Oracle® Tuxedo

Setting Up an Oracle Tuxedo Application

12*c* Release 2 (12.2.2)

April 2016

ORACLE

Setting Up an Oracle Tuxedo Application, 12*c* Release 2 (12.2.2)

Contents

4. About Transactions

5. Configuring Your ATMI Application to Use Transactions

6. Using Tuxedo with Oracle Real Application Clusters (RAC)

7. Enabling IPv6

8. Managing The Oracle Tuxedo Service Metadata Repository

9. Managing CORBA Interface Repositories

10. Distributing ATMI Applications Across a Network

11. Creating the Configuration File for a Distributed ATMI Application

14. About Workstation Clients

15. Setting Up Workstation Clients

16. Managing Remote Oracle Tuxedo CORBA Client Applications

17. Applying Service Version to Tuxedo Applications

18. Oracle Tuxedo Applications Packaging and Deployment

19. Configuring Tuxedo for Propagating ECID

20. Logging Last Resource Transaction Optimization

CHAPTER 1

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 Oracle Tuxedo system. Table 1-1 describes the required and optional tasks during the setup phase.

Table 1-1 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 Oracle Tuxedo system and the application (installation)	X	
Set up the Oracle 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
Apply service version to Oracle Tuxedo applications		X
After Tuxedo installation, deploy/undeploy the applications from a centralized control platform using deployment/undeployment tool		X

Run-time Tasks

With your Oracle 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. Table 1-2 describes the required and optional tasks during the run-time phase.

Table 1-2 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 Oracle Tuxedo Administration Console, the command-line interface, and the AdminAPI. Table 1-3 describes some of the circumstances in which your intervention may be needed.

Table 1-3 Circumstances Needed in Intervention

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 Oracle Tuxedo ATMI and CORBA Environments

For the Oracle Tuxedo CORBA environment, the Oracle 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 Oracle 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 Oracle Tuxedo ATMI data-dependent routing parameters continue to work without modification.
- In the Oracle 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 Oracle Tuxedo CORBA environments.

  However, the Oracle Tuxedo CORBA environment uses a different communications protocol to connect remote and foreign clients to Oracle Tuxedo server applications. The protocol is the standard Internet Inter-ORB Protocol (IIOP). Instead of the Oracle 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 Oracle 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 an Oracle Tuxedo CORBA environment system is known as factory-based routing. It is fundamentally different than the Oracle Tuxedo ATMI data-dependent routing mechanism.

  In the Oracle Tuxedo ATMI environment, you can examine any FML field used for a service invocation to determine the data-dependent routing criteria. In Oracle Tuxedo CORBA environments, the system designer must personally communicate the routing criteria of CORBA interfaces. For Oracle 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 Oracle 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?

    _____
    _____

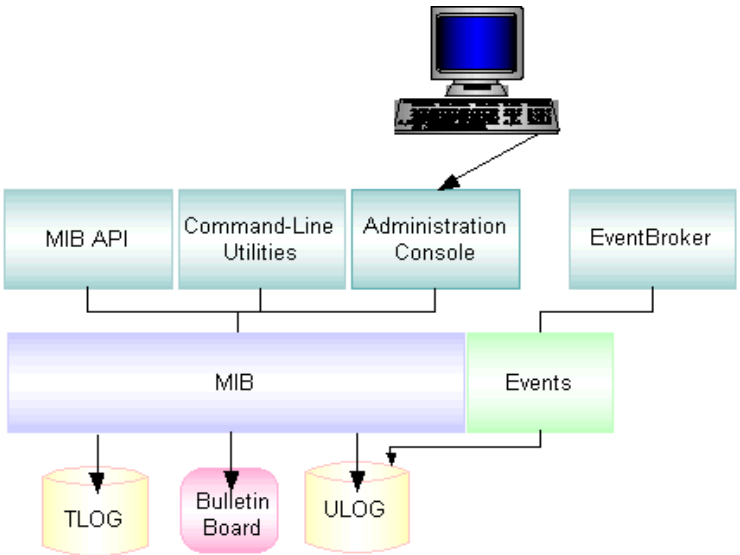
See Also

- "Tools to Help You Administer Your Application" on page 1-8

Tools to Help You Administer Your Application

The Oracle Tuxedo system gives you a choice of several methods for performing the same set of administrative tasks for either Oracle 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 an Oracle Tuxedo application. Figure 1-1 illustrates the tools you can use to write the configuration file and administer your Oracle Tuxedo application during run time.

Figure 1-1 Administration Tools



- Oracle Tuxedo Administration Console—a Web-based tool used to monitor an application, and to dynamically configure its operation.
- Oracle 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 *Oracle Tuxedo Command Reference*

Table 1-4

If You Use This Tool...	You Must...
Oracle 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.
Oracle Tuxedo MIB Application Programming Interface	Write a program that modifies the `TUXCONFIG` file for you.
Command-line interface	1. Create and edit the `UBBCONFIG` file (a text version of `TUXCONFIG`) with a text editor. 2. Run `tmloadcf` to convert the `UBBCONFIG` file into a `TUXCONFIG` (binary) file. (For specific details about the `tmloadcf` command options, see `tmloadcf(1)` in the *Oracle Tuxedo Command Reference*.)

See Also

- "Management Operations Using the Oracle Tuxedo Administration Console" in *Introducing Oracle Tuxedo ATMI*
- "Managing Operations Using the MIB" in *Introducing Oracle Tuxedo ATMI*
- "Managing Operations Using Command-Line Utilities" in *Introducing Oracle Tuxedo ATMI*
- Tasks an Administrator Performs
- "Oracle Tuxedo ATMI Architecture" in *Introducing Oracle Tuxedo ATMI*
- "The Tuxedo CORBA Programming Environment," in *Getting Started with Oracle 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 *Oracle Tuxedo Command Reference*

CHAPTER 2

About the Configuration File

This topic includes the following sections:

What Is the Configuration File?

Configuring each Oracle 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 Oracle 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 Oracle Tuxedo software provides three sample `UBBCONFIG` files—`ubbshm`, `ubbmp`, and `ubbsimple`—as part of the `bankapp` and `simpapp` applications. (See *Tutorials for Developing Oracle 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

Table 2-1 lists the nine sections of the configuration file and describes the purpose of each section.

Table 2-1 Contents of the Configuration File

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 Oracle 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 Oracle 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 Oracle 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 Oracle 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.
- Oracle 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.
- "Oracle Tuxedo Domains (Multiple-Domain) Servers" in *Introducing Oracle Tuxedo ATMI*
- "How to Create the TUXCONFIG File" in *Administering a Oracle Tuxedo Application at Run Time*
- For distributed Oracle Tuxedo CORBA applications, refer to the *Scaling, Distributing, and Tuning CORBA Applications* guide.

CHAPTER 3

Creating the Configuration File

This topic includes the following sections:

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 Oracle Tuxedo System to Take Advantage of Threads

See Also

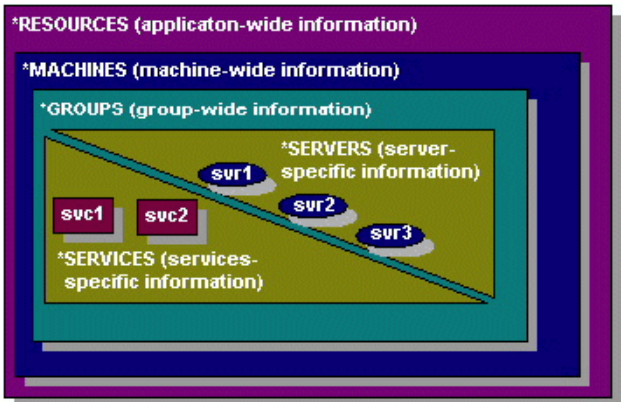
- "About the Configuration File" on page 2-1
- `UBBCONFIG(5)` in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*

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.



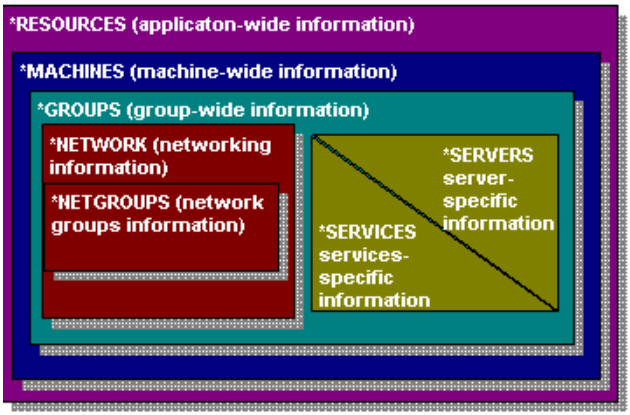
How to Create the Configuration File for a Multiple-machine (Distributed) Application

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 Oracle 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 shows the configuration tasks for a sample multiple-domain application.

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

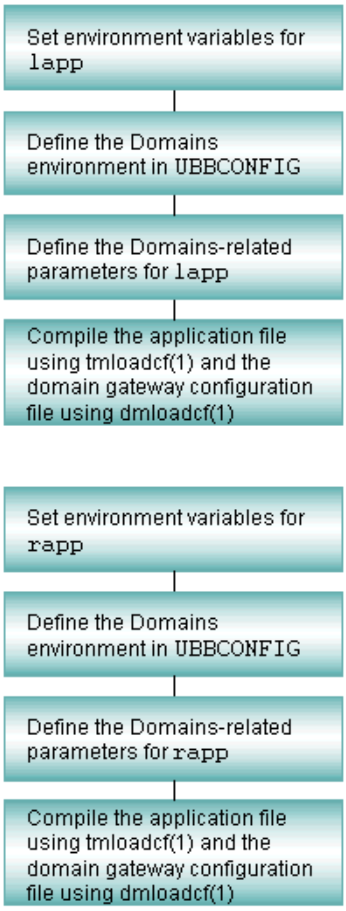
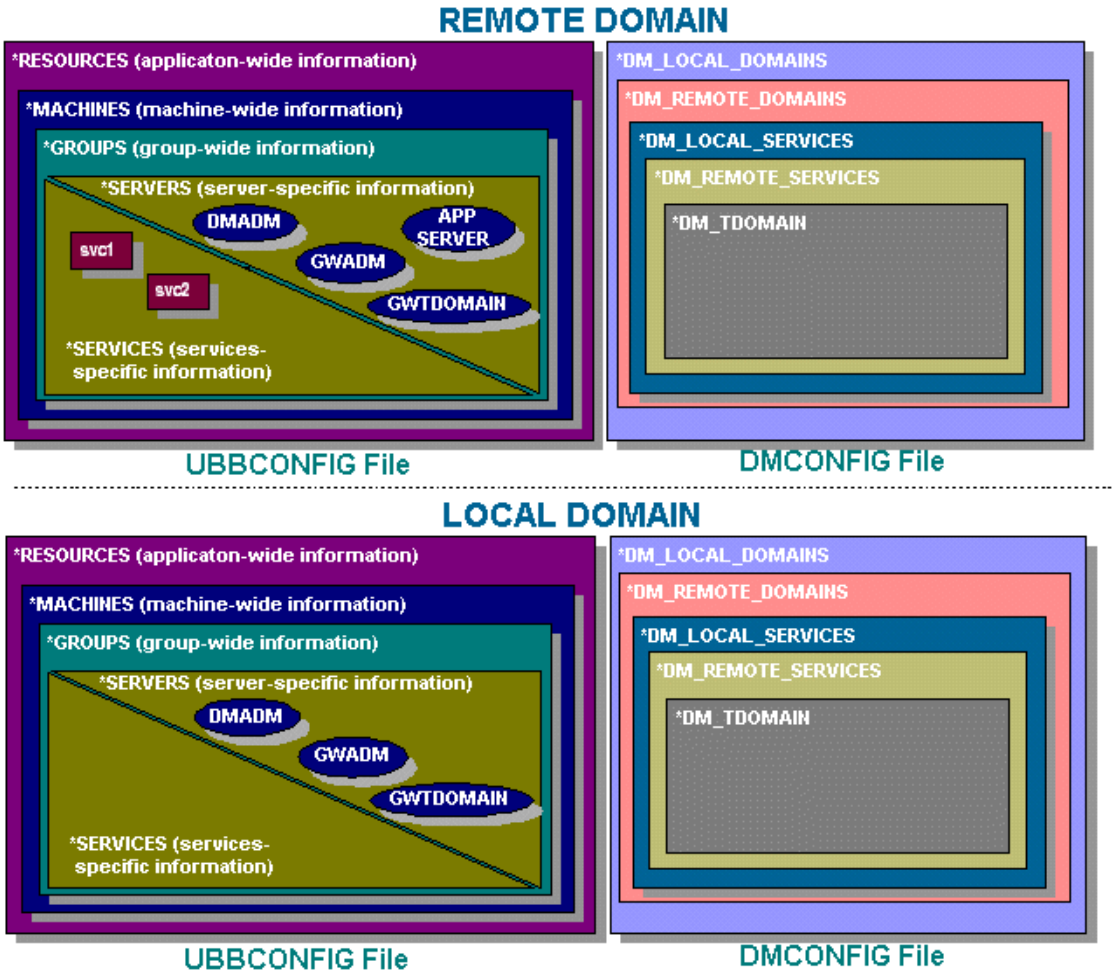


Figure 3-2 shows which sections of the UBBCONFIG and DMCONFIG files you need to configure for a two-domain application. One domain represents the local domain; the other, the remote domain.

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" in *Using the Oracle Tuxedo Domains Component*
- "Planning and Configuring ATMI Domains" in *Using the Oracle 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, Table 3-1 provides a description and links to reference pages and additional information.

Table 3-1 Description and Links to Reference Pages and Additional Information

To Specify This Information in the RESOURCES Section	Set This Parameter (Required/Optional)	For More Information, Click the Following
Unique address of interprocess communication (IPC) resources	IPCKEY (Required)	Shared memory address
Security access	UID, GID, and PERM (Optional)	Security access
Maximum number of processes that can be simultaneously connected to a bulletin board	MAXACCESSERS (Optional)	IPC limits
Maximum number of server table entries in a bulletin board	MAXSERVERS (Optional)	IPC limits
Maximum number of service table entries in a bulletin board	MAXSERVICES (Optional)	IPC limits
Maximum number of CORBA interfaces	MAXINTERFACES (Optional)	IPC limits
Maximum number of CORBA objects	MAXOBJECTS (Optional)	IPC limits
Distinguished Bulletin Board Liaison (DBBL) location at which booting, shutdown, and other administrative tasks are performed	MASTER (Required)	Master processor

Table 3-1 Description and Links to Reference Pages and Additional Information

To Specify This Information in the RESOURCES Section	Set This Parameter (Required/Optional)	For More Information, Click the Following
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 3-25

Defining the Application Type

Among the architectural decisions needed for an Oracle 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

Table 3-2 shows the characteristics of the model and OPTIONS parameters.

Table 3-2 Characteristics of the Model and OPTIONS Parameters

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

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	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. Example: `MAXBUFTYPE 20`
MAXBUFSTYPE	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. Example: `MAXBUFSTYPE 40`

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 *Oracle 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	Maximum number of overall processes that can be simultaneously connected to the bulletin board at any particular site in the Oracle 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 `tmadmin()`, which have reserved access slots to the bulletin board. The value of MAXACCESSERS must be greater than 0 and less than 32,768. If not specified, the default is 50. You can overwrite MAXACCESSERS, on a per-machine basis, in the MACHINES section.
MAXSERVERS	Maximum number of server processes available to the application. This number includes all system-supplied and application servers. The value of MAXSERVERS must be greater than 0 and less than 8,192. If not specified, the default is 50.
MAXSERVICES	Maximum number of different Oracle Tuxedo services that can be advertised in the application. The value of MAXSERVICES must be greater than 0 and less than 1,048,574. If not specified, the default is 100. **Note:** For CORBA environments, each CORBA interface is mapped to an Oracle Tuxedo service. Make sure you account for the number of services generated.

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

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 Oracle 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 Oracle 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 Oracle 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 Oracle Tuxedo service is offered by only one server.
- Do not use load balancing if every Oracle 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 Oracle 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 SCANUNITs 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 Oracle 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	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. The default is the user ID of the person who runs tmloadcf(1). Example: UID=3002 **Note:** On Windows, this value must be set to 0.

Parameter	Characteristics
GID	The numeric group ID of the administrator. The default is the group ID of the person who runs `tmloadcf(1)`. Example: `GID=100` **Note:** On Windows, this value must be set to 0.
PERM	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 Oracle Tuxedo system IPC structures against unauthorized access. These values should be specified for production applications. The default is 0600, which gives read/write access to all. Example: `PERM=0660`

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 Oracle 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.

Notes: LAUTHSVR must be set in the SERVERS section of the UBBCONFIG file to enable LDAP single security administration.

XAUTHSVR must be set in the SERVERS section of the UBBCONFIG file to enable the extensible security administration of authentication and authorization.

Characteristics of the SECURITY, AUTHSVC, and OPTIONS Parameters

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

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*
- *Oracle 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—Oracle Tuxedo libraries compiled with application code do not attach to shared memory while executing system code.
- FASTPATH—Oracle 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_OVERRIDDE	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 Oracle 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 Oracle 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	Value of IGNORE means clients should ignore unsolicited messages. Value of DIPIN means clients should receive unsolicited messages only when they call tpchkunsol() or when they make an ATMI call. Value of SIGNAL means clients should receive unsolicited messages by signals. Default is DIPIN Example: NOTIFY SIGNAL
USIGNAL	Value of SIGUSR1 and SIGUSR2 means notify clients with this type of signal. Default is SIGUSR2 Example: USIGNAL SIGUSR1

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 Oracle 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, Table 3-3 provides a description and links to reference pages and additional information.

Table 3-3 How to Create the MACHINES Section of the Configuration File

To Specify This Information in the MACHINES Section	Set This Parameter (Required/Optional)	For More Information, Click the Following
The number of entries in the cache used for ACL entries when SECURITY is set to ACL or MANDATORY_ACL.	MAXACLCACHE (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.	NETLOAD (Optional)	Additional loads
The address is the name of the physical processor, which all other entries describe. The LMID parameter specifies the logical name of the computer.	LMID (Required)	Address and machine ID
The number of attempts that should be made at user level to lock the bulletin board before blocking processes on a UNIX semaphore.	SPINCOUNT (Optional)	Bulletin board locking limit
A value used for grouping machines into classes.	TYPE (Optional)	Class grouping value

Table 3-3 How to Create the MACHINES Section of the Configuration File

To Specify This Information in the MACHINES Section	Set This Parameter (Required/Optional)	For More Information, Click the Following
The absolute pathname of the file or device where the binary TUXCONFIG file is found on this machine. **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.	TUXCONFIG (Required)	Configuration file location
The maximum number of simultaneous conversations in which processes on a particular machine can be involved.	MAXCONV (Optional)	Conversation limits
The numeric size, in pages, of the DTP transaction log for this machine.	TLOGSIZE (Optional)	DTP TLOG size
The name of the DTP transaction log for this machine.	TLOGNAME (Optional)	DTP transaction log name
A value that specifies that all clients and servers on the machine are to be executed with the environment specified in the named file.	ENVFILE (Optional)	Environment variable settings
The Oracle 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

Table 3-3 How to Create the MACHINES Section of the Configuration File

To Specify This Information in the MACHINES Section	Set This Parameter (Required/Optional)	For More Information, Click the Following
The number of accesser entries on this processor to be reserved for Workstation clients. The parameter is only used when the Oracle 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 Oracle 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 Oracle Tuxedo filesystem that contains the TUXCONFIG file for this machine.	TUXOFFSET (Optional)	Numeric offset containing the TUXCONFIG
The numeric group ID to be associated with the IPC structures created for the bulletin board. The valid range is 0-2147483647. If not specified, the default is the value specified in the RESOURCES section.	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

Table 3-3 How to Create the MACHINES Section of the Configuration File

To Specify This Information in the MACHINES Section	Set This Parameter (Required/Optional)	For More Information, Click the Following
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_PASSVAR	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 Oracle 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 (*string_value1*) and local processes (*string_value2*), 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 *uname -n* 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 *gumby*.
TUXDIR	The full path to the installed Oracle Tuxedo software (shown in double quotation marks).
APPDIR	The full path to the application directory (shown in double quotation marks).
TUXCONFIG	The full pathname of the configuration file (shown in double quotation marks). **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.
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).

Parameter	Meaning
MAXACCESSERS	For this machine, override the system-wide value (defined in the RESOURCES section) with 100.
MAXOBJECTS	(For the CORBA example.) For this machine, override the system-wide value (defined in the RESOURCES section) with 700.
MAXCONV	For this machine, override the system-wide value (defined in the RESOURCES section) with 15.

How to Customize the Sample MACHINES Section

You can customize the MACHINES section by indicating the following:

- Your machine name for *gumby*

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

- The full path of your Oracle 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 3-39

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 Oracle 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 Oracle 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 Oracle 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
- TMCMPLIMIT
- 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 Oracle 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 Oracle Tuxedo Filesystem Containing the TLOG

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

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

Specifying a Machine's Maximum Number of Simultaneous Global Transactions

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

Defining the Number of Accesser Entries on a Workstation Client

Use the MAXWSCLIENTS parameter to define the number of entries on a machine to be reserved for Workstation clients. Set the number of accesser slots reserved for MAXWSCLIENTS cautiously, since this number takes a portion of the total accesser slots specified with MAXACCESSERS for this machine; the accesser slots reserved for MAXWSCLIENTS are unavailable for use by other clients and servers on this machine. By setting this parameter to an appropriate value, you can help conserve IPC resources because Workstation client access to the system is multiplexed through an Oracle Tuxedo system-supplied surrogate, the Oracle 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 Oracle 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 Oracle Tuxedo system tables. In an Oracle 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 Oracle 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 Oracle 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 Oracle Tuxedo system tables. In an Oracle 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 Oracle 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 Oracle 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 Oracle Tuxedo software looks in TUXDIR/bin and APPDIR for executables.

Characteristics of the APPDIR and TUXDIR Parameters

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

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 3-39

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, Table 3-4 provides a description and links to reference pages and additional information.

Table 3-4 How to Create the GROUPS Section of the Configuration File

To Specify This Information in the GROUPS Section	Set This Parameter (Required/Optional)	For More Information, Click the Following
The logical name of the group.	*GROUPNAME* (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 `GROUPS` section.	`GRPNO` (Required)	Group number
The resource manager dependent information needed when closing the resource manager.	`CLOSEINFO` (Optional)	Information for closing the resource manager
The resource manager dependent information needed when opening the resource manager.	`OPENINFO` (Optional)	Information for opening the resource manager
The number of transaction manager servers to start for the associated group, if `TMSNAME` is specified.	`TMSCOUNT` (Optional)	Number of TMS servers in the group
Principal name of the process used for identification, location of private key of principal user, and the environment variable containing the password.	`SEC_PRINCIPAL_NAME`, `SEC_PRINCIPAL_LOCATION`, `SEC_PRINCIPAL_PASSVAR`	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

Table 3-4 How to Create the GROUPS Section of the Configuration File

To Specify This Information in the GROUPS Section	Set This Parameter (Required/Optional)	For More Information, Click the Following
A value that specifies that this group of servers resides on the machine symbolically named by *string_value1* 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/bankdl1:bankdb:readwrite"
BANKB2GRPNO=2 OPENINFO="TUXEDO/SQL:APPDIR1/bankdl2:bankdb:readwrite"
BANKB3GRPNO=3 OPENINFO="TUXEDO/SQL:APPDIR1/bankdl3:bankdb:readwrite"
```

Sample GROUPS Section for CORBA

The followiing 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 mst 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 3-89 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 Oracle 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 3-54

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
Group_name required_parameters [*optional_parameters*]	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=*string_value1* [,*string_value2*]	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 3-46

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 Oracle 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 Oracle 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 Oracle Tuxedo /Q XA interface, *qmconfig* is replaced with the name of the QMCONFIG (see qmadmin(1) in the *Oracle 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 Oracle 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/myubbconfig
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, Table 3-5 provides a description and links to reference pages and additional information.

Table 3-5 How to Create the NETWORK Section of the Configuration File

To Specify This Information in the NETWORK Section	Set This Parameter (Required/Optional)	For More Information, Click the Following
The device name to be used by the BRIDGE process placed on that LMID to access the network.	BRIDGE (Optional)	BRIDGE device name
The complete network address to be used by the BRIDGE process; that is, the listening address on the LMID.	NADDR (Required)	BRIDGE network address
The minimum level of encryption required when a network link to this machine is being established.	MINENCRYPTBITS (Optional)	Encryption levels
The maximum level of encryption allowed when a network link is being established.	MAXENCRYPTBITS (Optional)	Encryption levels

Table 3-5 How to Create the NETWORK Section of the Configuration File

To Specify This Information in the NETWORK Section	Set This Parameter (Required/Optional)	For More Information, Click the Following
The network group associated with this network entry. If unspecified, then the default, DEFAULTNET, is assumed. (If not set to DEFAULTNET, this parameter must be defined as a group name in the NETGROUPS section.)	NETGROUP (Optional)	Network group
The network address used by the tlisten(1) process servicing the network on the node identified by the LMID.	NLSADDR (Optional)	tlisten network address

Sample NETWORK Section

The following configuration file excerpt shows a NETWORK section for a 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 3-50

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 Oracle Tuxedo system prior to release 6.4 are assigned to the DEFAULTNET network group.

For each parameter in the NETGROUPS section, Table 3-6 provides a description and links to reference pages and additional information.

Table 3-6 How to Create the NETGROUPS Section of the Configuration File

To Specify This Information in the NETGROUPS Section (Optional)	Set This Parameter (Required/Optional)	For More Information, Click the Following
Allow more netgroups to be defined than the default (8). This value is specified in the RESOURCES section.	MAXNETGROUPS (Optional)	Maximum netgroups
The maximum size of data waiting for the network to become available. This value is specified in the MACHINES section.	MAXPENDINGBYTES (Optional)	Message space limits
The network group associated with this network entry.	NETGROUP (Required)	Network group name
A unique network group number that you must assign to use in failover and failback situations.	NETGRPNO (Required)	Network group number
The priority of this network group.	NETPRIO (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. Figure 3-3 shows machines A through E in the networks for which they have addresses.

Figure 3-3 Example of a Network Grouping

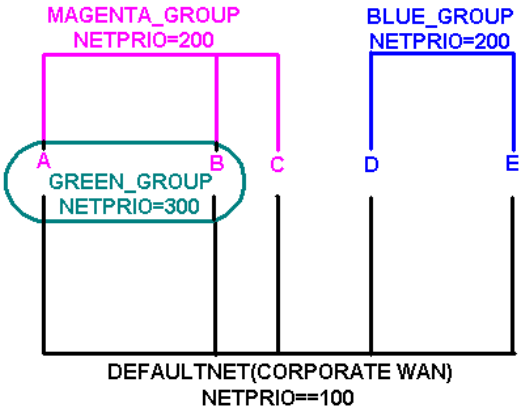


Table 3-7 shows which machines have addresses for which groups.

Table 3-7 Machines and Addresses for 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 Listing 3-1.

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 3-54
- "Setting Up the Network for a Distributed Application" on page 12-1

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 Oracle 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, Table 3-8 provides a description and links to reference pages and additional information.

Table 3-8 How to Create the SERVERS Section of the Configuration File

To Specify This Information in the SERVERS Section (Optional)	Set This Parameter (Required/Optional)	For More Information, Click the Following
Whether the server is a conversational server. Connections can be made only to conversational servers, and rpc requests (via tpacall(3c) or tpcall(3c)) can be made only to non-conversational servers.	CONV (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	SEC_PRINCIPAL_NAME, SEC_PRINCIPAL_LOCATION, SEC_PRINCIPAL_PASSVAR	Security attributes
When this server should be booted or shut down relative to other servers.	SEQUENCE (Optional boot parameter)	Server boot order
The minimum number of occurrences of the server to be booted by tmboot.	MIN (Optional boot parameter)	Server boot order
The maximum number of occurrences of the server that can be booted.	MAX (Optional boot parameter)	Server boot order
A list of servopts(5) options to be passed to a server process at boot time. If none are specified, the default is -A. *string_value* may contain up to 256 characters.	CLOPT (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 ENVFILE must be in the same location on both machines.	ENVFILE (Optional run-time parameter)	Server environment file
The name of the group in which the server is to run. *string_value* must be the logical name associated with a server group in the GROUPS section.	SRVGRP (Required)	Server group

Table 3-8 How to Create the SERVERS Section of the Configuration File

To Specify This Information in the SERVERS Section (Optional)	Set This Parameter (Required/Optional)	For More Information, Click the Following
An integer that uniquely identifies a server within a group. Identifiers must be between 1 and 30,000 inclusive.	`SRVID` (required)	Server ID
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 Oracle Tuxedo system libraries within application processes to gain access to Oracle Tuxedo system internal tables.	`SYSTEM_ACCESS` (Optional run-time parameter)	System access to servers

Table 3-8 How to Create the SERVERS Section of the Configuration File

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

Table 3-8 How to Create the SERVERS Section of the Configuration File

To Specify This Information in the SERVERS Section (Optional)	Set This Parameter (Required/Optional)	For More Information, Click the Following
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
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. SECURITY USER_AUTH or higher must be specified. Default uses the file $TUXDIR/udataobj/tpldap to get LDAP configuration information. Example: LAUTHSVR SRVGRP= "AUTH''SRVID=100 CLOPT="-A-- -f/usr/tuxedo/udataobj/tpldap"	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
RESTART=Y (default)	Restart the servers.
MAXGEN=5 (default)	The MAXGEN 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 GRACE parameter. The default is 1. If the server is to be restartable, MAXGEN must be >= 2. The number of restarts is at most *number* - 1 times. RESTART must be Y or MAXGEN is ignored.
GRACE=3600 (default)	If RESTART is Y, the GRACE parameter specifies the time period (in seconds) during which this server can be restarted as MAXGEN - 1 times. The number assigned must be equal to or greater than 0. The maximum is 2,147,483,648 seconds (or a little more than 68 years). If GRACE is not specified, the default is 86,400 seconds (24 hours). As soon as one GRACE period is over, the next grace period begins. Setting the grace period to 0 removes all limitations; the server can be restarted an unlimited number of times.
REPLYQ=N (default)	There is no reply queue.
CLOPT="-A" (default)	Specify -A on the command line of each server.
ENVFILE="/usr/home/envfile" (default)	Read environment settings from the file ENVFILE.
SYSTEM_ACCESS=PROTECTED (default)	Deny access to internal tables outside system code.

Parameter	Meaning
RINGUP1	Sample name of the first server to be booted.
SRVGRP=GROUP1 SRVID=1 MIN=3 RQADDR="ring1"	Three instances of the sample server will be booted in group GROUP1 with server IDs of 1, 2, and 3, respectively. The three servers will form an MSSQ set and will read requests from queue *ring1*. **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 3-73

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 3-61 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 an Oracle 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 Oracle Tuxedo CORBA University Basic application, which is one of the sample applications included with the Oracle 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 Oracle 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 Oracle Tuxedo CORBA provided server that
    # runs the NameManager and FactoryFinder services.

    # The NameManager service is a Oracle 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 9, "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 16, "Managing Remote Oracle 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
SEQUENCE	It is an optional parameter with a numeric range of 1 - 10,000. Smaller values are booted before larger values. Servers for which this parameter is not set are booted in the order in which they are listed in the SERVERS section. All sequenced servers are booted before any unsequenced servers.
MIN	It represents the minimum number of servers to boot during run time. If RQADDR is specified and MIN>1, an MSSQ set is created. All instances have the same server options. The range of values is 0 to 1000. The default is 1.
MAX	It represents the maximum number of servers to boot. The range of values for MAX is 0 to 1000. If MAX is not specified, the default is the value of MIN.

Specifying Server Command-line Options

The Oracle 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). Table 3-9 provides a partial list of the available options.

Table 3-9 Specifying Server Command-line Options

Use This Option	To
`-o filename`	Redirect standard output to file *`filename`*.
`-e filename`	Redirect standard error to file *`filename`*.
`-s services`	Advertise services. For example, `-s x,y,z` to advertise services `x`, `y`, and `z`.
`-s x,y,z:funcname`	Advertise services `x`, `y`, and `z`, but process requests for those services with function *`funcname`*. This is called *aliasing* a function name.
`-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
Server_name	It identifies the executable to be booted. It is built with `buildserver(1)` for ATMI. It is built with `buildobjserver(1)` 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 an Oracle 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 Min parameter.
RQPERM	Represents the permissions on a request queue. If no parameter is specified, the permissions of the bulletin board, as specified by PERM in the RESOURCES section, 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.

Parameter	Characteristics
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 an Oracle Tuxedo system, if the server is a member of an MSSQ set and contains services programmed to receive reply messages, REPLYQ should be set to Y so that an individual reply queue is created for this server. If not, the reply is sent to the request queue shared by all servers of the MSSQ set, and there is no way of assuring that it will be picked up by the server that is waiting for it. Multithreaded servers automatically create REPLYQs even if this parameter is not set.
RPPERM	Assigns permissions to the reply queue. This parameter is useful only when REPLYQ=Y. If requests and replies are read from the same queue, only RQPERM is needed; RPPERM is ignored.

Defining Server Restart Information

A properly debugged server should not terminate on its own. By default, servers that do terminate while the application is running are not restarted by the Oracle 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 Y enables a server to restart. The default is N.
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 MAXGEN. Zero represents unlimited restart. It must be between 0 and 2147,483,647 (2^{31} - 1). The default is 24 hours.

Defining Server Access to Shared Memory

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

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

Characteristics of the SYSTEM_ACCESS Parameter

The SYSTEM_ACCESS parameter has the following characteristics:

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

- A value of FASTPATH indicates that the server will attach to shared memory at all times.
- If NO_OVERRIDE is specified in the RESOURCES section, this parameter is ignored.
- The default is the value of the SYSTEM_ACCESS parameter in the RESOURCES section.
- The Oracle 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 Oracle 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 Oracle Tuxedo Security Pack.

The following table lists the 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, Table 3-10 provides a description and links to reference pages and additional information.

Table 3-10 How to Create the SERVICES Section of the Configuration File

To Specify This Information in the SERVICES Section	Set This Parameter (Required/Optional)	For More Information, Click the Following
Whether a transaction should be started automatically when a request message is received that is not already in transaction mode.	AUTOTRAN (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.	BUFTYPE (Optional)	Buffer types
A load factor to be imposed on the system by SVCNAM.	LOAD (Optional)	Load balancing
The name of the routing criteria used for this service when data- dependent routing is used.	ROUTING (Optional)	Routing criteria name
The name of the sever group from which SVCNAM gets all group parameter settings.	SRVGRP (Optional)	Server group parameters
The dequeuing priority of SVCNM.	PRIO (Optional)	Service priorities
Set the nontransactional blocking time value, in seconds, of the indicated service.	BLOCKTIME (Optional)	Specifying Nontransactional Service-Level Blocktime
The amount of time, in seconds, that is allowed for processing of the indicated service.	SVCTIMEOUT (Optional)	Service processing time
The default timeout interval, in seconds, for a transaction automatically started for the associated service.	TRANTIME (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 3-86

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 Oracle Tuxedo 8.0 or earlier, except that when a machine running Oracle 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 Oracle 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 Oracle 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 Oracle 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 Oracle Tuxedo system removes only frozen processes.

This section describes the causes and results of service timeout errors, and explains how the Oracle 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 Oracle 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 Oracle 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 Oracle 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 Oracle 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 `tpsprio()`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 Oracle Tuxedo.

The INTERFACES section in the configuration file is used to define parameters for CORBA environments in the Oracle 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)

The following table lists the AUTOTRAN, FACTORYROUTING, LOAD, PRIO, SRVGRP, TRANTIME, and TIMEOUT parameters characteristics.

Parameter	Characteristic
AUTOTRAN = {Y | N }	For each CORBA interface, set AUTOTRAN to Y if you want a transaction to start automatically when an operation invocation is received. AUTOTRAN=Y has no effect if the interface is already in transaction mode. The default is N. The effect of specifying a value for AUTOTRAN is dependent on the transactional policy specified by the system designer in the implementation configuration file (ICF) or Server Description File (XML) for the interface. This transactional policy will become the transactional policy attribute of the associated T_IFQUEUE MIB object at run time. The only time this value actually affects the behavior of the application is if the system designer specified a transaction policy of *optional*. **Note:** To work properly, this feature may be dependent on personal communication between the system designer and the system administrator. If the system administrator sets this value to Y without prior knowledge of the ICF or XML parameters set by the programmer, the actual run-time effort of the parameter might be unknown.
FACTORYROUTING = criterion-name	Specify the name of the routing criteria to be used for factory-based routing for this CORBA interface. You must specify a FACTORYROUTING parameter for interfaces requesting factory-based routing.
LOAD = number	This is an arbitrary number between 1 and 100 that represents the relative load that the CORBA interface is expected to impose on the system. The numbering scheme is relative to the LOAD numbers assigned to other CORBA interfaces used by this application. The default is 50. The number is used by the Oracle Tuxedo system to select the best server to route the request.

Parameter	Characteristic
PRIO = number	Specify the dequeuing priority number for all methods of the CORBA interface. The value must be greater than 0 and less than or equal to 100. 100 is the highest priority. The default is 50.
SRVGRP = server-group-name	Use SRVGRP to indicate that any parameter defined in this portion of the INTERFACES section applies to the interface within the specified server group. For a given CORBA interface, this feature lets you define different parameter values in different server groups.
TRANTIME = number	If AUTOTRAN is set to Y, you must set the TRANTIME parameter, which is the transaction timeout in seconds, for the transactions to be computed. The value must be greater than or equal to zero and must not exceed 2,147,483,647 (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 Oracle 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 3-89.

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 Oracle Tuxedo CORBA envirionments, 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 Oracle 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 Oracle Tuxedo, which introduces load balancing across mulitple servers in a local domain. For more information about parallel objects in Oracle 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 Oracle Tuxedo client or server application by assigning interface priorities using the PRIO parameter. For instance, Server 1 offers Interfaces A, B, and C. Interfaces A and B have a priority of 50 and Interface C has a priority of 70. An interface requested for C will always be dequeued before a request for A or B. Requests for A and B are dequeued equally with respect to one another. The system dequeues every tenth request in FIFO order to prevent a message from waiting indefinitely on the queue.

The PRIO parameter has the following characteristics:

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

Specifying Different Interface Parameters for Different Server Groups

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

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, Table 3-11 provides a description and links to reference pages and additional information.

Table 3-11 How to Create the ROUTING Section of the Configuration File

To Specify This Information in the ROUTING Section (Optional)	Set This Parameter (Required/Optional)	For More Information, Click the Following
Ranges and associated server groups for the routing field.	RANGES (Required)	Range criteria
The value must be a string with a maximum length of 15 characters. For ATMI, the routing criteria name specified as the value of the ROUTING parameter in the SERVICES section for data-dependent routing. For CORBA, the routing criteria name specified in the INTERFACES section as the FACTORYROUTING parameter factory-based routing.	criterion_name (required)	

Table 3-11 How to Create the ROUTING Section of the Configuration File

To Specify This Information in the ROUTING Section (Optional)	Set This Parameter (Required/Optional)	For More Information, Click the Following
Specifies the routing type. For ATMI, the default is `TYPE=SERVICE` to ensure that existing `UBBCONFIG` files used in Tuxedo ATMI environments continue to work properly. For CORBA, use `TYPE=FACTORY` when implementing factory-based routing for a CORBA interface.	`TYPE`	
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 `FLDTBLDIR` and `FIELDTBLS` 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. 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.	`FIELD` (Required)	Routing buffer field and type
A list of types and subtypes of data buffers for which this routing entry is valid. This parameter may contain up to 256 characters with a maximum of 32 type/subtype combinations.	`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	The name of the buffer field on which the routing is performed. It may contain up to 30 characters. In Oracle 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. In routing XML documents, the FIELD syntax contains either a *routing element type* (or name) or a *routing element attribute name*. You must define the FIELD parameter with the following syntax: `root_element[/child_element][/child_element][/. . .][/@attribute_name]` The element is assumed to be an *element type* (or *name*) or an *element attribute name* 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 Oracle Tuxedo system recognizes only the first occurrence of a given element type when processing an XML buffer for data-dependent routing. 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.
FIELDTYPE	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: FIELDTYPE=*type* where *type* is one of the following: string, char, short, long, float, or double. The default type of the routing field is string.

Specifying Range Criteria

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

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

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

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

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

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

Defining Buffer Types

For Oracle 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 Oracle Tuxedo. You can find the ubb_p.nt or ubb_p.mk UBBCONFIG

files for this sample in the directory where the Oracle 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 Oracle 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 Oracle 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 Oracle 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 Oracle Tuxedo System to Take Advantage of Threads

To configure a multicontexted application, edit your UBBCONFIG file as usual and add those parameters, listed in Table 3-12, that are needed for your application. Use a text editor or the Oracle Tuxedo Administration Console.

Table 3-12 Setting Parameters in the Configuration File to Use Threads

In This Section	Set These Parameters	With These Considerations.
RESOURCES	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.
	NOTIFY	Optional parameter that defines the default method to be used for unsolicited notification. Valid values for multicontexted applications are: • DIPIN • THREAD • IGNORE

Table 3-12 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 3-12 Setting Parameters in the Configuration File to Use Threads (Continued)

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

How to Compile a Configuration File

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

tmloadcf reads a file (or standard input written in UBBCONFIG syntax), checks the syntax, and optionally loads a binary configuration file called TUXCONFIG. The TUXCONFIG and (optionally) TUXOFFSET environment variables point to the TUXCONFIG file and (optional) offset where the

information should be stored. You can run `tmloadcf` only on the machine designated as `MASTER` in the `RESOURCES` section of the `UBBCONFIG` file, unless the `-c` or `-n` option is specified.

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

The 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.

CHAPTER 4

About Transactions

This topic includes the following sections:

- What Is a Transaction?
- Benefits of Using Transactions
- Example of a Global Transaction
- What Is the Oracle 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 an Oracle 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 Oracle 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 Oracle Tuxedo system.

Table 4-1 shows the ACID of Oracle Tuxedo Transactions.

Table 4-1 ACID Properties of Oracle 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	• The transaction *aborts*, that is, it terminates abruptly. • The transaction *rolls back*, 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 *commits* 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 Oracle Tuxedo system, including its communication APIs and protocols, is designed to support the use of transactions. The Oracle 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 enqueuing 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 Oracle 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 Oracle 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 Oracle 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 Oracle 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.

Table 4-2 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 *communication partners*.

Table 4-2 Actions Performed by the Transaction Manager

When . . .	The Transaction Manager . . .
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.

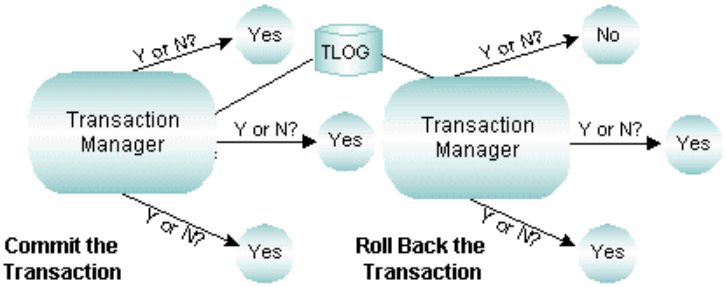
How the System Tracks Distributed Transaction Processing

Oracle 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 Oracle Tuxedo system stores tracking information about all the participants in a transaction in a dedicated file called a *transaction log*, or `TLOG`.

Figure 4-1 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 Oracle 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 Oracle Tuxedo system assigns a *global transaction identifier,* or `GTRID` to each transaction.

The Oracle 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.

Writing TLOG to an Oracle Database

If you want to write TLOG into an Oracle database, you must do the following steps:

1. Install Oracle database 10g client (or later), create link `libclntsh.so` for `libclntsh.so.x.x` (for example, `libclntsh.so.10.1`) and set `LD_LIBRARY_PATH` for link `libclntsh.so` on Linux platform.

2. Set `UBBCONFIG(5) TLOGDEVICE` or `DMCONFIG(5) DMTLOGDEV` using the following format: "`DB:Oracle_XA: ….`". For example:

   ```
   TLOGDEVICE="DB:Oracle_XA:ORACLE_XA+SqlNet=ORCL+ACC=P/scott/tiger"
   ```

   ```
   DMTLOGDEV="DB:Oracle_XA:ORACLE_XA+SqlNet=ORCL+ACC=P/scott/tiger"
   ```

3. Run `tmloadcf` to generate `TUXCONFIG`.

4. Create `TLOG` using the `tmadmin` and `dmadmin` commands. Below is an example to create `TLOG` using `tmadmin`. After `TLOG` command `crlog` is done, a table is created; the value that `TLOGNAME` defines in `UBBCONFIG` becomes the table name.

   ```
   $ tmadmin
   ```

   ```
   $ crlog -m <Machine>
   ```

Note the followings.

- You can only write `TLOG` to an Oracle database. Third party databases are not supported.
- There is no need for you to create `TLOG` using `tmadmin` command `crdl`.
- `TLOGDEVICE/DMTLOGDEV` points to Oracle database schema, which Tuxedo treats it as a database storage device.
- Below are the rules for `TLOGNAME` in `UBBCONFIG`.

- TLOGNAME in UBBCONFIG must not be empty. If multiple TLOG files are stored in the same schema of database, DBA should guarantee that TLOGNAME is unique for each TLOG, and Tuxedo exclusively accesses the database table that TLOGNAME specifies for the TLOG.
- Do not start TLOGNAME with "QS" or "TUX" because they are reserved by Tuxedo. DBA should guarantee that.
- Uppercase TLOGNAME when using TLOG2DB.
- Different platforms should share different database schemas.

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.

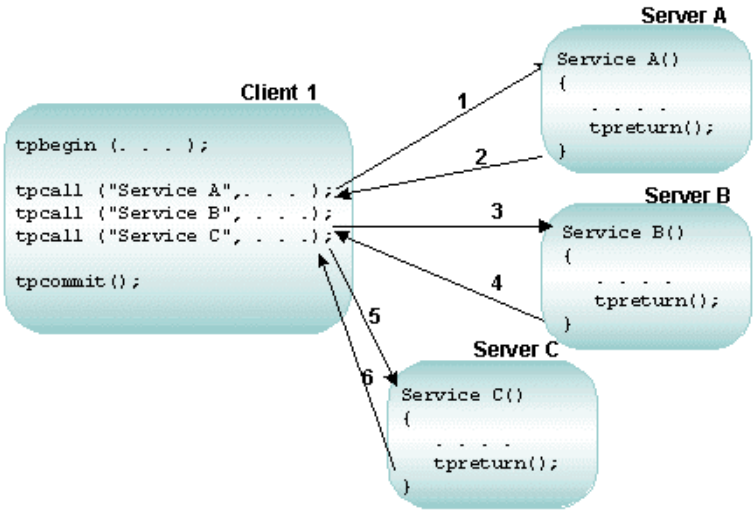
A two-phase transaction commit works in much the same way as the vacation planning project.

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 Oracle Tuxedo system tracks all participants to determine which of them should be involved in the two-phase commit. Figure 4-2 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 Oracle Tuxedo ATMI Applications Using C*
- "What You Can Do Using the ATMI" in *Introducing Oracle Tuxedo ATMI*
- For more information about using transactions in a Oracle Tuxedo CORBA environment, refer to *Using CORBA Transactions*

CHAPTER 5

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 an Oracle 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 `UBBCONFIG` file in the following ways.

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

Specifying Global Transaction Parameters in the RESOURCES Section

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

Set This Parameter . . .	To . . .
MAXGTT	Limit the total number of global transaction identifiers (GTRIDs) allowed on one machine at one time. The maximum value allowed is 2048; the minimum, 0; and the default, 100. You can override the value of MAXGTT on a per-machine basis in the MACHINES section. Entries remain in the table only while a global transaction is active, so this parameter has the effect of setting a limit on the number of simultaneous transactions.
CMTRET	Indicate the initial setting of the TP_COMMIT_CONTROL characteristic as one of the following: • LOGGED—the TP_COMMIT_CONTROL characteristic is set to TP_CMT_LOGGED, which means that tpcommit() returns when all the participants have successfully pre-committed. • COMPLETE—the TP_COMMIT_CONTROL characteristic is set to TP_CMT_COMPLETE, which means that tpcommit() does not return until all the participants have successfully committed. The default is COMPLETE. To determine the appropriate setting, consult your resource manager (RM) vendors. If any RM in the application uses the *late commit* implementation of the XA standard, the setting should be COMPLETE. If all RMs use the *early commit* implementation, the setting should be LOGGED for performance reasons. (You can override this setting with tpscmt().)
MAXTRANTIME	Specify the maximum length of the timeout for the transactions. Valid values are between 0 and 2,147,483,647 inclusive. 0 represents no limitation on transaction timeout value occurs. Default is 0. **Note:** 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).

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 Oracle 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 Oracle 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 . . .
TLOGNAME	The name of the DTP transaction log for the machine.
TLOGDEVICE	The Oracle 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 TLOG on the machine. The value may contain a maximum of 64 characters.
TLOGSIZEE	The size, in physical pages, of the TLOG file. The value must be between 1 and 2048; the default, 100. Assign a value that is large enough to hold the number of outstanding transactions on the machine at a given time. One transaction is logged per page. The default should be enough for most applications.
TLOGOFFSET	The offset, in pages, from the beginning of the TLOGDEVICE to the start of the VTOC that contains the transaction log for the machine.The value must be greater than or equal to 0, and less than the number of pages on the device. The default is 0. TLOGOFFSET is rarely necessary. However, if two VTOCs share the same device, or if a VTOC is stored on a device (such as a filesystem) that is shared with another application, you can use TLOGOFFSET to indicate a starting address relative to the address of the device.

Writing TLOG to an Oracle Database

If you want to write tlog into an Oracle database, you *do not* need to create a UDL. You must do the following steps:

1. Install Oracle database 10g client (or later), create link libclntsh.so for libclntsh.so.x.x (for example, libclntsh.so.10.1) and set LD_LIBRARY_PATH for link libclntsh.so on Linux platform.
2. Set UBBCONFIG(5) TLOGDEVICE or DMCONFIG(5) DMTLOGDEV using the following format: "DB:Oracle_XA: ….".

3. Create tlog using the `tmadmin` and `dmadmin` commands.

Note: You can only write tlog to an Oracle database. Third party databases are not supported.

Creating the Domains Transaction Log

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

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

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

See Also

- "What Is the Transaction Log (TLOG)?" in *Administering an Oracle 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 Oracle 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+LogDit=/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 Oracle Tuxedo system.

```
BANKB1 GRPNO=1 TMSNAME=TMS_SQL TMSCOUNT=2
OPENINFO="TUXEDO/SQL:APPDIR/bankdl1: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/bankdl1	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	Make a service the initiator of a transaction. 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. If a transaction already exists, a new one is not started. Default is N.
TRANTIME	Specify the length of the timeout for the AUTOTRAN transactions. Valid values are between 0 and 2,147,483,647 inclusive. 0 represents no timeout. Default is 30 seconds.
ROUTING	Point to an entry in the ROUTING section where data-dependent routing is specified for transactions that request this service.

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 Oracle Tuxedo filesystem that contains the Domains transaction log (DMTLOG) for this machine. The DMTLOG is stored as an Oracle Tuxedo VTOC table on the TLOGDEVICE (an Oracle 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	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 Oracle Tuxedo filesystem. The default is 100 pages. **Note:** The number of domains in a transaction determines the number of pages you must specify in the DMTLOGSIZE parameter. There is no one-to-one mapping between transactions and log pages.

Set This Parameter . . .	To Specify . . .
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 . . .
AUTOTRAN	By gateways to automatically start and terminate transactions for remote services. This capability is required if you want to enforce reliable network communication with remote services. To request this capability, set the AUTOTRAN parameter to Y in the entry for the appropriate remote service.
TRANTIME	To specify the default 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 Oracle Tuxedo 8.0 or earlier, except that when a

machine running Oracle 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 MAXTRANTIME parameter, or the node understands the MAXTRANTIME parameter but the parameter is not set, the timeout value for the transaction is determined by TRANTIME or by the timeout value passed in the tpbegin() call that started the transaction. If the TRANTIME or tpbegin() timeout value is exceeded, all Oracle 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 MAXTRANTIME parameter and the parameter is set for that node, the timeout value for the transaction is reduced to no greater than the MAXTRANTIME value on that node.

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

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

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).

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/bankdl1:bankdb:readwrite"
BANKB2           LMID=SITE2            GRPNO=2
 OPENINFO="TUXEDO/SQL:/usr/home/bankdl2:bankdb:readwrite"
BANKB3            LMID=SITE3            GRPNO=3
 OPENINFO="TUXEDO/SQL:/usr/home/bankdl3:bankdb:readwrite"

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

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

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

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" in *Programming Oracle Tuxedo ATMI Applications Using C*
- "What You Can Do Using the ATMI" in *Introducing Oracle Tuxedo ATMI*
- For more information about using transactions in an Oracle Tuxedo CORBA environment, refer to *Using CORBA Transactions*

CHAPTER 6

Using Tuxedo with Oracle Real Application Clusters (RAC)

This chapter includes the following sections.

This release also supports the following RAC features added in previous releases.

Instance Awareness

For the servers associated with Oracle Database, Tuxedo uses customized callback to retrieve Oracle Database instance information.

- XA Server

  Tuxedo uses customized callback to retrieve Oracle Database instance information by static callback registration for XA servers.

- Non-XA Server

Tuxedo uses customized callback to retrieve Oracle Database instance information by dynamic callback registration for non-XA servers with the following requirements.

– `$TUXDIR/lib/tuxociucb.so.1.0` package should be deployed in `$ORACLE_HOME/lib`.

– `ORA_OCI_UCBPKG` environment variable must include the package name.

It is necessary to deploy the package into `$ORACLE_HOME/lib` before running a non-XA server that needs instance awareness; otherwise, the server that uses dynamic callback registration will fail.

Non-XA server tries to automatically set the environment variable `ORA_OCI_UCBPKG` when booting up. If the server fails to do so, an error message will occur in `ULOG` and the instance awareness will be disabled in the server.

Using Tuxedo with XA Affinity

This section contains the following topics.

Overview

Oracle RAC environments have a one-to-many relationship between database and instances. Servers or groups of a Tuxedo application running on an Oracle RAC may connect to different instances.

It is XA Affinity that ensures all database operations to connect the same RAC instance when possible (no matter if those operations are in one global transaction branch or in different branches of one global transaction) and automatically exchanges the affinity information between Tuxedo domain and WebLogic Server via WTC (WebLogic Tuxedo Connector).

XA Affinity Priority

Tuxedo server selection rules are listed as below, in high to low priority order.

- `GWTDOMAIN` Transaction routing
- Oracle RAC routing for transaction affinity using RAC instances
- Service versioning
- Client/server affinity routing
- XA Affinity
- Load balance based on Oracle RAC LBA (Load Balancing Advisor)
- Tuxedo load balance

If XA Affinity is enabled, the Oracle RAC routing rule that environment variable `TUXRACGROUPS` specifies will be disabled.

Note: XA Affinity has no impact on domain routing. Only when a request arrives at a domain and starts to be routed to a service in that domain, XA Affinity affects server routing.

XA Affinity Policy

Tuxedo selects the server that is associated with the same instance name, DB name, and service name. If this attempt fails, Tuxedo follows the following policies to find the server.

- Tuxedo tries to find the server which is associated with both the same DB name and the same service name. The server's group must not be involved in the current global transaction.
- If the above attempt fails, Tuxedo tries to find the server which is associated with both the same DB name and the same instance name.
- If the above attempt fails, Tuxedo tries to find the server which is associated with the same DB name.
- If the above attempt fails, Tuxedo finds the server according to Tuxedo normal load balance.

Note: If Tuxedo finds multiple servers that are at the same priority, Tuxedo will find the server according to Tuxedo normal load balance.

Prerequisites

Software Requirements

The feature can be run on all Oracle Tuxedo supported platforms, except for Oracle Tuxedo 32-bit on Microsoft Windows platforms.

For specific platform software requirements, refer to Oracle Tuxedo 12c Release 2 (12.2.2) Platform Data Sheets.

Installation Notes

- The Oracle Tuxedo must be 12*c* Release 2 (12.1.3) or above.
- The Oracle Database must be 11.2.0.2.0 or above.

Configurations

As long as the option `XPP` in `OPTIONS` of `UBBCONFIG *RESOURCES` section is specified, XA Affinity feature is enabled by default.

Note: On Oracle Exalogic and Oracle SPARC SuperCluster platforms, the `OPTIONS` parameter must be set to `EECS`.

A new option, `NO_XAAFFINITY`, is introduced to `RMOPTIONS` of `UBBCONFIG *RESOURCES` section to explicitly disable XA Affinity.

```
RMOPTIONS {[...|NO_XAAFFINITY],*}
```

Listing 6-1 shows an example to configure `EECS`; Listing 6-2 lists an example to explicitly disable XA Affinity.

Listing 6-1 Example to Configure EECS

```
* RESOURCES
OPTIONS       EECS
```

Listing 6-2 Example to Explicitly Disable XA Affinity

```
* RESOURCES
```

```
OPTIONS       EECS
RMOPTIONS     NO_XAAFFINITY
```

This flag can also be specified in `T_DOMAIN` class via `TM_MIB`, when the tuxedo application is inactive. For more information, see File Formats, Data Descriptions, MIBs, and System Processes Reference.

Note: XA Affinity requires Tuxedo servers to retrieve Oracle Database instance information. Users can query a server's instance information through Tuxedo `TM_MIB` `T_SERVER` class's `TA_INSTSTR` field. For more information, see T_SERVER Class Definition.

Limitations

- Groups with MRM (multiple RM) are not supported.
- The max number of affinity context (database name+instance name+service name) in one transaction is 16.
- XA Affinity does not support multi-server single queue.
- XA Affinity does not support multi-threaded server.
- XA Affinity does not support cross-domain services.

Using Tuxedo with Common XID

This section contains the following topics.

Overview

In general, for global transactions, each participating group has its own transaction branch, and a distinguished transaction branch identifier (XID) identifies each branch. If a global transaction

involves multiple groups, Oracle Tuxedo adopts two-phase commit on each branch, taking the first participating group as the coordinator.

However, with common XID feature in this release, Oracle Tuxedo shares the coordinator group transaction branch with all other groups within the same transaction. The groups that connect to the same Oracle Database instance through the same service use the coordinator branch directly. For those groups, XA committing operations are not required and are saved.

In the most extreme case, where all groups in a global transaction use the coordinator branch directly, Oracle Tuxedo adopts one-phase commit instead of two-phase commit; no `TLOG` is written.

Typical Scenario

Assume there are two groups `GRP1` and `GRP2` in a Tuxedo application domain `DOM1`. Server `SERV1` belongs to `GRP1` and offers service `SVC1` while server `SERV2` belongs to `GRP2` and offers service `SVC2`. Both `SERV1` and `SERV2` connect to the same instance.

Then a native client begins a global transaction at first, invokes `SVC1` followed by `SVC2`, and commits the transaction.

If common XID functionality is enabled in the above case, Tuxedo invokes one-phase commit on the transaction and no `TLOG` is written; otherwise, two-phase commit is invoked.

Users are allowed to trace the above behaviors through `TMTRACE`. Please refer to `TMTRACE` for more information.

Prerequisites

Software Requirements

The feature can be run on all Oracle Tuxedo supported platforms, except for Oracle Tuxedo 32-bit on Microsoft Windows platforms.

For specific platform software requirements, refer to Oracle Tuxedo 12c Release 2 (12.2.2) Platform Data Sheets.

Installation Notes

- The Oracle Tuxedo must be 12*c* Release 2 (12.1.3) or above.
- The Oracle Database must be 11.2.0.2.0 or above.

Configurations

As long as the option EECS in OPTIONS of UBBCONFIG *RESOURCES section is specified, common XID feature is enabled by default. A new option, NO_COMMONXID, is introduced to RMOPTIONS of UBBCONFIG *RESOURCES section to explicitly disable common XID.

```
RMOPTIONS {[...|NO_COMMONXID],*}
```

Listing 6-1, "Example to Configure EECS," on page 6-4 shows an example to configure EECS; Listing 6-3 lists an example to explicitly disable common XID.

Listing 6-3 Example to Explicitly Disable Common XID

```
* RESOURCES

OPTIONS       EECS

RMOPTIONS     NO_COMMONXID
```

This flag can also be specified in T_DOMAIN class via TM_MIB, when the tuxedo application is inactive. For more information, see File Formats, Data Descriptions, MIBs, and System Processes Reference.

Note: Common XID requires Tuxedo servers to retrieve Oracle Database instance information. Users can query a server's instance information through Tuxedo TM_MIB T_SERVER class's TA_INSTSTR field. For more information, see T_SERVER Class Definition.

Limitations

- Groups with MRM (multiple RM) are not supported.
- Multi-threaded servers do not provide instance information via MIB; however, common XID still performs well on server-dispatched threads.
- In two-phase commit scenarios, GWTDOMAIN is always involved to do prepare and/or commit.
- If the coordinator group is the group where GWTDOMAIN locates, common XID does not work.

Using Tuxedo with Single Group Multiple Branches (SGMB)

This section contains the following topics.

Overview

In previous releases, servers in the same participated group use the same transaction branch in a global transaction. However the transaction branch would fail if these serves connect to different instances of a RAC. An XA error, `XAER_AFFINITY`, will be reported, meaning one branch cannot go through different instances. For this reason, the RAC service used by a Tuxedo group must be a singleton RAC service. A DTP service (if the DTP option, `-x` in `srvctl modify service`, is specified) or a service offered by only one instance could be a singleton RAC service.

In this release, using different transaction branches on different instances in a single group will solve the issue. The Tuxedo group can then use non-singleton service and take advantage of its benefits, such as load balance.

Prerequisites

Software Requirements

The feature can be run on all Oracle Tuxedo supported platforms, except for Oracle Tuxedo 32-bit on Microsoft Windows platforms.

For specific platform software requirements, refer to Oracle Tuxedo 12c Release 2 (12.2.2) Platform Data Sheets.

Installation Notes

- The Oracle Tuxedo must be 12*c* Release 2 (12.1.3) or above.
- The Oracle Database must be 11.2.0.2.0 or above.

Configurations

As long as the option EECS in OPTIONS of UBBCONFIG *RESOURCES section is specified, SGMB feature is enabled by default. A new option, SINGLETON, is introduced to RMOPTIONS of UBBCONFIG *RESOURCES section to explicitly disable SGMB.

```
RMOPTIONS {[...|SINGLETON],*}
```

This option indicates all RAC services used in the domain are singleton, so the SGMB feature is not necessary to work.

Listing 6-1, "Example to Configure EECS," on page 6-4 shows an example to configure EECS; Listing 6-4 lists an example to explicitly disable SGMB.

Listing 6-4 Example to Explicitly Disable SGMB

```
* RESOURCES

OPTIONS       EECS

RMOPTIONS     SINGLETON
```

This flag can also be specified in T_DOMAIN class via TM_MIB, when the tuxedo application is inactive. For more information, see File Formats, Data Descriptions, MIBs, and System Processes Reference.

Note: SGMB requires Tuxedo servers to retrieve Oracle Database instance information. Users can query a server's instance information through Tuxedo TM_MIB T_SERVER class's TA_INSTSTR field. For more information, see T_SERVER Class Definition.

Limitations

- Groups with MRM (multiple RM) are not supported.
- A transaction fails if more than 16 instances are involved in a single group.
- Read-Only optimization for XA does not work in a transaction if the preferred reserved group is a multi-branch group. If GWTDOMAIN is not the coordinator, the preferred reserved group is the coordinator group; otherwise, the preferred reserved group is the participated group coming next in the coordinator domain.

- Multi-threaded servers do not provide instance information via `MIB`; however, SGMB still performs well on server-dispatched threads.

Using Tuxedo with Fast Application Notification (FAN)

This section contains the following topics.

Overview

Tuxedo uses Fast Application Notification (FAN) to

- Provide rapid failure detection.
- Remove invalid DB connections from Tuxedo server and create valid DB connection. If Tuxedo server cannot create valid DB connection, Tuxedo removes these servers from the routing list.
- Perform graceful shutdown for planned and unplanned Oracle RAC node outages.
- Adapt to changes in topology, such as adding or removing a node.
- Distribute runtime work requests to all active Oracle RAC instances, including those rejoining a cluster.

Prerequisites

Software Requirements

The feature can be run on all Oracle Tuxedo supported platforms, except for Oracle Tuxedo 32-bit on Microsoft Windows platforms.

For specific platform software requirements, refer to Oracle Tuxedo 12c Release 2 (12.2.2) Platform Data Sheets.

Installation Notes

- The Oracle Tuxedo must be 12*c* Release 2 (12.1.3) or above.
- The Oracle Database must be 11.2.0.2.0 or above.
- Using the connection steering requires Oracle client 11.2.0.2.0 or Oracle client 12.1.0.1.0 with a specific patch (contact Oracle Support for the patch, or higher release of Oracle client).

Configurations

Configurations on DB

DB configuration includes the following topics.

ONS

On Oracle server side, ONS daemon must be enabled.

If Tuxedo is taken as a native client, ONS daemon on the client side must also be enabled. The ONS daemon configuration file is located in `$ORACLE_HOME/opmn/conf/ons.config`. After configuring ONS, start ONS daemon with `onsctl start` command. Please make sure that ONS daemon is running all the time.

If Tuxedo is taken as a remote client, ONS daemon on the client side is not used. It is the preferred mode.

Note: On the Oracle client side, if the Oracle version is lower than 12.1.0.1.0, ONS daemon must be enabled.

Load Balancing Advisor (LBA)

The ONS may publish LBA about a service if the service has load balancing advisory goal. You can use `-B` option to specify the goal via `srvctl` when creating or modifying the service.

TAF

If TAF is enabled, all Tuxedo servers can automatically do the reconnection by TAF; otherwise, only XA servers can automatically do the reconnection.

Reconnection is finished by TAF with the following requirements for user code.

- XA server

  `OPENINFO` must include `threads=t`.

- Non-XA server

  To monitor FAN event for the instance associated with the specific non-XA application server, `$TUXDIR/lib/tuxociucb.so.1.0` should be deployed in `$ORACLE_HOME/lib`, and the name of this binary must be specified in `ORA_OCI_UCBPKG` environment variable.

  `-L` option in the servopts must be used for a non-XA server to indicate that the server will connect to the Oracle Database. Since the ECID is enabled when `-L` is specified, a new option `-F` is introduced into servopts to close ECID. The usage is `F noECID`. The example is below.

  ```
  *SERVERS

  server1
  SRVGRP=GRP1 SRVID=1 ClOPT="-L libclntsh.so -F noECID"
  ```

  For TAF support, the OCI environment must be created in `OCI_THREADED` mode.

  Pro*C users should be able to precompile with `threads=yes` and use the embedded SQL statement as below before creating the first executable embedded SQL statement; otherwise, only XA servers can do the reconnection.

  ```
  EXEC SQL ENABLE THREADS;
  ```

Configurations on Tuxedo

It is required to configure `TMFAN` server in `UBBCONFIG *SERVERS` section and configure the option `EECS` in `OPTIONS` of `UBBCONFIG *RESOURCES` section. A new option, `NO_FAN`, is introduced to `RMOPTIONS` of `UBBCONFIG *RESOURCES` section to explicitly disable FAN.

```
RMOPTIONS {[...|NO_FAN],*}
```

Listing 6-1, “Example to Configure EECS,” on page 6-4 shows an example to configure `EECS`; Listing 6-5 lists an example to explicitly disable FAN.

Listing 6-5 Example to Explicitly Disable FAN

```
* RESOURCES

OPTIONS       EECS

RMOPTIONS     NO_FAN
```

This flag can also be specified in `T_DOMAIN` class via `TM_MIB`, when the tuxedo application is inactive. For more information, see File Formats, Data Descriptions, MIBs, and System Processes Reference.

Note: FAN requires Tuxedo servers to retrieve Oracle Database instance information. Users can query a server's instance information through Tuxedo `TM_MIB` `T_SERVER` class's `TA_INSTSTR` field. For more information, see T_SERVER Class Definition.

Limitations

- Groups with MRM (multiple RM) are not supported.
- If the customized server is going to use OCI to connect Oracle database, `OCI_NO_UCB` should not be set at OCI initialization time.
- Load balance based on Oracle RAC LBA (Load Balancing Advisor) does not support multi-server single queue.
- Load balance based on Oracle RAC LBA (Load Balancing Advisor) does not support multi-threaded server.
- Load balance based on Oracle RAC LBA (Load Balancing Advisor) does not support cross-domain services.

Using Tuxedo with Oracle Real Application Clusters (RAC)

This section contains the following topics.

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 or to obtain optimal RAC performance. This is because Oracle 10g does not allow the same database to be accessed from *multiple* RAC instances within the *same* XA transaction.

Note: Oracle 12c does allow the same database to be accessed from multiple RAC instances within the same XA transaction, but performance may be better if all accesses within a particular XA transaction occur from the same RAC instance.

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

Tuxedo 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.

Note: When using Oracle 10gR2, administrators should use an Oracle `<b>DTP Service</b>` to access the Oracle RAC system. This DTP service name should be specified in the OPENINFO string for the associated Tuxedo groups. Oracle 10gR2 verifies the service name, and migrates it to an alternate instance if required.

When using Oracle 12c or later release, the service name is transparently and automatically migrated to an alternate instance, if required, without any specific configuration.

Limitations

- Tuxedo supports Oracle RAC only when using Oracle 10g or later release, and does not support Oracle RAC when using Oracle 9i.

  Note: For Oracle 10gR1, patch set 10.1.0.3 or above is required.

  For Oracle 10gR2, patch set 10.2.0.2 or above is required due to the bug described at: https://metalink.oracle.com/metalink/plsql/f?p=130:14:3193163745563425327::::p14_database_id.

  For Oracle 12c, patch set 11.1.0.6 or above is preferred. Use of Oracle 12c is highly encouraged due to significant RAC improvements.

- 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

Software Requirements

For specific platform software requirements, refer to Oracle Tuxedo 12c Release 2 (12.2.2) Platform Data Sheets.

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 for Oracle RAC support:

- `Three environment variables`
  - `TUXRACGROUPS` (required for Oracle 10gR1 and 10gR2, optional for Oracle 11g and later releases)
  - `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 further information on using Oracle XA with RAC, refer to 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.

Note: When using Oracle 10g, 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.

Oracle 12c permits different RAC instances to operate on different transaction branches in RAC, and if transaction branches are on different instances, then they are tightly coupled and share locks and resources. So the `TUXRACGROUPS` environment variable is not necessary when using Oracle 12c. This environment variable still works in Oracle 12c and you can use it to associate Tuxedo groups with specific instances of Oracle RAC configurations.

`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: If the TUXRACGROUPS environment variable is used, it *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 1: Simple Configuration

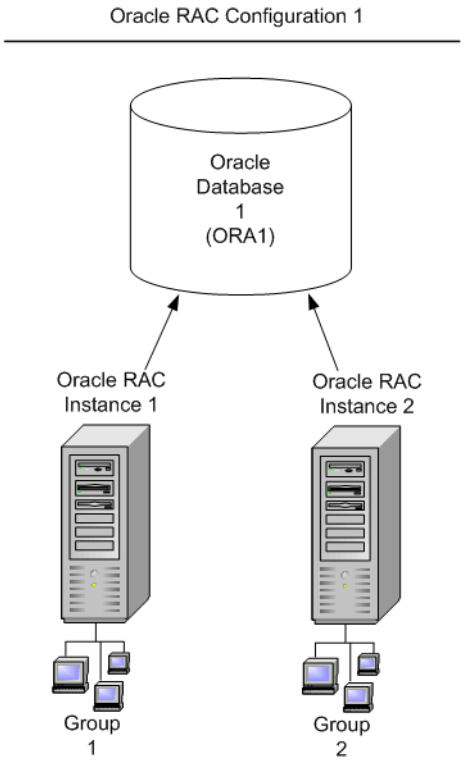
```
TUXRACGROUPS="G1;G2"
```

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.

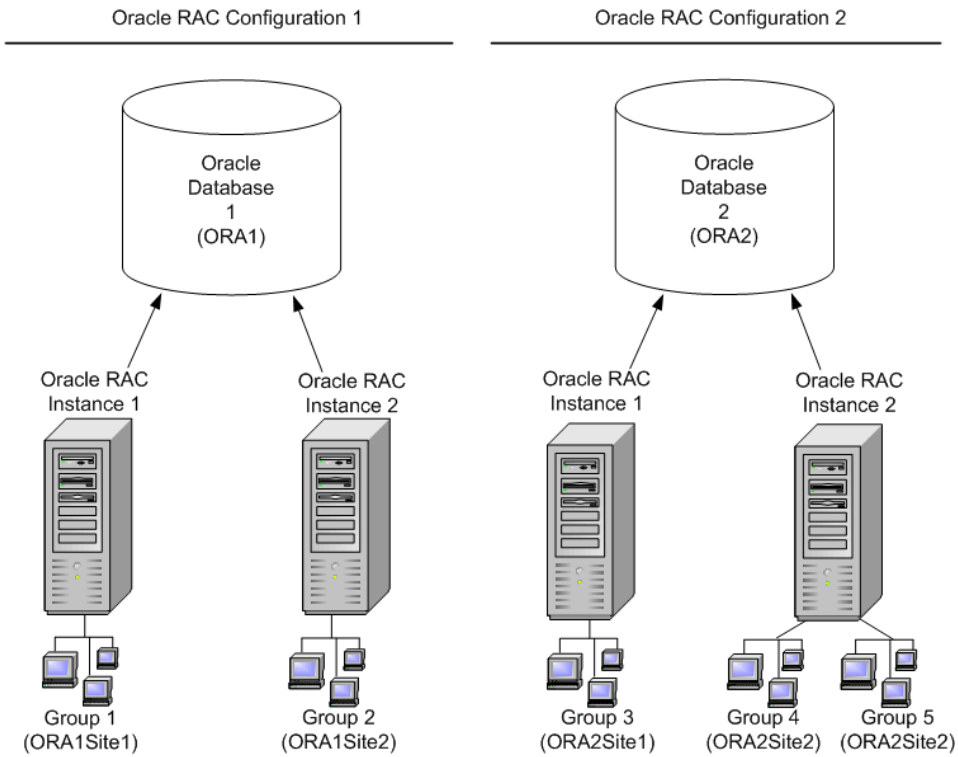
In this example, there are two Oracle databases: `ORA1` and `ORA2`. `ORA1` offers machine-specific services `ORA1SITE1` and `ORA1SITE2`, and `ORA2` offers machine-specific services `ORA2SITE1` and `ORA2SITE2`. The objective is to assign an approximately equal number of transactions and configure the 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 in Listing 6-6:

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

```
*GROUPS

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.

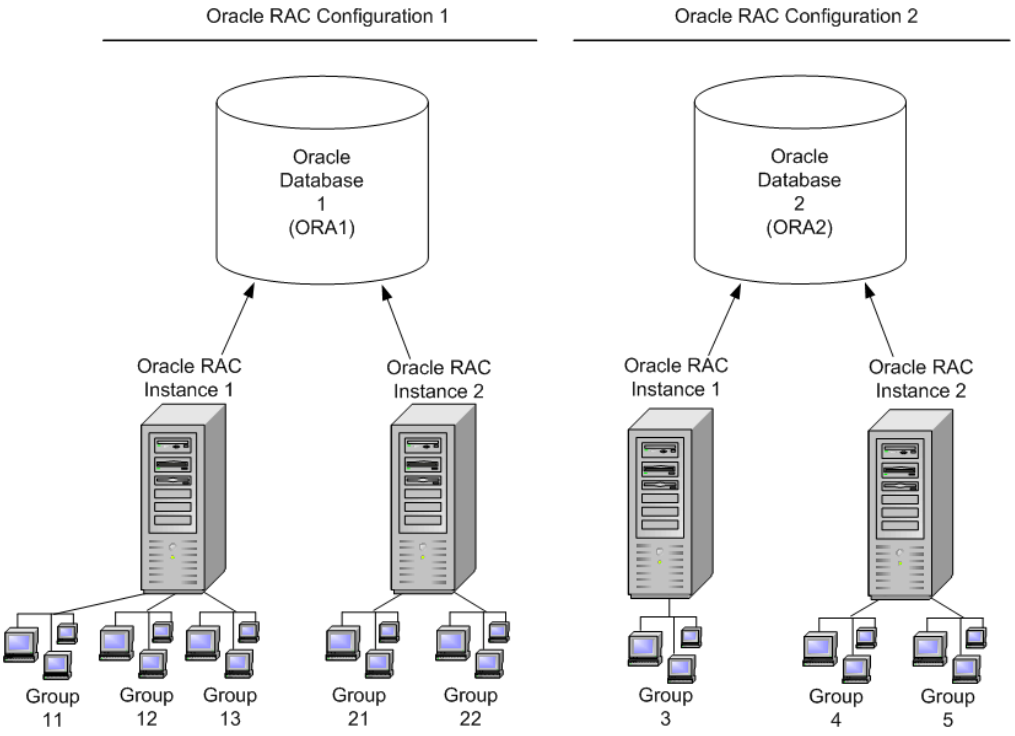
This example is similar to the previous example — except that GROUP11, GROUP12, and GROUP13 are all associated with the first RAC instance of the first RAC configuration, and GROUP21 and GROUP22 are both associated with the second RAC instance.

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
- tperrno 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.

Transaction Creation Behavior Using TUXRACGROUPS

Transactions are a pinned to Oracle RAC instances for as long as they exist. This is true independently, whether the call flow for such a transaction ever reaches a Tuxedo service associated with Oracle RAC or not.

There are two ways that transactions can be created:

- Transactions created in a group listed inside TUXRACGROUPS are pinned to the Oracle RAC instance configured via TUXRACGROUPS.
- Transactions created in groups not listed inside TUXRACGROUPS are pinned to one of the available Oracle RAC instances in a load-balancing-like algorithm.

Data Dependent Routing Using TUXRACGROUPS

Data dependent routing has been extended to support Oracle RAC configurations. It is possible to define multiple groups for the same routing range in the UBBCONFIG *ROUTING section. Listing 6-7 shows an example of different Tuxedo groups with the same range of values.

Listing 6-7 Tuxedo Groups with Same Range Values

```
RANGES="1-5:GROUP1A, 1-5:GROUP1B, 6-10:GROUP2B, 6-10:GROUP2A, *:*"
```

In this example, GROUP1A and GROUP1B are responsible for the same data range and GROUP2A and GROUP2B are responsible for the same data range. Tuxedo routes the service request to the group associated with the Oracle RAC instance that the transaction belongs to.

Data dependent routing for transactional services offered in RAC groups achieves the desired result only if:

- Each Oracle RAC Instance configuration 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 service instances connected to the same Oracle RAC Instance 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.

- Multiple `RANGES` entries for each routing value must be created for each RAC instance offering the service.

  If a routing was not configured for a special RAC instance a service calls for a transaction pinned to that Oracle RAC Instance will fail with `tperrno` set to `TPENOENT` and `tperror` detail set to `TPED_GROUP_FORBIDDEN`.

When transactional 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. All non-transactional requests for a special range of values are handled by one Oracle RAC instance only.

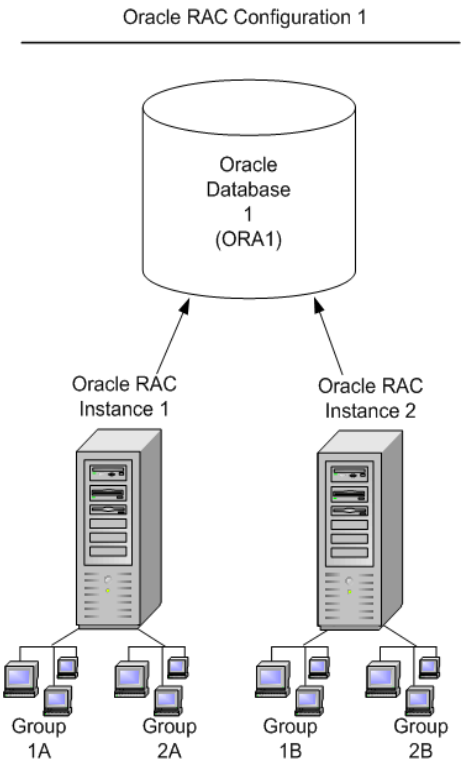
If routing is performed for a mixture of transactional and non-transactional requests, some applications may not require non-transactional request load balancing. You can vary the RAC instances listed first in your application for different data values so that non-transactional requests are balanced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

There is no way to enforce load balancing between all groups associated with the same routing range for non-transactional requests. If you want to enforce one-by-one load balancing, try the following:

- Varying the RAC instance listed first for each data value so that each RAC instance occurs first for approximately equal amounts of data, or
- Calling an intermediate AUTOTRAN service (in the UBBCONFIG file `*SERVICES` section) to enforce that each service call is associated with a transaction.

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

Figure 6-4 Routing Transactional/Non-Transactional Requests



The configuration shown in the example consists of 2 Oracle RAC instances. If 1,000 transactions are created in a group not listed in `TUXRACGROUPS`, around 500 transactions will be pinned to Oracle RAC instance 1 and can only access `GROUP1A` and `GROUP2A`. The other 500 transactions will be pinned to Oracle RAC instance 2 and can only access `GROUP1B` and `GROUP2B`.

Listing 6-8 shows an example of how the `*SERVICES` and `*ROUTING` sections of the `UBBCONFIG` file for this configuration might look:

Listing 6-8 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"
```

GROUP1A and GROUP2A belong to Oracle RAC instance 1. GROUP1B and GROUP2B belong to Oracle RAC instance 2. Requests with a BRANCH_ID 1 through 5 must be handled by GROUP1A or GROUP1B. Requests with a BRANCH_ID 6 through 10 must be handled by GROUP2A or GROUP2B.

For transactional requests, all transactions pinned to Oracle RAC instance 1; branches 1-5 map to GROUP1A and branches 6-10 map to GROUP2A. The other half is assigned to Oracle RAC instance 2; 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. These are the first groups specified that match the respective routing ranges. Requests with an invalid BRANCH_ID are mapped to any permitted group.

Note: Oracle RAC instance 1 is specified first for one data range and RAC instance 2 is specified first for the other data range in an attempt to achieve some non-transactional load balancing between RAC instances.

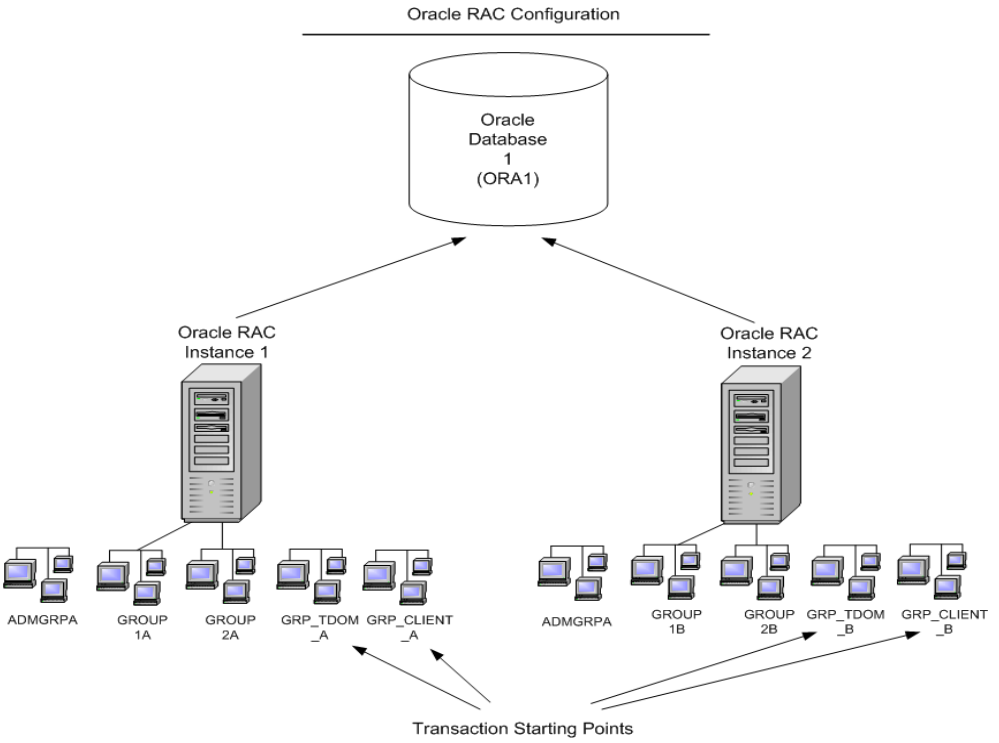
Assigning Transactions to Special Oracle RAC Instances

You may want to split your environment into multiple machines. For example, you may want a Tuxedo domain with some machines only accessing Oracle RAC instance 1 and other machines only accessing Oracle RAC instance 2 in order to enforce regional independency if Tuxedo installations and Oracle RAC installations are distributed over different buildings. The environment may be configured so that as few as possible calls should be sent outside of a building.

Figure 6-5 shows an example with, machine 1 serving GROUP1A and GROUP2A; machine 2 serving GROUP1B and GROUP2B. In addition, calls might be made and transactions might be created from a Tuxedo /Domain Gateway, for Tuxedo /WS clients, Tuxedo Native Clients, Tuxedo /Q, or any server linked with another Resource Manager such as MQ Series.

Whenever a request is sent, the transaction should be pinned to the local machine and avoid hopping between different machines as much as possible.

Figure 6-5 Assigning Transactions to Special Oracle RAC Instances



Listing 6-9 shows a UBBCONFIG file example with two physical machines, `TUXM1` and `TUXM2`, running Tuxedo. Both machines have two groups connecting to an Oracle RAC. Groups `GROUP1A` and `GROUP2A` are running on machine `TUXM1` connecting to RAC instance 1. Groups `GROUP1B` and `GROUP2B` are running on machine `TUXM2` connecting to RAC instance 2.

Listing 6-9 UBBCONFIG File Example

```
*MACHINES

DEFAULT:
```

```
        APPDIR="/path/to/appdir"
        ENVFILE="/path/to/oracle.env"
        TUXDIR="/path/to/tuxdir"
        TUXCONFIG="/path/to/tuxconfig"
        TLOGDEVICE="/path/to/TLOG"

"machine1"    LMID=TUXM1

"machine2"    LMID=TUXM2

*GROUPS
ADMGRPA       LMID=TUXM1    GRPNO=10      OPENINFO=NONE
ADMGRPB       LMID=TUXM2    GRPNO=20      OPENINFO=NONE

GROUP1A LMID=TUXM1 GRPNO=101 TMSNAME=TMS_ORA

OPENINFO="Oracle_XA:Oracle_XA+ACC=P/user/password+Sqlnet=ORA1SITE1+SesTm=1
00+LogDir=.+MaxCur=5"
GROUP1B LMID=TUXM2 GRPNO=102 TMSNAME=TMS_ORA

OPENINFO="Oracle_XA:Oracle_XA+ACC=P/user/password+Sqlnet=ORA1SITE2+SesTm=1
00+LogDir=.+MaxCur=5"

GROUP2A LMID=TUXM1 GRPNO=201 TMSNAME=TMS_ORA

OPENINFO="Oracle_XA:Oracle_XA+ACC=P/user/password+Sqlnet=ORA1SITE1+SesTm=1
00+LogDir=.+MaxCur=5"
GROUP2B LMID=TUXM2 GRPNO=202 TMSNAME=TMS_ORA

OPENINFO="Oracle_XA:Oracle_XA+ACC=P/user/password+Sqlnet=ORA1SITE2+SesTm=1
00+LogDir=.+MaxCur=5"

GROUP_TDOM_A LMID=TUXM1 GRPNO=301
GROUP_TDOM_B LMID=TUXM2 GRPNO=302

GROUP_CLIENT_A LMID=TUXM1 GRPNO=401 TMSNAME=TMS
GROUP_CLIENT_B LMID=TUXM2 GRPNO=402 TMSNAME=TMS
```

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

TMSYSEVT SRVGRP="ADMGRPA" SRVID=10
TMUSREVT SRVGRP="ADMGRPA" SRVID=20

TMSYSEVT SRVGRP="ADMGRPB" SRVID=10 CLOPT="-A -- -S "
TMUSREVT SRVGRP="ADMGRPB" SRVID=20 CLOPT="-A -- -S "

EMPLOYEE_SVR SRVGRP=GROUP1A SRVID=1
EMPLOYEE_SVR SRVGRP=GROUP1B SRVID=2
BANKING_SVR SRVGRP=GROUP2A SRVID=3
BANKING_SVR SRVGRP=GROUP2B SRVID=4

DMADM  SRVGRP="GROUP_TDOM_A"       SRVID=100
GWADM  SRVGRP="GROUP_TDOM_A"       SRVID=110
GWTDOMAIN SRVGRP="GROUP_TDOM_A"    SRVID=111 REPLYQ=Y   RQADDR="GWGRP_M1"
GWADM  SRVGRP="GROUP_TDOM_B"       SRVID=110
GWTDOMAIN SRVGRP="GROUP_TDOM_B"    SRVID=111 REPLYQ=Y   RQADDR="GWGRP_M2"
```

Additionally, there is a group for administrative services, as well as one group for Tuxedo /Domain gateways and one group for native Tuxedo clients on both machines. All transactions are created by GWTDOMAIN and native clients. Even if GWTDOMAIN and the native Tuxedo clients never connect to an Oracle RAC directly, they must be included in `TUXRACGROUPS` as shown in Listing 6-10 to ensure that the opened transactions belong to the correct RAC instance and are handled locally.

Note: Native clients must set `tpinfo->grpname` to the local group to ensure the right behavior. For more information, see "Avoiding Transactions Created by Tuxedo Native Clients Being Sent to a Remote Machine" on page 6-31.

Listing 6-10 TUXGROUPS

```
TUXRACGROUPS="GROUP_TDOM_A,GROUP_CLIENT_A,GROUP1A,GROUP2A;GROUP_TDOM_B,
GROUP_CLIENT_B,GROUP1B,GROUP2B"
```

TUXRAGROUPS Transaction Use Cases

Dealing with Service Calls that are Made Outside of Transactions

As long as no transaction is involved, Tuxedo will try to handle as many requests as possible on the local machine as long as the load allows and requests will only go to remote machines if no local services are idle according to the load balancing algorithm. Summarized this means one does not have to care about requests sent to remote machines if all services are available on all machines.

What an administrator always has to ensure is that he includes all service groups into the `TUXRACGROUPS` environment variable that are accessed during the call flow and that are candidates for opening a new transaction even if they are not linked with the Oracle RM and/or are not physically associated with any Oracle RAC instance. The environment variable `TUXRACGROUPS` does not have any impact for non-transactional service calls.

Avoiding Transactions Created by a Group Handling an External Resource Manager Being Sent to a Remote Machine

If you have a Tuxedo server built with another RM such as MQSeries or another database, you can force newly started transactions to be pinned to your local machine by including this group into the `TUXRACGROUPS` environment variable as well.

Listing 6-11 MQSeries Example

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

In this example `MQSGROUPA`, `GROUP1A` and `GROUP2A` are located on machine 1 and `MQSGROUPB`, `GROUP1B` and `GROUP2B` are located on machine 2.

If a server inside group `MQSGROUPA` creates a transaction, all Tuxedo service calls for services under groups `GROUP1A`, `GROUP2A`, `GROUP1B` and `GROUP2B` will only go to `GROUP1A` and `GROUP2A`. `GROUP1B` and `GROUP2B` are ignored as they belong to RAC instance 2 and the transaction was already created for RAC instance 1 via group `MQSGROUPA`.

Avoiding Transactions Created by GWTDOMAIN Being Sent to a Remote Machine?

Create on local Tuxedo /Domain Gateway on each machine. Set the `TUXRACGROUPS` environment variable as shown in Listing 6-12.

Listing 6-12 GWTDOMAIN Example

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

In this example GWTGROUPA, GROUP1A and GROUP2A are located on machine 1 and GWTGROUPB, GROUP1B and GROUP2B are located on machine 2.

If GWTDOMAIN on machine 1 creates a new transaction because it receives an external request, all Tuxedo service calls for services under groups GROUP1A, GROUP2A, GROUP1B and GROUP2B will only go to GROUP1A and GROUP2A. GROUP1B and GROUP2B are ignored as they belong to RAC instance 2 and the transaction was already created for RAC instance 1 via group GWTGROUPA.

Avoiding Transactions created by TMQFORWARD Being Sent to a Remote Machine

Create a local Tuxedo /Q configuration on each machine. Set your TUXRACGROUPS environment variable as shown in Listing 6-13.

Listing 6-13 TMQFORWARD Example

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

In this example QUEUEGROUPA, GROUP1A and GROUP2A are located on machine 1 and QUEUEGROUPB, GROUP1B and GROUP2B are located on machine 2.

If TMQFORWARD on machine 1 transactionally forwards a new message to such an Oracle service, all Tuxedo service calls for services under groups GROUP1A, GROUP2A, GROUP1B and GROUP2B will only go to GROUP1A and GROUP2A. GROUP1B and GROUP2B are ignored as they belong to RAC instance 2 and the transaction was already created for RAC instance 1 via group QUEUEGROUPA.

Avoiding Transactions Created by Tuxedo Native Clients Being Sent to a Remote Machine

You can also bind native clients to a special server group. You just have to build the client using the command `buildclient -r <RM_of_the_group> -f <source_file> -o <binary_file>` and initiate `tpinit()` with the group name that you want to use.

For example, you can create two additional groups CLIENTGROUPA and CLIENTGROUPB,and start at least two TMS in each group. Set your TUXRACGROUPS environment variable as shown in Listing 6-14.

Listing 6-14 Remote Machine Example

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

Whenever you initiate tpinit(TPINIT *tpinfo) with a TPINIT structure where tpinfo->grpname is set to CLIENTGROUPA the client is associated with CLIENTGROUPA. When tpinfo->grpname is set to CLIENTGROUPB, the client is associated with CLIENTGROUPB.

Native clients on machine 1 should always call tpinit() with tpinfo->grpname = CLIENTGROUPA; native clients on machine 2 should always call tpinit() with tpinfo->grpname = CLIENTGROUPB if CLIENTGROUPA is running on machine 1 and CLIENTGROUPB is running on machine 2. When a Tuxedo Native Client calls tpbegin(), the transaction is associated with RAC instance 1 in case of CLIENTGROUPA and with RAC instance 2 in case of CLIENTGROUPB

Avoiding Sending Transactions Created by Tuxedo /WS Clients to Remote Machines

The grpname value must be the NULL string (0-length string) for Workstation clients. You cannot set any group name and you cannot pin /WS clients to special groups. tpbegin() inside the Tuxedo /WS clients is always unspecified and the opened transaction is distributed in equal parts over all RAC instances.

The best practice to use with Tuxedo /WS Clients is to avoid transaction handling on the client side, and start the transaction with the first server that is called by the Tuxedo /WS Client. For example, you can automatically force creating a transaction when setting the AUTOTRAN parameter for the called service in the UBBCONFIG file *SERVICES section.

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

Listing 6-15 is an example of an Oracle FAN script.

Listing 6-15 Oracle FAN Script Example

```
//This File should be placed at ORA_CRS_HOME/racg/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 6-6, 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.

Configuring Transaction Recovery for Oracle 12c

For Oracle 12c, no specific configuration is needed; transaction recovery is transparent.

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” in *Programming an Oracle Tuxedo ATMI Application Using C*
- Oracle Real Application Clusters Home Page
- Oracle Application Server Adapters for Tuxedo
- Best Practices for Using XA with RAC

CHAPTER 7

Enabling IPv6

This topic includes the following sections:

Overview

IPv6 is the next generation internet protocol. It fixes a number of problems in IPv4, such as the limited number of available IPv4 addresses. It also adds many improvements to IPv4 in areas such as routing and network autoconfiguration. IPv6 has strong mobile device support, and has attractive features for ISPs or Telecom companies, such as QoS and security. IPv6 is expected to gradually replace IPv4, with the two coexisting for a number of years during a transition period.

Note: Oracle Tuxedo 11*g* Release 1 (11.1.1.0) only supports IPv6 basic functionality in this release. Advanced IPv6 features (for example, QoS and flow control) are not supported.

Enabling IPv6

A Tuxedo process can only supports one IP version at the same time. In order to switch between IPv4 and IPv6, you must use the `TMUSEIPV6` environment variable. For more information, see

tuxenv(5) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference* in the Tuxedo 11*g* Release 1 (11.1.1.0) Reference Guide.

The default value is `n|N` (IPv4). If `TMUSEIPV6` is set to `y|Y` IPv6 is used as the network protocol.

`TMUSEIPV6` can be set in the `*MACHINES`, `*GROUPS`, `*SERVERS` sections in the UBBCONFIG file, or you can set it before booting Tuxedo.

IPv6 Address Format

The following are valid IPv6 formats:

- `fe80:0:0:0:202:55ff:fecf:50b`
- `fe80::202:55ff:fecf:50b`

Tuxedo support two formats of V6 address:

```
//[IPv6 address]:port
//hostname:port
```

The IPv6 address in the URL is enclosed by square brackets. For hostname, it does not need to be enclosed by square brackets. For example: `//[fe80::202:55ff:fecf:50b]:9010` or `//bjaix5:9010`

You can use`[::]` or `[0:0:0:0:0:0:0:0]` as IPv6 wildcard addresses. For example:

For a server booted on bjaix5 (a dual stack machine), the wildcard address can be `//[::]:60120`

or `//[0:0:0:0:0:0:0:0]:60120`

The server listens on `60120` on all bjaix5 interfaces (`172.22.34.45` and `fe80::202:55ff:fecf:50b`). It can accept IPv6 and IPv4 protocol.

Tuxedo Component IPv6 Support

Following Tuxedo components support IPv6:

- BRIDGE & BSBRIDGE
- `tlisten`
- GWTDOMAIN
- WSL/WSH
- WS

- CERT-C
- Jolt
- ISL/ISH
- CORBA client
- SNMP
- SALT
- CORBA & ATMI SSL LDAP

Notes: Tuxedo invokes database XA call back to operate with database. For XA IPv6 depends on the database vendor support.

WEBGUI does not support IPv6

IPv4 and IPv6 Interoperability

Tuxedo supports the following TCP/IP address formats:

- IPv4 only
- IPv6 only
- IPv4 and IPv6 mixed environment

Note: Windows 2000, 20003, and XP platforms do not support dual stack.

Table 7-1 summarizes IPv4 and IPv6 interoperability.

Table 7-1 IPv4 and IPv6 Interoperability

	IPv4 Server IPv4 Host Only	IPv6 Server IPv6 Host Only	IPv4 Server Dual Host Stack	IPv6 Server Dual Host Stack
IPv4 client, IPv4-only host	IPv4	No	IPv4	IPv4(1)
IPv6 client, IPv6-only host	No	IPv6	No	IPv6

Table 7-1 IPv4 and IPv6 Interoperability

IPv4 client, dual-stack host	IPv4	No	IPv4	IPv4(1)
IPv6 client, dual-stack host	IPv4	IPv6	IPv4(2)	IPv6

1. On Linux and UNIX platforms, the server must listen using the IPv6 wildcard address (::).
2. IPv6 client can connect to an IPv4 server on Dual-stack host with textual V4 IP address only (for example, //10.130.5.144:10002).

Oracle Tuxedo MP Mode Interoperability

If a master uses IPv6 and NADDR & NLSADDR are configured as `//[IPv6 address]:port`, all slave nodes *must* use IPv6 as well. Slave nodes using IPv4 cannot start.

If master is using IPv4, all slave nodes must use IPv4 as well. Slave nodes using IPv6 cannot start.

Note: Oracle Tuxedo MP mode cannot be configured using a wildcard address (`[::]`) in UBBCONFIG. If you use a wildcard address in MP mode, `tmloadcf` fails and an ERROR message is sent to ULOG.

CHAPTER 8

Managing The Oracle Tuxedo Service Metadata Repository

This topic includes the following sections:

Oracle Tuxedo Service Metadata Repository

The Oracle Tuxedo service metadata repository contains Oracle Tuxedo service definitions that allow Oracle Tuxedo clients to access Oracle Tuxedo service parameter information. It provides Oracle Tuxedo application developers and administrators the ability to store and retrieve detailed service parameter information on any or all Oracle Tuxedo application services.

The Oracle 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 Oracle Tuxedo service metadata repository

- `TMMETADATA(5)`: Oracle 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 Oracle 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 Oracle Tuxedo service metadata repository.
- `tpsetrepos(3c)`: programmatically uses FML32 buffers to add, delete, or update service parameter information to the metadata repository file.

MIB(5) Similarities and Differences

Programmatic access to the Oracle Tuxedo System Metadata Repository is accomplished through the use of a FML32 buffer format that is very similar to the Oracle Tuxedo MIB format. However, there are also some distinct difference as noted in Table 8-1:

Table 8-1 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 Oracle Tuxedo Service Metadata Repository

The metadata repository file contains all the service parameter information that is accessed in the Oracle 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 Oracle 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 Oracle Tuxedo service metadata repository service-level keywords are as follows in Table 8-2:

Table 8-2 Service-Level Keyword, Abbreviations, and Values

Service-Level Keyword	Keyword Abbreviation	Value
`service`	sv	Any Oracle Tuxedo service name Note: This key valued can only be once per Metadata Repository instance. It cannot be duplicated within the same Metadata Repository.
`tuxservice`	tsv	Actual Oracle Tuxedo service name Note: The difference between the `service` and `tuxservice` keywords is: • `service` represents the service entry stored in the Metadata Repository. • `tuxservice` represents the actual Oracle Tuxedo service name. Two or more `service` definitions can have the same value as `tuxservice`. When used together, these two keywords make it possible to have multiple service definitions for one Oracle Tuxedo service. By default, `tuxservice` has the same value as `service`
`servicetype`	st	Service invocation type. Legal values are: `request` - `response` - the service is a synchronous `oneway` - the service will not send response to the client `queue` - the service is a /Q related application `conv` - the service is conversional
`SNAISC`	ISC	Enables outbound Tuxedo service requests to map to APPC transaction programs or CICS programs. It is only valid when the value of servicemode is "sna". Its valid value list is: APPC,ATI, DPL, DTP. The default value APPC indicates the remote service is a transaction program that may or may not be running under CICS. The DPL value indicates the remote service maps to a program running under CICS.

Table 8-2 Service-Level Keyword, Abbreviations, and Values (Continued)

Service-Level Keyword	Keyword Abbreviation	Value
`servicemode`	sm	Type of service origination(Optional). Legal values are: `tuxedo` - the service is an Oracle Tuxedo originated service `webservice` - proxy service converted from external Web Service Interface `sna` - import service for SNA gateway. For export service, `tuxedo` is the default value. If not specified, `tuxedo` is the default value. **Note:** Do not specify `webservice` for any Oracle Tuxedo service, `webservice` is reserved for SALT proxy service only.
`export`	`ex`	Y (default) or N. This keyword is used to determine service availability to the Oracle Jolt client. In the Oracle Tuxedo repository, this keyword does not have any meaning, but is nevertheless accepted to maintain compatibility with existing Oracle Jolt bulk loader files. **Note:** If `export` is set to N, the service will not be exported for C pseudo-code or text format.
`inbuf`	`bt`	Oracle Tuxedo service request (input) buffer type. Select one of the following type values (case sensitive): `FML`, `FML32`, `VIEW`, `VIEW32`, `STRING`, `CARRAY`, `XML`, `X_OCTET`, `X_COMMON`, `X_C_TYPE`, `MBSTRING` or other arbitrary string representing an application defined custom buffer type. **Note:** The `"inbuf"` value of each service definition cannot be NULL.

Table 8-2 Service-Level Keyword, Abbreviations, and Values (Continued)

Service-Level Keyword	Keyword Abbreviation	Value
outbuf	BT	Oracle Tuxedo service response (output) buffer type with TPSUCCESS. Select one of the following type values (case sensitive): FML, FML32, VIEW, VIEW32, STRING, CARRAY, XML, X_OCTET, X_COMMON, X_C_TYPE, MBSTRING or other arbitrary string representing an application defined custom buffer type. **Note:** The "outbuf" value of each service definition cannot be NULL for a "service" typed service or a "queue" typed service.
errbuf	ebt	Oracle Tuxedo service response (error) buffer type with TPFAIL. Select one of the following type values (case sensitive): FML, FML32, VIEW, VIEW32, STRING, CARRAY, XML, X_OCTET, X_COMMON, X_C_TYPE, MBSTRING or other arbitrary string representing an application defined custom buffer type.
inview	vn	View name for input buffer(Optional) **Note:** This keyword is mandatory only if one of the following buffer types is used: VIEW, VIEW32, X_COMMON, X_C_TYPE, FML and FML32.
outview	VN	View name for output buffer (Optional) **Note:** This keyword is mandatory only if one of the following buffer types is used: VIEW, VIEW32, X_COMMON, X_C_TYPE, FML and FML32.
errview	evn	View name for error buffer (Optional) **Note:** This keyword is mandatory only if one of the following buffer types is used: VIEW, VIEW32, X_COMMON, X_C_TYPE.

Table 8-2 Service-Level Keyword, Abbreviations, and Values (Continued)

Service-Level Keyword	Keyword Abbreviation	Value
inbufschema	isc	Customized message schema association for input buffer (Optional). Value format is: XSD_E:<element_local_name>@namespaceURI For example, XSD_E:Book@http://example.org represents the input buffer is associated with a XML element <Book> defined in the XML namespace "http://example.org". **Note:** This keyword is introduced for supporting Oracle SALT extensible message mapping and conversion feature. For more information about SALT message conversion, see Data Type Mapping and Message Conversion in *Oracle SALT Programming Web Services*.
outbufschema	osc	Customized message schema association for output buffer (Optional). Value format is: XSD_E:<element_local_name>@namespaceURI
errbufschema	esc	Customized message schema association for error buffer (Optional). Value format is: XSD_E:<element_local_name>@namespaceURI
svcdescription	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 8-2 Service-Level Keyword, Abbreviations, and Values (Continued)

Service-Level Keyword	Keyword Abbreviation	Value
rcvqueue	RQN	Receive queue name. Optional for a "queue" typed service.
version	vs	This parameter is exclusive to the Oracle Tuxedo service metadata repository and accommodates any string value used by the application. Oracle Tuxedo does not interpret this parameter.
attributes	att	This parameter is exclusive to the Oracle Tuxedo service metadata repository and accommodates any string value used by the application. Oracle Tuxedo does not interpret this parameter.
fieldtbls	ftb	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 fieldtbls 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 Oracle Tuxedo service metadata repository parameter-level keywords are as follows in Table 8-3:

Table 8-3 Parameter-Level Keyword, Abbreviations, and Values

Parameter-Level Keyword	Metadata Repository Abbreviation	Value
`param`	`pn`	Any parameter name
`type`	`pt`	`byte`, `short`, `integer`, `float`, `double`, `string`, `carray`, `dec_t`, `xml`, `ptr`, `fml32`, `view32`, `mbstring`. **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.
`subtype`	`pst`	A view name for a view32 typed parameter

Table 8-3 Parameter-Level Keyword, Abbreviations, and Values (Continued)

Parameter-Level Keyword	Metadata Repository Abbreviation	Value
access	pa	in, out, err, inout, inerr, outerr, inouterr, noaccess. in - indicates a parameter that is used for input only. out - indicates a parameter that is used for output only. err - indicates a parameter that is used for error output only. inout - indicates a parameter that is used for both input and output. inerr - indicates a parameter that is used for both input and error output. outerr - indicates a parameter that is used for both output and error output. inouterr - indicates a parameter that is used for input, output and error output. noacesss - 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. The set of parameters expected on input is those specified with in, inout, inerr, inouterr, or noaccess access The set of parameters returned on output is those specified with out, inout, outerr, or inouterr access. The set of parameters returned on error output is those specified with err, inerr, outerr, or inouterr access.
count	po	Maximum number of occurrences (default is 1). The value for unlimited occurrences is 0. The value range is [0, 32767]. In the Oracle Tuxedo repository, this parameter is stored for display and is also used by tmunloadrepos(1)pseudocode generation options.
paramdescription	pd	Any string value. A new-line break can be used to improve readability if the string is too long.

Table 8-3 Parameter-Level Keyword, Abbreviations, and Values (Continued)

Parameter-Level Keyword	Metadata Repository Abbreviation	Value
`size`	`pl`	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. The following parameter types expect this value: `carray`, `string`, `xml`, `mbstring`.
`requiredcount`	`ro`	Minimum number of times that the parameter must be specified.The value range is `[0, 32767]`.
`fieldindex`	`fi`	The occurrence number of the field. This applies to FML/FML32 only. It only supports Jolt client.
`fieldname`	`fn`	FML's field name. This applies to FML/FML32 only. It only supports Jolt client. **Note:** For FML/FML32, you should use this "`fieldname`" to specify the field name instead of another parameter-level keyword "`param`".
`fldnum`	`fno`	This optional parameter indicates the field number of the parameter if it is a `FML/FML32` `field`. **Note:** It is not recommended that you use this information if the `fieldtbl` files have already been defined by indicating field table directories using environment `FLDTBLDIR(32)` and indicating field table files using environment `FIELDTBLS(32)`or `fieldtbl` service-level keyword. **Note:** If you configured the `fldnum` field, you will receive the responding `fldid` according to the `fldnum` value instead of the `param` value.
`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)`

Table 8-3 Parameter-Level Keyword, Abbreviations, and Values (Continued)

Parameter-Level Keyword	Metadata Repository Abbreviation	Value
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.
paramschema	psc	This parameter is optional to save the XML Schema information for the decomposed FML32 field. **Note:** This parameter keyword is introduced especially for "servicemode=webservice" typed service definition, i.e. SALT proxy service for outbound call. The parameter value is generated by Oracle SALT wsdlcvt utility from converting an external WSDL file. Do not manually specify or modify this keyword value.
primetype	pxt	This parameter is optional to save the original XML primitive data type for the decomposed FML32 field. **Note:** This parameter keyword is introduced especially for "servicemode=webservice" typed service definition, i.e. SALT proxy service for outbound call. The parameter value is generated by Oracle SALT wsdlcvt utility from converting an external WSDL file. Do not manually specify or modify this keyword value.
isarray	arr	This parameter is optional for SALT RESTful service. If it set to "Y", SALT REST/JSON service maps Tuxedo buffer to JSON array type - even if there is only one occurrence of a field. Currently, only FML/FML32 buffer types are supported.
inheader		Retrieved from the SOAP header portion of the SOAP envelope message received. Message can be a request (native Tuxedo service) or reply (external web service call).

Table 8-3 Parameter-Level Keyword, Abbreviations, and Values (Continued)

Parameter-Level Keyword	Metadata Repository Abbreviation	Value
outheader		Added to the SOAP header portion of the SOAP envelope message sent. Message can be a reply (native Tuxedo service) or request (external web service call).
inoutheader		Combination of inheader and outheader.This parameter is both added to and retrieved from the SOAP header portion of the SOAP message.
(		Indicates the beginning of the description of the parameters contained in an embedded FML32 or VIEW32 buffer field. 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. A closing right parenthesis ')' ends the embedded parameter description.
)		Ends an embedded FML32 or VIEW32 parameter definition of that began with an opening matching left parenthesis '('. 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. In addition, the maximum embedded level depends on the upper limit of embedded FML32 nesting level (18 at present).

Parameter Occurrences

As a generally applied Oracle 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 8-4 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
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 8-5 Service Buffer Type (SMALL CAPS)/Service Parameter Type (lower case) Matching Table II

	bool	Unsigned char	Signed char	Wchar_t	Unsigned int	Unsigned long	Long long	Unsigned long long	Long double
VIEW	X	X	X	X	X	X	X	X	X
VIEW32	X	X	X	X	X	X	X	X	X

Table 8-6 Service Buffer Type (SMALL CAPS)/Service Parameter Type (lower case) Matching Table III

	carray	dec_t	xml	ptr	fml32	view32	mbstring
CARRAY	X						
FML	X						
FML32	X			X	X	X	X
STRING							
VIEW	X	X					
VIEW32	X	X				X	X
X_COMMON							
X_C_TYPE							
XML			X				
X_OCTET	X						
MBSTRING	X						X

Configuring The Oracle Tuxedo Service Metadata Repository Server

To configure the Oracle 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 Oracle Tuxedo metadata server is running, the .TMMETAREPOS service is automatically activated. .TMMETAREPOS is an Oracle 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 Oracle Tuxedo Service Metadata Repository Servers

Setting up multiple TMMETADATA servers on a particular Oracle 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 Oracle Tuxedo Service Metadata Repository File

The Oracle 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 Oracle Tuxedo service metadata repository access.

  `tpgetrepos(3c)` and `tpsetrepos(3c)` can access the Oracle 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)

CHAPTER 9

Managing CORBA Interface Repositories

This topic, which is specific to Oracle Tuxedo CORBA environments, includes the following sections:

Note: The Oracle Tuxedo CORBA Java client and Oracle Tuxedo CORBA Java client ORB were deprecated in Tuxedo 8.1 and are no longer supported in Tuxedo 9.x. All Oracle Tuxedo CORBA Java client and Oracle 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. Oracle Tuxedo does not provide any technical support or documentation for third party CORBA Java ORBs.

An Interface Repository contains the interface descriptions of the CORBA objects that are implemented within the Oracle Tuxedo domain. Administration of the Interface Repository is done using tools specific to Oracle 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 then one Interface Repository server, you must decide whether to have replicated databases, shared databases, or a combination of the two. There are advantages and disadvantages to each configuration. Replicated Interface Repository databases allow for local file access that can potentially increase performance.

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

Using Administration Commands to Manage Interface Repositories

Use the following commands to manage the Interface Respository for an Oracle 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 Respository

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

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

For a detailed description of this command, see the *File Formats, Data Descriptions, MIBs, and System Processes Reference* in the Oracle 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 Oracle 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 Oracle 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 Oracle Tuxedo CORBA applications is critical. An Oracle 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 3-61 in Chapter 3, "Creating the Configuration File."

Notice that the `TMIFRSVR` Interface Repository server is the fifth server started.

```
*SERVERS

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

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

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

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

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

For a description of the `TMIFRSVR -f filename` parameter, refer to the *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 Oracle 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 3-54 in Chapter 3, "Creating the Configuration File," for details about parameters such as `SRVGRP`, `SRVID`, `RESTART`, `MAXGEN`, and `GRACE`.

CHAPTER 10

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 Oracle 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

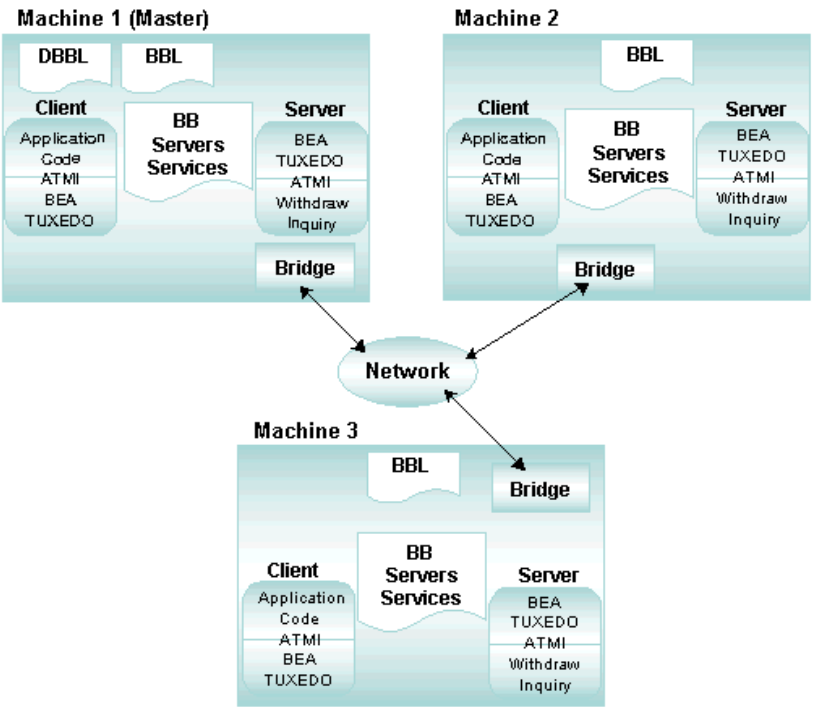
State of the art telecommunications and data networks are making distributed operations of this sort increasingly common. Applications developed to implement this type of strategy allow businesses to reduce costs and enhance their offerings of services to customers around the world.

The Oracle 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 Oracle Tuxedo configuration file.

Example of a Distributed Application

Figure 10-1 illustrates the basic parts of an application distributed across three machines.

Figure 10-1 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 11-1
- "Setting Up the Network for a Distributed Application" on page 12-1
- "Managing the Network in a Distributed Application" in *Administering an Oracle 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 Oracle 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 Oracle 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 Oracle Tuxedo ATMI*
- "What Is Data-Dependent Routing?" in *Introducing Oracle Tuxedo ATMI*
- *Scaling, Distributing, and Tuning CORBA Applications*

CHAPTER 11

Creating the Configuration File for a Distributed ATMI Application

This section includes the following topics:

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

Configuration File Requirements for a Distributed Oracle Tuxedo ATMI Application

A distributed Oracle 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 Oracle 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 OracleTuxedo application, see UBBCONFIG(5) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*.

Table 11-1 shows the resources section parameters.

Table 11-1 RESOURCES Section Parameters

Parameter	Description
BBLQUERY (Optional)	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. 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.
BLOCKTIME (Optional)	BLOCKTIME sets a multiplier of the basic SCANUNIT after which a blocking call (for example, receiving a reply) times out. The value of BLOCKTIME must be greater than 0. If this parameter is not specified, the default is set so that (SCANUNIT * BLOCKTIME) is approximately 60 seconds.
DBBLWAIT (Optional)	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. 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.

Table 11-1 RESOURCES Section Parameters (Continued)

Parameter	Description
IPCKEY (Required)	IPCKEY specifies the numeric key for the bulletin board. In a single-processor environment, this key *names* the bulletin board. In a multiprocessor environment, this key *names* 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. The value of IPCKEY must be greater than 32,768 and less than 262,143.
MASTER (Required)	MASTER (*string_value1*[,*string_value2*]) 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. *string_value2* 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. The value of both *string_value1* and *string_value2* must be LMIDs of machines defined in the MACHINES section. Each string may contain up to 30 characters.
MAXGROUPS (Optional)	MAXGROUPS specifies the maximum number of configured server groups to be accommodated in the group table of the bulletin board. The value of MAXGROUPS must be greater than or equal to 100 and less than 32,768. The default is 100.
MAXSERVERS (Optional)	MAXSERVERS specifies the maximum number of servers to be accommodated in the server table of the bulletin board. The value of MAXSERVERS must be greater than 0 and less than 8192. The default is 50.
MAXSERVICES (Optional)	MAXSERVICES specifies the maximum number of services to be accommodated in the services table of the bulletin board. The value of MAXSERVICES must be greater than 0 and less than 1,048,575. The default is 100.

Table 11-1 RESOURCES Section Parameters (Continued)

Parameter	Description
SANITYSCAN (Optional)	SANITYSCAN sets a multiplier of the basic SCANUNIT between sanity checks of the system. The value of SCANUNIT must be greater than 0. The default is set so that (SCANUNIT * SANITYSCAN) is approximately 120 seconds. Sanity checks are performed on servers as well as on the bulletin board data structure itself.
SCANUNIT (Optional)	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 tpbegin(3c) 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. 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.

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. Table 11-2 describes the parameters available for defining machine names and other machine-specific parameters for each machine that participates in a distributed application.

Table 11-2 MACHINES Section Parameters

Parameter	Description
ENVFILE (Optional)	ENVFILE specifies a file that defines the environment with which all clients and servers on the machine are to be executed. Lines must be in the form *ident=value* where *ident* contains only underscores and/or alphanumeric characters, and begins with an underscore or a letter of the alphabet. If the value of ENVFILE is an invalid filename, no values are added to the environment.
MAXACCESSERS (Optional)	MAXACCESSERS specifies the maximum number of processes that can access the bulletin board on this processor at any one time. When calculating the appropriate number, you are not required to count system administration processes, such as the BBL and tmadmin, but you must count all application servers and clients, and TMS servers. The value of MAXACCESSERS must be greater than 0 and less than 32,768. The default is the value specified in the RESOURCES section.
MAXCONV (Optional)	MAXCONV specifies the maximum number of simultaneous conversations allowed for processes on a particular machine. 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.
MAXWSCLIENTS (Optional)	MAXWSCLIENTS specifies the number of accesser entries on this processor to be reserved for Workstation clients only. This parameter is used only when the Oracle 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 an Oracle Tuxedo system-supplied surrogate, the workstation handler. 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.

Creating the GROUPS Section

In the GROUPS section you identify each server group in your application so that the Oracle 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.

Table 11-3 describes the parameters in the GROUPS section.

Table 11-3 GROUPS Section Parameters

Parameter	Description
ENVFILE	ENVFILE specifies a file that defines the environment with which all servers in the group are executed. Lines must be in the form *ident=value* where *ident* contains only underscores and/or alphanumeric characters. If the value of ENVFILE is an invalid filename, no values are added to the environment.
GRPNO (Required)	GRPNO associates a number with a particular server group. The number must be greater than 0 and less than 30,000. It must be unique among entries in the GROUPS section.
LMID (Required)	LMID identifies the machine on which the server group being defined runs. A second LMID value can be specified (separated from the first by a comma) for an alternate machine to which this server group can be migrated if the MIGRATE option has been specified. Servers in the group can be migrated if RESTART=Y to migrate is specified in the GROUPS section. The values of LMID must be the values assigned to the LMID parameter in the MACHINES section.

Creating the SERVICES Section

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

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

Table 11-4 describes the parameters in the SERVICES section that are available for defining distributed applications.

Table 11-4 SERVICES Section Parameters

Parameter	Description
LOAD (Optional)	LOAD specifies the size of the load imposed by SVCNM on the system. The value of LOAD must be a number between 1 and 32,767, inclusive. A higher number indicates a greater load. The default is 50.
PRIO (Optional)	PRIO specifies the dequeuing priority of SVCNM. The value of PRIO must be greater than 0 and less than or equal to 100, with 100 being the highest priority. The default is 50.
ROUTING (optional)	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. The value of ROUTING may contain up to 127 characters. If multiple entries exist for the same service name but with different SRVGRP parameters, the ROUTING parameter must be the same for all entries.

Table 11-4 SERVICES Section Parameters

Parameter	Description

SRVGRP (Optional)	SRVGRP specifies the host server group for the service that is specified by SVCNM and controlled by the parameters set in this section.
	
	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:
	`WITHDRAW  ROUTING=123 LOAD=60 SRVGRP=GROUP1`
	`WITHDRAW  ROUTING=123 LOAD=60 SRVGRP=GROUP2`
	
	The value of SRVGRP may contain up to 30 characters.

SVCTIMEOUT (Optional)	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. 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.

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.

Table 11-5 describes the parameters in the ROUTING section.

Table 11-5 ROUTING Section Parameters

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

See Also

- "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 an Oracle 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 Oracle 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 Oracle 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.

Parameter	Description
FIELD (Optional)	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. If a field in an FML32 buffer is used for routing, it must have a field number less than or equal to 8191.
RANGES (Optional)	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 *range/RACCESSPOINT* pairs. The value of *range* may be either a single value (a signed numeric value or a character string enclosed in single quotes), or a range of the form *lower*-*upper* (where *lower* and *upper* are both signed numeric values or character strings in single quotes). The value of *lower* must be less than or equal to *upper*. A single quote embedded in a character string value, as in "O'Brien," for example, must be preceded by two back slashes: "O\\'Brien". Use MIN to indicate the minimum value for the data type of the associated FIELD. For strings and carrays, it is the null string; for character fields, it is 0; for numeric values, it is the minimum numeric value that can be stored in the field. Use MAX to indicate the maximum value for the data type of the associated FIELD. For strings and carrays, it is effectively an unlimited string of octal-255 characters; for a character field, it is a single octal-255 character; for numeric values, it is the maximum numeric value that can be stored in the field. Thus, MIN - -5 is all numbers less than or equal to -5, and 6 - MAX is all numbers greater than or equal to 6. The metacharacter * (wildcard) in the position of a range indicates any values not covered by 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).

Parameter	Description
BUFTYPE (Optional)	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. If multiple buffer types are specified for a single routing entry, the data types of the routing field for each buffer type must be the same. If the field value is not set (for FML buffers), or does not match any specific range, and a wildcard range has not been specified, an error is returned to the application process that requested the execution of the remote service.

Routing Field Description

The 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 an Oracle Tuxedo domain. The fourth branch belongs to another TP domain, and OSI-TP is used to communicate with that domain.

Listing 11-1 shows the Oracle 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 11-1 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]
#
tlr_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" in *Using the Oracle Tuxedo Domains Component*
- "Setting Up a Domains Configuration" in *Using the Oracle Tuxedo Domains Component*
- *Scaling, Distributing, and Tuning CORBA Applications*

CHAPTER 12

Setting Up the Network for a Distributed Application

This topic includes the following sections:

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 Oracle 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 Oracle 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 of Table 12-1.

Table 12-1 Configuring the Network for a Distributed Application

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.

Table 12-1 Configuring the Network for a Distributed Application

In This Section . . .	Set This Parameter . . .	To . . .

MACHINES	TYPE=*string* (Optional)	Determine whether encoding is required when messages are exchanged by two machines. The TYPE 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.
		If the machines in question both use the same type of data representation, however, the system skips the encoding/decoding process.
		
		Example 1
		`LMID_1 TYPE = “abc”`
		`LMID_2 TYPE = “abc”`
		Encoding is not used in this case.
		
		Example 2
		`LMID_1 TYPE = “HP”`
		`LMID_2 TYPE = “SUN”`
		Encoding is used in this case.
		
		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 TYPE=s*tring* only for the HP. For the SPARC machines, the default null string identifies them as the same type.

Table 12-1 Configuring the Network for a Distributed Application

In This Section . . .	Set This Parameter . . .	To . . .
	CMPLIMIT=*remote* [, *local*] (Optional)	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 *remote* and *local* is a number between 0 and MAXLONG. If CMPLIMIT is set to only one value, it is assumed that the specified value is the *remote* argument and that messages sent to local destinations are never compressed. For example, if you set CMPLIMIT=1024, than any message greater than 1024 bytes bound for a remote location is compressed. Compression thresholds can also be specified with the variable TMCMPLIMIT. See the discussion, in tuxenv(5), about the variable TMCMPPRFM, which sets the degree of compression in a range of 1 to 9.
	NETLOAD=*number* (Optional)	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.
NETGROUPS (Optional)	NETGROUP (Required)	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.
	NETGRPNO=*number* (Required)	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.
	NETPRIO=*number* (Optional)	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.

Table 12-1 Configuring the Network for a Distributed Application

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=*string* (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=*string* (Required)	Specify the network address for the tlisten process on this LMID. Valid formats are the same as the valid formats for NADDR.
	NETGROUP=*string* (Optional)	Specify a NETWORK group name. The value of *string* 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.

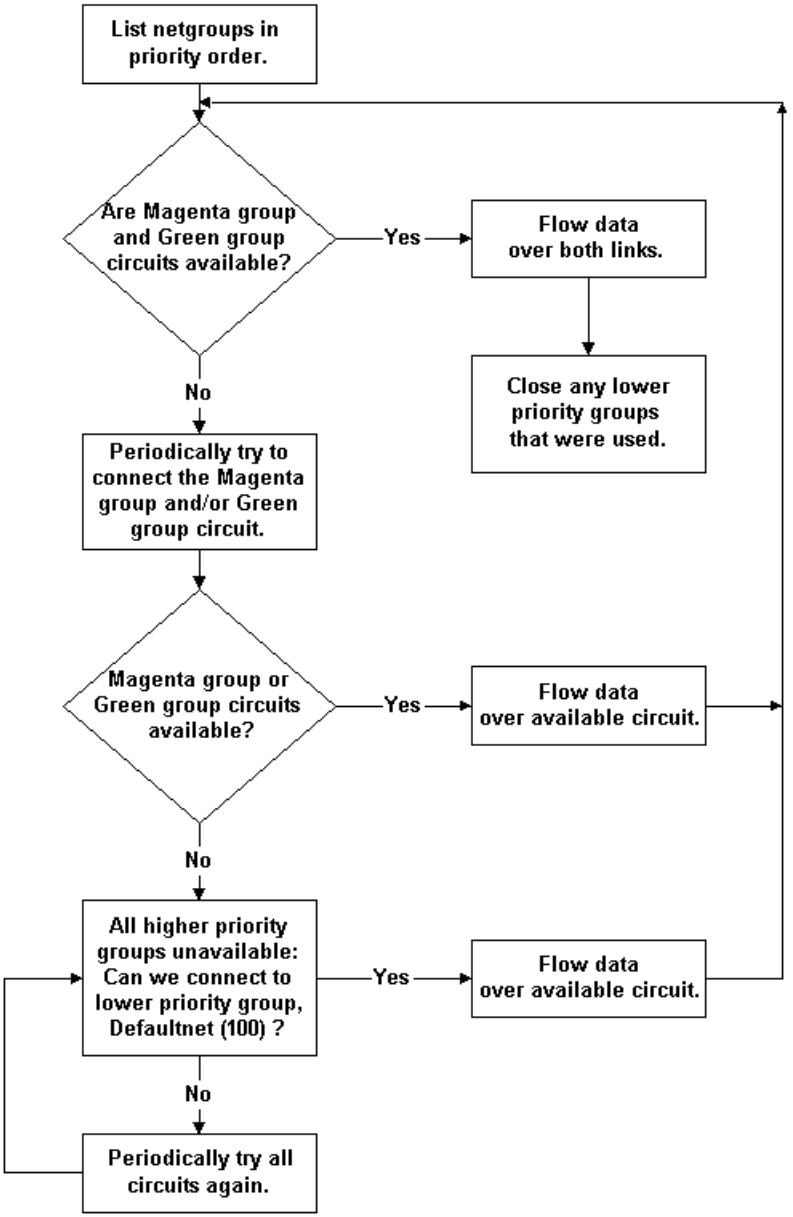
Before making a decision to use parallel data circuits, however, you should determine whether it will be important, in your application, for messages to be kept in sequence. The system guarantees that conversational messages are kept in the correct sequence by binding the conversation connection to one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

Figure 12-1 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 12-1 Flow of Data over the BRIDGE


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

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="//mach1:51669″
           NLSADDR="//mach1:31669"
#
   SITE2   NADDR="//mach386:51669"
           NLSADDR="//mach386:31669"
```

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).

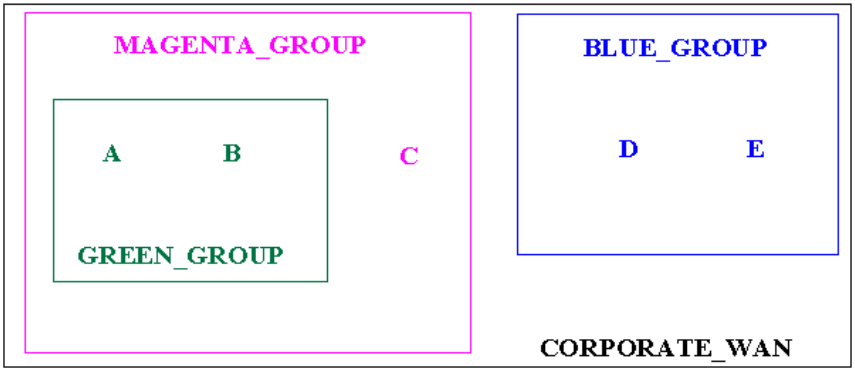
In the following example, it is assumed that in addresses written in the form `//A_CORPORATE:5345`, the string `A_CORPORATE` is specified in the `/etc/hosts` file or in DNS.

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`. Figure 12-2 illustrates group assignments for the network.

Figure 12-2 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

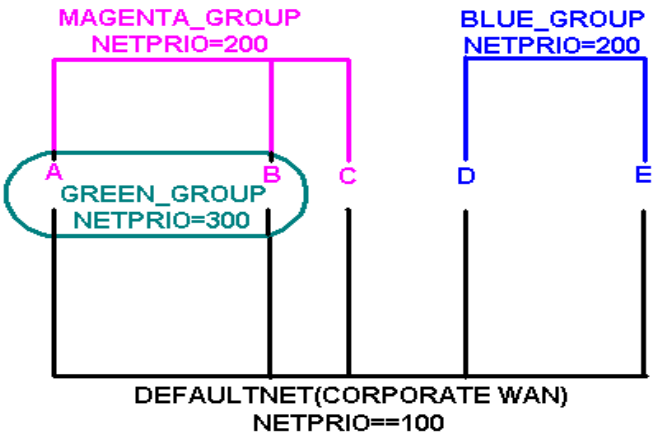
Assigning priorities appropriately for each NETGROUP enables you to maximize the capability of network BRIDGE processes. When determining NETGROUP priorities, keep in mind the following considerations:

- 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

Figure 12-3 shows how the First State Bank administrator assigns priorities to the available network groups.

Figure 12-3 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"
```

Using Oracle Tuxedo Distributed Caching (TDC) with Oracle Coherence

This topic contains the following sections:

For a quick start, see the following samples.

For more information about the Oracle Tuxedo ATMI APIs that relates to this feature, see Oracle Tuxedo Distributed Caching (TDC) Related ATMI APIs.

Overview

This feature leverages Oracle Coherence, and Oracle Tuxedo java server, which is working as a client of Oracle Coherence, assuring you taking all advantages that Oracle Coherence has for caching.

This feature supports the following caching strategies.

- Data Caching for Clients and Servers
- Result Caching for Oracle Tuxedo Services

As Oracle Coherence and TMJAVASVR are used, it requires you to configure them both before actually using TDC.

- Configuring Oracle Coherence
- Configuring Oracle Tuxedo Java Server

After these configurations, you can use Oracle Tuxedo Distributed Caching.

- Using Data Caching for Clients and Servers
- Using Result Caching for Oracle Tuxedo Services

Data Caching for Clients and Servers

When you enable data caching, you can store data in cache, and clients on other servers can retrieve the data from the cache. This offers you a new way of sharing data between clients and servers, especially sharing data for or from other servers.

This feature provides you

- High performance

  Unlike storing and retrieving data through queues, files or databases which usually means occupying a large amount of sources and wasting time, storing and retrieving through cache is faster and lighter.

  Oracle Tuxedo also adopts other ways to achieve high performance, such as minimizing local buffer copies (ideally providing some zero copy use cases) and focusing on primarily read intensive operations (for example, make 2:1 read/write ratio or higher).

- Various buffer types support

  This feature supports many Oracle Tuxedo buffer types so that you do not need to manage to match your data with the data types that Oracle Tuxedo supports. The supported Tuxedo buffer types are CARRAY, FML, FML32, MBSTRING, STRING, VIEW, VIEW32, XML, RECORD, X_C_TYPE, and X_COMMON.

- Easy approach

This feature can be transparent for you so you can use it without making any code changes, providing you an easy approach. Oracle Tuxedo encapsulates Oracle Coherence functions in java server container so that you can just use java server to implement all functions that pertain to this feature.

Result Caching for Oracle Tuxedo Services

When you enable result caching, Oracle Tuxedo first reaches the result from the cache entry instead of reaching the backend service; if it fails to reach it or if the cache or cache entry is expired, Oracle Tuxedo reaches the backend service and the result is stored in the cache for Oracle Tuxedo to reach it next time.

This feature provides you

- High performance

  Synchronous services that return results that do not change often are good candidates to have their results cached by Oracle Tuxedo. This can improve performance by reducing network overhead to access the backend service.

  Oracle Tuxedo also adopts other ways to achieve high performance, such as minimizing local buffer copies (ideally providing some zero copy use cases) and focusing on primarily read intensive operations (for example, make 2:1 read/write ratio or higher).

- High availability and scalability

  By integrating with Oracle Coherence, Oracle Tuxedo takes advantage of its specialized scalable protocol and its creation of a cluster. A cluster can be seamlessly expanded to add more memory, processing power or both, and can avoid single point of failure as it transparently fails over if a cluster member fails. As a result this feature provides you a highly availability and scalability.

  Also, taking advantage of Oracle Coherence, any cache entry can be replicated across two or more machines, and the data processing can be farmed out to where the data is and return results to you. This assures your data to be scalable.

- Various buffer types support

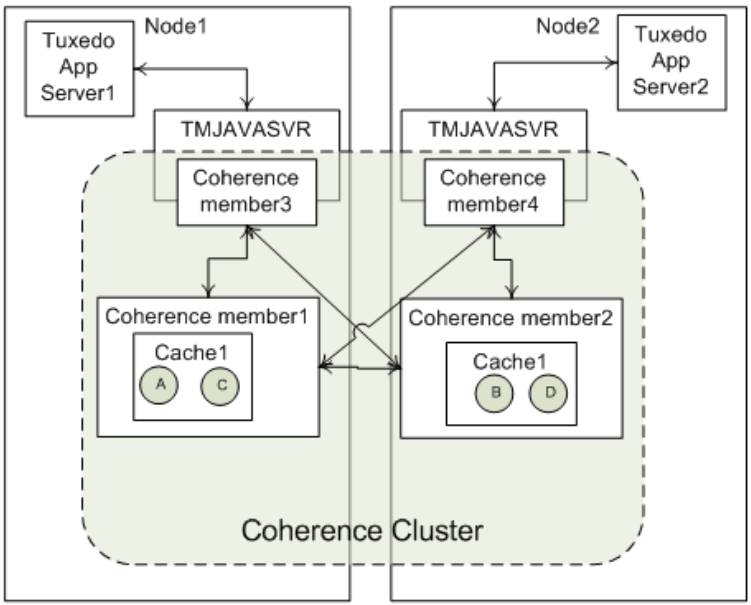
  This feature supports many Oracle Tuxedo buffer types so that you do not need to manage to match your data with the data types that Oracle Tuxedo supports. The supported Tuxedo buffer types are `CARRAY`, `FML`, `FML32`, `MBSTRING`, `STRING`, `VIEW`, `VIEW32`, `XML`, `RECORD`, `X_C_TYPE`, and `X_COMMON`.

- Easy approach

This feature can be transparent for you so you can use it without making any code changes, providing you an easy approach. Oracle Tuxedo encapsulates Oracle Coherence functions in java server container so that you can just use java server to implement all functions that pertain to this feature.

Figure 13-1 illustrates a typical deployment in an Oracle Tuxedo MP domain to use TDC based on Oracle Coherence.

Figure 13-1 Result Caching for Oracle Tuxedo Services



As you can see from this figure, Oracle Tuxedo java server is taken as Oracle Coherence's client (member). You can directly use Oracle Tuxedo java server to cache results from multiple machines without worrying about how cluster members located in different machines communicate with each other.

Configuring Oracle Coherence

Configure the following files just like you configure on Oracle Coherence. See *Oracle Fusion Middleware Developing Applications with Oracle Coherence* for detailed instruction.

- tangosol-coherence-override.xml

- coherence-cache-config.xml

You can deploy the above configuration files into any path as long as this path is in the java class path and prior to where `coherence.jar` is. For example, you can put the configuration files into `${APPDIR}/config` and then start Oracle Coherence server like this:

Listing 13-1 Oracle Coherence Cluster Deployment

```
java -server -showversion $JAVA_OPTS -Dtangosol.coherence.mode=prod -cp
$APPDIR/config: ${COHERENCE_HOME}/lib/coherence.jar
com.tangosol.net.DefaultCacheServer
```

tangosol-coherence-override.xml

Listing 13-2 is an example; note the properties in bold. In this example, `coherence_tux` is the name of the Oracle Coherence cluster whose multicast port number is `51697` and unicast port number is `51687`.

Listing 13-2 tangosol-coherence-override.xml

```
<?xml version='1.0'?>

<coherence  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"


xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"


xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-operational-co
nfig coherence-operational-config.xsd">

  <cluster-config>

    <member-identity>

      <cluster-name
system-property="tangosol.coherence.cluster">coherence_tux</cluster-name>

    </member-identity>

    <unicast-listener>
```

```
      <address
system-property="tangosol.coherence.localhost">localhost</address>
      <port system-property="tangosol.coherence.localport">51687</port>
    </unicast-listener>
    <multicast-listener>
        <port system-property="tangosol.coherence.clusterport">51697</port>
    </multicast-listener>
  </cluster-config>
</coherence>
```

coherence-cache-config.xml

Listing 13-3 is an example; note the properties in bold. In this example, we create an Oracle Coherence cache named `tux_distributed`.

Listing 13-3 coherence-cache-config.xml

```
<?xml version="1.0"?>
<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"

xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
coherence-cache-config.xsd">
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>tux_distributed</cache-name>
      <scheme-name>distributed</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>
```

```
  <caching-schemes>
    <distributed-scheme>
      <scheme-name>distributed</scheme-name>
      <service-name>DistributedCache</service-name>
        <lease-granularity>member</lease-granularity>
      <backing-map-scheme>
        <local-scheme/>
      </backing-map-scheme>
      <autostart>true</autostart>
    </distributed-scheme>
  </caching-schemes>
</cache-config>
```

Configuring Oracle Tuxedo Java Server

- Configuring Oracle Tuxedo Java Server Cofiguration file
- Configure Oracle Tuxedo Distributed Caching (TDC) Property File

Configuring Oracle Tuxedo Java Server Cofiguration file

Oracle Tuxedo Distributed Caching (TDC) introduces a new system-supplied Oracle Tuxedo java server, which converts the caching request to Oracle Coherence and sends back the reply. It works as a client of Oracle Coherence.

The java package used by this java server is `com.oracle.tuxedo.tdcj.jar` located in `${TUXDIR}/udataobj/tuxj/tdc`.

Listing 13-4 is an example of Oracle Tuxedo Java Server Configuration File that enables TDC. For more information, see Java Server Configuration Schema File for version 2.0.

- In general, you should not change the code lines in bold. The class `com.oracle.tuxedo.tdc.TCache4Coherence` is the main class used by TDC which is located in `${TUXDIR}/udataobj/tuxj/tdc/com.oracle.tuxedo.tdcj.jar`.

- It is necessary to change the code lines in Italic due to different environment. Note that <APPDIR> must be replaced with the real path to make the file work.

Listing 13-4 Configuring Oracle Tuxedo Java Server Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<TJSconfig version="2.0">
    <java-config>
       <jvm-options>-XX:MaxPermSize=192m</jvm-options>
       <jvm-options>-server</jvm-options>

<jvm-options>-Dtangosol.coherence.distributed.localstorage=false</jvm-options>
       <jvm-options>-Dtangosol.coherence.mode=prod</jvm-options>
    </java-config>
<tux-config>
       <server-clopt>-f <APPDIR>/config/tdcsvr_coh.conf</server-clopt>
    </tux-config>
    <classpath-config>
    </classpath-config>
    <tux-resources>
    </tux-resources>
    <jdbc-resources>
    </jdbc-resources>
    <tux-server-config>

<classpath>${TUXDIR}/udataobj/tuxj/tdc/com.oracle.tuxedo.tdcj.jar</classpath>
        <classpath>${TUXDIR}/udataobj/tuxj/tdc/dms.jar</classpath>
        <classpath>${TUXDIR}/udataobj/tuxj/tdc/ojdl.jar</classpath>
        <classpath>${APPDIR}/config</classpath>
```

```
        <classpath>${COHERENCE_HOME}/lib/coherence.jar</classpath>
        <server-class name="com.oracle.tuxedo.tdc.TCache4Coherence">
        </server-class>
    </tux-server-config>
</TJSconfig>
```

Configure Oracle Tuxedo Distributed Caching (TDC) Property File

It is required to add a new property file which specifies all properties about the caches for TDC. Oracle Tuxedo java server advertises services using the name of these caches. You can define many Oracle Tuxedo caches in a single property file.

Listing 13-5 shows a template of the property file. You can find this template in `$TUXDIR/udataobj/tuxj/tdc/tdcsvr_coh.conf.template`. In this template, two Oracle Tuxedo cache names are configured: `tc1` and `tc2`. `tc1` uses Oracle Coherence cache `tux_distributed` and `tc2` uses Oracle Coherence cache `tux2_distributed`.

See Oracle Tuxedo Distributed Caching (TDC) Property File Properties for more information about its properties.

Listing 13-5 TDC Property File Template

```
#* global option encoding setting
#options.encoding=no

#* configurations for Tuxedo cache "tc"

#* option encoding setting
#cache.options.encoding.tc=no
#* physical cache used in Oracle Coherence
coh.cache.name.tc=tux_distributed
```

```
#* configurations for Tuxedo cache "tc2"
#
#* option encoding setting
#cache.options.encoding.tc2=no
#* physical cache used in Oracle Coherence
#coh.cache.name.tc2=tux2_distributed
```

Using Data Caching for Clients and Servers

Steps for Using Data Caching for Clients and Servers

- Configure Oracle Coherence
- Start Oracle Coherence Cluster
- Configure Oracle Tuxedo Java Server
- Configure UBBCONFIG
- Put an Oracle Tuxedo buffer associated with a key into an Oracle Tuxedo cache
- Get an Oracle Tuxedo buffer from an Oracle Tuxedo cache according to the key

Configure Oracle Coherence

For how to configure Oracle Coherence, see Configuring Oracle Coherence.

Start Oracle Coherence Cluster

If there is no Oracle Coherence Cluster is running, start your own cluster, making sure you configure the path of the configuration files into the Java Class Path and ahead of where `coherence.jar` is. See Listing 13-1 for an example.

Configure Oracle Tuxedo Java Server

See Configuring Oracle Tuxedo Java Server for instruction.

Configure UBBCONFIG

Configure TMJAVASVR on UBBCONFIG. TMJAVASVR uses multi-threaded configuration to improve performance. Multi-instances configuration is also used to gain higher availability.

Listing 13-6 UBBCONFIG for TMJAVASVR

```
*RESOURCE
...
MODEL          SHM
...
*MACHINES
"m1"           LMID=L1
...
*GROUPS
JGRP1  LMID=L1 GRPNO=10


...
TMJAVASVR    SRVGRP=JGRP1 SRVID=10
             MINDISPATCHTHREADS=4 MAXDISPATCHTHREADS=4 MIN=2 MAX=2
             CLOPT="-- -c /home/scott/tuxedo/dom1/config/tdcsvr_coh.xml"
...
```

Put an Oracle Tuxedo buffer associated with a key into an Oracle Tuxedo cache

On Oracle Tuxedo native client or workstation client, use TDC API tpgetcache and tpcacheput to put an Oracle Tuxedo buffer associated with a key into an Oracle Tuxedo cache. For more information about tpgetcache and tpcacheput, see Oracle Tuxedo Distributed Caching (TDC) Related ATMI APIs.

Listing 13-7 Put an Oracle Tuxedo buffer associated with a key into an Oracle Tuxedo cache

```
...
        TCACHE* mycache = NULL;
        char mykey[128];
        char* databuf = NULL;

        tpinit(NULL);
        databuf = tpalloc("STRING", NULL, 256);

        mycache = tpgetcache("tc");
        strcpy(mykey, "myname");
        strcpy(databuf, "scott");
        tpcacheput(mycache, mykey, databuf, 0, 0L);

        tpfree(databuf);
...
```

Get an Oracle Tuxedo buffer from an Oracle Tuxedo cache according to the key

On Oracle Tuxedo native client or workstation client, use TDC API `tpgetcache` and `tpcacheget` to get an Oracle Tuxedo buffer associated with a key into an Oracle Tuxedo cache. For more information about `tpgetcache` and `tpcacheget`, see Oracle Tuxedo Distributed Caching (TDC) Related ATMI APIs.

Listing 13-8 Get an Oracle Tuxedo buffer from an Oracle Tuxedo cache according to the key

```
...
        TCACHE* mycache = NULL;
```

```
        char mykey[128];
        char* databuf = NULL;

        tpinit(NULL);
        databuf = tpalloc("STRING", NULL, 256);

        mycache = tpgetcache("tc");
        strcpy(mykey, "myname");
        tpcacheget(mycache, mykey, &databuf, NULL, 0L);

        tpfree(databuf);
...
```

Sample: Using Data Caching for Clients and Servers

Suppose `${APPDIR}` is `/home/scott/tuxedo/dom1`.

- Sample: Configure Oracle Coherence
- Sample: Start Oracle Coherence cluster
- Sample: Configure Oracle Tuxedo Java Server
- Sample: Configure TMJAVASVR in UBBCONFIG
- Sample: Put an Oracle Tuxedo buffer associated with a key into an Oracle Tuxedo cache
- Sample: Get an Oracle Tuxedo buffer from an Oracle Tuxedo cache according to the key

Sample: Configure Oracle Coherence

- Prepare `tangosol-coherence-override.xml` in `${APPDIR}/config`. See Listing 13-9.

  Configure Oracle Coherence cluster `coherence_tux` whose multicast port number is `51697` and unicast port number is `51687`.

- Prepare `coherence-cache-config.xml` in `${APPDIR}/config`. See Listing 13-10.

  Configure Oracle Coherence cache `tux_distributed`.

Listing 13-9 Prepare tangosol-coherence-override.xml

```
<?xml version='1.0'?>
<coherence  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"

xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-operational-co
nfig coherence-operational-config.xsd">
  <cluster-config>
    <member-identity>
      <cluster-name
system-property="tangosol.coherence.cluster">coherence_tux</cluster-name>
    </member-identity>
    <unicast-listener>
      <address
system-property="tangosol.coherence.localhost">localhost</address>
      <port system-property="tangosol.coherence.localport">51687</port>
    </unicast-listener>
    <multicast-listener>
        <port system-property="tangosol.coherence.clusterport">51697</port>
    </multicast-listener>
  </cluster-config>
</coherence>
```

Listing 13-10 Prepare coherence-cache-config.xml

```
<?xml version="1.0"?>
<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"

xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config
coherence-cache-config.xsd">
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>tux_distributed</cache-name>
      <scheme-name>distributed</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>

  <caching-schemes>
    <distributed-scheme>
      <scheme-name>distributed</scheme-name>
      <service-name>DistributedCache</service-name>
        <lease-granularity>member</lease-granularity>
      <backing-map-scheme>
        <local-scheme/>
      </backing-map-scheme>
      <autostart>true</autostart>
    </distributed-scheme>
  </caching-schemes>
</cache-config>
```

Sample: Start Oracle Coherence cluster

If no Oracle Coherence cluster is running, you can start your own cluster.

```
java -server -showversion $JAVA_OPTS -Dtangosol.coherence.mode=prod -cp
${APPDIR}/config: ${COHERENCE_HOME}/lib/coherence.jar
com.tangosol.net.DefaultCacheServer
```

Sample: Configure Oracle Tuxedo Java Server

Preparing tdcsvr_coh.xml for Oracle Tuxedo java server

Preparing `tdcsvr_coh.xml` for Oracle Tuxedo java server in `${APPDIR}/config`.

See Listing 13-11, where the `<server-clopt>-f /home/scott/tuxedo/dom1/config/tdcsvr_coh.conf</server-clopt>` property specifies the TDC property file.

Listing 13-11 Configure Oracle Tuxedo Java Server Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<TJSconfig version="2.0">
    <java-config>
        <jvm-options>-XX:MaxPermSize=192m</jvm-options>
        <jvm-options>-server</jvm-options>
<jvm-options>-Dtangosol.coherence.distributed.localstorage=false</jvm-options>
        <jvm-options>-Dtangosol.coherence.mode=prod</jvm-options>
    </java-config>
<tux-config>
       <server-clopt>-f
/home/scott/tuxedo/dom1/config/tdcsvr_coh.conf</server-clopt>
    </tux-config>
    <classpath-config>
    </classpath-config>
    <tux-resources>
```

```
    </tux-resources>
    <jdbc-resources>
    </jdbc-resources>
    <tux-server-config>
<classpath>${TUXDIR}/udataobj/tuxj/tdc/com.oracle.tuxedo.tdcj.jar</classpath>
       <classpath>${TUXDIR}/udataobj/tuxj/tdc/dms.jar</classpath>
       <classpath>${TUXDIR}/udataobj/tuxj/tdc/ojdl.jar</classpath>
       <classpath>${COHERENCE_HOME}/lib/coherence.jar</classpath>
       <classpath>${APPDIR}/config</classpath>
       <server-class name="com.oracle.tuxedo.tdc.TCache4Coherence">
       </server-class>
    </tux-server-config>
</TJSconfig>
```

Prepare tdcsvr_coh.conf for TDC property file

Prepare `tdcsvr_coh.conf` for TDC property file in `${APPDIR}/config`.

See Listing 13-12, where it configures Oracle Tuxedo cache `tc`, which actually uses Oracle Coherence cache `tux_distributed`.

Listing 13-12 Configure Oracle Tuxedo Distributed Caching (TDC) Property File

```
#* global option encoding setting
#options.encoding=no

#* configurations for Tuxedo cache "tc"

#* option encoding setting
```

```
#cache.options.encoding.tc=no
#* physical cache used in Oracle Coherence
coh.cache.name.tc=tux_distributed
```

Sample: Configure TMJAVASVR in UBBCONFIG

Configure TMJAVASVR in UBBCONFIG.

See Listing 13-13, where multi-threaded configuration is enabled and TMJAVASVR configuration file tdcsvr_coh.xml is set.

Listing 13-13 Configure TMJAVASVR in UBBCONFIG

```
*RESOURCES
...
MODEL  SHM
...
*MACHINES
"m1"   LMID=L1
...
*GROUPS
JGRP1  LMID=L1 GRPNO=10

...
TMJAVASVR    SRVGRP=JGRP1 SRVID=10
              MINDISPATCHTHREADS=4 MAXDISPATCHTHREADS=4 MIN=2 MAX=2
              CLOPT="-- -c /home/scott/tuxedo/dom1/config/tdcsvr_coh.xml"
...
```

Sample: Put an Oracle Tuxedo buffer associated with a key into an Oracle Tuxedo cache

Put an Oracle Tuxedo `STRING` buffer associated with a key `mykey` into an Oracle Tuxedo cache `tc`.

Listing 13-14 Put an Oracle Tuxedo buffer

```
...
       TCACHE* mycache = NULL;
       char mykey[128];
       char* databuf = NULL;

       tpinit(NULL);
       databuf = tpalloc("STRING", NULL, 256);

       mycache = tpgetcache("tc");
       strcpy(mykey, "myname");
       strcpy(databuf, "scott");
       tpcacheput(mycache, mykey, databuf, 0, 0L);

       tpfree(databuf);
...
```

Sample: Get an Oracle Tuxedo buffer from an Oracle Tuxedo cache according to the key

Get an Oracle Tuxedo typed buffer from an Oracle Tuxedo cache `tc` according to a key `mykey`.

Listing 13-15 Get an Oracle Tuxedo buffer

```
...
        TCACHE* mycache = NULL;
        char mykey[128];
        char* databuf = NULL;

        tpinit(NULL);
        databuf = tpalloc("STRING", NULL, 256);

        mycache = tpgetcache("tc");
        strcpy(mykey, "myname");
        tpcacheget(mycache, mykey, &databuf, NULL, 0L);

        tpfree(databuf);
...
```

Using Result Caching for Oracle Tuxedo Services

Steps for Using Result Caching for Oracle Tuxedo Services

- Configure Oracle Coherence
- Start Oracle Coherence Cluster
- Configure Oracle Tuxedo Java Server
- Configure UBBCONFIG

You can also use MIB to dynamically make changes for TDC.

- Use MIB to Dynamically Make Changes for TDC

Configure Oracle Coherence

See Configuring Oracle Coherence for instruction.

Start Oracle Coherence Cluster

If there is no Oracle Coherence Cluster is running, start your own cluster, making sure you configure the path of the configuration files into the Java Class Path and ahead of where `coherence.jar` is. See Listing 13-1 for an example.

Configure Oracle Tuxedo Java Server

See Configuring Oracle Tuxedo Java Server for instruction.

Configure UBBCONFIG

Configure TDC on `UBBCONFIG`.

- SERVICES Section
- CACHING Section

SERVICES Section

Specify `CACHING=string_value` as the name of the caching criteria used for caching for this service. For more information, see Oracle Tuxedo Distributed Caching (TDC) Related UBBCONFIG Parameters.

CACHING Section

Specify this `CACHING` section on `UBBCONFIG`. For more information, see Oracle Tuxedo Distributed Caching (TDC) Related UBBCONFIG Parameters.

Listing 13-16 shows an example, where

- `Svccache1` uses Oracle Tuxedo cache `tc1`. Other configurations are default. (It means `KEY=$service+$request` and `KEY_BUFTYPE=STRING`).
- `Svccache2` uses Oracle Tuxedo cache `tc1`. The request is used to figure out the key when caching the response data. Other configurations are default. (It means `KEY_BUFTYPE=STRING`).
- `Svccache3` uses Oracle Tuxedo cache `tc1`. The fixed string `key1` is used as the key. Other configurations are default.

- `Svccache4` uses Oracle Tuxedo cache `tc1`. The buffer type of the request message to the service is `VIEW32` whose subtype is `mystruct1`. The value of the field `name` in the subtype `mystruct1` is used as the key.
- `Svccache5` uses Oracle Tuxedo cache `tc1`. The buffer types of the `FML32` and `VIEW32: mystruct1` have the same `field1` and `field2` (name and data type should be the same; value can be different); the request message will use the values of `field1` and `field2` as the key.

Listing 13-16 UBBCONFIG CACHING Section Configuration

```
...
*CACHING
Svccache1
       CACHENAME="tc1"

Svccache2
       CACHENAME="tc1"
       KEY="$request"

Svccache3
       CACHENAME="tc1"
       KEY="key1"

Svccache4
       CACHENAME="tc1"
       KEY="$request"
       KEY_BUFTYPE="IEW32:mystruct1"
       KEY_FIELD="name"

Svccache5
```

```
        CACHENAME="tc1"
        KEY="mykey_$request"
        KEY_BUFTYPE="FML32;VIEW32:mystruct1"
           KEY_FIELD="field1+field2"

...
```

Use MIB to Dynamically Make Changes for TDC

You can use MIB to dynamically make changes for TDC.

For more information, see Oracle Tuxedo Distributed Caching (TDC) Related MIB Attributes.

Sample: Using Result Caching for Oracle Tuxedo Services

- Sample: Configure VIEWTABLE
- Sample: Configure UBBCONFIG
- Sample: Set on Server Side
- Sample: Set on Client Side

Sample: Configure VIEWTABLE

Configure your `VIEWTABLE`. See Listing 13-17, where the buffer type is `VIEW` and the subtype is `mystruct1`.

Listing 13-17 Configure VIEWTABLE

```
...
VIEW mystruct1
# type    cname     fbname            count  flag  size    null
string    name      -                  1      -      31      -
string    address   -                  1      -     255      -
```

```
char      age        -                 1     -      -        -
END
...
```

Sample: Configure UBBCONFIG

Configure your `UBBCONFIG`. See Listing 13-18, where

- `TMJAVASVR` is configured

  Multi-threaded configuration is enabled and the configuration file `tdcsvr_coh.xml` is set.

- Caching is enabled

  Oracle Tuxedo service `mysvc1` uses caching entry `svccache1` to improve performance. `svccache1` uses Oracle Tuxedo cache `tc1` to cache the service result. The corresponding key of the response is the value of the request data.

Listing 13-18 Configure UBBCONFIG

```
...
*GROUPS
JGRP1  LMID=L1 GRPNO=10


...
TMJAVASVR    SRVGRP=JGRP1 SRVID=10
              MINDISPATCHTHREADS=4 MAXDISPATCHTHREADS=4 MIN=2 MAX=2
              CLOPT="-- -c /home/scott/tuxedo/dom1/config/tdcsvr_coh.xml"
...
*SERVICES
mysvc1
       ...
       CACHING="svccache1"
```

```
...
*CACHING
svccache1
        CACHENAME="tc1"
        KEY=$request
...
```

Sample: Set on Server Side

Configure on server side. See Listing 13-19, where the request of `mysvc1` is set as `STRING` and the response of `mysvc1` is a `VIEW32` `mystruct1`.

Listing 13-19 Configure on Server Side

```
...
struct mystruct1* rsp;
int tpsvrinit(int argc, char *argv[])
{
        rsp = tpalloc("VIEW32", "mystruct1", sizeof(struct mystruct1));
}
...
void mysvc1(TPSVCINFO *rqst)
{
        int ret = 0;
        /*rqst->data is the name, getrsp will get data from the database and
store into rsp*/
        ret = getrsp(rqst->data, rsp);
        if(ret < 0){
                tpreturn(TPFAIL, 0, NULL, 0L, 0);
```

```
        }
        tpreturn(TPSUCCESS, 0, rsp, 0L, 0);
}
...
```

Sample: Set on Client Side

Assume a data file is like this:

Listing 13-20 Data File Example

```
...
Scott
Mike
Andy
Scott
Ben
Brian
Scott
Clark
...
```

Set on your client set like Listing 13-21.

At the first time where `Scott` is taken as the request, `mysvc1` is invoked and the response is sent back and the response is cached into Oracle Tuxedo cache `tc1` with a key `Scott`. As long as the data in the cache `tc1` is not expired, all following requests for `Scott` to service `mysvc1` will get response from the cache `tc1` instead of invoking the service itself.

Listing 13-21 Set on Client Set

```
...
struct mystruct1* rsp;
char* req;
int main(int argc, char *argv[])
{
       int ret;
       long olen = 0;
       rsp = tpalloc("VIEW32", "mystruct1", sizeof(struct mystruct1));
       req = tpalloc("STRING", NULL, 32);
       /*get name from the data file*/
       while(getname(req) == 0){
              tpcall("mysvc1",req, 0, &rsp, &olen,0);
}
}
...
```

Propagating Execution Context ID (ECID) to Oracle Coherence

Oracle Coherence can use the Execution Context ID (ECID) in its logs. This globally unique ID can be attached to requests between Oracle components. ECID allows you to track log messages pertaining to the same request when multiple requests are processed in parallel. Oracle Coherence logs will include ECID only if you already have an activated ECID prior to calling Oracle Coherence operations. ECID may be passed from another component or obtained in the client code.

Working as a client of Oracle Coherence, Oracle Tuxedo TDC enables you to propagate ECID to Oracle Coherence.

Enabling ECID

- Enable ECID in Oracle Tuxedo

  Oracle Tuxedo has two flags which you can add to `OPTIONS` in `UBBCONFIG` to control ECID.

  - `ECID_CREATE`

    Indicates that the ECID (Execution Context Identifier) creation function is enabled. In this case, boundary nodes (including Native/WS/Jolt clients and domain gateways) can generate the ECID.

  - `ECID_USERLOG`

    If the identifier `ECID_USERLOG` is set and the ECID is not a null string, ECID will be appended to the userlog.

- Enable ECID in Oracle Coherence

  See *Oracle Coherence documentation* for instructions.

Enabling ECID for TDC

Prepare `tangosol-coherence-override.xml` in `${APPDIR}/config`. More specifically,

- `<destination>` element is used to configure path and file name for emitting log messages to a file. The specified path must already exist.
- Add `ecid` into `< message-format >` element to enable ECID.
- `<severity-level>` element can be used to change log level.

Listing 13-22

```
<?xml version='1.0'?>

<coherence  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"


xmlns="http://xmlns.oracle.com/coherence/coherence-operational-config"


xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-operational-co
nfig coherence-operational-config.xsd">

<cluster-config>
```

```
       <member-identity>

       <cluster-name
system-property="tangosol.coherence.cluster">coherence_tux</cluster-name>

       </member-identity>

       <unicast-listener>

       <address
system-property="tangosol.coherence.localhost">localhost</address>

       <port system-property="tangosol.coherence.localport">51687</port>

       </unicast-listener>

       <multicast-listener>

       <port system-property="tangosol.coherence.clusterport">51697</port>

       </multicast-listener>

</cluster-config>


<logging-config>

       <destination
system-property="tangosol.coherence.log">/tmp/coherence.log</destination>

       <severity-level
system-property="tangosol.coherence.log.level">9</severity-level>

       <message-format>{date}/{uptime} {product} ecid={ecid} {version}
&lt;{level}&gt;(thread={thread},member={member}):{text}</message-format>

</logging-config>

</coherence>
```

Oracle Tuxedo Distributed Caching (TDC) Related ATMI APIs

Table 13-1 Oracle Tuxedo Distributed Caching (TDC) Related ATMI APIs

Name	Description
tpgetcache(3c)	Get an Oracle Tuxedo Cache handle according to the configuration
tpcacheput(3c)	Put an Oracle Tuxedo typed buffer into a cache, associating that buffer with a key
tpcacheget(3c)	Get the Oracle Tuxedo typed buffer associated with the key from a cache
tpcacheremove(3c)	Remove the cache entry associated with the parameter key from a cache
tpcachemremove(3c)	Remove cache entries associated with the parameter keyarray from a cache
tpcacheremoveall(3c)	Remove all cache entries from a cache

tpgetcache(3c)

Name

`tpgetcache` - Get an Oracle Tuxedo Cache handle according to the configuration

Synopsis

```
#include "atmi.h"

TCACHE* tpgetcache(const char* name);
```

Description

`tpgetcache(3c)` gets an Oracle Tuxedo cache handle according to Oracle Tuxedo cache name, which indicates the name of Oracle Tuxedo cache to be retrieved. The name must be 78 characters or less in length. `tpgetcache(3c)` is a thread-level API. The return handle `TCACHE` can only be used in the same thread.

Return Values

Upon success, `tpgetcache(3c)` returns a handle typed `TCACHE` while is an internal structure.

Upon failure, `tpgetcache(3c)` returns `NULL` and sets `tperrno` to indicate the error condition. If a call fails with a particular `tperrno` value, a subsequent call to `tperrordetail(3c)`, with no intermediate ATMI calls, may provide more detailed information about the generated error. Refer to the `tperrordetail(3c)` reference page for more information.

Errors

[TPEINVAL]
Invalid arguments were given (for example, conf is NULL).

[TPENOENT]
The requested cache does not exist.

[TPETIME]
This error code indicates that a timeout has occurred

[TPESYSTEM]
An Oracle Tuxedo system error has occurred. The exact nature of the error is added to the ULOG.

[TPEOS]
An operating system error has occurred.

[TPESVCFAIL]
The Oracle Tuxedo cache server meets an error. `tpurcode` contains the exact error value. The possible values of `tpurcode` are list below:

TDC_ERR_CACHE_NOTEXIST
The error code indicates the requested Oracle Tuxedo cache does not exist.

TDC_ERR_CACHE_UNAVAIL
The error code indicates the requested Oracle Tuxedo cache is unavailable.

TDC_ERR_CACHENAME_INVALID
The error code indicates the requested Oracle Tuxedo cache is invalid.

tpcacheput(3c)

Name

`tpcacheput` - put an Oracle Tuxedo typed buffer into a cache, associating that buffer with a key

Synopsis

```
#include "atmi.h"

int tpcacheput(TCACHE* tc, char* key, char* data, long len, long flags);
```

Description

`tpcacheput(3c)` puts an Oracle Tuxedo typed buffer into a cache, associating that buffer with a key. `tc` is returned by `tpgetcache(3c)`. `data` points to the tuxedo typed buffer allocated by `tpalloc(3c)`. `len` is the length of the data. If the type of the data does not require a length to be specified (for example, an FML fielded buffer), `len` is ignored (and may be 0). `flags` is reserved and must be `0L`.

Return Values

Upon success, `tpcacheput(3c)` returns 0.

Upon failure, `tpcacheput(3c)` returns -1 and sets `tperrno` to indicate the error condition. If a call fails with a particular `tperrno` value, a subsequent call to `tperrordetail(3c)`, with no intermediate ATMI calls, may provide more detailed information about the generated error. Refer to the `tperrordetail(3c)` reference page for more information.

Errors

[TPEINVAL]
Invalid arguments were given.

[TPENOENT]
The requested cache does not exist.

[TPETIME]
This error code indicates that a timeout has occurred

[TPESYSTEM]
An Oracle Tuxedo system error has occurred. The exact nature of the error is added to the ULOG.

[TPEOS]
An operating system error has occurred.

[TPESVCFAIL]
The Oracle Tuxedo cache server meets an error. `tpurcode` contains the exact error value. The possible values of `tpurcode` are list below:

TDC_ERR_CACHE_NOTEXIST
The error code indicates the requested Oracle Tuxedo cache does not exist.

TDC_ERR_CACHE_UNAVAIL
The error code indicates the requested Oracle Tuxedo cache is unavailable.

TDC_ERR_CACHENAME_INVALID
The error code indicates the requested Oracle Tuxedo cache is invalid.

tpcacheget(3c)

Name

`tpcacheget` - get the Oracle Tuxedo typed buffer associated with the key from a cache

Synopsis

```
#include "atmi.h"

int tpcacheget(TCACHE* tc, char* key, char** odata, long* olen, long flags);
```

Description

`tpcacheget(3c)` gets the Oracle Tuxedo typed buffer associated with the key from a cache. `tc` is returned by `tpgetcache(3c)`. `odata` is the address of a pointer to the buffer where the data of the key is read into. It must point to a buffer originally allocated by `tpalloc(3c)`. `olen` points to the length of the data. `flags` is reserved and must be `0L`.

Return Values

Upon success, `tpcacheget(3c)` returns 0.

Upon failure, `tpcacheget(3c)` returns -1 and sets `tperrno` to indicate the error condition. If a call fails with a particular `tperrno` value, a subsequent call to `tperrordetail(3c)`, with no intermediate ATMI calls, may provide more detailed information about the generated error. Refer to the `tperrordetail(3c)` reference page for more information.

Errors

[TPEINVAL]
Invalid arguments were given.

[TPENOENT]
The requested cache does not exist.

[TPETIME]
This error code indicates that a timeout has occurred.

[TPESYSTEM]
An Oracle Tuxedo system error has occurred. The exact nature of the error is added to the ULOG.

[TPEOS]
An operating system error has occurred.

[TPESVCFAIL]
The Oracle Tuxedo cache server meets an error. `tpurcode` contains the exact error value. The possible values of `tpurcode` are list below:

TDC_ERR_CACHE_NOTEXIST
The error code indicates the requested Oracle Tuxedo cache does not exist.

TDC_ERR_CACHE_UNAVAIL
The error code indicates the requested Oracle Tuxedo cache is unavailable.

TDC_ERR_CACHENAME_INVALID
The error code indicates the requested Oracle Tuxedo cache is invalid.

TDC_ERR_KEY_NOTEXIST
The error code indicates the requested entry does not exist according to the specified key.

tpcacheremove(3c)

Name

`tpcacheremove` - remove the cache entry associated with the parameter key from a cache

Synopsis

```
#include "atmi.h"

int tpcacheremove(TCACHE* tc, char* key, long flags);
```

Description

`tpcacheremove`(3c) removes the cache entry associated with the parameter key from a cache. `tc` is returned by `tpgetcache(3c)`. `flags` is reserved and must be `0L`.

Return Values

Upon success, `tpcacheremove(3c)` returns 0.

Upon failure, `tpcacheremove(3c)` returns -1 and sets `tperrno` to indicate the error condition. If a call fails with a particular `tperrno` value, a subsequent call to `tperrordetail(3c)`, with no intermediate ATMI calls, may provide more detailed information about the generated error. Refer to the `tperrordetail(3c)` reference page for more information.

Errors

[TPEINVAL]
Invalid arguments were given.

[TPENOENT]
The requested cache does not exist.

[TPETIME]
This error code indicates that a timeout has occurred.

[TPESYSTEM]
An Oracle Tuxedo system error has occurred. The exact nature of the error is added to the ULOG.

[TPEOS]
An operating system error has occurred.

[TPESVCFAIL]
The Oracle Tuxedo cache server meets an error. `tpurcode` contains the exact error value. The possible values of `tpurcode` are list below:

TDC_ERR_CACHE_NOTEXIST
The error code indicates the requested Oracle Tuxedo cache does not exist.

TDC_ERR_CACHE_UNAVAIL
The error code indicates the requested Oracle Tuxedo cache is unavailable.

TDC_ERR_CACHENAME_INVALID
The error code indicates the requested Oracle Tuxedo cache is invalid.

tpcachemremove(3c)

Name

`tpcachemremove` - remove cache entries associated with the parameter keyarray from a cache

Synopsis

```
#include "atmi.h"

int tpcachemremove(TCACHE* tc, char* keyarray[], int size, long flags);
```

Description

`tpcachemremove(3c)` removes cache entries associated with the parameter keyarray from a cache. `tc` is returned by `tpgetcache(3c)`. `keyarray` is an array of keys to be removed. `size` is the size of the keyarray. `flags` is reserved and must be `0L`.

Return Values

Upon success, `tpcachemremove(3c)` returns 0.

Upon failure, `tpcachemremove(3c)` returns -1 and sets `tperrno` to indicate the error condition. If a call fails with a particular `tperrno` value, a subsequent call to `tperrordetail(3c)`, with no intermediate ATMI calls, may provide more detailed information about the generated error. Refer to the `tperrordetail(3c)` reference page for more information.

Errors

[TPEINVAL]
Invalid arguments were given.

[TPENOENT]
The requested cache does not exist.

[TPETIME]
This error code indicates that a timeout has occurred

[TPESYSTEM]
An Oracle Tuxedo system error has occurred. The exact nature of the error is added to the ULOG.

[TPEOS]
An operating system error has occurred.

[TPESVCFAIL]
The Oracle Tuxedo cache server meets an error. `tpurcode` contains the exact error value. The possible values of `tpurcode` are list below:

TDC_ERR_CACHE_NOTEXIST
The error code indicates the requested Oracle Tuxedo cache does not exist.

TDC_ERR_CACHE_UNAVAIL
The error code indicates the requested Oracle Tuxedo cache is unavailable.

TDC_ERR_CACHENAME_INVALID
The error code indicates the requested Oracle Tuxedo cache is invalid.

tpcacheremoveall(3c)

Name

`tpcacheremoveall` - remove all cache entries from a cache

Synopsis

```
#include "atmi.h"

int tpcacheremoveall(TCACHE* tc, long flags);
```

Description

`tpcacheremoveall(3c)` removes all entries from a cache. `tc` is returned by `tpgetcache(3c)`. `flags` is reserved and must be `0L`.

Return Values

Upon success, `tpcacheremoveall(3c)` returns 0.

Upon failure, `tpcacheremoveall(3c)` returns -1 and sets `tperrno` to indicate the error condition. If a call fails with a particular `tperrno` value, a subsequent call to `tperrordetail(3c)`, with no intermediate ATMI calls, may provide more detailed information about the generated error. Refer to the `tperrordetail(3c)` reference page for more information.

Errors

[TPEINVAL]
Invalid arguments were given.

[TPENOENT]
The requested cache does not exist.

[TPETIME]
This error code indicates that a timeout has occurred.

[TPESYSTEM]
An Oracle Tuxedo system error has occurred. The exact nature of the error is added to the ULOG.

[TPEOS]
An operating system error has occurred.

[TPESVCFAIL]
The Oracle Tuxedo cache server meets an error. `tpurcode` contains the exact error value. The possible values of `tpurcode` are list below:

TDC_ERR_CACHE_NOTEXIST
The error code indicates the requested Oracle Tuxedo cache does not exist.

TDC_ERR_CACHE_UNAVAIL
The error code indicates the requested Oracle Tuxedo cache is unavailable.

TDC_ERR_CACHENAME_INVALID
The error code indicates the requested Oracle Tuxedo cache is invalid.

Oracle Tuxedo Distributed Caching (TDC) Property File Properties

Oracle Tuxedo TDC property file is a file in a simple line-oriented format.

Properties are processed in terms of lines. There are two kinds of line, natural lines and logical lines.

A natural line is defined as a line of characters that is terminated either by a set of line terminator characters (`\\n` or `\\r` or `\\r\\n`) or by the end of the stream. A natural line may be either a blank line, a comment line, or hold all or some of a key-element pair.

A logical line holds all the data of a key-element pair, which may be spread out across several adjacent natural lines by escaping the line terminator sequence with a backslash character `\\`.

A natural line that contains only white space characters is considered blank and is ignored. A comment line has an ASCII `'#'` or `'!'` as its first non-white space character; the key contains all of the characters in the line starting with the first non-white space character and up to, but not including, the first unescaped `'=', ':'`. Any white space after the key is skipped; if the first non-white space character after the key is `'='` or `':'`, then it is ignored and any white space characters after it are also skipped. All remaining characters on the line become part of the associated element string; if there are no remaining characters, the element is the empty string `""`.

If there are several properties with the same key, the last property of them will be used.

Table 13-2 lists the supported TDC property file properties.

Table 13-2 Oracle Tuxedo Distributed Caching (TDC) Property File Properties

Property	Description
`options.encoding`	Values can be • `yes` (all caching data must be encoded) • `no` (default) This value can be overridden by `cache.options.encode.[cachename]`.
`cache.options.encoding.[cachename]`	`yes` indicates all caching data in Oracle Tuxedo cache `[cachename]` must be encoded. It should be enabled when caching users are located in machines that have different data representation. If this property is not set, `options.encoding` value is used.
`coh.cache.name.[cachename]`	Indicates the used cache name in the Coherence cluster for the Oracle Tuxedo cache `[cachename]`. This is necessary for TDC property file if Oracle Coherence is used for TDC.

Note: `[cachename]` must be 78 characters or less in length.

Oracle Tuxedo Distributed Caching (TDC) Related UBBCONFIG Parameters

UBBCONFIG SERVICES Section

Table 13-3 Oracle Tuxedo Distributed Caching (TDC) Related UBBCONFIG SERVICES Section Parameters

Parameter	Description
`CACHING=string_value` `(Mandatory)`	Specifies the name of the caching criteria used for this service when caching the response. The `string_value` should be the `CACHING_CRITERIA_NAME` defined in the `CACHING` section. You must specify this parameter; otherwise, service caching will not be enabled. The `string_value` must be 127 characters or less in length.

UBBCONFIG CACHING Section

Table 13-4 Oracle Tuxedo Distributed Caching (TDC) Related UBBCONFIG Parameters

Parameter	Description
`CACHING_CRITERIA_NAME required_parameters` `(Mandatory)`	`CACHING_CRITERIA_NAME` (`string_value`) is the name assigned to the `CACHING` parameter for a particular service entry in the `SERVICES` section. `CACHING_CRITERIA_NAME` must be 127 characters or less in length.
`CACHENAME=string_value` `(Mandatory)`	Specifies the name of Oracle Tuxedo cache to be used. `string_value` must be 78 characters or less in length.

Table 13-4 Oracle Tuxedo Distributed Caching (TDC) Related UBBCONFIG Parameters

Parameter	Description
KEY=string_value (Optional)	Specifies how to compose the key associating the cached data. string_value must be 127 characters or less in length. This parameter is optional. If not specified, the key is generated as a combination of the service name and the serialized user data in the request. string_value can import the built-in variables to compose key by $. The variable request indicates to use part of or total user data in the request. For example, mykey_$request indicates the key's format is mykey_servicename_[user data in request]. KEY_BUFTYPE and KEY_FIELD parameters can help to specify which part of the user data will be used.

Table 13-4 Oracle Tuxedo Distributed Caching (TDC) Related UBBCONFIG Parameters

Parameter	Description
KEY_BUFTYPE="type1[:subtype1[,subtype2 . . .]][;type2[:subtype3[, . . .]]] . . ." (Optional)	A list of types and subtypes of data buffers for which this caching entry is valid to use to generate the key. This parameter can be up to 255 characters in length and a maximum of 32 type/subtype combinations. The type must be one of the following: STRING, CARRAY, FML, FML32, XML, VIEW, VIEW32, RECORD, MBSTRING, X_C_TYPE, or X_COMMON. No subtypes can be specified for types STRING, CARRAY, FML, FML32, MBSTRING, or XML. Subtypes are required for type VIEW, VIEW32, RECORD, X_C_TYPE, and X_COMMON (* is not allowed). Subtype name should not contain semicolon, colon, comma, or asterisk characters. Duplicate type/subtype pairs cannot be specified for the same caching criteria name; more than one caching entry can have the same criteria name as long as the type/subtype pairs are unique. If multiple buffer types are specified for a single caching entry, the data types of the caching field for each buffer type must be the same. STRING is used if this parameter is not specified. This parameter is ignored if KEY is not set to request.

Table 13-4 Oracle Tuxedo Distributed Caching (TDC) Related UBBCONFIG Parameters

Parameter	Description
KEY_FIELD= "field1[+field2[+field3[+¡-]]]" (Optional)	Specifies the name of the fields in the buffer used to generate the key. It must be 255 characters or less and a maximum of 8 fields are allowed. Each field can be an XML element, element attribute, or field name in an FML field table (using FLDTBLDIR and FIELDTBLS environment variables, or FLDTBLDIR32 and FIELDTBLS32 environment variables), or an FML view table (using VIEWDIR and VIEWFILES environment variables, or VIEWDIR32 and VIEWFILES32 environment variables). This information is used to get the associated field value for service caching when sending a message. For XML element content or element attribute in the XML buffer, define the name of the field with the following syntax: root_element[/child_element][/child_element][/. . .][/@attribute_name] The element name and attribute name combined may contain no more than 35 characters. Oracle Tuxedo recognizes only the first occurrence of a given element type when processing an XML buffer for service caching. XML strictly defines the set of characters that may be used in an attribute name. An attribute name must be a string consisting of a single letter, underscore, or colon followed by one or more name characters. Both element names and attribute names are case-sensitive. See XML documentation for more information. If KEY_BUFTYPE is set to STRING or CARRAY, define the field with the following syntax: [index1, index2] Index1 indicates the beginning index of the field in the buffer and index2 indicates the ending index of the field. The whole buffer is taken as the key if this parameter is not defined. The index should be set from 1. This parameter is ignored if KEYBUFTYPE is not set or KEY is not set to request.

Oracle Tuxedo Distributed Caching (TDC) Related MIB Attributes

T_SERVICE Class Definition

Table 13-5 Oracle Tuxedo Distributed Caching (TDC) Related MIB Attributes

Attribute	Type	Permission	Values	Default
TA_CACHING	string	rwxr--r--	string[0..127]	""

TA_CACHING

This T_SERVICE object indicates the caching criteria name. Any updates on this attribute will be reflected in all associated T_SVCGRP objects.

T_CACHING Class Definition

This T_CACHING class represents configuration attributes of caching specifications for an application. These attribute values identify and characterize application caching criteria (such as buffer types, field names and caching definitions).

Table 13-6 Oracle Tuxedo Distributed Caching (TDC) Related MIB Attributes

Attribute	Type	Permission	Values	Default
TA_CACHING_NAME(r)(*)	string	ru--r--	string[0..127]	N/A
TA_CACHING_CACHENAME(r)	string	rw-r--r--	string[1..78]	N/A
TA_CACHING_KEY	string	rw-r--r--	string[0..127]	""
TA_CACHING_KEY_BUFTYPE(*)	string	ru--r--	string[0..255]	"STRING"
TA_CACHING_KEY_FIELD	string	rw-r--r--	string[0..255]	""
TA_STATE(k)	string	rw-r--r--	GET: "VAL" SET: "{NEW | INV}"	N/A N/A

Notes:

- (k) GET key field
- (r) required field for object creation (SET TA_STATE NEW)
- (*) GET/SET key, one or more required for SET operations
- The specified u (uniqueness) permission means aht the combination of TA_CACHING_NAME and TA_CACHING_KEY_BUFTYPE must be unique.

TA_CACHING_NAME: string[1..127]
Specifies the caching criteria name.

TA_CACHING_CACHENAME: string[1..78]
Specifies the name of the Oracle Tuxedo cache to be used.

TA_CACHING_KEY: string[0..127]
Specifies how to compose the key associating the cached data.

TA_CACHING_KEY_BUFTYPE: string[0..255]
Specifies a list of types and subtypes of data buffers for which this caching entry is valid to use to generate the key. This parameter can be up to 255 characters in length and a maximum of 32 type/subtype combinations.
The type must be one of the following: STRING, CARRAY, FML, FML32, XML, VIEW, VIEW32, RECORD, MBSTRING, X_C_TYPE, or X_COMMON. No subtypes can be specified for types STRING, CARRAY, FML, FML32, MBSTRING, or XML. Subtypes are required for type VIEW, VIEW32, RECORD, X_C_TYPE, and X_COMMON (* is not allowed). Subtype name should not contain semicolon, colon, comma, or asterisk characters. Duplicate type/subtype pairs cannot be specified for the same routing caching criteria name; more than one routing caching entry can have the same criteria name as long as the type/subtype pairs are unique. If multiple buffer types are specified for a single caching entry, the data types of the caching field for each buffer type must be the same.

TA_CACHING_KEY_FIELD: string[0..255]
Specifies the name of the fields in the buffer used to generate the key. It must be 255 characters or less and a maximum of 8 fields are allowed.
Each field can be an XML element, element attribute, or field name in an FML field table (using FLDTBLDIR and FIELDTBLS environment variables, or FLDTBLDIR32 and FIELDTBLS32 environment variables), or an FML view table (using VIEWDIR and VIEWFILES environment variables, or VIEWDIR32 and VIEWFILES32 environment variables). This information is used to get the associated field value for service caching when sending a message.
For XML element content or element attribute in the XML buffer, define the name of the field with the following syntax:

```
root_element[/child_element][/child_element][/. .
.][/@attribute_name]
```

The element name and attribute name combined may contain no more than 35 characters. Oracle Tuxedo recognizes only the first occurrence of a given element type when processing an XML buffer for service caching. XML strictly defines the set of characters that may be used in an attribute name. An attribute name must be a string consisting of a single letter, underscore, or colon followed by one or more name characters. Both element names and attribute names are case-sensitive. See XML documentation for more information.

If KEY_BUFTYPE is set to STRING or CARRAY, define the field with the following syntax:

```
[index1, index2]
```

Index1 indicates the beginning index of the field in the buffer and index2 indicates the ending index of the field. The whole buffer is taken as the key if this parameter is not defined. The index should be set from 1.

TA_STATE

GET: "{VALid}"

A GET operation will retrieve configuration information for the selected T_CACHING object(s). The following state indicates the meaning of a TA_STATE returned in response to a GET request. States not listed will not be returned.

VALid: T_CACHING object is defined. Note that this is the only valid state for this class.

SET: "{NEW | INValid}"

A SET operation will update configuration information for the selected T_CACHING object. The following states indicate the meaning of a TA_STATE set in a SET request. States not listed may not be set.

NEW: Create T_CACHING object for application. State change allowed only when in the INValid state. Successful return leaves the object in the VALid state.

INValid: Delete T_CACHING object for application. State change allowed only when in the VALid state. Successful return leaves the object in the INValid state.

CHAPTER 14

About Workstation Clients

This topic includes the following sections:

What Is the Workstation Component?

The Workstation component of the Oracle 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:

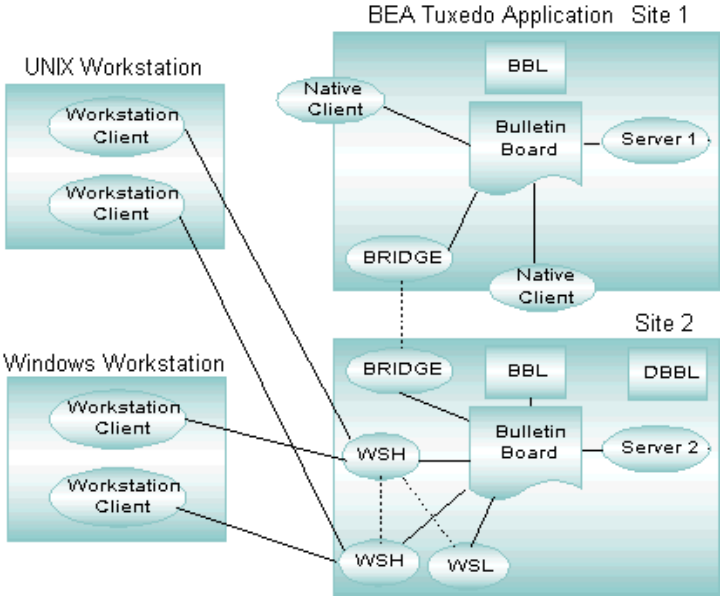
- Send and receive messages

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

Sample Application with Four Workstation Clients

Figure 14-1 shows an example of an application with four Workstation clients.

Figure 14-1 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.

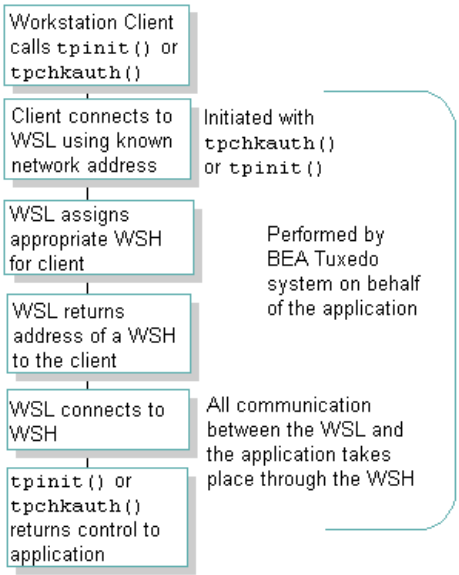
Administrative servers and application servers are located on `SITE1` and `SITE2`. Any service request by a Workstation client to the application is sent over the network to the WSH. This process forwards the request to the appropriate server, gets a reply from the server, and sends the reply to the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

CHAPTER 15

Setting Up Workstation Clients

This topic includes the following sections:

Defining Workstation Clients

Before a Workstation client can join an Oracle Tuxedo application, the application environment must be prepared to accommodate it. The Oracle Tuxedo system provides the variables described in Table 15-1 table for setting up your environment. Two (`TUXDIR` and `WSNADDR`) are required; the rest are optional. Defaults are available for all parameters except `WSENVFILE`.

Table 15-1 Defining Workstation Clients

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)
Specifies the security principal name identification string to be used for authentication purposes when SSL is initiated. This parameter may contain a maximum of 511 characters (excluding the terminating NULL character).	SEC_PRINCIPAL_NAME (Optional)
Specifies the location of the file or device where the decryption (private) key for the principal specified in SEC_PRINCIPAL_NAME resides. This parameter may contain a maximum of 1023 characters (excluding the terminating NULL character).	SEC_PRINCIPAL_LOCATION (Optional)
Specifies the variable in which the password for the principal specified in SEC_PRINCIPAL_NAME is stored. This parameter may contain a maximum of 31 characters (excluding the terminating NULL character).	SEC_PRINCIPAL_PASSWORD (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, 128, or 256 (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, 128, or 256 (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)

Table 15-1 Defining Workstation Clients

To Specify . . .	Set This Environment Variable . . .
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 *as is*—no encoding conversion. Setting TPMBACONV to any value, say Y (yes), turns on the automatic conversion.	TPMBACONV (Optional)
The location of the Oracle Tuxedo system software on this workstation. *The client cannot connect unless this environment variable is set.*	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)
Specifies the directory in which the workstation machine caches DTD, schema, and entity files. The URLENTITYCACHEDIR variable specifies the absolute pathname for the cached files. If URLENTITYCACHEDIR is not specified, the default directory becomes URLEntityCachedir, which will be created in the current working directory of the Workstation client process provided that the appropriate write permissions are set.	URLENTITYCACHEDIR (Optional)
The network device to be used. The default is an empty string.	WSDEVICE (Optional)
The name of the file in which all environment variables may be set. There is no default for this variable.	WSENVFILE (Optional)
The network address used by the Workstation client when connecting to the Workstation listener or Workstation handler. This variable, along with the WSFRANGE variable, determines the range of TCP/IP ports to which a Workstation client attempts to bind before making an outbound connection. This address must be a TCP/IP address	WSFADDR (Optional)

Table 15-1 Defining Workstation Clients

To Specify . . .	Set This Environment Variable . . .
The range of TCP/IP ports to which a Workstation client process attempts to bind before making an outbound connection. The WSFADDR parameter specifies the base address of the range.	WSFRANGE (Optional)
A list of one or more network addresses of the WSL that the client wants to contact. This address must match the address of a WSL process in the application configuration file.	WSNADDR (Required)
The amount of core memory to be used for buffering application replies. The default is 256,000 bytes.	WSRPLYMAX (Optional)
The machine type. If the value of WSTYPE matches the value of TYPE in the configuration file for the WSL machine, no encoding/decoding is performed. The default is the empty string.	WSTYPE (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 Oracle 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	Specifies the maximum number of WSCs that may connect to a machine. The syntax is MAXWSCLIENTS=*number*. The default is 0. If MAXWSCLIENTS is not specified, WSCs may not connect to the machine being described.

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 shown in Table 15-2 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*.

Table 15-2 Using Command-line Options Set with CLOPT

Use This Command-line Option . . .	To Specify . . .
-n *netaddr* (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 -n.
[-d *device*] (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.
[-t *timeout*]	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 timeout by the value of the SCANUNIT 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: • USER_AUTH • ACL • MANDATORY_ACL • APP_PW

Table 15-2 Using Command-line Options Set with CLOPT

Use This Command-line Option . . .	To Specify . . .
[-w *name*]	The name of the WSH process that should be booted for this listener. The default is WSH, which is the name of the handler provided. If another handler process is built with the buildwsh(1) command, that name is specified here.
[-m *number*]	The minimum number of handlers that should be booted and always available. The default is 0.
[-M *number*]	The maximum number of handlers that can be booted. The default is the value of MAXWSCLIENTS for the machine being configured, divided by the multiplexing value (specified with -x).
[-x *number*]	The maximum number of clients that a WSH can multiplex at one time. The value must be greater than 0. The default is 10.
[-T *client_timeout*]	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.
[-p *minwshport*] and [-P *maxwshport*]	The range for port numbers available for use by WSHs associated with this listener server. Port numbers must fall in the range between 0 and 65535. The default is 2048 for *minwshport* and 65535 for *maxwshport*.
[-z] and [-Z]	The range of bits that can be used, on the WSL side, for link-level encryption: use -z to specify the minimum number of bits, and -Z to specify the maximum number of bits.

Table 15-2 Using Command-line Options Set with CLOPT

Use This Command-line Option . . .	To Specify . . .
[-N *network_timeout*]	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.
[-K {client | handler | both | none}]	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`.

Table 15-3 shows the keep-alive option.

Table 15-3 Using the Keep-alive Option

Use This Option...	To...
`-K client`	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 `tperrno` of `TPESYSTEM`.
`-K handler`	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 `-x`) with stale clients.
`-K both`	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.
`-K none`	Turn off the keep-alive option. Using this setting has the same effect as not specifying `-K` 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.

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 Oracle 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 Oracle 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 Oracle 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 in Listing 15-1 shows how you can add the Workstation component to the `bankapp` application. It contains modifications to the `MACHINES` and `SERVERS` sections.

Listing 15-1 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).

CHAPTER 16

Managing Remote Oracle Tuxedo CORBA Client Applications

This chapter explains how to configure connections from remote Oracle Tuxedo CORBA client applications to CORBA objects via the standard Internet Inter-ORB Protocol (IIOP). This chapter is specific to Oracle Tuxedo CORBA servers.

Note: The Oracle Tuxedo CORBA Java client and Oracle Tuxedo CORBA Java client ORB were deprecated in Tuxedo 8.1 and are no longer supported in Tuxedo 9.x. All Oracle Tuxedo CORBA Java client and Oracle 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. Oracle Tuxedo does not provide any technical support or documentation for third party CORBA Java ORBs.

This topic includes the following sections:

- 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 an Oracle Tuxedo domain. An Oracle Tuxedo CORBA server is built with the Oracle Tuxedo CORBA `buildobjserver` command. CORBA Servers implement Oracle Tuxedo functionality, such as security, transactions, and object state management. Servers can make invocations on any server, inside or outside an Oracle Tuxedo domain.

Native Client

A client located within an Oracle Tuxedo domain, using the CORBA ORB to make invocations on objects either inside or outside the Oracle Tuxedo domain. A native client's host contains the Oracle 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 an Oracle Tuxedo domain. A remote client can use the CORBA ORB to make invocations on objects either inside or outside the Oracle Tuxedo domain. A remote client's host does not contain Oracle 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 an Oracle 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 Oracle Tuxedo CORBA administrative and infrastructure components, such as tmadmin, FactoryFinder, and ISL/ISH (hence none of Oracle Tuxedo's scalability and reliability attributes), it does not use the Oracle 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 an Oracle Tuxedo domain. The joint client/server does not use the Oracle 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 Oracle Tuxedo administrative and infrastructure components, such as tmadmin, FactoryFinder, and ISL/ISH (hence, none of Oracle Tuxedo's scalability and reliability attributes).

Oracle 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 Oracle Tuxedo CORBA servers; that is, they are part of an Oracle Tuxedo domain and use the Oracle 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 an Oracle 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 Oracle 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:

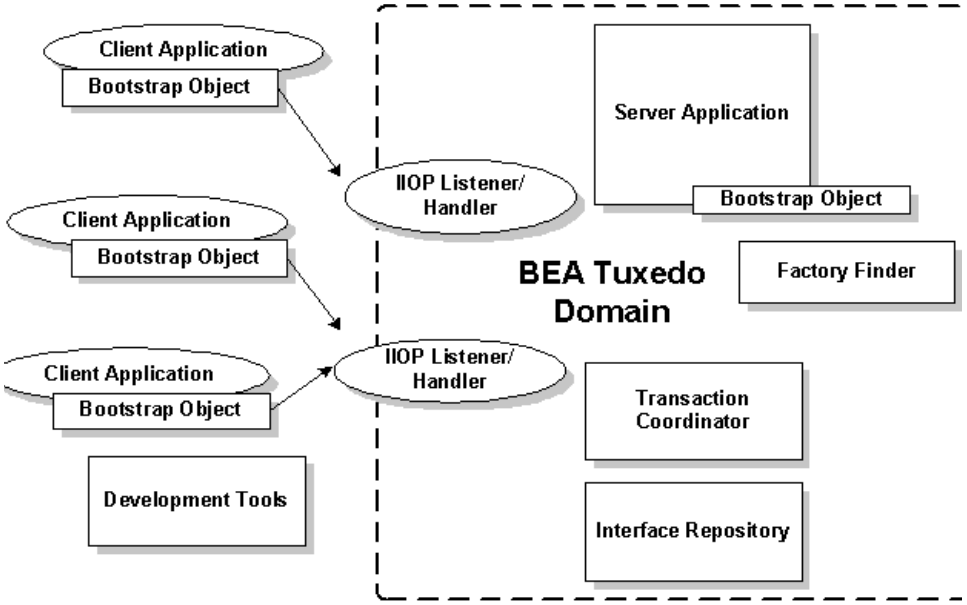
- Invoke methods on remote CORBA objects
- Begin, roll back, or commit transactions
- Be required to pass application security

Note: A client process communicates with the native domain through the ISH.

Illustration of an Application with Remote CORBA Clients

Figure 16-1 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 16-1 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 Oracle 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 Oracle 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
MAXWSCLIENTS	Specifies the maximum number of remote clients that may connect to a machine. The default is 0. If a value is not specified, remote clients may not connect to the machine being described. The syntax is MAXWSCLIENTS=*number*.

Configuring a Listener for a Remote 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 Oracle 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 *Oracle Tuxedo Command Reference*.

Modifying the Configuration File to Support Remote CORBA Clients

Listing 16-1 shows a sample UBBCONFIG file to support remote clients, as follows:

- The MACHINES section shows the default MAXWSCLIENTS as being overridden for two sites. For SITE1, the default is raised to 150, while it is lowered to 0 for SITE2, which does not have remote clients connected to it.
- The SERVERS section shows an ISL process listed for group BANKB1. Its server ID is 500 and it is marked as restartable.
- The command line options show the following:
  - The IIOP Listener/Handler will advertise all of its services (-A).
  - The IIOP Listener/Handler will listen at host TRIXIE on port 2500.
  - The network provider is /dev/tcp (-d).
  - The minimum number of ISH processes to boot is 5 (-m).
  - The maximum number of ISH processes to boot is 30 (-M).
  - Each handler can have a maximum of 5 clients connected at any one time (-x).

Listing 16-1 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 Oracle 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 Oracle 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 Oracle 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 16-2.)

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 Oracle WebLogic Enterprise release 4.2 or later C++ GIOP 1.2 servers, clients, and joint client/servers)

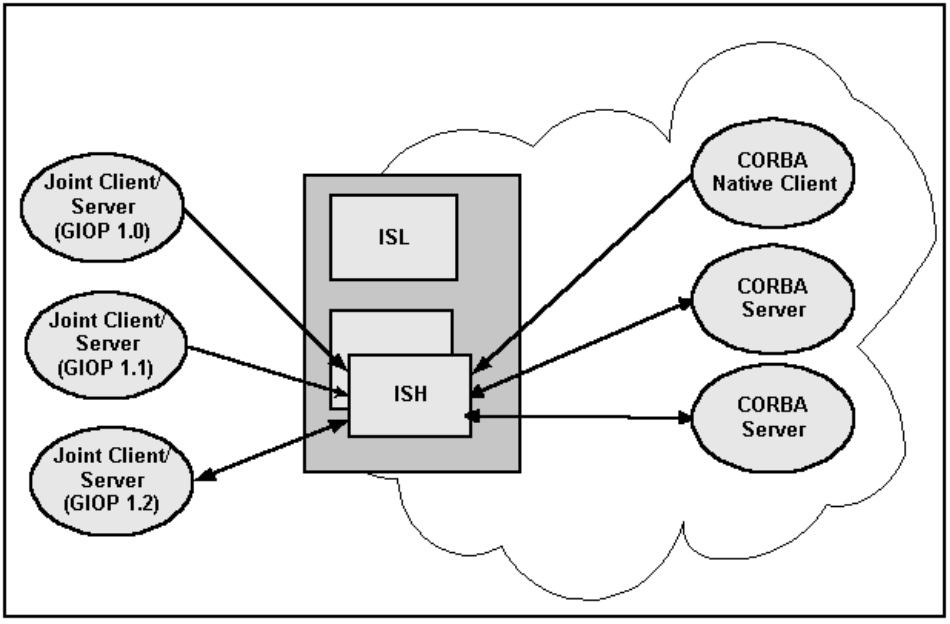
- Asymmetric—outbound IIOP via a second connection (supported for GIOP 1.0, GIOP 1.1, and GIOP 1.2 servers, clients, and joint client/server applications)
- Dual-paired connection—outbound IIOP (supported for GIOP 1.0, GIOP 1.1, and GIOP 1.2 servers, clients, and joint client/server applications)

  Note: GIOP 1.2 is supported only by Oracle WebLogic Enterprise release 4.2 (and later) and Oracle Tuxedo release 8.0 (and later) C++ clients, servers, and joint client/servers. Oracle 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 Oracle 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 16-2 Joint Client/Server IIOP Connections Supported

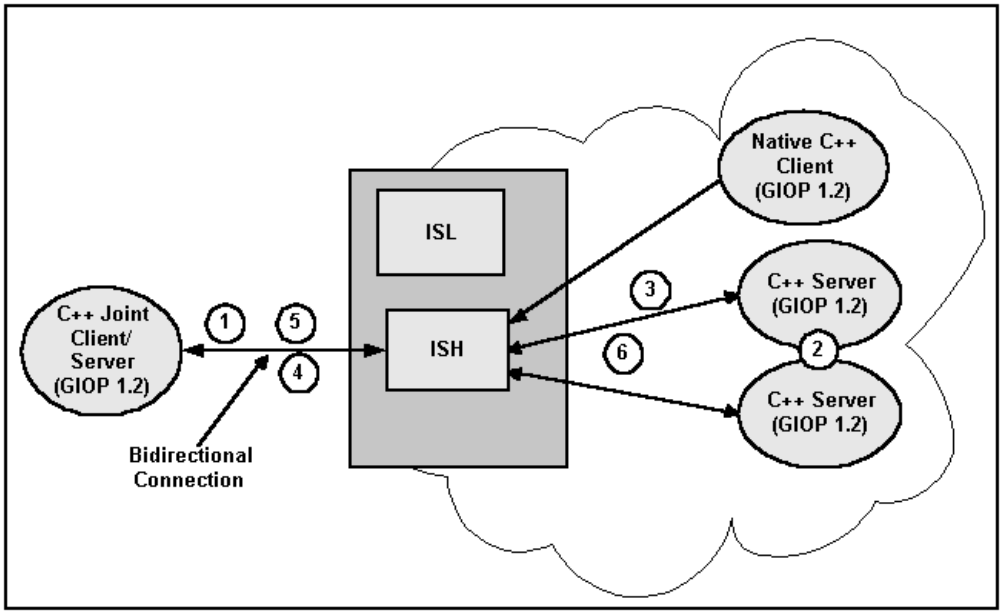


Bidirectional Outbound IIOP

With bidirectional outbound IIOP, the following operations are executed (see Figure 16-3):

1. A client creates an object reference and invokes on a Oracle 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 Oracle Tuxedo CORBA server.
2. When unmarshaling the object reference, the Oracle 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 Oracle Tuxedo CORBA servers.
3. At some point in time, an Oracle 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 Oracle Tuxedo CORBA server.

Figure 16-3 Bidirectional Connection



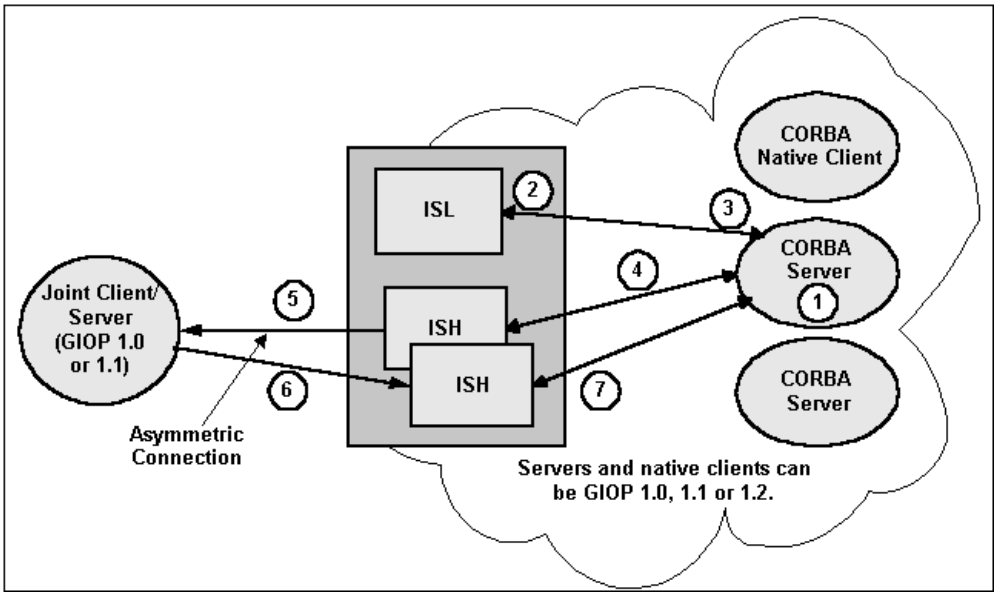
Asymmetric Outbound IIOP

With asymmetric outbound IIOP, the following operations are executed (see Figure 16-4):

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 Oracle 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 Oracle Tuxedo CORBA server.
4. The Oracle 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 Oracle Tuxedo CORBA server.

Figure 16-4 Asymmetric Outbound IIOP



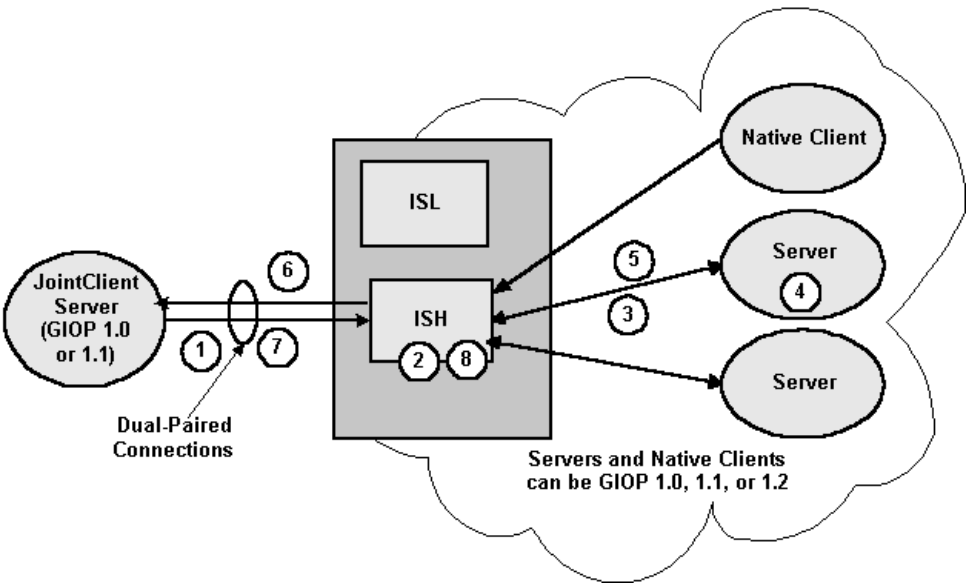
Dual-paired Connection Outbound IIOP

With dual-paired connection outbound IIOP, the following operations are executed (see Figure 16-5):

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 an Oracle 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 Oracle Tuxedo CORBA server.
4. When unmarshaling the object reference, the Oracle 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 Oracle Tuxedo CORBA servers.

5. At some point in time, an Oracle 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 Oracle Tuxedo CORBA server. If the client disconnects from the ISH, the second connection is also disconnected.

Figure 16-5 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 Oracle 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-Oracle Tuxedo CORBA ORB or from a remote Oracle Tuxedo CORBA joint client/server.

Types of Object References

There are two kinds of remote object references:

- Object references created by Oracle Tuxedo CORBA remote joint client/servers outside of the Oracle 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 Oracle 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 *Oracle Tuxedo Command Reference*.

Applying Service Version to Tuxedo Applications

Overview

It is common that the user wants to keep existing functionality but also want to add new functionality into a service as time going. To reduce the compatible risk, it had better to provide two different version services with the same service name, one for old functionality, and one for new functionality. The old client can still use the existing functionality without any code change while the new client can use the new functionality.

The application service version feature offer a configuration driven way which can be used by Tuxedo customers to plan, develop, test, scale, and deploy their Tuxedo applications in each stage. The user can use the version to partition current Tuxedo application into different virtual application domains, different virtual machines, and different virtual server groups on current Tuxedo management hierarchy. It provide a flexible method to let customers setup their application zone according to a defined version (from this perspective, version can be endowed a new meaning: logical partition identity) to respond all kinds of special business access logic, and on the other hand customers can use version to solve some upgrading requirements in non-stop mode and change the service business logic seamlessly for the end users.

Enabling and Disabling Application Service Versioning

The user can enable/disable the application service version feature in UBB configuration file or through MIB.

Enable/Disable Application Service Version Using UBB Config File

To enable the application service version, add the APPVER option to the OPTIONS parameter in *RESOURCES section. For example:

```
*RESOURCES

OPTIONS        APPVER, LAN
```

To disable the application service version, remove the APPVER option from the OPTIONS parameter in *RESOURCES section. For example:

```
*RESOURCES

OPTIONS        LAN
```

Note: If the application version is disabled, the user cannot configure the application service version related configuration in the*RESOURCE and *GROUP sections.

Enable/Disable Application Service Version Using MIB

To enable the application service version through MIB, add the APPVER option to TA_OPTIONS in T_DOMAIN class. For example:

```
SRVCNM   .TMIB

TA_OPERATION   SET

TA_CLASS       T_DOMAIN

TA_OPTIONS     APPVER,LAN
```

To disable the application service version through MIB, remove the APPVER option from TA_OPTIONS in T_DOMAIN class. For example:

```
SRVCNM   .TMIB

TA_OPERATION   SET

TA_CLASS       T_DOMAIN

TA_OPTIONS     LAN
```

Note: To disable the application service version through MIB, the user should remove the configured application service version related configuration firstly. For more information, see

UBB Config File Application Service Version Configuration

Three attributes (REQUEST_VERSION, VERSION_POLICY and VERSION_RANGE), are used in configuration files to specify what version and what allowable version range in a configured Tuxedo management entity. These three attributes can be configured in the *GROUP and *RESOUCEsection of the UBB Config File as shown in Listing 16-2

For more information, see UBBCONFIG(5), Section 5 - File Formats, Data Descriptions, MIBs, and System Processes Reference in the Oracle Tuxedo Reference Guide.

Listing 16-2 UBB Config File Application Service Version Configuration

```
*RESOUCE
DOMAINID LOCALDOM
OPTIONS  LAN,APPVER
REQUEST_VERSION 1 VERSION_RANGE  "1-2"
*GROUP
GRP1  GRPNO=1 REQUEST_VERSION=2
VERSION_POLICY="PROPAGATE"
GRP2 GRPNO=2 VERSION_RANGE="3-4"
GRP3 GRPNO=3 REQUEST_VERSION=3  VERSION_RANGE="1-3"
DMGRP   GRPNO=4 LMID=SITE1
GWGRP   GRPNO=5 LMID=SITE1
WSGRP GRPNO=6 LMID=SITE1 REQUEST_VERSION=4
JGRP GRPNO=7 LMID=SITE1 REQUEST_VERSION=3
*SERVER
SERVER1 SVRGRP=GRP1
SERVER2 SVRGRP=GRP2
SERVER3 SVRGRP=GRP3
DMADM  SRVGRP=DMGRP
GWADM  SRVGRP=GWGRP
GWTDOMAIN SRVGRP=GWGRP
WSL SRVGRP=WSGRP
JSL SRVGRP=JGRP
```

server1 advertises SVC2, SVC3. Because server1 belongs to GRP1, the REQUEST_VERSION of the server1, SVC2, SVC3 is inherited from GRP1. The configured REQUEST_VERSION of the GRP1 is 2, so REQUEST_VERSION of the server1, SVC2, SVC3 is 2.

The VERSION_RANGE, VERSION_POLICY of SVC2, SVC3 are inherited from GRP1. There is no configured VERSION_RANGE for GRP1, so it inherit from the *RESOURCE section, which is "1-2"

The VERSION_POLICY of SVC2, SVC3 are inherited from GRP1. The configured VERSION_POLICY of GRP1 is PROPAGATE, so the VERSION_POLICY of SVC2, SVC3 is PROPAGATE.

server2 advertises SVC1, SVC2, SVC3. According to the same rule described for server1, the REQUEST_VERSION of server2, SVC1, SVC2, SVC3 is 1, the VERSION_RANGE of SVC1, SVC2, SVC3 are "3-4", the VERSION_POLICY of the SVC1, SVC2, SVC3 is non-PROPAGATE.

server3 advertises SVC1, SVC2. According to the same rule described for server1, the REQUEST_VERSION of the server3, SVC1, SVC2 is 3, the VERSION_RANGE of SVC1, SVC2 are "1-3", the VERSION_POLICY of the SVC1, SVC2 is non-PROPAGATE.

If a native client joins the application without specifying the group name, its REQUEST_VERSION is 1.

If a native client joins the application with a specific group name, such as GRP3, its REQUEST_VERSION is 3.

If a /WS client joins the application, its REQUEST_VERSION is determined by the WSL, whose REQUEST_VERSION is 4 according to the UBB config file. So the REQUEST_VERSION of the /WS client is 4.

If a JOLT client joins the application, its REQUEST_VERSION is determined by the JSL, whose REQUEST_VERSION is 3 according to the UBB config file. So the REQUEST_VERSION of the /WS client is 4.

Domain Configuration File Application Service Version Configuration

Listing 16-3 shows a domain configuration file application service configuration example.

Listing 16-3 Domain Configuration File Application Service Version Configuration

```
*DM_LOCAL
LOCALDOM TYPE=TDOMAIN
DOMAINID="LOCALDOM"
```

```
*DM_REMOTE
REMOTEDOM1   TYPE=TDOMAIN
DOMAINID= "DOM1" MTYPE="Linux"
REMOTEDOM2   TYPE=TDOMAIN
DOMAINID= "DOM2" MTYPE="Linux"
REQUEST_VERSION=4
*DM_IMPORT
R_SVC1   RDOM= REMOTEDOM1  VERSION_RANGE="1-3"
R_SVC2   RDOM= REMOTEDOM2  VERSION_RANGE="4-6"
R_SVC3   RDOM= REMOTEDOM2
```

No REQUEST_VERSION is configured for REMOTEDOM1, so the domain gateway will propagate the request version of all the requests come from REMOTEDOM1, I.e the domain gateway will not change the incoming request version.

The REQUEST_VERSION of the REMOTEDOM2 is configured as 4, so the domain gateway will change the request version of all the requests come from REMOTEDOM2 to 4.

The LOCALDOM import R_SVC1 service from REMOTEDOM1 and specify the VERSION_RANGE as "1-3". So the VERSION_RANGE of the R_SVC1 service in the LOCALDOM is "1-3".

The LOCALDOM import R_SVC2 service from REMOTEDOM2 and specify the VERSION_RANGE as "4-6". So the VERSION_RANGE of the R_SVC2 service in the LOCALDOM is "4-6".

The LOCALDOM import R_SVC3 service without specified VERSION_RANGE. Because the VERSION_RANGE of the imported service is still determined by VERSION_RANGE configuration of the *GROUP and *RESOURCE, the VERSION_RANGE of the *RESOUCE is "1-2", so the VERSION_RANGE of R_SVC3 is "1-2".

For more information, seeUBB Config File Application Service Version Configuration.

Version Based Routing

When the application service feature is enabled, the system dispatches the request to the service according to both the service name and the version range of the service. We call the this mechanism as Version Based Routing (VBR). When a service entry matching the requested service name is found, the VBR is used to further routing decision.

VBR only does a simple numeric comparison using current request version number with the two boundary values of version range. VBR return "no entry is found" error to the caller when all of services with matching name are not allowable for this versioned request.

Tuxedo already offers several routing mechanisms: DDR (Data Dependent Routing), TAR (Transaction Affinity Routing), and RAR (Request Affinity Routing). VBR (Version Based Routing) is also a new routing mechanism that can owns same functions as these of existing routing algorithms.

VBR can be used together with the other routing mechanisms; Tuxedo will choose the services that match all criteria if there are multiple routing mechanisms. But the user had better to understand how the interaction among these routing mechanisms if use them together.

Suppose the configuration describe as above section.

1. If `server3` needs to call `SVC2` during its initialization period, The `REQUEST_VERSION` of server3 is `3`, the candidate services are:

   ```
   Server1:SVC2 1-2
   Server2:SVC2 3-4
   ```

   So `server3` will call `Server2:SVC2`.

2. If the native client need to call SVC3, the `REQUEST_VERSION` of the native client is 1, the candidate services are:

   ```
   Server1:SVC1 1-2
   Server2:SVC1 3-4
   ```

   So the native client will call `Server1:SVC1`

3. If `Server1:SVC1` needs to call `SVC3`, the `SVC1` will propagate the incoming `REQUEST_VERSION`, in this case the incoming `REQUEST_VERSION` is `1`, so the current `REQUEST_VERSION` of `Server1:SVC1` is `1`, the candidate services are:

   ```
   Server2:SVC3 3-4
   Server3:SVC3 1-3
   ```

   So `Server1:SVC1` calls `Server3:SVC3`

4. If a request come from `REMOTEDOM2`, suppose the original `REQUEST_VERSION` is `6`, then the `REQUEST_VERSION` of the incoming request is changed to `4`.

5. If a request comes from `REMOTEDOM1`, suppose the original `REQUEST_VERSION` is `2`, then the REQUEST_VERSION of the incoming request will still be `2`.

Resetting the User Configured Service Version Information Using MIB

You can configure the REQUEST_VERSION, VERSION_RANGE or VERSION_POLICY in the *GROUP or *RESOURCE section in the UBB Config file. The low-level configuration overrides the high level-configuration.

If there is no user configured service version configuration at any level, the system uses the default value. So the result will be very different for the user configured configuration and default value. If the user modifies the REQUEST_VERSION, VERSION_RANGE or VERSION_POLICY through MIB, it is the user configured service version configuration. It is necessary to provide a method to reset this modification to the default value through MIB, otherwise you cannot restore the UBB config file to its original state.

To reset the REQUEST_VERSION, VERSION_RANGE or VERSION_POLICY to default value, you just must set value as DEFAULT.

For example the you can modify the REQUEST_VERSION through MIB as shown in Listing 16-4

Listing 16-4 Resetting the User Configured Service Version Information Using MIB

```
SRVCNM  .TMIB
TA_OPERATION    SET
TA_CLASS        T_GROUP
TA_SRVGRP       APPGRP1
TA_GRPNO        1
TA_CURLMID      SITE1
TA_REQUEST_VERSION      4
Then the user reset the REQUEST_VERSION to default value through MIB:
SRVCNM  .TMIB
TA_OPERATION    SET
TA_CLASS        T_GROUP
TA_SRVGRP       APPGRP1
TA_GRPNO        1
TA_CURLMID      SITE1
TA_REQUEST_VERSION      DEFAULT
```

CHAPTER 17

Applying Service Version to Tuxedo Applications

This topic contains the following sections:

Overview

It is common that the user wants to keep existing functionality but also want to add new functionality into a service as time goes by. To reduce the compatible risk, it is better to provide two different version services with the same service name, one for old functionality, and the other for new functionality. The old client can still use the existing functionality without changing any code while the new client can use the new functionality.

Application Service Versioning feature offer a configuration driven way which can be used by Tuxedo customers to plan, develop, test, scale, and deploy their Tuxedo applications in each stage. The user can use the version to partition current Tuxedo application into different virtual application domains, different virtual machines, and different virtual server groups on current

Tuxedo management hierarchy, so as to respond to various of special business access logics and on the other hand satisfy upgrading requirements in non-stop mode.

This feature supports for COBOL application and programming environment without requiring special changes for COBOL environment.

This feature supports FORWARD queue only for /Q. With Application Service Versioning enabled, when a client puts a message into FORWARD queue, the FORWARD queue forwards the queued message to the service that supports the client request version.

Enabling and Disabling Application Service Versioning

You can enable/disable the application service versioning feature using UBB configuration file or MIB.

Enable/Disable Application Service Versioning Using UBB Config File

To enable the application service versioning in UBB configuration file, add the APPVER option to the OPTIONS parameter in *RESOURCES section.

For example:

```
*RESOURCES

OPTIONS       APPVER, LAN
```

To disable the application service version in UBB configuration file, remove the APPVER option from the OPTIONS parameter in *RESOURCES section.

For example:

```
*RESOURCES

OPTIONS       LAN
```

Note: If the application service versioning is disabled, you cannot configure the application service versioning related attributes in *RESOURCE and *GROUP sections.

Enable/Disable Application Service Versioning Using MIB

To enable the application service versioning in MIB, add the APPVER option to TA_OPTIONS in the T_DOMAIN class.

For example:

```
SRVCNM   .TMIB
TA_OPERATION  SET
TA_CLASS      T_DOMAIN
TA_OPTIONS    APPVER,LAN
```

To disable the application service versioning in MIB, remove the APPVER option from TA_OPTIONS in the T_DOMAIN class.

For example:

```
SRVCNM   .TMIB
TA_OPERATION  SET
TA_CLASS      T_DOMAIN
TA_OPTIONS    LAN
```

Note: Before disabling the application service versioning, you should remove the application service versioning related options that were already configured in MIB. For more information, see Resetting the User Configured Service Version Information Using MIB.

Application Service Version Configurations

UBB Config File Configuration

Three attributes, REQUEST_VERSION, VERSION_POLICY, and VERSION_RANGE, are used in configuration files to specify the version and acceptable version range in a configured Tuxedo management entity. These three attributes can be configured in the *GROUPS and *RESOURCES section of the UBB configuration file, as shown in Listing 17-1

For more information, see UBBCONFIG(5)in Section 5 - File Formats, Data Descriptions, MIBs, and System Processes Reference in the Oracle Tuxedo Reference Guide.

Listing 17-1 UBB Config File Application Service Version Configuration

```
*RESOURCES
DOMAINID LOCALDOM
OPTIONS  LAN,APPVER
REQUEST_VERSION 1 VERSION_RANGE  "1-2"
*GROUPS
```

```
GRP1  GRPNO=1 REQUEST_VERSION=2
VERSION_POLICY="PROPAGATE"
GRP2 GRPNO=2 VERSION_RANGE="3-4"
GRP3 GRPNO=3 REQUEST_VERSION=3  VERSION_RANGE="1-3"
DMGRP   GRPNO=4 LMID=SITE1
GWGRP   GRPNO=5 LMID=SITE1
WSGRP GRPNO=6 LMID=SITE1 REQUEST_VERSION=4
JGRP GRPNO=7 LMID=SITE1 REQUEST_VERSION=3
*SERVERS
SERVER1 SVRGRP=GRP1
SERVER2 SVRGRP=GRP2
SERVER3 SVRGRP=GRP3
DMADM  SRVGRP=DMGRP
GWADM  SRVGRP=GWGRP
GWTDOMAIN SRVGRP=GWGRP
WSL SRVGRP=WSGRP
JSL SRVGRP=JGRP
```

Take Listing 17-1 for an example, application service version has the following rules:

- A server inherits the REQUEST_VERSION attribute from the group it belongs to. When a server advertises services, the services inherit version attributes from the group, to which the server belongs. If there is no version attribute value specified to the group, they will inherit upward from the attributes specified in *RESOURCES section. Based on this rule:
  - When server1 advertises SVC2 and SVC3, the REQUEST_VERSION, VERSION_RANGE, and VERSION_POLICY of SVC2 and SVC3 are 2, “1-2”, and PROPAGATE, respectively.
  - When server2 advertises SVC1, SVC2, and SVC3, the REQUEST_VERSION, VERSION_RANGE, and VERSION_POLICY of SVC1, SVC2, and SVC3 are 1, “3-4”, and non-PROPAGATE, respectively.
  - When server3 advertises SVC1 and SVC2, the REQUEST_VERSION, VERSION_RANGE, and VERSION_POLICY of SVC1 and SVC2 are 3, "1-3", and non-PROPAGATE, respectively.
- If a native client joins the application without specifying the group name, its REQUEST_VERSION is 1.

- If a native client joins the application with a specific group name, such as GRP3, its REQUEST_VERSION is 3.
- If a /WS client joins the application, its REQUEST_VERSION is determined by the WSL, whose REQUEST_VERSION is 4 according to the UBB config file. So the REQUEST_VERSION of the /WS client is 4.
- If a JOLT client joins the application, its REQUEST_VERSION is determined by the JSL, whose REQUEST_VERSION is 3 according to the UBB config file. So the REQUEST_VERSION of the JOLT client is 3.

Domain Config File Configuration

Listing 17-2 shows a domain configuration file application service version configuration example.

Listing 17-2 Domain Configuration File Application Service Version Configuration

```
*DM_LOCAL
LOCALDOM TYPE=TDOMAIN
DOMAINID="LOCALDOM"
*DM_REMOTE
REMOTEDOM1   TYPE=TDOMAIN
DOMAINID= "DOM1" MTYPE="Linux"
REMOTEDOM2   TYPE=TDOMAIN
DOMAINID= "DOM2" MTYPE="Linux"
REQUEST_VERSION=4
*DM_IMPORT
R_SVC1   RDOM= REMOTEDOM1  VERSION_RANGE="1-3"
R_SVC2   RDOM= REMOTEDOM2  VERSION_RANGE="4-6"
R_SVC3   RDOM= REMOTEDOM2
```

From Listing 17-2, the application service version are configured as follows:

- No REQUEST_VERSION is configured to REMOTEDOM1. Therefore the domain gateway will propagate the request version of all the requests come from REMOTEDOM1 without changing the incoming request version.

- The REQUEST_VERSION of the REMOTEDOM2 is configured to 4. Therefore the domain gateway will change the request version of all the requests come from REMOTEDOM2 to 4.
- The LOCALDOM imports R_SVC1 service from REMOTEDOM1 and specifies the VERSION_RANGE to "1-3". Therefore the VERSION_RANGE of the R_SVC1 service in the LOCALDOM is "1-3".
- The LOCALDOM imports R_SVC2 service from REMOTEDOM2 and specifies the VERSION_RANGE to "4-6". Therefore the VERSION_RANGE of the R_SVC2 service in the LOCALDOM is "4-6".
- The LOCALDOM imports R_SVC3 service without specifying VERSION_RANGE. Because the VERSION_RANGE of the imported service is still determined by VERSION_RANGE configuration of the *GROUPS and *RESOURCES, the VERSION_RANGE of the *RESOURCES is "1-2", and the VERSION_RANGE of R_SVC3 is "1-2".

For more information, see UBB Config File Configuration.

Version Based Routing

When the application service versioning feature is enabled, the system dispatches the requests to the service according to both the service name and service version range. We call this mechanism Version Based Routing (VBR). When a service entry matching the requested service name is found, VBR is used for further routing decision.

VBR only does a simple numeric comparison using the current request version number with two boundary values of version range. VBR returns "no entry is found" error to the caller when all the services with the matching name are not allowable for this versioned request.

As a routing mechanism, VBR functions the same as the existing routing mechanisms, like DDR (Data Dependent Routing), TAR (Transaction Affinity Routing), and RAR (Request Affinity Routing).

VBR can work with other routing mechanisms. Oracle Tuxedo chooses the services that match all criteria if there are multiple routing mechanisms. It is recommended you are clear about the interoperability among these routing mechanisms before using them together.

Using Listing 17-1 as an illustration:

- Suppose server3 needs to call SVC2 during the initializing period, so the candidate services are:

```
Server1:SVC2 1-2
Server2:SVC2 3-4
```

Since the REQUEST_VERSION of server3 configured in Listing 17-1 is 3, the server3 will call Server2:SVC2.

- Suppose the native client needs to call SVC3, so the candidate services are:

  Server1:SVC3 1-2

  Server2:SVC3 3-4

  Since the REQUEST_VERSION of the native client configured in Listing 17-1 is 1, the native client will call Server1:SVC3

- Suppose the native client calls Server1:SVC1 and Server1:SVC1 needs to call SVC3, so the candidate services are:

  Server2:SVC3 3-4

  Server3:SVC3 1-3

  As configured in Listing 17-1, the Server1:SVC1 will propagate the incoming REQUEST_VERSION which is 1, as a result the REQUEST_VERSION of Server1:SVC1 will become to 1, rather than its own REQUEST_VERSION 2, therefore the Server1:SVC1 will call Server3:SVC3

- If a request comes from REMOTEDOM2, suppose the original REQUEST_VERSION is 6, then the REQUEST_VERSION of the incoming request is changed to 4.

- If a request comes from REMOTEDOM1, suppose the original REQUEST_VERSION is 2, then the REQUEST_VERSION of the incoming request is still 2.

Resetting the User Configured Service Version Information Using MIB

You can configure REQUEST_VERSION, VERSION_RANGE, and VERSION_POLICY in the *GROUPS or *RESOURCES section of UBB config file. The low-level configuration overrides the high level-configuration.

If there is no user-configured service version configuration at any level, the system uses the default value. That causes the result very different for the user configured configuration and default value. If you modify the REQUEST_VERSION, VERSION_RANGE or VERSION_POLICY using MIB, it is the user-configured service version configuration. It is necessary to provide a method to reset this modification to the default value using MIB, otherwise you cannot restore the UBB config file to its original state through MIB operation.

To reset the REQUEST_VERSION, VERSION_RANGE, and VERSION_POLICY to default value, you just need to simply set value as DEFAULT.

For example, modify the REQUEST_VERSION in MIB as shown in Listing 17-3.

Listing 17-3 Resetting the User Configured Service Version Information Using MIB

```
SRVCNM  .TMIB
TA_OPERATION    SET
TA_CLASS        T_GROUP
TA_SRVGRP       APPGRP1
TA_GRPNO        1
TA_CURLMID      SITE1
TA_REQUEST_VERSION      4
Then the user reset the REQUEST_VERSION to default value through MIB:
SRVCNM  .TMIB
TA_OPERATION    SET
TA_CLASS        T_GROUP
TA_SRVGRP       APPGRP1
TA_GRPNO        1
TA_CURLMID      SITE1
TA_REQUEST_VERSION      DEFAULT
```

Interoperability

You can control how the JCA/WTC/old Tuxedo domain interoperates with the new Tuxedo domain using the domain configuration file, as follows:

- control the requests coming from JCA/WTC/old Tuxedo domain by setting the REQUEST_VERSION and VERSION_POLICY attributes in the DM_REMOTE section
- control the requests going to JCA/WTC/old Tuxedo domain by setting VERSION_RANGE in the DM_IMPORT section.

If the request coming from the old Tuxedo domain enters in the Tuxedo 12c domain which has no REQUEST_VERSION configured for the corresponding remote domain, the request version is changed to 0.

The default request version is 0-65535, which means the new domain can call all the imported service from JCA/WTC/old Tuxedo domain by default.

In MP environment, if a local client runs on a machine that is installed the old version Tuxedo, the client can call any version service because there is no version control for the old version Tuxedo.

Likewise, for the /WS or Jolt client connecting to an old version WSL or JSL server, there is no version control for them.

Oracle Tuxedo Applications Packaging and Deployment

This topic contains the following sections:

Overview

This feature provides a centralized control platform to allow users to automatically deploy/undeploy one Tuxedo application (domain) on different remote machines using a set of new commands on the master node of domain. Deployment process typically contains several steps: application packages distribution, Tuxedo system environment setup, Tuxedo configuration, Tuxedo system booting and so on.

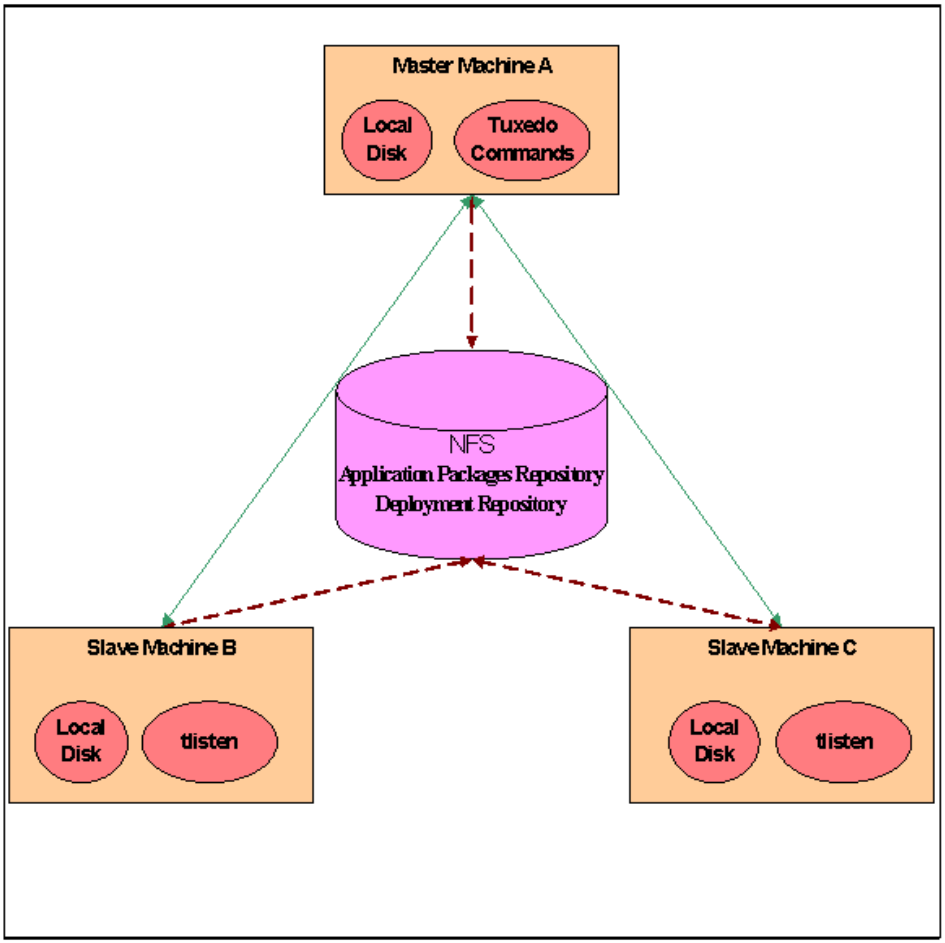
Components

This feature contains the following main components:

- Application Packages Repository
- Deployment Repository
- New-added Tuxedo commands `tmcrdom` and `tmdeldom`
- Enhanced Tuxedo `tlisten daemon`.

The relationship among these components is illustrated in Figure 18-1:

Figure 18-1 Applications Packaging and Deployment Component Relationship



- Application Packages Repository: It is a place in NFS (Network File System). Users need to maintain (add/delete) their Application Packages in this place by themselves. This place can be accessed by all the compute nodes which will be deployed the Tuxedo Application to.
- Deployment Repository: It is a place in NFS (Network File System). This feature will maintain all the domains' information generated by this feature itself in this place. This place can be accessed by all the compute nodes which will be deployed the Tuxedo Application to.

- New-added Tuxedo Commands: Two new added Tuxedo commands, which are responsible for creating domain, configuring domain, connecting with tlisten to do the deployment tasks and so on.
- Enhanced tlisten daemon: This daemon will do the real deployment tasks on every compute node once it receives the Tuxedo deployment command notification.

To use this feature, first, the users need to allocate two big enough disk space or something like this to act as the Application Packages Repository and Deployment Repository on the NFS. And make sure the two places are accessible by all the Tuxedo System such as its new added commands, tlisten and so on. Second, the Tuxedo System must have been installed on every compute node which the Tuxedo Application will be deployed to. Last, you need to start up tlisten on every compute nodes. All the Tuxedo servers' running dependent libraries, such as DataBase client libraries, are not in this feature's deployment scope, it needs the customers themselves to assure their availabilities.

Constraints

This feature doesn't support to deploy machine level ENVFILE (which is specified in UBBCONFIG MACHINES section by ENVFILE parameter), groups level ENVFILE, servers level ENVFILE, servers level RCMD file to other directories than APPDIR.

The JAVA JDK has been installed on the master node on which will run the tmcrdom command.

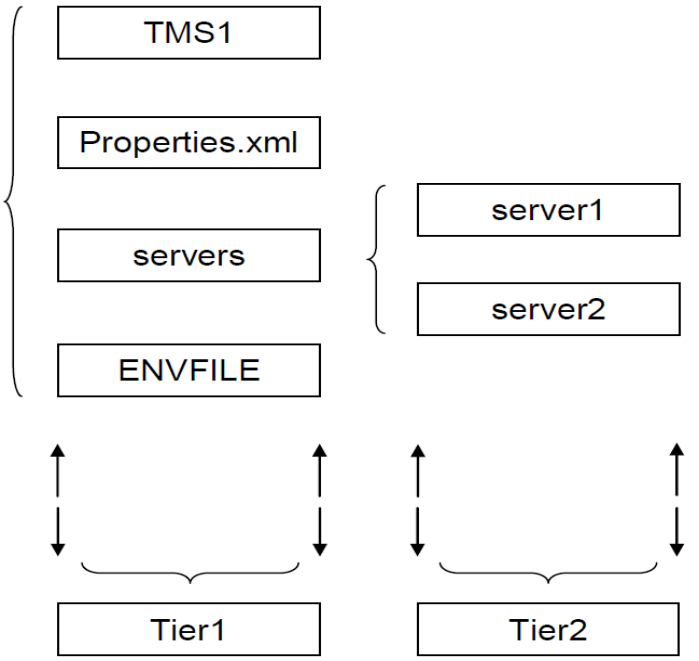
How to Deploy/Undeploy Tuxedo Applications

Introduction to Application Package Organization and Contents

A Tuxedo application (domain), as defined in a TUXCONFIG (UBBCONFIG) configuration file, is the set of machines, groups, servers, and other resources. It can exist on a single machine or cross multiple network-connected machines. For this feature, to deploy the whole Tuxedo application, users need to add their Application Packages to the Application Packages Repository by themselves first. One Application Package is an entity which holds all the binary or non-binary files referenced by one or more groups defined in a UBBCONFIG file. That means, every Application Package is mapped to one or more groups defined in a UBBCONFIG file, and all these groups must belong to the same machine in UBBCONFIG. So a Tuxedo application can consist of one or more application packages. Also every Application Package can be repeatly deployed to one or more domains.

The application package is a user generated .zip file and can have several tiers in it. For example:

Figure 18-2 Application Package



Every Application Package is required to contain a file named "Properties.xml" in Tier1. This file is a group level's part UBBCONFIG file. It contains some properties in `GROUPS`, `RMS`, `SERVERS`, and `SERVICES` sections of a complete UBBCONFIG file and is mainly used to describe the relationship and parameters of all the servers within this package's groups. The Properties.xml file will be used to generate the ultimate UBBCONFIG file when decide to deploy this package to a machine using tmcrdom and its content can be modified in the according deployment plan.

All the items in the `GROUPS`, `RMS`, `SERVERS`, `SERVICES` section of UBBCONFIG are divided into four categories:

- Mandatory and Changeable. It is a kind of index value. The user must fill it in the Properties.xml file, but it will be replaced by the value specified in the deployment plan when using tmcrdom command.
- Optional and Unchangeable. If needed, it must be filled by the user in the Properties.xml file. And it will not allow to modify it when assemble the UBBCONFIG. That means, this

kind of parameter can only appear in the Properties.xml if needed, and can not appear in the deployment plan.

- Mandatory and Unchangeable. It must be filled by the user in the Properties file, and its value will not be modified when assemble the UBBCONFIG.
- Forbidden. It can't be filled in the Application Packages' Properties.xml file. The user will fill it in the deployment plan when assemble the UBBCONFIG if needed.

Table 18-1 GROUPS Section Properties

GROUPS Section	
Items	**Category**
`GROUPNAME`	Mandatory and Changeable
`GRPNO = number`	Mandatory and Changeable
`LMID = string_value1 [,string_value2]`	Forbidden
`ENVFILE = string_value`	Optional and Unchangeable ENVFILE must be a relative path of application package, tmcrdom will add absolute path to it.
`TMSNAME = string_value`	Optional and Unchangeable
`MRM = {Y | N}`	Optional and Unchangeable
`SIGNATURE_REQUIRED = {Y | N}`	Optional and Unchangeable
`ENCRYPTION_REQUIRED = {Y | N}`	Optional and Unchangeable
`OPENINFO = string_value`	Forbidden
`CLOSEINFO = string_value`	Forbidden
`TMSCOUNT = number`	Forbidden
`SEC_PRINCIPAL_NAME = string_value`	Forbidden
`SEC_PRINCIPAL_LOCATION = string_value`	Forbidden

Table 18-1 GROUPS Section Properties

GROUPS Section	
Items	**Category**
`SEC_PRINCIPAL_PASSVAR = string_value`	Forbidden
`REQUEST_VERSION`	Forbidden
`VERSION_RANGE`	Forbidden
`VERSION_POLICY`	Forbidden

Table 18-2 RMS Section Properties

RMS Section	
Item	**Category**
`RMSNAME`	Mandatory and Changeable
`RMID = number`	Mandatory and Changeable
`TMSNAME = string_value`	Optional and Unchangeable
`OPENINFO = string_value`	Forbidden
`CLOSEINFO = string_value`	Forbidden
`TMSCOUNT = number`	Forbidden
`AUTO = {Y | N}`	Forbidden
`SRVGRP`	Forbidden

Table 18-3 SERVERS Section Properties

SERVERS Section	
Item	**Category**
`SRVID = number`	Mandatory and Changeable
`AOUT`	Mandatory and Unchangeable. It must be a relative path of application package, tmcrdom will add absolute path to it.
`ENVFILE = string_value`	Optional and Unchangeable. It must be a relative path of application package, tmcrdom will add absolute path to it..
`RCMD = string_value`	Optional and Unchangeable. It must be a relative path of application package, tmcrdom will add absolute path to it..
`CONV = {Y | N}`	Optional and Unchangeable
`CLOPT = string_value`	Forbidden
`SEQUENCE = number`	Forbidden
`RQADDR = string_value`	Forbidden
`MIN = number`	Forbidden
`MAX = number`	Forbidden
`RQPERM = number`	Forbidden
`REPLYQ = {Y | N}`	Forbidden
`RPPERM = number`	Forbidden
`MAXGEN = number`	Forbidden
`GRACE = number`	Forbidden
`RESTART = {Y | N}`	Forbidden

Table 18-3 SERVERS Section Properties

SERVERS Section	
Item	**Category**
`SYSTEM_ACCESS = identifier[,identifier]`	Forbidden
`MAXDISPATCHTHREADS = number`	Forbidden
`MINDISPATCHTHREADS = number`	Forbidden
`THREADSTACKSIZE = number`	Forbidden
`SEC_PRINCIPAL_NAME = string_value`	Forbidden
`SEC_PRINCIPAL_LOCATION = string_value`	Forbidden
`SEC_PRINCIPAL_PASSVAR = string_value`	Forbidden
`SICACHEENTRIESMAX = string_value`	Forbidden
`CONCURR_STRATEGY=PER_REQUEST`	Forbidden
`CONCURR_STRATEGY = PER_OBJECT`	Forbidden

Table 18-4 SERVICES Section Properties

SERVICES Section	
Item	**Category**
`SVCNM`	Mandatory and Unchangeable
`BUFTYPE = "type1[:subtype1[,subtype2 . . . ]][;type2[:subtype3[, . . . ]]] . . ."`	Optional and Unchangeable
`SIGNATURE_REQUIRED = {Y | N}`	Optional and Unchangeable
`ENCRYPTION_REQUIRED = {Y | N}`	Optional and Unchangeable

Table 18-4 SERVICES Section Properties

SERVICES Section	
Item	**Category**
`LOAD = number`	Forbidden
`PRIO = number`	Forbidden
`BUFTYPECONV = {XML2FML | XML2FML32}`	Optional and Unchangeable
`BLOCKTIME numeric_value`	Forbidden
`SVCTIMEOUT = number`	Forbidden
`SESSIONROLE`	Forbidden
`AFFINITYSCOPE`	Forbidden
`AFFINITYSTRICT`	Forbidden
`AUTOTRAN = {Y | N}`	Forbidden
`ROUTING = string_value`	Forbidden
`TRANTIME = number`	Forbidden

In the Properties.xml file, users must assure all the parameters' references are valid. For example, the GROUP which is indicated by SRVGRP in the SERVICES section must have already been defined in this Properties.xml file's GROUPS section or it will report error when system generates the UBBCONFIG.

Besides the items of the GROUPS, RMS, SERVERS, SERVICES section of UBBCONFIG in Properties.xml, this file also contains some package global attributes at the beginning:

Table 18-5 Description Information of the Package

Item	Description
`PackageName`	The global unique application package name, for example, APP1.zip.
`TuxedoVersion`	The Tuxedo version this package is built on. This item is checked when assembling Tuxedo domain to decide if the package is suitable for certain Tuxedo installation.

Table 18-5 Description Information of the Package

Item	Description
SupportedOS	The Operation System this package can be deployed to. This information is compared with the corresponding item in Machine list entry when deploying the package. The possible values are Linux, SunOS, AIX, and HP-UX.
TuxedoWordSize	The Tuxedo word size this package is built on. The value can be 32 or 64 (bit). This information is compared with the corresponding item in Machine list entry when deploying the package.
MachineArch	The machine architecture this package can apply to. The possible values are: x86_64, SUNW, powerpc, and IA64. This information is compared with the corresponding item in Machine list when deploying the package.
LibPath	The path where the library locates in the package, if there is a library in it.

Actually the values of these package global attributes will not be checked by Tuxedo tmcrdom/tmdeldom commands. The reason to keep these values in the Properties.xml is that, the contents requirement of the Application Package are the same with the similar Tuxedo EM function. So we suggest you to fill these values as the table described, then when you want to use the Tuxedo EM deploy/undeploy function, you can use these packages directly without any change.

For more information, please refer to Properties.xml Schema.

Uploading/Deleting an Application Package

User can add/delete the Application Packages to/from the Application Package Repository freely by themselves. That means it's the users' duty to maintain the Application Packages Repository.

Creating and Deploying a Domain

Users need to use `tmcrdom` to create and deploy the domain. This command must be run on the master node of the domain which will be created by this command.

Before running this command, the customer need to export their `JAVA_HOME` and `JVMLIBS` environment.

The command syntax is:

```
tmcrdom -d "domain_name" -f "deployment_plan"
```

Where:

- "domain_name" will be checked. If the domain has already existed in the Deployment Repository, it will report error, or it will create this domain in the repository.
- "deployment_plan" is a text file. This feature assumes that it has already been created by the customer before invoking this command. It contains the deployment information needed by this domain. It can be specified in the command as an absolute path or relative path on the master node. Its format is similar to UBBCONFIG.

Note: In some scenarios, although tmcrdom execution fails, it actually has created a domain in the Deployment Repository, thus if you create the domain again you will receive an error message saying the domain already exists. To solve this, you need to delete the domain using the tmdeldom command with -f option before you can create the domain.

deployment_plan

A deployment_plan file is made up of nine possible specification sections. Allowable section names are:

- RESOUCES Section
- MACHINES Section
- GROUPS Section
- RMS Section
- NETGROUPS Section
- NETWORK Section
- SERVERS Section
- ROUTING
- SERVICES Section

RESOUCES Section

Required parameters are:

`IPCKEY numeric_value`
Refer to the same parameter in UBBCONFIG.

`MASTER string_value1[,string_value2]`
Refer to the same parameter in UBBCONFIG.

`MODEL {SHM | MP}`
Refer to the same parameter in UBBCONFIG.

`APPREPOSITORY string_value`
It specifies the repository path which the Application Packages will be put on. This parameter is required. The place which is specified by this parameter must be NFS accessible. The string length can't be larger than `256`. It must be an absolute path.

`DEPREPOSITORY string_value`
It specifies the repository path which all the domain deployment information will be put. This parameter is required. The place which is specified by this parameter must be NFS accessible. The string length can't be larger than `256`. It must be an absolute path.

All other parameters are same as `RESOURCES` section parameters in UBBCONFIG.

MACHINES Section

It's same as the UBBCONFIG's MACHINES section. It must be included in the deployment plan. Its contents will be used as the MACHINES section in the ultimate UBBCONFIG.

Besides these, this feature introduce some new parameters:

`CONFIGSCRIPT = string_value[0..256]`
It specifies the configuration script which will be run on each compute node before booting the Tuxedo System. It must be an absolute path on the master node. The tmcrdom command will copy this script to the deployment repository specified by DEPREPOSITORY and deploy it to the destination machine later. After deployed to the destination machine, it will be run under the destination machine's APPDIR.

`BOOTSCRIPT = string_value[0..256]`
It specifies the booting script which will be run on the mater node to boot the Tuxedo System. It must be an absolute path on the master node. The tmcrdom command will copy this script to the deployment repository specified by DEPREPOSITORY and deploy it to the destination machine later. After deployed to the destination machine, it will be run under the destination machine's APPDIR.

`SHUTDOWNSCRIPT = string_value[0..256]`
It specifies the shutdown script which will be run on the mater node to shutdown the Tuxedo System. It must be an absolute path on the master node. The tmcrdom command will copy this script to the deployment repository specified by DEPREPOSITORY and deploy it to the destination machine later. After deployed to the destination machine, it will be run under the destination machine's APPDIR.

`UNCONFIGSCRIPT = string_value[0.. 256]`
It specifies the unconfigure script which will be run on each compute node after the Tuxedo System is shutdown. It must be an absolute path on the master node. The tmcrdom

command will copy this script to the deployment repository specified by DEPREPOSITORY and deploy it to the destination machine later. After deployed to the destination machine, it will be run under the destination machine's APPDIR.

For every machine, machine address, `LMID`, `TUXCONFIG`, `TUXDIR` and `APPDIR` are required. For master machine, `CONFIGSCRIPT`, `BOOTSCRIPT` and `SHUTDOWNSCRIPT` are required. For slave machine, `BOOTSCRIPT` and `SHUTDOWNSCRIPT` are forbidden.

Note: The scripts specified by `CONFIGSCRIPT`, `BOOTSCRIPT`, `SHUTDOWNSCRIPT`, and `UNCONFIGSCRIPT` must be able to be found by `tmcrdom` on the master machine where the command runs.

GROUPS Section

Required parameters are:

`GROUPNAME`
Refer to the same parameter in UBBCONFIG. This group name will be used in the ultimate UBBCONFIG.

`LMID = string_value1 [,string_value2]`
Refer to the same parameter in UBBCONFIG.

`GRPNO = number`
Refer to the same parameter in UBBCONFIG. This group number will be used in the ultimate UBBCONFIG.

Optional parameters are:

- `OPENINFO = string_value`
- `CLOSEINFO = string_value`
- `TMSCOUNT = number`
- `SEC_PRINCIPAL_NAME = string_value [0..511]`
- `SEC_PRINCIPAL_LOCATION = string_value [0..1023]`
- `SEC_PRINCIPAL_PASSVAR = string_value [0..31]`
- `REQUEST_VERSION = { numeric_value| * }`
- `VERSION_RANGE = string_value`
- `VERSION_POLICY = string_value { PROPAGATE }`

For information about above parameters, refer to the same parameters in UBBCONFIG.

These parameters below are new for the Optional parameters:

`PAKNAME = string_value[0..256]`

It associates this group with one Application Package. That means, this Application Package will be deployed to the machine which is specified by this group's LMID parameter. If PAKNAME parameter is specified, the GROUPS parameters which can be specified in the properties.xml file like:ENVFILE, TMSNAME, MRM, SIGNATURE_REQUIRED, ENCRYPTION_REQUIRED can't appear in this group definition in the deployment plan, their values will come from the Application Package's properties.xml file. If PAKNAME is not specified, this group is a normal one. Then all the parameters in this group will be the same with the UBBCONFIG. We recommend the customers to use this normal group to hold their Tuxedo System level servers, for example, GWADM.

If PAKNAME is specified, PAKGRPNAME and PAKINSTANCE must also be specified for this group entry.

`PAKGRPNAME= string_value [1..30]`

It specifies the logical name of the group in the Application Package, and the Application Package's name is specified by PAKNAME parameter. It cannot contain an asterisk (*), comma, or colon. If a non-empty value is specified, PAKNAME and PAKINSTANCE must also be specified for this group entry. Two group entries with the same PAKNAME and PAKINSTANCE cannot have the same PAKGRPNAME. That means two group entries cannot be associated with one same group entry in one Application Package.

`PAKINSTANCE = number [1...30000]`

It specifies the index for the same Application Package. This is for the situation that one Application Package being deployed to the same domain for more than once. This number must be greater than 0 and less than 30000. If PAKINSTANCE is specified, PAKNAME and PAKGRPNAME must also be specified.

If PAKNAME is not specified, the parameters below are the same as the parameters in UBBCONFIG:

- `ENVFILE = string_value[0..256] (up to 78 bytes for Oracle Tuxedo 8.0 or earlier)`
- `TMSNAME = string_value[0..256] (up to 78 bytes for Oracle Tuxedo 8.0 or earlier)`
- `MRM = {Y | N}`
- `SIGNATURE_REQUIRED = {Y | N}`
- `ENCRYPTION_REQUIRED = {Y | N}`

RMS Section

If one rms entry belongs to a group which is not associated with any Application Package, all the parameters are same as those in UBBCONFIG.

If not:

Required parameters are:

`RMSNAME`
: Refer to the same parameter in UBBCONFIG. This rms entry name will be used in the ultimate UBBCONFIG.

`SRVGRP = string_value`
: Refer to the same parameter in UBBCONFIG. All the entries in this RMS section which belong to this group must be associated with one Application Package group's all rms entries, or it will report error. The according group in the Application Package is specified by the group's PAKGRPNAME in the deployment plan.

`RMID = number`
: Refer to the same parameter in UBBCONFIG.

`PAKRMID = number`
: This specifies the RMID appears in the Application Package. It must be between 1 and 31 inclusive. The rms entries belong to the same group cannot have the same PAKRMID. If one RMS entry belong to a group which has PAKNAME defined, it must specify PAKRMID parameter in the deployment plan, or it will report error.

Optional parameters are:

- `TMSCOUNT = number`
- `OPENINFO = string_value`
- `CLOSEINFO = string_value`
- `AUTO = {Y | N}`

For information about above parameters, refer to the same parameters in UBBCONFIG.

`TMSNAME = string_value[0..256]` can't be specified in the deployment plan. It will be from the properties.xml file.

NETGROUPS Section

Same as NETGROUPS section in UBBCONFIG. It must be included in the deployment plan if customer needs it in the ultimate generated UBBCONFIG. Its contents will be used as the NETGROUPS section in the ultimate UBBCONFIG.

NETWORK Section

Same as NETWORK section in UBBCONFIG. It must be included in the deployment plan. Its contents will be used as the NETWORK section in the ultimate UBBCONFIG.

SERVERS Section

If one server belongs to a group which is not associated with any Application Package, then all the parameters are the same with UBBCONFIG.

If not:

Required parameters are:

`AOUT`
: It must be the same with the aout in the corresponding Application Package's properties.xml.

`SRVGRP = string_value`
: Refer to the same parameters in UBBCONFIG.

`SRVID = number`
: Refer to the same parameters in UBBCONFIG. This value will be used in the ultimate UBBCONFIG.

`PAKSRVID = number`
: It specifies the SRVID in the Application Package's properties file. Every server in the Application Package's properties file must be associated with one entry in this section in the deployment plan.
If one server belongs to a group which has PAKNAME defined, then PAKSRVID must be specified for this server. And two servers belong to the same group can not have the same PAKSRVID. If one server belongs to a normal group which has no PAKNAME defined, then it can not define PAKSRVID parameter.

Optional parameters listed below are same as those in UBBCONFIG:

- `CLOPT = string_value`
- `SEQUENCE = number`
- `MIN = number`
- `MAX = number`
- `RQADDR = string_value`
- `RQPERM = number`
- `REPLYQ = {Y | N}`
- `RPPERM = number`

- MAXGEN = number
- GRACE = number
- RESTART = {Y | N}
- SYSTEM_ACCESS = identifier[,identifier]
- MAXDISPATCHTHREADS = number
- MINDISPATCHTHREADS = number
- THREADSTACKSIZE = number
- SEC_PRINCIPAL_NAME = string_value [0..511]
- SEC_PRINCIPAL_LOCATION = string_value [0..1023]
- SEC_PRINCIPAL_PASSVAR = string_value [0..31]
- SICACHEENTRIESMAX = string_value
- CONCURR_STRATEGY=PER_REQUEST
- CONCURR_STRATEGY = PER_OBJECT

The parameters listed below cannot appear in this section, the ultimate UBBCONFIG will keep the properties.xml file value.

- ENVFILE = string_value[0..256]
- RCMD = string_value[0..256]
- CONV = {Y | N}

ROUTING

It's same as UBBCONFIG's ROUTING section. It must be included in the deployment plan if customer needs it in the ultimate generated UBBCONFIG. Its contents will be used as the ROUTING section in the ultimate UBBCONFIG.

SERVICES Section

If one service belongs to a group which is not associated with any Application Package, then all the parameters are same as those in UBBCONFIG.

If not:

Required parameters are:

SVCNM
: It must be the same with the SVCNM in the corresponding Application Package's properties.xml. All the services in the properties.xml file must have an associated entry in this section.

SRVGRP = string_value
: Refer to the same parameters in UBBCONFIG

Optional parameters listed below are same as those in UBBCONFIG:

- LOAD = number
- PRIO = number
- ROUTING = string_value
- BLOCKTIME numeric_value
- SVCTIMEOUT = number
- SESSIONROLE
- AFFINITYSCOPE
- AFFINITYSTRICT
- AUTOTRAN = {Y | N}
- TRANTIME = number

The parameters listed below can't appear in this section, the ultimate UBBCONFIG will keep the properties.xml file value:

- BUFTYPE = "type1[:subtype1[,subtype2 . . .]][;type2[:subtype3[, . . .]]] . . . "
- SIGNATURE_REQUIRED = {Y | N}
- ENCRYPTION_REQUIRED = {Y | N}
- BUFTYPECONV = {XML2FML | XML2FML32}

This command will generate one domain's all information including UBBCONFIG and save it to the Deployment Repository.

This command will configure every compute node according to the configure script including creating the TLOG device and so on.

This command will boot the whole Tuxedo System.

Undeploying a Domain

Users can use `tmdeldom` to shut down and undeploy the domain. This command must be run on the master node of the domain.

The command syntax is:

```
tmdeldom -d "domain_name" -r  "deprepository"  -f
```

The command functions are:

- Shut down the whole domain.
- Unconfigure every node if needed.
- Undeploy the whole domain and delete it also from the Deployment Repository.

The parameter domain_name is the domain's name which is specified in the previous tmcrdom command. The deprepository is the Deployment Repository which is specified by the DEPREPOSITORY parameter in the previous deployment plan.

Without "-f", once any step of tmdeldom fail, the command will not continue to execute.

If -f is specified, even if step 1, 2, or 3 is failed, this command will also delete the domain information in the repository.

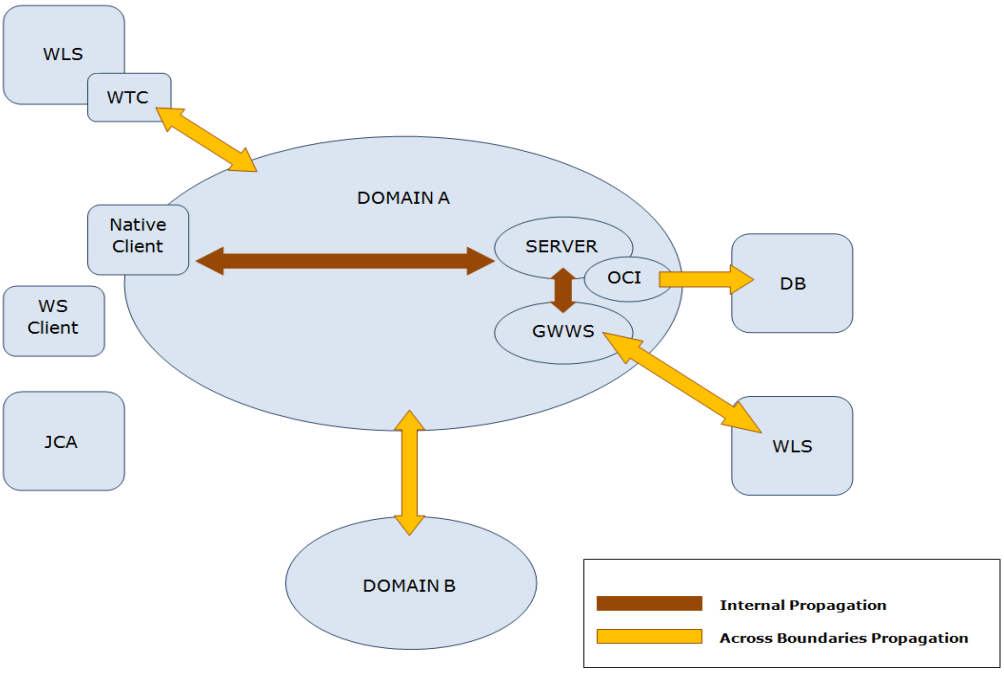
Configuring Tuxedo for Propagating ECID

This topic includes the following sections:

Overview

Tuxedo is enhanced to support ECID propagation from its 12c (12.1.1) release. As Figure 19-1 shows, it supports the propagation of ECID across various Tuxedo boundaries.

Figure 19-1 ECID Propagation



With this feature enabled, as the receiving side, Tuxedo will check whether there is an ECID in each request call across boundaries. If yes, the received ECID will be propagated along the whole call path; otherwise, as the boundary node, the domain gateway will try to generate an ECID for this call and propagate it once correctly configured.

For a call initiated by a Tuxedo component, say, a native client, a WS client, or a Jolt client, the client can generate an ECID for it once correctly configured.

Note: JCA support is not in the scope of this feature.

This section describes following four scenarios of ECID propagation.

- Propagating ECID from Tuxedo to Database
- Propagating ECID Between Tuxedo and WLS
- Propagating ECID within Tuxedo
- Generating ECID by Native/WS/Jolt clients and Domain Gateway

- Interoperability

Propagating ECID from Tuxedo to Database

Tuxedo supports to propagate the ECID to Oracle DB via Oracle Call Interface (OCI). This is a one-way propagation. Since the OCI/DB never initiates a request call in theory, it's not necessary to support the propagation in opposite direction (from DB to Tuxedo).

The OCI provides a registration API and a callback function interface. The implementation of the callback function handles the details of propagation. The process is showed as below:

1. When a tuxedo server process is up and with ECID enabled, it will call the registration API to register the implementation of the callback function.
2. OCI invokes the callback function and passes the relative context which can be used to identify the DB connection/session before a roundtrip.
3. The callback implementation gets the associated ECID from Tuxedo context and passes such ECID to Oracle DB.

Propagating ECID Between Tuxedo and WLS

WTC provides interoperability between WLS applications and Tuxedo services. As an interface, WTC takes full responsibility of ECID's propagation between Tuxedo and WLS.

- For a request call from WLS to Tuxedo, the process is:
  a. WTC sets the ECID into the special field of the message.
  b. WTC sends the message to Tuxedo.
  c. Tuxedo domain gateway gets the ECID from received message.
- For a request call from Tuxedo to WLS, the process is:
  a. Tuxedo domain gateway sends the message with ECID to WTC.
  b. WTC gets the ECID from received message.
  c. WTC sets the ECID for current Execution Context.

ECID can also be propagated between Tuxedo and WLS* when using Web Services through the SALT gateway (GWWS), either in SOAP or REST mode, and regardless of direction (Tuxedo

calling WLS or WLS calling Tuxedo). In this mode, the ECID will be included in an HTTP header in the same manner as WLS* uses to propagate ECID contexts with other Oracle products.

Note: * WLS should be WLS 12c (12.1.3) or higher releases of WLS.

Propagating ECID within Tuxedo

The ECID propagation within Tuxedo (both intra-domain and across domains) utilizes the `META_TCM` field in the messages. Tuxedo client, server, and domain gateway are all enhanced to support to get or set ECID in `META_TCM` section of a message.

Inside the Tuxedo, the ECID generation/propagation happens with following three ATMI APIs:

- tpcall
- tpacall
- tpforward

Generating ECID by Native/WS/Jolt clients and Domain Gateway

As boundary components, native/WS/Jolt client and domain gateway can generate the ECID for each request message once configured correctly.

Interoperability

If there is a domain/machine with an Oracle Tuxedo version lower than 12c in the call path, the end-to-end ECID propagation depends on specific user scenarios.

For example, let's consider the following interoperability scenario.

There are three domains: DOM1, DOM2 and DOM3. DOM1 and DOM3 are in Oracle Tuxedo 12c, DOM2 is in Tux11gR1; DOM1 enables both `ECID_USERLOG` and `ECID_CREATE` but DOM3 only enables `ECID_USERLOG`.

On one hand, if client on DOM1 invokes tpcall service1 on DOM2 and then service1 invokes tpcall service2 on DOM3 (allocating a new buffer via tpalloc), ECID will not be found in DOM3's ULOG. Thus, ECID information from DOM1 cannot be kept.

On the other hand, if client on DOM1 invokes tpcall service1 on DOM2 and then service1 invokes tpcall service2 on DOM3 (reusing the buffer with which service1 is invoked), ECID will

be found in DOM3's ULOG. Thus, ECID information from DOM1 can be propagated and retrieved by DOM3.

Configurations

This section covers the following configurations to enable ECID propagation:

- Enabling and Disabling ECID Propagation
- Configuring the Server to Propagate ECID via OCI

Enabling and Disabling ECID Propagation

In RESOURCES section of `UBBCONFIG(5)`, the `OPTIONS` field is extended with new flags to enable and specify the behavior for handling the ECID.

Option '`ECID_CREATE`': ECID creation functionality is enabled. The boundary nodes (including Native/WS/Jolt client and domain gateway) can generate the ECID.

Option '`ECID_USERLOG`': With this option on, if not a null string, the ECID will be appended to the userlog.

If neither of above options is set, the ECID propagation around Tuxedo is enabled by default, but ECID can neither be created nor printed in ULOG in this domain. In such scenario, "`tmadmin > psr -v`" can be used to get the ECID. The two of above options can also be updated by MIB operation.

Configuring the Server to Propagate ECID via OCI

To perform ECID propagation to Oracle DB via OCI callback functions, the related server must be specified by a "`-L`" option. For example:

```
simpserv    CLOPT="-A -L default --" SRVGRP=GROUP1 SRVID=2
```

Once activated with a specified "`-L`", a server will call the registration API of OCI for ECID propagation.

`-LD` or `-L default`

Loads default OCI library, that is, "oci.dll" for WIN32 platform and "libclntsh.so" for other platforms.

```
-L oci_lib_name
```

Loads the OCI library with the name specified by `oci_lib_name`. Such `oci_lib_name` can be an absolute path or a leaf name, which can be found through the system library path.

Tracing ECID with Tuxedo System

The userlog is enhanced to print out the ECID of current active call if '`ECID_USERLOG`' is configured in `UBBCONFIG`. For example:

```
233331.lclnx16!simpserv.19461.2664420416.0: ECID
<004fVWEOfCE6iKO5IjK6yf0004k8000000>: sleep 15
```

The `tmadmin printserver (psr -v)` command is enhanced to print out ECID if the server is currently in service. For example:

```
      Group ID: GROUP1, Server ID: 1

      Machine ID: SITE1

     Process ID: 19461, Request Qaddr: 1009713159, Reply Qaddr: 1009713159

      Server Type: USER

      Prog Name:
/nfs/ucfhomes/xuhchen/lclnx16/TUX12gR164/LC/bld/samples/atmi/simpapp/simps
erv

      Queue Name: 00001.00001

         Options: ( none )

      Generation: 1, Max message type: 1073741824

   Creation time: Tue Sep 27 23:32:44 2011

         Up time: 0:05:22

   Requests done: 2

       Load done: 100

 Current Service: TOUPPER,(ecid:
c9ca469656eb23b5:79482783:132ae1bef3b:-8000-0000000000000010)
```

See Also

File Formats, Data Descriptions, MIBs, and System Processes Reference

CHAPTER 20

Logging Last Resource Transaction Optimization

This topic includes the following sections:

Overview

Logging Last Resource (LLR) transaction optimization is a performance enhancement option that enables a non-XA resource to participate in a global transaction.

The LLR resource uses a local transaction for its transaction work. The Oracle Tuxedo transaction manager prepares all other resources in the transaction and determines the commit decision for the global transaction based on the outcome of the LLR resource's local transaction.

In a global two-phase commit (2PC) transaction with an LLR participant, the Oracle Tuxedo transaction manager follows these basic steps:

- Calls prepare on all other (XA-compliant) transaction participants.
- Inserts a commit record to a table on the LLR participant (rather than to the file-based transaction log).

- Commits the LLR participant's local transaction (which includes both the transaction commit record insert and the application's SQL work).
- Calls commit on all other transaction participants.
- After the transaction completes successfully, lazily deletes the database transaction log record as part of a future transaction.

In the recovery, the transaction records left in the LLR table are taken as completing transactions.

Logging Last Resource Configurations

Configuring LLR Library in RM File

The LLR library provides an XA switch to emulate XA operations and several uniform APIs for LLR server/TMS. Configure LLR library in the RM file under `${TUXDIR}/udataobj` as follows:

```
[LLR XA switch name]:[LLR XA switch variable]:[Link options]
```

The Table 20-1 shows the LLR library details:

Table 20-1 LLR Library

LLR XA Switch Name	LLR XA Switch Variable	Link Options	RM	Implementation	LLR Library
`tuxllr_oraesql`	`tuxllrsw_oraesql`	`-lllroraesql -L${ORACLE_LIB} -lclntsh`	Oracle	Embedded SQL	`${TUXDIR}/lib/libllroraesql.so`

In general, the LLR XA switch variable is `tuxllrsw_[name]` and the LLR XA switch name is `tuxllr_[name]`. The related library name is `libllr[name]`, where `[name]` consists of RM ID and implementation ID. For example, `ora` indicates Oracle database, and `esql` indicates embedded SQL.

Configuring OPENINFO in UBBCONFIG File

Configure `OPENINFO` in the `UBBCONFIG` file in the following format:

```
[LLR XA switch name]:[open string]
```

The Table 20-2 shows the details.

Table 20-2 OPENINFO Format

LLR XA Switch Name	Format

`tuxllr_oraesql`	`ORACLE_XA{+required_fields...}`
	`[+optional_fields...]`
	For more information about SqlNet, Acc and DB, refer to Oracle online document.

`TuxLLR` is a specific optional field for Tuxedo LLR, which defines the LLR table name. If not defined, the default name is used.

The LLR table is used to store the committing TLOG for transactions that involves LLR server. The default table name is `TUXLLR_[DOMAINID]`. If `[DOMAINID]` is empty, `DOM` is used. When you specify the LLR table name using `TuxLLR`, make sure different LLR tables are used for different tuxedo domains in a same non-XA resource; otherwise the recovery might work incorrectly.

In addition, ensure the RM user account used by LLR switch has the privilege to create and update the LLR table. Otherwise, errors might occur.

For example, a user might get the following message in Oracle database:

```
ORA-01950: no privileges on tablespace 'string'
```

This is because the user does not have privilege to allocate an extent in the specified tablespace. To solve the issue, grant the user appropriate system privilege or grant the user space resource on the tablespace.

Configuring LLR Options in UBBCONFIG File

`LLR_DELSWAPSIZE numeric_value`
: Specifies the maximum completed LLR involved 2PC global transaction ID (gtrid) that can be stored in the swap area. By default, the value is 0. To enable LLR feature, specify a positive value.

`LLR_DELDELAY numeric_value`
: Specifies a multiplier of the basic `SCANUNIT` between LLR completed records lazy deletions. The value must be greater than 0. If this parameter is not specified, the default is set so that `SCANUNIT * LLR_DELDELAY` is approximately 30 seconds, however, if the `SCANUNIT` value is greater than 30 seconds, `LLR_DELDELAY` is set to 1 if not specified.

Building LLR Server/TMS

Oracle Tuxedo takes an XA switch as a Oracle Tuxedo-specific LLR XA switch if the XA switch name prefix is `tuxllr_`.

As long as an application server is built with the LLR XA switch, Oracle Tuxedo treats it as an LLR server. The TMS servers built with the same LLR XA switch in the same group are treated as LLR TMS. The group that LLR server belongs to is an LLR group. The transaction involving an LLR server is an LLR involved transaction. The TLOG record of this transaction is stored in the LLR table instead of Oracle Tuxedo traditional TLOG file.

An LLR server/TMS must use the correct LLR XA switch. It means the RM and the implementation must be the same as the LLR library that provides the XA switch.

Typical Configuration Example

Following is an example of the LLR server (server.pc), which uses the embedded SQL to insert a record into an Oracle database table within a local transaction.

Listing 20-1

```
…
EXEC SQL INCLUDE sqlca;
EXEC SQL BEGIN DECLARE SECTION;
char   mylog[200];
int    mypid;
EXEC SQL END DECLARE SECTION;
int tpsvrinit(int argc, char *argv[])
{
       tpopen();
       mypid=getpid();
       return(0);
}
void tpsvrdone()
```

```
{
       tpclose();
}
void ECHO(TPSVCINFO *rqst)
{
       strncpy(mylog, rqst->data,rqst->len);
       mylog[rqst->len-1] = 0;
       EXEC SQL INSERT INTO TUX_RAC_TAB(OWNER, DATA) VALUES(:mypid,
:mylog);
       tpreturn(TPSUCCESS, 0, rqst->data, 0L, 0);/*do not commit the local
transaction*/
}
…
```

Listing 20-2 Configuring the RM File

```
tuxllr_oraesql:tuxllrsw_oraesql: -lllroraesql -L${ORACLE_LIB} -lclntsh
```

Listing 20-3 Configuring UBBCONFIG

```
*RESOURCE
…
MODEL                SHM
SCANUNIT              5
MAXGTT               600
LLR_DELDELAY          6
LLR_DELSWAPSIZE       500
…
```

```
*MACHINES
“m1”   LMID=L1TLOGSIZE=100
…
*GROUPS
GRP1   LMID=L1 GRPNO=10 TMSNAME="TMS_LLRORAESQL" TMSCOUNT=2

OPENINFO="tuxllr_oraesql:ORACLE_XA+SqlNet=orcl.tux1+ACC=P/scott/tiger”
…
server1SRVGRP=GRP1 SRVID=10
…
```

Listing 20-4 Building LLR Servers and TMS

```
…
${PROC} ${PROCFLAGS} iname=server.pc
buildserver -r tuxllr_oraesql -o server1 -f server.c
buildtms -r tuxllr_oraesql -o TMSLLRORAESQL
…
```

Lazy Deletion on TLOG Records of Completed LLR Transactions

The TLOG records of completed 2PC global transaction IDs (gtrid) in LLR table are purged in a lazy way. The purge is based on a timer. You can configure it using the parameter `LLR_DELDELAY`.

A new introduced swap area in BB is used to temporarily store the completed LLR involved gtrids. BBL may move these gtrids in the swap area to each local cache scan unit and purge the corresponding TLOG records in the LLR table according to the gtrids when time is out. You can specify the swap area size using the attribute `LLR_DELSWAPSIZE`.

If the swap area is too small, the completed gtrids that cannot be stored in the swap area at that time are cached by the related coordinator TMS process temporarily.

Unexpected node crash (e.g. BB is lost) leaves the completed records in the LLR table that does not be purged in time. Each record needs on entry in the transaction table during Tuxedo recovery. The transaction table size in BB is specified by `MAXGTT`. If `MAXGTT` is not big enough to store these records, BBL fails to boot up when the node is restored. When that happens, increase the `MAXGTT` value and retry.

It is recommended that specifying a sufficient `LLR_DELSWAPSIZE` to accommodate at least all completed LLR involved 2PC gtrids in a scan unit.

BBL does not purge the completed gtrids records in the LLR table. It invokes the system supplied LLR helper process to do this job. The process can also help BBL to retrieve records in LLR data during Tuxedo recovery.

Note: A "`.llr`" directory is created under `$APPDIR` to store temporary files used by LLR.

Constrains and Limitations

- General
  - Like a normal XA server, the customer codes in an LLR server should manage the connection through `tpopen/tpclose`.
  - Only one LLR server can be involved in a global transaction.
  - According to the MRM group, the `OPENINFO` used by the LLR server must be specified in the `GROUP` section
  - LLR server does not support `tpsuspend` and `tpresume`
  - According to MP domain, the slave node using an Oracle Tuxedo release that does not support LLR fails to boot up if LLR is enabled.
  - BBL spends more time than usual to boot up if some LLR groups are specified because BBL needs to retrieve possible TLOG records from the LLR tables of all related non-XA resources.
  - Only Linux 64-bit platform is supported
- RM-Specific (specific to `libllroraesql`)
  - Multi-threaded LLR server is not supported

The simulating XA switch `tuxllrsw_oraesql` does not support thread context. According to LLR involved MRM group, the other XA connections to Oracle database should not enable the property `Threads` in the `OPENINFO`. Otherwise the SQL jobs on LLR connection in the application server cannot work.

– An LLR server cannot handle any new request before current local transaction is finished.