GROUP1A`, and branches 6-10 map to `GROUP2B` because these are the first groups specified that match the respective routing ranges.

Requests with an invalid `BRANCH_ID` are mapped to any permitted group.

Configuring Transaction Recovery

`TMS_rac_refresh(1)`, `XARETRYDURATIONSECONDS`, and `XARETRYINTERVAL` specifically handle transaction recovery issues.

`TMS_rac_refresh(1)` is called when an Oracle RAC group fails over to an alternate group.

`TMS_rac_refresh(1)` *should not* be executed manually from the command line; the proper way to invoke `TMS_rac_refresh(1)` is to use Oracle Fast Application Notification (FAN).

Note: For more details on configuring Oracle FAN, refer to Oracle 10g documentation.

The `XARETRYDURATIONSECONDS` and `XARETRYINTERVAL` environment variables are used to retry transaction recovery operations (`xa_recover()`) as required by Oracle RAC.

`XARETRYDURATIONSECONDS`

Specifies the time interval during which the Tuxedo Transaction Manager Server (TMS) retries `xa_recover()` operations when `TMS_rac_refresh(1)` is called. If it is not set or set to 0, then `xa_recover()` is performed once only.

The default value for `XARETRYDURATIONSECONDS` is 0.

Note: For Oracle 10.1, it is recommended that `XARETRYDURATIONSECONDS` is set to 120.

XARETRYINTERVAL

Specifies the interval in seconds that `xa_recover()` operations are retried during the `XARETRYDURATIONSECONDS` interval. The `XARETRYINTERVAL` value is relevant only if `XARETRYDURATIONSECONDS` is set to a value greater than 0.

The default value for `XARETRYINTERVAL` is 30.

Configuring Oracle 10g Fast Application Notification (FAN)

A key process in configuring Tuxedo for Oracle RAC is setting up Oracle FAN to invoke `TMS_rac_refresh(1)` with the appropriate group parameter on group failover. (More group parameter and group failover information is provided in [Configuring Transaction Propagation](#).)

More information regarding Oracle FAN can be found in the [Workload Management with Oracle Real Application Clusters \(PDF\) White Paper](#).

Oracle FAN Script Example

The following is an example of an Oracle FAN script.

Listing 6-3 Oracle FAN Script Example

```
//This File should be placed at ORA_CRS_HOME/rac/usrco//
-----calout.sh-----

#!/bin/ksh

#parse the event

AWK=awk
NOTIFY_EVENTTYPE=$1 # Event type is handled differently

for ARGS in $*
do
    PROPERTY=`echo $ARGS|$AWK -F"=" '{print $1}'`
    VALUE=`echo $ARGS|$AWK -F"=" '{print $2}'`
    case ${PROPERTY} in
        VERSION|version)NOTIFY_VERSION=$VALUE;;
        SERVICE|service)NOTIFY_SERVICE=$VALUE;;
        DATABASE|database)NOTIFY_DATABASE=$VALUE;;
```

```

        INSTANCE|instance) NOTIFY_INSTANCE=$VALUE;;
        HOST|host) NOTIFY_HOST=$VALUE ;;
        STATUS|status) NOTIFY_STATUS=$VALUE;;
        REASON|reason) NOTIFY_REASON=$VALUE;;
        CARD|card) NOTIFY_CARDINALITY=$VALUE ;;
        TIMESTAMP|timestamp) NOTIFY_LOGDATE=$VALUE;; # catch
event
        ??:?:??) NOTIFY_LOGTIME=$PROPERTY;; # catch event time
        (hh24:mi:ss)
    esac
done

#Set the REFRESH_DIR environment variable.
. /home/oracle/callout.env

#Make a log to record events.
FAN_LOGFILE=/home/oracle/app/products/10.1.0.3.0/db_1/calloutlog/`hostname
`_upti
me.log
touch ${FAN_LOGFILE}
echo ${1} >>${FAN_LOGFILE}

#invoke the TMS_rac_refresh command.
if [ ${NOTIFY_EVENTTYPE} = "INSTANCE" -a ${NOTIFY_STATUS} = "down" ]
then
    ${REFRESH_DIR}/rac_refresh >> ${FAN_LOGFILE} 2>&1
fi
-----callout.sh end-----

-----callout.env-----

#!/bin/ksh
#TUXEDO and Oracle RAC server are not one the same machine.
export REFRESH_DIR=/tmp
-----callout.env end-----

-----rac_refresh-----

```

```
#!/bin/ksh
#If TUXEDO and Oracle RAC server on different machine

. /home/oracle/callout.env

rsh -l ${LOGNAME} ${TUX_MASTER_MACHINE} ${REFRESH_DIR}/rac_refresh
>/tmp/run1.log 2>&1

rsh -l ${LOGNAME} ${TUX_NONMASTER_MACHINE}
${REFRESH_DIR}/rac_refresh >/tmp/run1.log 2>&1

#If TUXEDO and Oracle RAC server are on same machine
#set up environment variable
#export APPDIR=/tmp
#export ORACLE_HOME=/home/oracle/Ora10g
#export TUXDIR=/nfs/users/libo/r902/BJ/bld
#export PATH=.:${PATH}:${TUXDIR}/bin
#. $TUXDIR/tux.env
#export TUXCONFIG=${APPDIR} /tuxconfig

#invoke TMS_rac_refresh
#TMS_rac_refresh RACDBGRP1
#TMS_rac_refresh RACDBGRP3
-----rac_refresh end-----
```

Configuring Transaction Recovery for Oracle 10gR2

For Oracle 10gR2, it is much simpler to configure transaction recovery. The database services specified in the `OPENINFO` string for each group associated with Oracle RAC should be declared in Oracle as DTP services.

For example, in [Listing 6-1](#), `GROUP1` accessed Oracle via service `ORA1SITE1` and `GROUP2` accessed Oracle via service `ORA1SITE2`. In Oracle 10gR2, service `ORA1SITE1` should be declared with `DTP=TRUE`, with preferred instance `SITE1`, and with available instance `SITE2`. Service `ORA1SITE2` should be declared with `DTP=TRUE`, with preferred instance `SITE2`, and with available instance `SITE1`. A similar process should be followed for groups `GROUP3`, `GROUP4`, and `GROUP5`.

By declaring different preferred instances, the application will be able to get the benefit of load balancing during normal operation when both instances are available.

The setting of the `TUXRACGROUPS` environment variable will ensure that different instances of the RAC configuration are not combined in the same transaction in order to obtain optimal performance. If one of the RAC instances goes down, Oracle will transfer the DTP service to the non-preferred instance while maintaining transactional integrity.

When using Oracle 10gR2 DTP services, it is not necessary and is not recommended to configure Oracle FAN, use `TMS_rac_refresh(1)` or set the `XARETRYDURATIONSECONDS` or `XARETRYINTERVAL` environment variables.

Specifying Environment Variables in the UBBCONFIG File

Although the Tuxedo Oracle RAC environment variables can be initiated at the operating system command line, it is highly recommended that you use the `ENVFILE` parameter specified in the `*MACHINES` section of the `UBBCONFIG` file to initiate these environment variables.

Apply the following syntax considerations when setting the environment variables for Oracle RAC.

- When Tuxedo environment variables are set using `ENVFILE`, which is the preferred method, quotation marks *are not* permitted around the environment variable value.
- If environment variables are set at the command line, quotation marks *are required* if environment variable values contain characters that could be interpreted as *special* by the command line interpreter. An example of a *special* character is a semicolon.
- Ensure that the Tuxedo Oracle RAC environment variables are set consistently on all nodes in a RAC configuration.

See Also

- [buildtms\(1\)](#)
- [UBBCONFIG\(5\)](#)
- “About Transactions” on page 4-1
- “Configuring Your ATMI Application to Use Transactions” on page 5-1
- “Writing Global Transactions” on page 9-1
- [Oracle Database 10g, Oracle Real Application Clusters Home Page](#)

- [Oracle Application Server 10g Adapters for Tuxedo](#)
- [Best Practices for Using XA with RAC](#)

Managing The Tuxedo Service Metadata Repository

This topic includes the following sections:

- [Tuxedo Service Metadata Repository](#)
- [Creating The Tuxedo Service Metadata Repository](#)
- [Configuring The Tuxedo Service Metadata Repository Server](#)
- [Accessing The Tuxedo Service Metadata Repository File](#)

Tuxedo Service Metadata Repository

The Tuxedo service metadata repository contains BEA Tuxedo service definitions that allow Tuxedo clients to access BEA Tuxedo service parameter information. It provides Tuxedo application developers and administrators the ability to store and retrieve detailed service parameter information on any or all Tuxedo application services.

The Tuxedo service metadata repository is designed to process interactive queries by developers and administrators during application development or modification. It is not designed to process high volumes of automated queries during the application production phase.

Five utilities are used in conjunction with the Tuxedo service metadata repository

- [TMMETADATA \(5\)](#) : Tuxedo service metadata repository server. It provides one service, `.TMMETAREPOS`, which uses an FML32 input and output buffer format described in [METAREPOS \(5\)](#)

Note: The `.TMMETAREPOS` buffer format is similar to [MIB \(5\)](#).

- `tmloadrepos (1)` : creates or updates the binary metadata repository file and loads it with service parameter information.
- `tmunloadrepos (1)` : displays service information from the Tuxedo service metadata repository. Output can be optionally specified as plain text format, WSDL format, or C pseudocode
- `tpgetrepos (3c)` : programmatically uses FML32 buffers to output service information from the Tuxedo service metadata repository
- `tpsetrepos (3c)` : programmatically uses FML32 buffers to add, delete, or update service parameter information to the metadata repository file

JOLT Repository Similarities and Differences

BEA Jolt also provides a service repository that allows applications to manually enter Tuxedo service information, including service names, input and output buffer types, parameter names, parameter data types, the number of times each parameter is expected, and whether each parameter is for input, output, or both. All of which seem very similar to the Tuxedo service metadata repository. However, there are also some distinct difference as noted in the following table:

Table 7-1 JOLT Repository Similarities and Differences

	Jolt Repository	Tuxedo Service Metadata Repository
Function	Designed for JAVA client communication with Tuxedo servers	Designed for Web service based communication with Tuxedo servers
GUI interface	Yes	No
Uses plain text repository input file	Yes (via bulkloader)	Yes (via <code>tmloadrepos</code>)
Service and Parameter keywords and values	Yes	Yes (but more than Jolt Repository to provide the user with more detailed service information)
Repository file format	Plain text	Binary (service indexed for quick access)

Table 7-1 JOLT Repository Similarities and Differences

Access method	Administration utility, direct editing	Administration utility, programming API, system service
Unload output format	Plain text	Plain text, WDSL, and C-pseudocode
Inter operability	Cannot read Tuxedo service metadata repository file	Can read, but not modify, existing Jolt Repository file. Cannot create a Jolt Repository file.

MIB(5) Similarities and Differences

Programmatic access to the Tuxedo System Metadata Repository is accomplished through the use of a FML32 buffer format that is very similar to the Tuxedo MIB format. However, there are also some distinct difference as noted in the following table:

Table 7-2 MIB(5) Similarities and Differences

	MIB (5)	METAREPOS (5)
Input/out buffers	FML32	FML32
Generic MIB fields	Yes	Yes, but with some limitations. See METAREPOS (5)
Authentic MIB class entities	Many	No authentic MIB class entities, but uses similar type
Service entry	.TMIB in BBL	.TMMETAREPOS in TMMETADA server

Creating The Tuxedo Service Metadata Repository

The metadata repository file contains all the service parameter information that is accessed in the Tuxedo service metadata repository. The `tmloadrepos` command is used to create a metadata repository file. Metadata repository file service parameter information is input directly from the

computer console (*standard* input) if a repository input file is not specified or from a specified plain text *repository input file*. For example:

```
tmloadrepos -i/usr/tuxedo/repository_input_file/usr/tuxedo/  
service_metatdata_repository
```

The Tuxedo Service Metadata Repository Input File

The `repository_input_file` contains service parameter keywords and their associated values. Keywords are divided into two categories: service-level and parameter-level.

Note: Keyword abbreviations are also supported. Both keywords and abbreviations are case sensitive. For more information on keywords, abbreviations, and values, see [Using Service-Level Keywords and Values](#) and [Using Parameter-Level Keywords and Values](#).

No more than one keyword/value combination can be specified per line. The maximum line character length is 1024 bytes. String parameter values do not need to be set off with quotation marks.

The `repository_input` file uses the following syntax: `<keyword>=<value>` and has the following input conventions:

`" (" and ") "`

When a parameter must define a sub-parameter, a line consisting of a single left parenthesis '(' and a line consisting of a single right parenthesis ')' denotes the beginning and end of the embedded sub-parameter portion of the parameter. The left and right parentheses can be used recursively.

`\ and \"`

You can include blank lines in the `repository_input` file as needed for readability. A new line is preceded by a `\` character. To use an actual `\` character it must be written as `\\`.

`#`

Lines starting with a `#` are interpreted as comment lines. Unlike comments specified via the `svcdescription` or `paramdescription` keywords, comments are not stored in the binary `repository_file` or output by `tmunloadrepos`.

The `repository_input` file can consist of zero or more service parameter definitions. Each service definition starts with a line beginning with the `<service>` keyword followed by zero or more lines beginning with one of the other service-level keywords, followed by parameter-level keywords. A particular service-level keyword may not be repeated for a particular service.

Using Service-Level Keywords and Values

A service definition must begin with the keyword `service<=NAME>` or the abbreviation `sv<=NAME>`. Services using `CARRAY`, `STRING`, or `XML` buffer types can have only one parameter per service. The Tuxedo service metadata repository service-level keywords are as follows:

Table 1 Service-Level Keyword, Abbreviations, and Values

Service-Level Keyword	Keyword Abbreviation	Value
<code>service</code>	<code>sv</code>	Any Tuxedo service name Note: This key valued can only be once per Metadata Repository instance. It cannot be duplicated within the same Metadata Repository.
<code>tuxservice</code>	<code>tsv</code>	Actual Tuxedo service name Note: The difference between the <code>service</code> and <code>tuxservice</code> keywords is: <ul style="list-style-type: none"> <code>service</code> represents the service entry stored in the Metadata Repository. <code>tuxservice</code> represents the actual Tuxedo service name. Two or more <code>service</code> definitions can have the same value as <code>tuxservice</code>. When used together, these two keywords make it possible to have multiple service definitions for one Tuxedo service. By default, <code>tuxservice</code> has the same value as <code>service</code>
<code>servicetype</code>	<code>st</code>	Service invocation type. Legal values are: <code>service</code> - the service is a synchronous <code>oneway</code> - the service will not send response to the client <code>queue</code> - the service is a /Q related application
<code>export</code>	<code>ex</code>	Y (default) or N. In the Jolt repository, this keyword is used to determine service availability to the Jolt client. In the Tuxedo repository, this keyword does not have any meaning, but is nevertheless accepted to maintain compatibility with existing Jolt bulk loader files.

Table 1 Service-Level Keyword, Abbreviations, and Values (Continued)

Service-Level Keyword	Keyword Abbreviation	Value
inbuf	bt	Select one of these buffer types: FML, FML32, VIEW, VIEW32, STRING, CARRAY, XML, X_OCTET, X_COMMON, X_C_TYPE, MBSTRING or an application defined buffer type. Note: "inbuf" value of each service definition cannot be NULL.
outbuf	BT	Select one of these buffer types: FML, FML32, VIEW, VIEW32, STRING, CARRAY, XML, X_OCTET, X_COMMON, X_C_TYPE, MBSTRING or an application defined buffer type. Note: The "outbuf" value of each service definition cannot be NULL for a "service" typed service or a "queue" typed service.
inview	vn	View name for input parameters (Optional) Note: This keyword is mandatory only if one of the following buffer types is used: VIEW, VIEW32, X_COMMON, X_C_TYPE.)
outview	VN	View name for output parameters (Optional) Note: This keyword is mandatory only if one of the following buffer types is used: VIEW, VIEW32, X_COMMON, X_C_TYPE.
svcdescripti on	sd	Any string value. A new-line break can be used to improve readability if the string is too long.
sendqspace	sqS	Send queue space name. Optional for a "queue" typed service.
sendqueue	sqn	Send queue name. Optional for a "queue" typed service.
rplyqueue	rqn	Reply queue name. Optional for a "queue" typed service.
errqueue	eqn	Error queue name. Optional for a "queue" typed service.
rcvqspace	RQS	Receive queue space name. Optional for a "queue" typed service.

Table 1 Service-Level Keyword, Abbreviations, and Values (Continued)

Service-Level Keyword	Keyword Abbreviation	Value
<code>rcvqueue</code>	<code>RQN</code>	Receive queue name. Optional for a "queue" typed service.
<code>version</code>	<code>vs</code>	This parameter is exclusive to the Tuxedo service metadata repository and accommodates any string value used by the application. BEA Tuxedo does not interpret this parameter.
<code>attributes</code>	<code>att</code>	This parameter is exclusive to the Tuxedo service metadata repository and accommodates any string value used by the application. BEA Tuxedo does not interpret this parameter.
<code>fieldtbls</code>	<code>ftb</code>	This parameter is optional and specifies a comma-separated list of field tables where the FML or FML32 fields used by this service can be found. The <code>fieldtbls</code> parameter is intended for reference use by application developers.

Using Parameter-Level Keywords and Values

A parameter begins with the keyword `<param>=<NAME>` or the abbreviation `<pn>=<NAME>` followed by a listing of parameter keywords. It ends with another `<param>` or `<service>` keyword, or when end-of-file is encountered. The parameters can be listed in any order after `<param>=<NAME>`.

Note: A particular service can specify multiple occurrences of the `<param>` keyword. That is to say, more than one parameter can exist for a particular service. For example, a parameter with an `FML` or `VIEW` buffer.

The Tuxedo service metadata repository parameter-level keywords are as follows:

Table 2 Parameter-Level Keyword, Abbreviations, and Values

Parameter-Level Keyword	Metadata Repository Abbreviation	Value
param	pn	Any parameter name
type	pt	<p>byte, short, integer, float, double, string, carray, dec_t, xml, ptr, fml32, view32, mbstring.</p> <p>Note: The parameter type must be consistent with its service buffer type. For example, an FML16 buffer only allow parameters with the following type: byte (char), short, integer, long, float, double, string, carray. All other type parameters are not permitted. See following buffer type/parameter type matching table.</p>
subtype	pst	A view name for a view32 typed parameter
access	pa	<p>in, out, inout, noaccess.</p> <p>in - indicates a parameter that is used for input only. out - indicates a parameter that is used for output only. inout - indicates a parameter that is used for both input and output. noaccess - indicates a parameter that must be provided on input but which is not referenced in the server, such as an obsolete parameter or a parameter that must be provided as a filler field in a view.</p> <p>The set of parameters expected on input is those specified with in, inout, or noaccess access</p> <p>The set of parameters returned on output is those specified with either out or inout access.</p>
count	po	<p>Maximum number of occurrences (default is 1). The value for unlimited occurrences is 0. The value range is [0 , 32767].</p> <p>In the Jolt repository, this parameter is used only by the Repository Editor to format test screens. In the Tuxedo repository, this parameter is stored for display and is also used by tmunloadrepos (1) pseudocode generation options.</p>

Table 2 Parameter-Level Keyword, Abbreviations, and Values (Continued)

Parameter-Level Keyword	Metadata Repository Abbreviation	Value
paramdescription	pd	Any string value. A new-line break can be used to improve readability if the string is too long.
size	pl	<p>This optional parameter indicates the number of bytes allocated for the parameter. It is used in pseudo code generation for non-numeric parameters and can be used for programmer reference purposes.</p> <p>The following parameter types expect this value: <code>carray</code>, <code>string</code>, <code>xml</code>, <code>mbstring</code>.</p>
requiredcount	ro	Minimum number of times that the parameter must be specified. The value range is [0, 32767].
fldnum	fno	<p>This optional parameter indicates the field number of the parameter if it is a FML/FML32 field.</p> <p>Note: Users are not encouraged to provide this information if they have already defined <code>fieldtbl</code> files by indicating field table directories using environment <code>FLDTBLDIR(32)</code> and indicating field table files using environment <code>FLDTBLS(32)</code> or <code>fieldtbl</code> service-level keyword.</p>
vfbname	vfb	This parameter is optional for view structure members. It is used to indicate the field name in the fielded buffer. Please reference viewfile(5)
vflag	vfl	This parameter is optional for view structure members. Legal values are combination of the following options: 'C', 'F', 'L', 'N', 'P', 'S'. Please reference viewfile(5) .
vnull	vnu	This parameter is optional for view structure members. It indicates the view member default null value.

Table 2 Parameter-Level Keyword, Abbreviations, and Values (Continued)

Parameter-Level Keyword	Metadata Repository Abbreviation	Value
(<p>Indicates the beginning of the description of the parameters contained in an embedded FML32 or VIEW32 buffer field.</p> <p>It contains no associated value and is specified separately on a line by itself. It is valid only if a previous type keyword has been specified for this parameter with a FML32 or VIEW32 value.</p> <p>A closing right parenthesis ')' ends the embedded parameter description.</p>
)		<p>Ends an embedded FML32 or VIEW32 parameter definition of that began with an opening matching left parenthesis '('.</p> <p>It contains no associated value and is specified separately on a line by itself. It is valid only if a previous only if there is a previous matching '(' keyword.</p> <p>In addition, the maximum embedded level depends on the upper limit of embedded FML32 nesting level (18 at present).</p>

Parameter Occurrences

As a generally applied Tuxedo rule, only FML/FML32, VIEW/VIEW32, X_COMMON, and X_C_TYPE typed buffers can specify multiple parameters (due to their information structure). All other typed buffers have only one parameter with the corresponding parameter type. For example, a CARRAY type buffer has only one CARRAY typed parameter to describe the necessary information that it contains. You must follow this rule to define application services.

Table 3 Service Buffer Type (SMALL CAPS)/Service Parameter Type (lower case) Matching Table I

	byte (char)	short	integer	long	float	double	string
CARRAY							
FML	X	X	X	X	X	X	X

Table 3 Service Buffer Type (SMALL CAPS)/Service Parameter Type (lower case) Matching Table I (Continued)

	byte (char)	short	integer	long	float	double	string
FML32	X	X	X	X	X	X	X
STRING							X
VIEW	X	X	X	X	X	X	X
VIEW32	X	X	X	X	X	X	X
X_COMMON		X		X			X
X_C_TYPE	X	X	X	X	X	X	X
XML							X
X_OCTET							
MBSTRING							

Table 4 Service Buffer Type (SMALL CAPS)/Service Parameter Type (lower case) Matching Table II

	carray	dec_t	xml	ptr	fml32	view32	string
CARRAY	X						
FML	X						
FML32	X			X	X	X	X
STRING							
VIEW	X	X					
VIEW32	X	X					
X_COMMON							
X_C_TYPE							
XML			X				

Table 4 Service Buffer Type (SMALL CAPS)/Service Parameter Type (lower case) Matching Table II (Continued)

	carray	dec_t	xml	ptr	fml32	view32	string
X_OCTET	X						
MBSTRING	X						X

Configuring The Tuxedo Service Metadata Repository Server

To configure the Tuxedo service metadata repository you must:

- add `TMMETADATA` to the `*SERVERS` section of the `UBBCONFIG (5)` .
- run `tmloadcf (1)` on the `UBBCONFIG` file.
- use `tmloadrepos (1)` to create and enter service parameter information into the metadata repository file.
- boot the server.

Once the Tuxedo metadata server is running, the `.TMMETAREPOS` service is automatically activated. `.TMMETAREPOS` is a Tuxedo system service and cannot be modified.

All requests made to the server are responded to on a first-come-first-served basis.

Configuring Multiple Tuxedo Service Metadata Repository Servers

Setting up multiple `TMMETADATA` servers on a particular Tuxedo node requires adherence to two crucial configuration rules:

- Each `TMMETADATA` server must be configured to access the same metadata repository file or an exact copy of the file to provide consistent request results. Therefore, it is strongly recommended that a stable version of the metadata repository is made available for multiple `TMMETADATA` server access.
- Permission settings must be consistently applied (either read only or read/write) for multiple `TMMETADATA` servers on a particular node.

Accessing The Tuxedo Service Metadata Repository File

The Tuxedo service metadata repository facilitates native and remote client access in order to view, update, add, or delete service metadata repository parameter information.

- For native clients *exclusively*, `tpgetrepos(3c)`, and `tpsetrepos(3c)` are used for Tuxedo service metadata repository access.

`tpgetrepos(3c)` and `tpsetrepos(3c)` can access the Tuxedo service metadata repository whether the server is booted or not.
- For remote and native clients, `TMMETADATA(5)` can be used.

See Also

`TMMETADATA(5)`, `METAREPOS(5)`, `tmloadrepos(1)`, `tmunloadrepos(1)`, `tpgetrepos(3c)`, `tpsetrepos(3c)`

Managing CORBA Interface Repositories

This topic, which is specific to BEA Tuxedo CORBA environments, includes the following sections:

Note: The BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB were deprecated in Tuxedo 8.1 and are no longer supported in Tuxedo 9.x. All BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB text references, associated code samples, etc. should only be used:

- to help implement/run third party Java ORB libraries, and
- for programmer reference only.

Technical support for third party CORBA Java ORBs should be provided by their respective vendors. BEA Tuxedo does not provide any technical support or documentation for third party CORBA Java ORBs.

- [Administration Considerations](#)
- [Using Administration Commands to Manage Interface Repositories](#)
- [Configuring the UBBCONFIG File to Start One or More Interface Repository Servers](#)

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

For related programming information, see the *CORBA 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) .

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 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 than 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 Repository for a BEA Tuxedo domain:

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

Prerequisites

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

On Windows:

```
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:

```
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 Repository

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 exist, the command loads the specified interface definitions into it. The format of the command is as follows:

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

For a detailed description of this command, see the *File Formats, Data Descriptions, MIBs, and System Processes Reference* in the BEA Tuxedo online documentation.

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

Displaying or Extracting the Content of an Interface Repository

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

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

For a detailed description of this command, see the *File Formats, Data Descriptions, MIBs, and System Processes Reference* in the BEA Tuxedo online documentation.

Deleting an Object from an Interface Repository

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

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

For a detailed description of this command, see the *File Formats, Data Descriptions, MIBs, and System Processes Reference* in the BEA Tuxedo online documentation.

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

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

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

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

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

Note: Other server entries are shown in the following sample to emphasize that the order in which servers are started for BEA Tuxedo CORBA applications is critical. A BEA Tuxedo CORBA application will not boot if the order is changed.

For more information, see the section [“Required Order in Which to Boot CORBA C++ Servers” on page 3-63 in Chapter 3, “Creating the 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"
```

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

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

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

For a description of the `TMIFRSVR -f filename` parameter, refer to the *File Formats, Data Descriptions, MIBs, and System Processes Reference*. 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 [“How to Create the SERVERS Section of the Configuration File”](#) on page 3-56 in [Chapter 3, “Creating the Configuration File,”](#) for details about parameters such as `SRVGRP`, `SRVID`, `RESTART`, `MAXGEN`, and `GRACE`.

Distributing ATMI Applications Across a Network

This topic includes the following sections:

- [What Is a Distributed ATMI Application?](#)
- [Why Distribute an ATMI Application Across a Network?](#)

Note: For detailed information about distributing BEA Tuxedo CORBA applications across a network, refer to the *Scaling, Distributing, and Tuning CORBA Applications* guide.

What Is a Distributed ATMI Application?

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

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

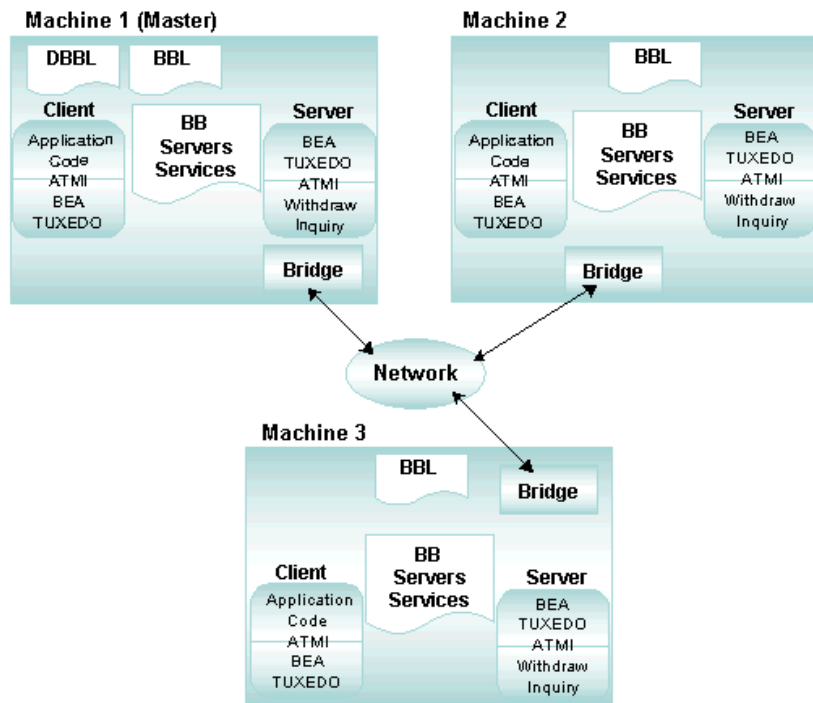
State of the art telecommunications and data networks are making distributed operations of this sort increasingly common. Applications developed to implement this type of strategy allow businesses to reduce costs and enhance their offerings of services to customers around the world.

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

Example of a Distributed Application

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

Figure 9-1 Sample of a Distributed Application



Implementing a Distributed Application

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

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

See Also

- [“Creating the Configuration File for a Distributed ATMI Application” on page 10-1](#)
- [“Setting Up the Network for a Distributed Application” on page 11-1](#)
- [“Managing the Network in a Distributed Application” on page 4-1](#) in *Administering a BEA Tuxedo Application at Run Time*
- *Scaling, Distributing, and Tuning CORBA Applications*

Why Distribute an ATMI Application Across a Network?

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

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

A distributed application allows you to do the following:

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

Features of a Distributed Application

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

See Also

- [“How to Create the Configuration File for a Multiple-machine \(Distributed\) Application” on page 3-3.](#)
- [“What Is Load Balancing?”](#) in *Introducing BEA Tuxedo ATMI*
- [“What Is Data-Dependent Routing?”](#) in *Introducing BEA Tuxedo ATMI*
- *Scaling, Distributing, and Tuning CORBA Applications*

Creating the Configuration File for a Distributed ATMI Application

This section includes the following topics:

- [Configuration File Requirements for a Distributed BEA Tuxedo ATMI Application](#)
- [Creating the RESOURCES Section](#)
- [Creating the MACHINES Section](#)
- [Creating the GROUPS Section](#)
- [Creating the SERVICES Section](#)
- [Creating the ROUTING Section](#)
- [Example Configuration File for a Distributed Application](#)
- [Modifying the Domain Gateway Configuration File to Support Routing](#)

Note: For detailed information about creating a configuration file for a distributed BEA Tuxedo CORBA application, refer to the *Scaling, Distributing, and Tuning CORBA Applications* guide.

Configuration File Requirements for a Distributed BEA Tuxedo ATMI Application

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

a distributed configuration, you must create a configuration file that includes the following sections:

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

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

Creating the RESOURCES Section

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

Note: The parameters described in the tables in this topic are used only for distributed applications. For a description of the basic parameters that are available for any kind of BEATuxedo application, see [UBBCONFIG \(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*.

Table 10-1 RESOURCES Section Parameters

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

Table 10-1 RESOURCES Section Parameters (Continued)

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

Table 10-1 RESOURCES Section Parameters (Continued)

Parameter	Description
SANITYSCAN (Optional)	<p>SANITYSCAN sets a multiplier of the basic SCANUNIT between sanity checks of the system.</p> <p>The value of SCANUNIT must be greater than 0. The default is set so that (SCANUNIT * SANITYSCAN) is approximately 120 seconds.</p> <p>Sanity checks are performed on servers as well as on the bulletin board data structure itself.</p>
SCANUNIT (Optional)	<p>SCANUNIT sets the time interval (in seconds) between scans by the bulletin board liaison for timed-out transactions and blocking calls within service requests. This value is used as the basic unit of scanning by the BBL. It affects the granularity with which transaction timeout values can be specified on <code>tpbegin(3c)</code> and the blocking timeout value specified with the BLOCKTIME parameter. The SANITYSCAN, BBLQUERY, DBBLWAIT, and BLOCKTIME parameters are multipliers of this unit for other timed operations within the system.</p> <p>The value of SCANUNIT must be a multiple of 2 or 5 greater than 0 and less than or equal to 60 seconds. The default is 10 seconds.</p>

Creating the MACHINES Section

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

Table 10-2 MACHINES Section Parameters

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

Creating the GROUPS Section

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

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

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

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

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

Table 10-3 GROUPS Section Parameters

Parameter	Description
<code>ENVFILE</code>	<p><code>ENVFILE</code> specifies a file that defines the environment with which all servers in the group are executed.</p> <p>Lines must be in the form <i>ident=value</i> where <i>ident</i> contains only underscores and/or alphanumeric characters.</p> <p>If the value of <code>ENVFILE</code> is an invalid filename, no values are added to the environment.</p>
<code>GRPNO</code> (Required)	<p><code>GRPNO</code> associates a number with a particular server group.</p> <p>The number must be greater than 0 and less than 30,000. It must be unique among entries in the <code>GROUPS</code> section.</p>
<code>LMID</code> (Required)	<p><code>LMID</code> identifies the machine on which the server group being defined runs. A second <code>LMID</code> value can be specified (separated from the first by a comma) for an alternate machine to which this server group can be migrated if the <code>MIGRATE</code> option has been specified. Servers in the group can be migrated if <code>RESTART=Y</code> to migrate is specified in the <code>GROUPS</code> section.</p> <p>The values of <code>LMID</code> must be the values assigned to the <code>LMID</code> parameter in the <code>MACHINES</code> section.</p>

Creating the SERVICES Section

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

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

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

Table 10-4 SERVICES Section Parameters

Parameter	Description
LOAD (Optional)	<p>LOAD specifies the size of the load imposed by <code>SVCNM</code> on the system.</p> <p>The value of <code>LOAD</code> must be a number between 1 and 32,767, inclusive. A higher number indicates a greater load. The default is 50.</p>
PRIOR (Optional)	<p>PRIOR specifies the dequeuing priority of <code>SVCNM</code>.</p> <p>The value of <code>PRIOR</code> must be greater than 0 and less than or equal to 100, with 100 being the highest priority. The default is 50.</p>
ROUTING (optional)	<p>ROUTING specifies the name of the routing criteria used for this service when data-dependent routing is being performed. If this parameter is not specified, data-dependent routing is not performed for this service.</p> <p>The value of <code>ROUTING</code> may contain up to 15 characters. If multiple entries exist for the same service name but with different <code>SRVGRP</code> parameters, the <code>ROUTING</code> parameter must be the same for all entries.</p>

Table 10-4 SERVICES Section Parameters

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

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

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

```
*SERVICES

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

Creating the ROUTING Section

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

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

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

```
CRITERION_NAME required_parameters
```

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

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

Table 10-5 ROUTING Section Parameters

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

See Also

- [“How to Create the Configuration File for a Multiple-machine \(Distributed\) Application” on page 3-3](#)
- [UBBCONFIG \(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*
- *Scaling, Distributing, and Tuning CORBA Applications*

Example Configuration File for a Distributed Application

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

```
*GROUPS
BANKB1          GRPNO=1
BANKB2          GRPNO=2
BANKB3          GRPNO=3
#
*SERVICES
```

```

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

```

Modifying the Domain Gateway Configuration File to Support Routing

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

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

Note: For more information about the `DMCONFIG` file, refer to [DMCONFIG\(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*.

Description of ROUTING Section Parameters in DMCONFIG

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

CRITERION_NAME required_parameters

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

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

Modifying the Domain Gateway Configuration File to Support Routing

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

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

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

Routing Field Description

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

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

Example of a 5-Site Domain Configuration Using Routing

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

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

Listing 10-1 Domains Configuration File for Five Sites

```
# TUXEDO DOMAIN CONFIGURATION FILE FOR THE CENTRAL BANK
#
#
*DM_LOCAL
# local_domain_name Gateway_Group_name domain_type domain_ID log_device
#
#           [audit log] [blocktime]
#           [log name] [log offset] [log size]
#           [maxaccesspoint] [maxraptran] [maxtran]
#           [maxdatalen] [security]
#           [tuxconfig] [tuxoffset]
#
#
#
DEFAULT: SECURITY = NONE
c01      GWGRP = bankg1
        TYPE = TDOMAIN
        ACCESSPOINTID = "BA.CENTRAL01"
        DMTLOGDEV = "/usr/apps/bank/DMTLOG"
        DMTLOGNAME = "DMTLG_C01"
c02      GWGRP = bankg2
        TYPE = OSITP
        ACCESSPOINTID = "BA.CENTRAL01"
        DMTLOGDEV = "/usr/apps/bank/DMTLOG"
        DMTLOGNAME = "DMTLG_C02"
        NWDEVICE = "OSITP"
        URCH = "ABCD"
#
#
*DM_REMOTE
#remote_domain_name domain_type domain_ID
#
b01      TYPE = TDOMAIN
        ACCESSPOINTID = "BA.BANK01"
b02      TYPE = TDOMAIN
        ACCESSPOINTID = "BA.BANK02"
```

```

b03      TYPE = TDOMAIN
        ACCESSPOINTID = "BA.BANK03"
b04      TYPE = OSITP
        ACCESSPOINTID = "BA.BANK04"
        URCH = "ABCD"
#
*DM_TDOMAIN
#
#      local_or_remote_domain_name network_address [nwdevice]
#
# Local network addresses
c01      NWADDR = "//newyork.acme.com:65432"      NWDEVICE = "/dev/tcp"
c02      NWADDR = "//192.76.7.47:65433"      NWDEVICE = "/dev/tcp"
# Remote network addresses: second b01 specifies a mirrored gateway
b01      NWADDR = "//192.11.109.5:1025" NWDEVICE = "/dev/tcp"
b01      NWADDR = "//194.12.110.5:1025" NWDEVICE = "/dev/tcp"
b02      NWADDR = "//dallas.acme.com:65432" NWDEVICE = "/dev/tcp"
b03      NWADDR = "//192.11.109.156:4244" NWDEVICE = "/dev/tcp"
#
*DM_OSITP
#
#local_or_remote_domain_name apt aeq
#                                [aet] [acn] [apid] [aeid]
#                                [profile]
#
c02      APT = "BA.CENTRAL01"
        AEQ = "TUXEDO.R.4.2.1"
        AET = "{1.3.15.0.3},{1}"
        ACN = "XATMI"
b04      APT = "BA.BANK04"
        AEQ = "TUXEDO.R.4.2.1"
        AET = "{1.3.15.0.4},{1}"
        ACN = "XATMI"
*DM_EXPORT
#service_name [Local_Domain_name] [access_control] [exported_svcname]
#            [inbuftype] [outbuftype]
#
open_act      ACL = branch
close_act     ACL = branch
credit
debit
balance
loan          LACCESSPOINT = c02      ACL = loans
*DM_IMPORT
#service_name [Remote_domain_name] [local_domain_name]
#            [remote_svcname] [routing] [conv]
#            [trantime] [inbuftype] [outbuftype]
#
t1r_add      LACCESSPOINT = c01  ROUTING = ACCOUNT

```



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

See Also

- [“Understanding the Domains Configuration File” on page 1-15](#) in *Using the BEA Tuxedo Domains Component*
- [“Setting Up a Domains Configuration” on page 2-27](#) in *Using the BEA Tuxedo Domains Component*
- *Scaling, Distributing, and Tuning CORBA Applications*

Setting Up the Network for a Distributed Application

This topic includes the following sections:

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

Configuring the Network for a Distributed Application

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

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

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

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

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

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

How Data Moves Over a Network

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

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

How Data Moves Over Parallel Networks

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

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

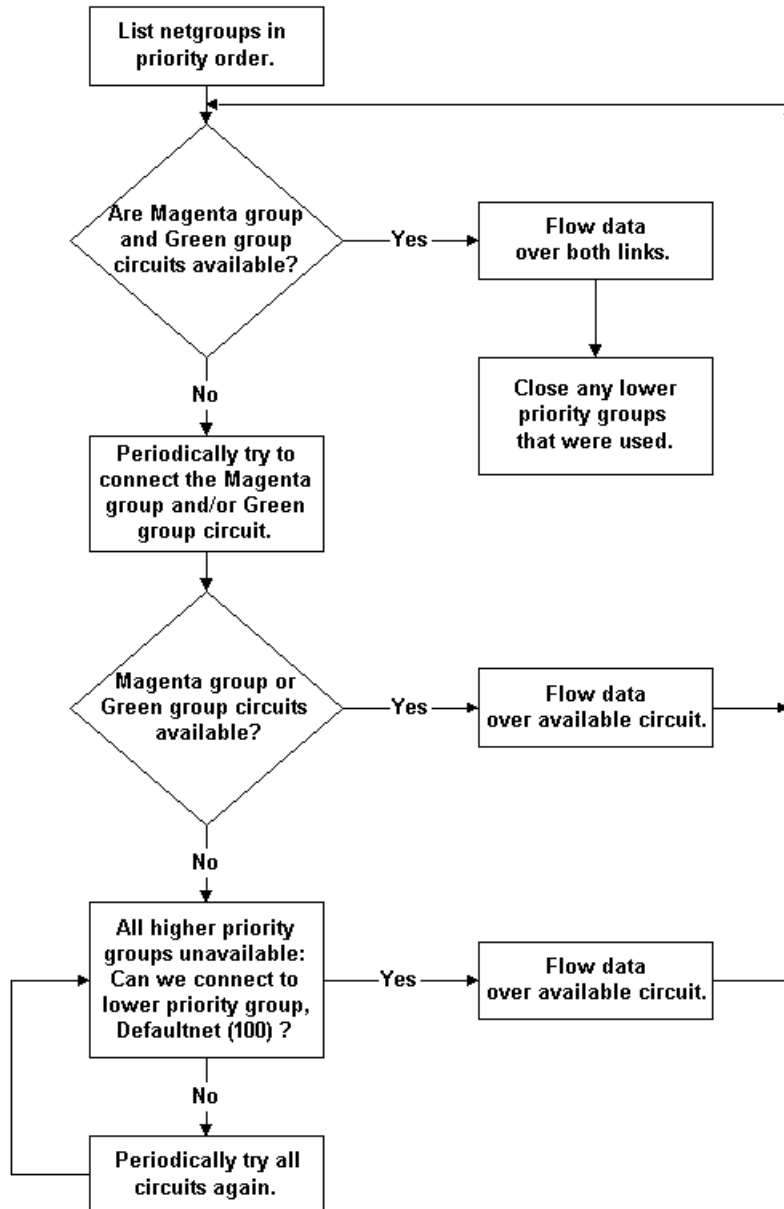
Before making a decision to use parallel data circuits, however, you should determine whether it will be important, in your application, for messages to be kept in sequence. The system guarantees that conversational messages are kept in the correct sequence by binding the conversation connection to one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

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

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

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

Figure 11-1 Flow of Data over the BRIDGE



Example of a Network Configuration for a Simple Distributed Application

The following example shows how to configure a simple network:

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

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

How Failover and Failback Work in Scheduling Network Data

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

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

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

Example Configuration of Multiple Netgroups

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

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

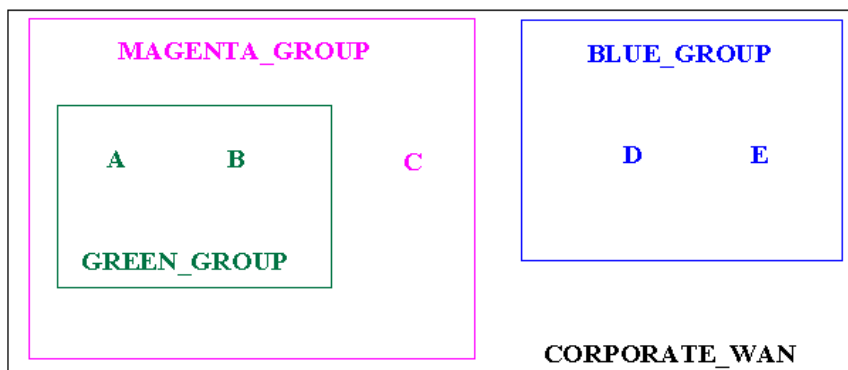
In the following example, it is assumed that in addresses written in the form `//A_CORPORATE:5345`, the string `A_CORPORATE` is specified in the `/etc/hosts` file or in DNS.

The four groups in the First State Bank network include:

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

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

Figure 11-2 Example Network Groups



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

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

Machine C has addresses for the following:

- `DEFAULTNET` (the corporate WAN)

- MAGENTA_GROUP (LAN)

Machines D and E have addresses for the following:

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

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

Configuration File for the Sample Network

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

*NETGROUPS

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

*NETWORK

A	NETGROUP=DEFAULTNET	NADDR="//A_CORPORATE:5723"
A	NETGROUP=MAGENTA_GROUP	NADDR="//A_MAGENTA:5724"
A	NETGROUP=GREEN_GROUP	NADDR="//A_GREEN:5725"
B	NETGROUP=DEFAULTNET	NADDR="//B_CORPORATE:5723"
B	NETGROUP=MAGENTA_GROUP	NADDR="//B_MAGENTA:5724"
B	NETGROUP=GREEN_GROUP	NADDR="//B_GREEN:5725"
C	NETGROUP=DEFAULTNET	NADDR="//C_CORPORATE:5723"
C	NETGROUP=MAGENTA_GROUP	NADDR="//C_MAGENTA:5724"
D	NETGROUP=DEFAULTNET	NADDR="//D_CORPORATE:5723"
D	NETGROUP=BLUE_GROUP	NADDR="//D_BLUE:5726"
E	NETGROUP=DEFAULTNET	NADDR="//E_CORPORATE:5723"
E	NETGROUP=BLUE_GROUP	NADDR="//E_BLUE:5726"

Assigning Priorities for Each Network Group

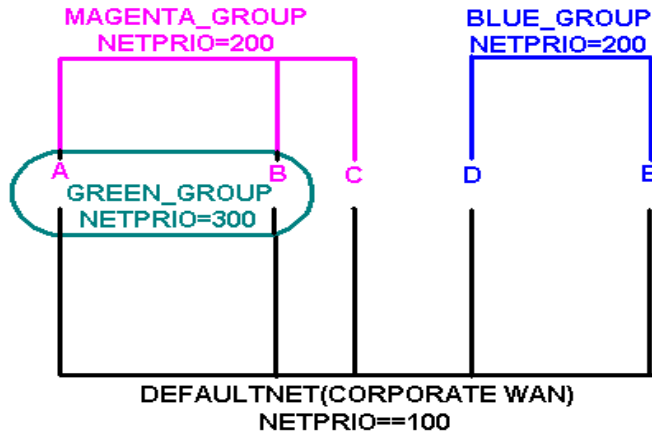
Assigning priorities appropriately for each `NETGROUP` enables you to maximize the capability of network `BRIDGE` processes. When determining `NETGROUP` priorities, keep in mind the following considerations:

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

Example Assignment of Priorities to Network Groups

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

Figure 11-3 Assigning Priorities to Network Groups



The following priorities are assigned:

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

Example NETGROUP and NETWORK Sections

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

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

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

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

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

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

*NETWORK

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


About Workstation Clients

This topic includes the following sections:

- [What Is the Workstation Component?](#)
- [Sample Application with Four Workstation Clients](#)
- [How the Workstation Client Connects to an Application](#)

What Is the Workstation Component?

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

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

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

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

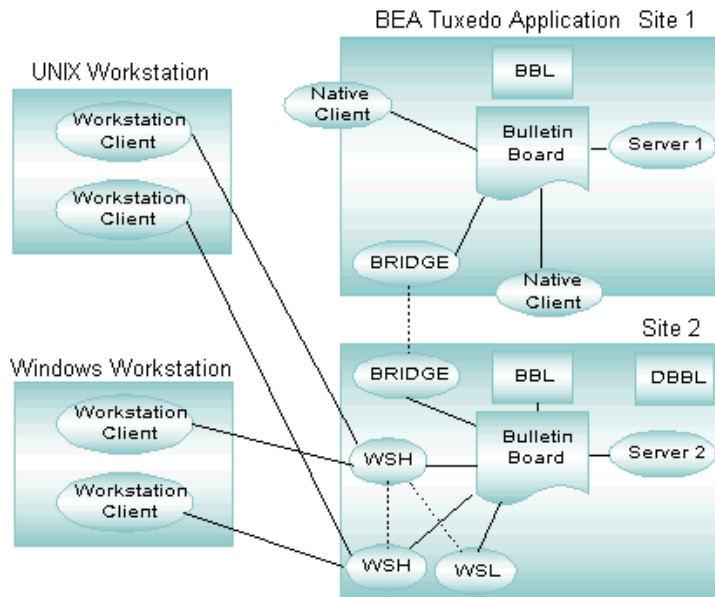
- Send and receive messages

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

Sample Application with Four Workstation Clients

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

Figure 12-1 Bank Application with Four Workstation Clients



Two workstation clients are running on a UNIX system; another two Workstation clients, on Windows. All workstation clients initially joined the application through the Workstation listener (WSL), which delegates subsequent communication to a Workstation handler. This process differs from the process that occurs when native clients join an application: in the latter case, the native clients attach directly to the bulletin board upon joining.

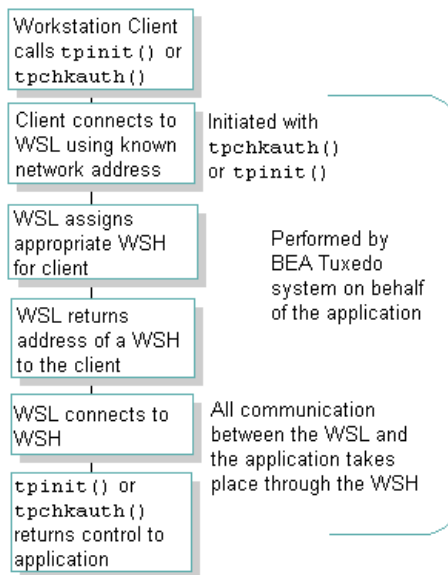
Administrative servers and application servers are located on SITE1 and SITE2. Any service request by a Workstation client to the application is sent over the network to the WSH. This process forwards the request to the appropriate server, gets a reply from the server, and sends the reply to the Workstation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

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

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

How the Workstation Client Connects to an Application

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



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

Setting Up Workstation Clients

This topic includes the following sections:

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

Defining Workstation Clients

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

To Specify . . .	Set This Environment Variable . . .
The application password. (Useful only for applications in which security is implemented through password usage.) Clients that run from scripts can get the application password from this variable.	APP_PW (Optional)
The maximum number of significant bits of the encryption key for link-level encryption. Value can be 0 (if no encryption is used), or 40, 56, or 128 (if the number specified is the number of significant bits in the encryption key).	TMMAXENCRYPTBITS (Optional)
The minimum number of significant bits of the encryption key for link-level encryption. Value can be 0 (if no encryption is used), or 40, 56, or 128 (if the number specified is the number of significant bits in the encryption key).	TMMINENCRYPTBITS (Optional)
The directory in which replies are stored when the WSRPLYMAX limit has been reached. The default is the working directory.	TMPDIR (Optional)
Specifies the code-set encoding name that the workstation machine includes in an allocated MBSTRING typed buffer. TPMBENC has no default value. For a Workstation client using MBSTRING typed buffers, TPMBENC must be defined on the workstation machine.	TPMBENC (Optional)
Specifies whether the workstation machine automatically converts the data in a received MBSTRING buffer to the encoding defined in TPMBENC. By default, the automatic conversion is turned off, meaning that the data in the received MBSTRING buffer is delivered to the Workstation client <i>as is</i> —no encoding conversion. Setting TPMBACNV to any value, say Y (yes), turns on the automatic conversion.	TPMBACNV (Optional)
The location of the BEA Tuxedo system software on this workstation. <i>The client cannot connect unless this environment variable is set.</i>	TUXDIR (Required)
Specifies whether the workstation machine caches Document Type Definition (DTD), XML schema, and entity files. By default, the caching is turned on (Y). Setting URLENTITYCACHING to N (no) turns off the caching.	URLENTITYCACHING (Optional)

To Specify . . .	Set This Environment Variable . . .
Specifies the directory in which the workstation machine caches DTD, schema, and entity files. The <code>URLENTITYCACHEDIR</code> variable specifies the absolute pathname for the cached files. If <code>URLENTITYCACHEDIR</code> is not specified, the default directory becomes <code>URLEntityCachedir</code> , which will be created in the current working directory of the Workstation client process provided that the appropriate write permissions are set.	<code>URLENTITYCACHEDIR</code> (Optional)
The network device to be used. The default is an empty string.	<code>WSDEVICE</code> (Optional)
The name of the file in which all environment variables may be set. There is no default for this variable.	<code>WSENVFILE</code> (Optional)
The network address used by the Workstation client when connecting to the Workstation listener or Workstation handler. This variable, along with the <code>WSFRANGE</code> variable, determines the range of TCP/IP ports to which a Workstation client attempts to bind before making an outbound connection. This address must be a TCP/IP address	<code>WSFADDR</code> (Optional)
The range of TCP/IP ports to which a Workstation client process attempts to bind before making an outbound connection. The <code>WSFADDR</code> parameter specifies the base address of the range.	<code>WSFRANGE</code> (Optional)
A list of one or more network addresses of the WSL that the client wants to contact. This address must match the address of a WSL process in the application configuration file.	<code>WSNADDR</code> (Required)
The amount of core memory to be used for buffering application replies. The default is 256,000 bytes.	<code>WSRPLYMAX</code> (Optional)
The machine type. If the value of <code>WSTYPE</code> matches the value of <code>TYPE</code> in the configuration file for the WSL machine, no encoding/decoding is performed. The default is the empty string.	<code>WSTYPE</code> (optiOnal)

Specifying the Maximum Number of Workstation Clients

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

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

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

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

Defining a Workstation Listener (WSL) as a Server

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

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

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

Passing Information to a WSL Process

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

```
CLOPT="[ -A ] [servopts_options] -- -n netaddr [-d device]
        [-w WSHname] [-t timeout_factor] [-T Client_timeout]
        [-m minh] [-M maxh] [-x mpx_factor ]
        [-p minwshport] [-P maxwshport]
        [-I init_timeout] [-c compression_threshold]
        [-k compression_threshold]
        [-z bits] [-Z bits] [-H external_netaddr]
        [-N network_timeout] [-K{client|handler|both|none}] "
```

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

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

Using Command-line Options Set with CLOPT

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

Note: For a complete list of the `CLOPT` command-line options, see [servopts\(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*.

Use This Command-line Option . . .	To Specify . . .
<code>-n <i>netaddr</i></code> (Required)	The network address used by WSCs to contact the listener. The WSC must set the appropriate environment variable (WSNADDR) to the value specified after <code>-n</code> .
<code>[-d <i>device</i>]</code> (Required for some transport interfaces)	Specify the network device name. This is an optional parameter because only some transport interfaces require it. Sockets, for example, does not require this parameter.
<code>[-t <i>timeout</i>]</code>	The amount of time to allow for a client to connect to the WSH. To calculate the total amount of time to allow for this purpose, the system multiplies the value of <code>timeout</code> by the value of the <code>SCANUNIT</code> parameter. The default is 3 in a nonsecure application, and 6 in a secure application. In this context we refer to an application as secure if one of the following parameters is set: <ul style="list-style-type: none"> • <code>USER_AUTH</code> • <code>ACL</code> • <code>MANDATORY_ACL</code> • <code>APP_PW</code>
<code>[-w <i>name</i>]</code>	The name of the WSH process that should be booted for this listener. The default is <code>WSH</code> , which is the name of the handler provided. If another handler process is built with the <code>buildwsh(1)</code> command, that name is specified here.
<code>[-m <i>number</i>]</code>	The minimum number of handlers that should be booted and always available. The default is 0.
<code>[-M <i>number</i>]</code>	The maximum number of handlers that can be booted. The default is the value of <code>MAXWSClients</code> for the machine being configured, divided by the multiplexing value (specified with <code>-x</code>).
<code>[-x <i>number</i>]</code>	The maximum number of clients that a WSH can multiplex at one time. The value must be greater than 0. The default is 10.

Use This Command-line Option . . .	To Specify . . .
<code>[-T <i>client_timeout</i>]</code>	The amount of time (in minutes) that a client can remain idle without being disconnected. If a client does not make any requests within this time period, the WSH disconnects the client. If this argument is not given or is set to 0, the timeout is infinite.
<code>[-p <i>minwshport</i>]</code> and <code>[-P <i>maxwshport</i>]</code>	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 <i>minwshport</i> and 65535 for <i>maxwshport</i> .
<code>[-z]</code> and <code>[-Z]</code>	The range of bits that can be used, on the WSL side, for link-level encryption: use <code>-z</code> to specify the minimum number of bits, and <code>-Z</code> to specify the maximum number of bits.
<code>[-N <i>network_timeout</i>]</code>	The minimum amount of time (in seconds) that a Workstation client is allowed to wait to receive a response from the WSL/WSH. A value of 0 indicates no network timeout.
<code>[-K {client handler both none}]</code>	The viability of a network connection between the Workstation handler and a Workstation client if no traffic has occurred over that connection within a specified period of time.

See Also

- [servopts\(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*

Detecting Network Failures

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

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

Using the Keep-alive Option

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

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

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

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

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

In the example, `-K` turns on keep-alive checking on both the Workstation client and the server. For details about the format of a `WSL` entry in `UBBCONFIG`, see [WSL \(5\)](#) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*.

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

Limitations When Using the Keep-alive Option

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

- Tru64 UNIX
- HP UX
- Windows

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

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

Using the Network Timeout Option

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

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

How Network Timeout Works

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

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

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

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

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

Limitations When Using the Network Timeout Option

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

Setting the Network Timeout Option

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

Sample Configuration File that Supports Workstation Clients

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

Listing 13-1 Sample UBBCONFIG File Supporting Workstation Clients

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

SITE2
    ...
    MAXWSCLIENTS=0
    ...

*SERVERS
    ...

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

Modifying the MACHINES and SERVERS Sections

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

- In the `MACHINES` section, the default for `MAXWSCLIENTS` is overridden in the entries for two sites. For `SITE1`, the default is raised to 150, while it is lowered to 0 for `SITE2`, because no Workstation clients will be connected to that site.

- In the `SERVERS` section, a WSL process is specified for group `BANKB1`. The WSL has a server ID of 500 and it is marked as restartable.
- The command-line options show the following:
 - The WSL will advertise all of its services (`-A`).
 - The WSL will listen at network address `//ws.beasys.com:5120` (`-n`).
 - A minimum of five WSHs will be booted (`-m`).
 - A maximum of 30 WSHs will be booted (`-M`).
 - Each handler will be allowed a maximum of five clients connected at any one time (`-x`).

Managing Remote BEA Tuxedo CORBA Client Applications

This chapter explains how to configure connections from remote BEA Tuxedo CORBA client applications to CORBA objects via the standard Internet Inter-ORB Protocol (IIOP). This chapter is specific to BEA Tuxedo CORBA servers.

Note: The BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB were deprecated in Tuxedo 8.1 and are no longer supported in Tuxedo 9.x. All BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB text references, associated code samples, etc. should only be used:

- to help implement/run third party Java ORB libraries, and
- for programmer reference only.

Technical support for third party CORBA Java ORBs should be provided by their respective vendors. BEA Tuxedo does not provide any technical support or documentation for third party CORBA Java ORBs.

This topic includes the following sections:

- [CORBA Object Terminology](#)
- [Remote CORBA Client Overview](#)
- [Setting Environment Variables for Remote CORBA Clients](#)
- [Setting the Maximum Number of Remote CORBA Clients](#)
- [Configuring a Listener for a Remote CORBA Client](#)
- [Modifying the Configuration File to Support Remote CORBA Clients](#)

- [Configuring Outbound IIOP for Remote Joint Client/Servers](#)
- [Using the ISL Command to Configure Outbound IIOP Support](#)

CORBA Object Terminology

The following terms are used in this chapter.

DLL

Dynamic Link Libraries. A DLL is a collection of functions grouped into a load module that is dynamically linked with an executable program at run time for a Windows 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 BEA Tuxedo domain. A BEA Tuxedo CORBA server is built with the BEA Tuxedo CORBA `buildobjserver` command. CORBA Servers implement BEA Tuxedo functionality, such as security, transactions, and object state management. Servers can make invocations on any server, inside or outside a BEA Tuxedo domain.

Native Client

A client located within a BEA Tuxedo domain, using the CORBA ORB to make invocations on objects either inside or outside the BEA Tuxedo domain. A native client's host contains the BEA Tuxedo administrative and infrastructure components, such as `tmadmin`, `FactoryFinder`, and `ISL/ISH`. Native clients use the environmental objects to access CORBA objects. You build native C++ clients with the `buildobjclient` command or native Java clients using the tools provided by the third-party ORB.

Remote Client

A client not located within a BEA Tuxedo domain. A remote client can use the CORBA ORB to make invocations on objects either inside or outside the BEA Tuxedo domain. A remote client's host does not contain BEA Tuxedo administrative and infrastructure components, such as `tmadmin`, `FactoryFinder`, and `ISL/ISH`; it does contain supporting software (the CORBA ORB) that allows remote clients to invoke objects. Remote clients use the environmental objects to access CORBA objects. You build remote C++ clients with the `buildobjclient` command or remote Java clients using the tools provided by the third-party ORB.

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 BEA Tuxedo domain. You build native joint C++ client/servers with the `buildobjclient` command. Java native joint client/servers are not supported.

Note: The server role of the native joint client/server is considerably less robust than that of a server. It has none of the BEA Tuxedo CORBA administrative and infrastructure components, such as `tmadmin`, `FactoryFinder`, and `ISL/ISH` (hence none of BEA Tuxedo's scalability and reliability attributes), it does not use the BEA Tuxedo TP Framework, and it requires more direct interaction between the client and the ORB.

Remote Joint Client/server

A process that has two purposes: (1) execute code acting as the starter for some business actions and (2) execute method code for invocations on objects. A joint client/server located outside a BEA Tuxedo domain. The joint client/server does not use the BEA Tuxedo TP Framework and requires more direct interaction between the Client and the ORB. You build remote joint C++ client/servers with the `buildobjclient` command or remote Java client/servers using the tools provided by the third-party ORB.

Note: A joint client/server is different from a server that acts as a client as part of its server role. Once the server completes processing of an invocation, it returns to dormancy. A joint client/server is always in the active mode, executing code not related to a server role; the server role temporarily interrupts the active client role, but the client role is always resumed.

Note: The server role of the remote joint client/server is considerably less robust than that of a server. Neither the client nor the server has any of the BEA Tuxedo administrative and infrastructure components, such as `tmadmin`, `FactoryFinder`, and `ISL/ISH` (hence, none of BEA Tuxedo's scalability and reliability attributes).

BEA Tuxedo CORBA object

A CORBA object that is implemented using TP Framework and that implements security, transactions, and object state management. CORBA objects are implemented in BEA Tuxedo CORBA servers; that is, they are part of a BEA Tuxedo domain and use the BEA Tuxedo infrastructure.

Callback Object

A CORBA object supplied as a parameter in a client's invocation on a target object. The target object can make invocations on the callback object either during the execution of the target object or at some later time (even after the invocation on the target object has been completed). A callback object might be located inside or outside a BEA Tuxedo domain.

Remote CORBA Client Overview

In this section, the term “remote client” represents a CORBA client application that is deployed on systems that do not have the full BEA Tuxedo CORBA 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

A client process can run on 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:

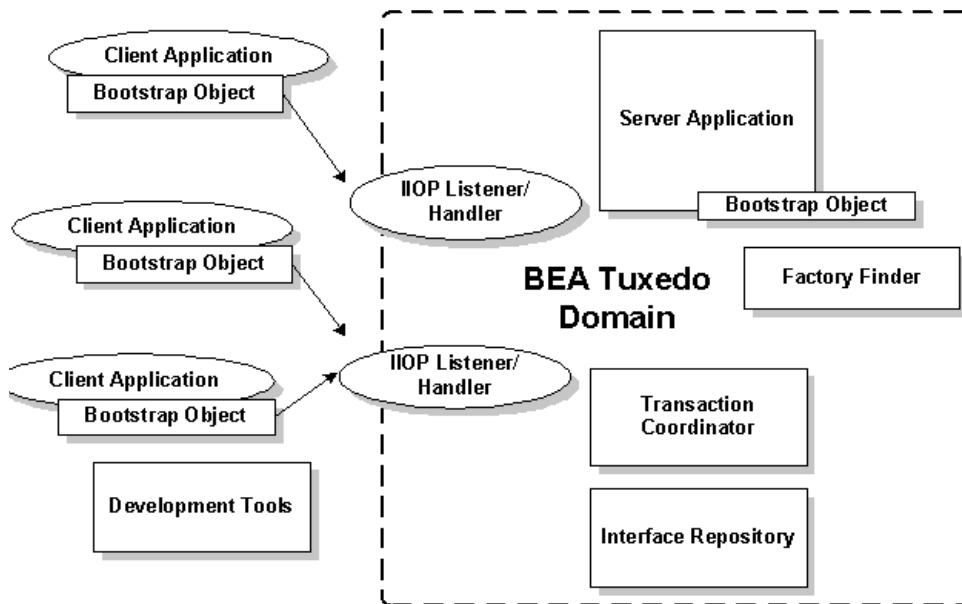
- Invoke methods on remote CORBA objects
- Begin, roll back, or commit transactions
- Be required to pass application security

Note: A client process communicates with the native domain through the ISH.

Illustration of an Application with Remote CORBA Clients

Figure 14-1 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 14-1 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 Remote CORBA Clients

For CORBA C++ clients, environment variables can be used to pass information to the system, as follows:

- **TUXDIR**—the location of the BEA Tuxedo CORBA client software on this remote client. It must be set for the client to connect.
- **TOBJADDR**—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 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 CORBA Clients

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

`MAXWSCLIENTS` tells the BEA Tuxedo 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
<code>MAXWSCLIENTS</code>	<p>Specifies the maximum number of remote clients that may connect to a machine.</p> <p>The default is 0. If a value is not specified, remote clients may not connect to the machine being described.</p> <p>The syntax is <code>MAXWSCLIENTS=<i>number</i></code>.</p>

Configuring a Listener for a Remote CORBA Client

Remote clients access your application through the services of an ISL process and one or more ISH processes. The ISL is specified in one entry as a server supplied by the BEA Tuxedo 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 the ISL command in the *BEA Tuxedo Command Reference*.

Modifying the Configuration File to Support Remote CORBA Clients

[Listing 14-1](#) 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 IOP Listener/Handler will advertise all of its services (`-A`).
 - The IOP 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 14-1 Sample UBBCONFIG File Configuration

```
*MACHINES
SITE1
    ...
    MAXWSCLIENTS=150
    ...
SITE2
    ...
    MAXWSCLIENTS=0
    ...
*SERVERS
    ...
ISL SRVGRP="BANKB1" SRVID=500 RESTART=Y
    CLOPT="-A -- -n //TRIXIE:2500 -d /dev/tcp
        -m 5 -M 30 -x 5"
    ..
```

Configuring Outbound IIOP for Remote Joint Client/Servers

Support for outbound IIOP provides native clients and servers acting as native clients the ability to invoke on a remote object reference outside of the BEA Tuxedo 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.

Administrators have the option of booting the ISLs with the default parameters. However, the default BEA Tuxedo ISL startup parameters do not enable use of outbound IIOP.

Note: Outbound IIOP is not supported for transactions or security.

Functional Description

Outbound IIOP support is required to support client callbacks. In BEA WebLogic Enterprise versions 4.0 and 4.1, the ISL/ISH was an inbound half-gateway. Outbound IIOP support adds the outbound half-gateway to the ISL/ISH. (See [Figure 14-2](#).)

There are three types of outbound IIOP connections available, depending on the version of GIOP supported by the native server and the remote joint client/server application:

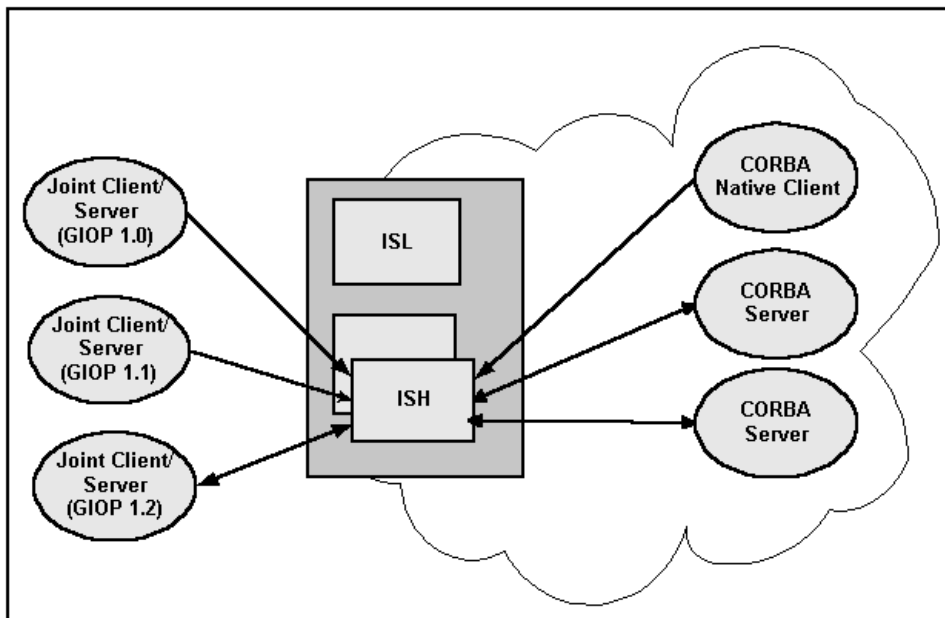
- **Bidirectional**—outbound IIOP reusing the same connection (supported only for BEA WebLogic Enterprise release 4.2 or later C++ GIOP 1.2 servers, clients, and joint client/servers)
- **Asymmetric**—outbound IIOP via a second connection (supported for GIOP 1.0, GIOP 1.1, and GIOP 1.2 servers, clients, 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 BEA WebLogic Enterprise release 4.2 (and later) and BEA Tuxedo release 8.0 (and later) C++ clients, servers, and joint client/servers. BEA WebLogic Enterprise releases 4.0 and 4.1 C++ clients and servers support GIOP versions 1.0 and 1.1, but not GIOP 1.2. Java clients, servers, and joint client/servers only support GIOP 1.0.

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 BEA Tuxedo CORBA 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 14-2 Joint Client/Server IIOP Connections Supported



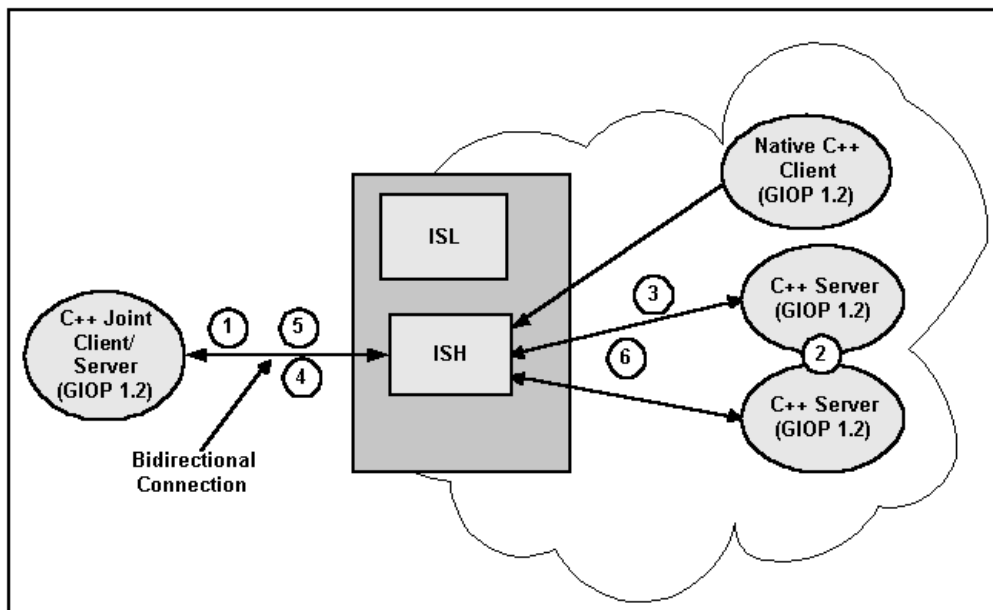
Bidirectional Outbound IIOP

With bidirectional outbound IIOP, the following operations are executed (see [Figure 14-3](#)):

1. A client creates an object reference and invokes on a BEA Tuxedo CORBA server. The client ORB identifies the connection as being bidirectional using the service context. The service context travels with the message to the BEA Tuxedo CORBA server.

2. When unmarshaling the object reference, the BEA Tuxedo CORBA server compares the host/port in the service context with the host/port in the object reference. If they match, the ORB adds the ISH client information needed for routing to the ISH. This client information travels with the object reference whenever it is passed to other BEA Tuxedo CORBA servers.
3. At some point in time, a BEA Tuxedo CORBA server or native client invokes on the object reference, and the routing code invokes on the appropriate ISH, given the client information.
4. The ISH sends the request to the client over the same client connection.
5. The client executes the method and sends the reply back to the ISH via the client connection.
6. The ISH receives the reply and sends it to the BEA Tuxedo CORBA server.

Figure 14-3 Bidirectional Connection

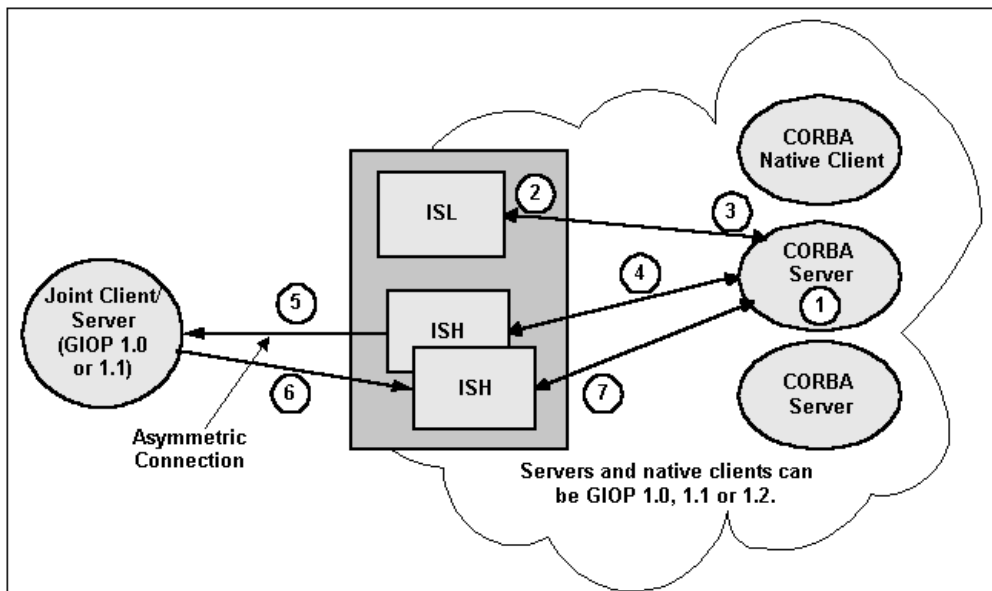


Asymmetric Outbound IIOP

With asymmetric outbound IIOP, the following operations are executed (see [Figure 14-4](#)):

1. A server gets an object reference from some source. It could be a naming service, a `string_to_object`, or it could be passed in through a client, but not located in that client. Since the object reference is not located in a client connected to an ISH, the outgoing call cannot be made using the bidirectional method. The BEA Tuxedo CORBA server invokes on the object reference.
2. On the first invoke, the routing code invokes a service in the ISL and passes in the host/port.
3. The ISL selects an ISH to handle the outbound invoke and returns the ISH information to the BEA Tuxedo CORBA server.
4. The BEA Tuxedo CORBA server invokes on the ISH.
5. The ISH determines which outgoing connection to use to send the request to the client. If none is connected, the ISH creates a connection to the host/port.
6. The client executes the method and sends the reply back to the ISH.
7. The ISH receives the reply and sends it to the BEA Tuxedo CORBA server.

Figure 14-4 Asymmetric Outbound IIOP

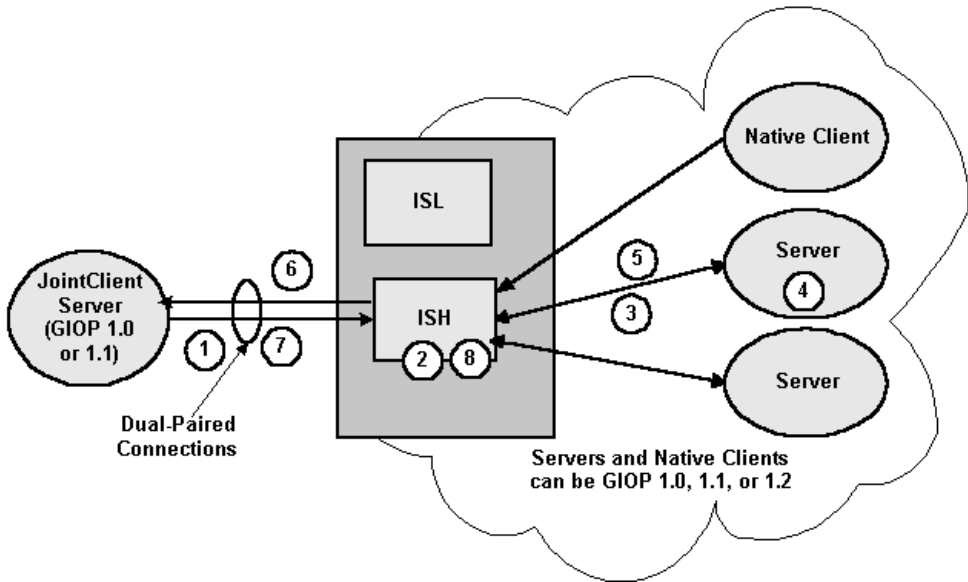


Dual-paired Connection Outbound IIOP

With dual-paired connection outbound IIOP, the following operations are executed (see [Figure 14-5](#)):

1. A client creates an object reference and calls the Bootstrap function (`register_callback_port`) and passes the object reference.
2. The ISH gets the host/port from the IOR and stores it with the client context.
3. The client invokes on a BEA Tuxedo CORBA server and passes the object reference. From the `register_callback_port` call, the ISH creates a service context containing the host/port. The service context travels with the message to the BEA Tuxedo CORBA server.
4. When unmarshaling the object reference, the BEA Tuxedo CORBA server compares the host/port in the service context with the host/port in the object reference. If they match, the ORB adds the ISH client information to the object reference. This client information travels with the object reference whenever it is passed to other BEA Tuxedo CORBA servers.
5. At some point in time, a BEA Tuxedo CORBA server or native client invokes on the object reference. The routing code invokes on the appropriate ISH, passing the client information.
6. The ISH creates a second connection to the client. It sends the request to the client over the second connection.
7. The client executes the method and sends the reply back to the ISH via the first client connection.
8. The ISH receives the reply and sends it to the BEA Tuxedo CORBA server. If the client disconnects from the ISH, the second connection is also disconnected.

Figure 14-5 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-BEA Tuxedo CORBA ORB or from a remote BEA Tuxedo CORBA joint client/server.

Types of Object References

There are two kinds of remote object references:

- Object references created by BEA Tuxedo CORBA remote joint client/servers outside of the BEA Tuxedo 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 BEA Tuxedo 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 the ISL command in the *BEA Tuxedo Command Reference*.

