BEA Tuxedo®

# Using the BEA Tuxedo Domains Component

Version 9.1
Document Released: May 15, 2006

# Copyright



# Restricted Rights Legend



# Trademarks or Service Marks

# Contents

## About This Document



## About Domains

# Planning and Configuring ATMI Domains

## Planning and Configuring CORBA Domains

# Administering Domains

# About This Document

This document explains how to configure and administer the BEA Tuxedo Domains component for both BEA Tuxedo ATMI and CORBA environments.

This document covers the following topics:

- Chapter 1, "About Domains," provides an overview of the BEA Tuxedo Domains component.
- Chapter 2, "Planning and Configuring ATMI Domains," explains how to plan and configure a domain for a BEA Tuxedo ATMI Domains environment.
- Chapter 3, "Planning and Configuring CORBA Domains," explains how to configure a domain for a BEA Tuxedo CORBA Domains environment.
- Chapter 4, "Administering Domains," explains how to administer a BEA Tuxedo Domains environment.

## What You Need to Know

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

## e-docs Web Site

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

## How to Print the Document

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

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

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

## Related Information

The following BEA Tuxedo documents contain information that is relevant to the BEA Tuxedo Domains component:

- *BEA Tuxedo Product Overview*
- *Setting Up a BEA Tuxedo Application*

- *Administering a BEA Tuxedo Application at Run Time*
- *Scaling, Distributing, and Tuning CORBA Applications*

For more information about BEA Tuxedo ATMI and CORBA environments, see *Bibliography*.

# Contact Us!

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

In your e-mail message, please indicate that you are using the documentation for the BEA Tuxedo 9.1 release.

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

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

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

# Documentation Conventions

The following documentation conventions are used throughout this document.

| Convention | Item |
|---|---|
| **boldface text** | Indicates terms defined in the glossary. |
| Ctrl+Tab | Indicates that you must press two or more keys simultaneously. |
| *italics* | Indicates emphasis or book titles. |

| Convention | Item |
|---|---|
| `monospace text` | Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames and their extensions. Monospace text also indicates text that you must enter from the keyboard.<br>*Examples*:<br>`#include <iostream.h> void main ( ) the pointer psz`<br>`chmod u+w *`<br>`\tux\data\ap`<br>`.doc`<br>`tux.doc`<br>`BITMAP`<br>`float` |
| **`monospace boldface text`** | Identifies significant words in code.<br>*Example*:<br>`void` **`commit`** `( )` |
| *`monospace italic text`* | Identifies variables in code.<br>*Example*:<br>`String` *`expr`* |
| UPPERCASE TEXT | Indicates device names, environment variables, and logical operators.<br>*Example*s:<br>LPT1<br>SIGNON<br>OR |
| `{ }` | Indicates a set of choices in a syntax line. The braces themselves should never be typed. |
| `[ ]` | Indicates optional items in a syntax line. The brackets themselves should never be typed.<br>*Example*:<br>`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`<br>`[-l` *`file-list`*`]...` |
| \| | Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed. |

| Convention | Item |
| --- | --- |
| . . . | Indicates one of the following in a command line:<br>• That an argument can be repeated several times in a command line<br>• That the statement omits additional optional arguments<br>• That you can enter additional parameters, values, or other information<br>The ellipsis itself should never be typed.<br>*Example*:<br>`buildobjclient [-v] [-o name ] [-f file-list]...`<br>`[-l file-list]...` |
| .<br>.<br>. | Indicates the omission of items from a code example or from a syntax line.<br>The vertical ellipsis itself should never be typed. |

CHAPTER 1

# About Domains

The following sections provide an overview of the BEA Tuxedo Domains component:

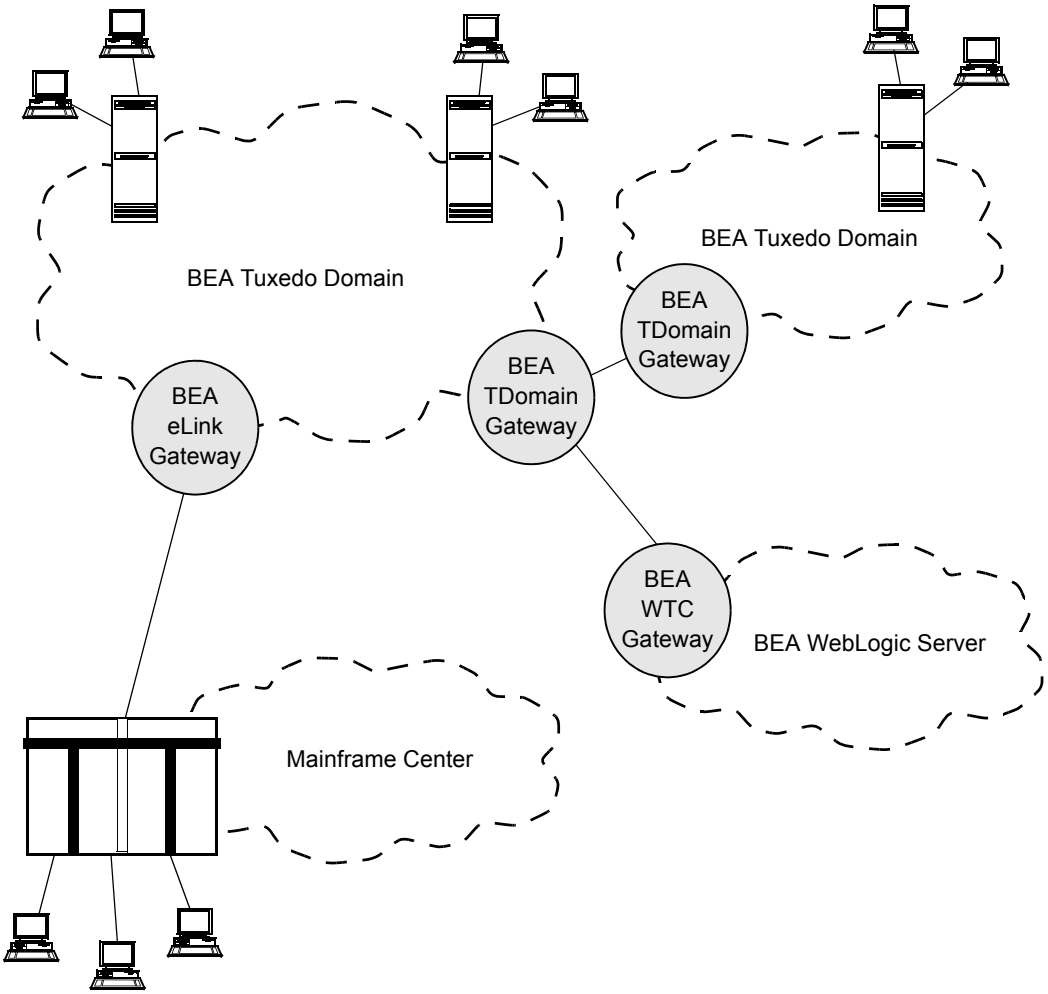


## What Is the BEA Tuxedo Domains Component?

As a company's business grows, application engineers may need to organize the business information management into distinct applications, each having administrative autonomy, based on functionality, geographical location, or confidentiality. These distinct business applications,

known as *domains*, need to share information. The BEA Tuxedo Domains component provides the infrastructure for interoperability among the domains of a business, thereby extending the BEA Tuxedo client/server model to multiple transaction processing (TP) domains. The following figure shows how the BEA Tuxedo Domains component can tie multiple domains together.

**Figure 1-1 Interdomain Communications Using the BEA Tuxedo Domains Component**



## Interoperability Among Domains

By transparently making services of a remote domain available to users of the local domain, and making services of the local domain available to users of a remote domain, the BEA Tuxedo Domains component breaks down the walls between a company's business applications. In addition, the Domains component enables a company running a BEA Tuxedo application to expand its business by interoperating with applications running on other transaction processing (TP) systems, such as BEA's WebLogic Server, IBM/Transarc's Encina, and IBM's CICS.

Because a company often uses the nature of a business application as part of its name, applications have names like the "accounting" domain or the "order entry" domain. A BEA Tuxedo domain is a single computer or network of computers controlled by a single configuration file known as the `UBBCONFIG` file. (The BEA Tuxedo configuration file may have any name as long as the content of the file conforms to the format described on reference page `UBBCONFIG(5)` in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*.) A BEA Tuxedo domain is administered as a single unit.

## Types of Domain Gateways

The BEA Tuxedo Domains component offers different types of gateways to be able to communicate with different types of networks and domains. Specifically, the Domains component offers the following domain gateways:

- *BEA Tuxedo TDomain gateway* (implemented by the `GWTDOMAIN` server process)—provides interoperability between two or more BEA Tuxedo domains through a specially designed TP protocol that flows over network protocol TCP/IP. Working with the WebLogic Tuxedo Connector (WTC) gateway, a BEA WebLogic Server component, the BEA Tuxedo TDomain gateway can also provide interoperability between Tuxedo domains and WebLogic Server applications.
- *BEA eLink Adapter for Mainframe TCP gateway* (implemented by the `GWIDOMAIN` server process) provides interoperability between BEA Tuxedo domains and applications running under IBM OS/390 Customer Information Control System (CICS) and Information Management System (IMS) over network protocol TCP/IP. The gateway supports only non-transactional tasks.
- *BEA eLink Adapter for Mainframe SNA gateway* (implemented by the `GWSNAX` server process)—provides interoperability between BEA Tuxedo domains and applications running on any System Network Architecture (SNA) Advanced Program-to-Program Communications (APPC) or Common Programming Interface for Communications

(CPI-C) supported platform, including IBM OS/400, OS/390 CICS and IMS systems and VSE/CICS. The gateway supports communication with multiple SNA networks.

- *BEA eLink Adapter for Mainframe OSI TP gateway* (implemented by the `GWOSITP` server process)—provides interoperability between BEA Tuxedo domains and other transaction processing applications that use the Open Systems Interconnection (OSI) transaction processing (TP) standard. OSI TP is a protocol for distributed transaction processing defined by the International Standards Organization (ISO). The gateway supports global transactions and various non-transactional tasks.

The discussions that follow focus on the BEA TDomain gateway and the communication between BEA Tuxedo domains. For information about the WTC gateway, see:

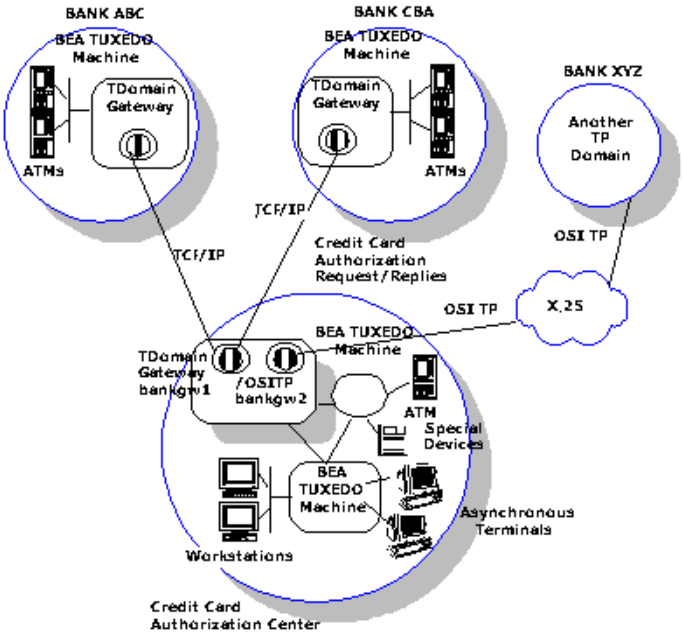
- *BEA Tuxedo Product Overview*
- *WebLogic Tuxedo Connector* at `http://e-docs.bea.com/wls/docs81/wtc.html`

For information about BEA eLink gateways, see *BEA eLink Documentation* at `http://e-docs.bea.com/elink/mainfram/mainfram.htm`.

# Example of a Domains Configuration

The following figure shows an example Domains configuration involving four domains, three of which are BEA Tuxedo domains.

**Figure 1-2 A Banking Domains Configuration—Example**




The BEA Tuxedo credit card authorization center at the bottom of the figure has two gateway groups: a TDomain gateway group named `bankgw1` and an OSI TP gateway group named `bankgw2`. `bankgw1` provides access to two remote BEA Tuxedo domains, Bank ABC and Bank CBA, using network protocol TCP/IP. `bankgw2` provides access to one remote domain, Bank XYZ, using network protocol OSI TP.

In this example, Bank ABC generates service requests to the credit card authorization center. These requests are received by a domain gateway server process named `GWTDOMAIN` running within group `bankgw1`. This gateway issues a service request, on behalf of the remote domain, to the credit card authorization service provided by another locally running server process. This server handles the request and sends the reply to the gateway, and the gateway forwards the reply to Bank ABC.

The credit card authorization center may also issue service requests. For example, the authorization center may send balance inquiries to Bank XYZ via a domain gateway server process named `GWOSITP`.

The BEA Tuxedo Domains component makes the interdomain communications possible through domain gateway server processes that advertises remote services—services available in other domains—as if they were local services.

# Functionality Supported by Domain Gateways

Domain gateways support the following functionality:

- *Multinetwork support*—gateways can communicate with other domains via a variety of network protocols, such as TCP/IP, IPX/SPX, OSI, and others. However, a gateway is limited by the capabilities of the networking library to which it is linked. In other words, a gateway typically supports a single type of network protocol. As an example, the BEA Tuxedo TDomain gateway supports only TCP/IP.
- *Multidomain Interaction*—gateways can communicate with multiple domains.
- *Transaction management*—gateways enable ATMI applications to interoperate with other domains within a transaction. The gateway coordinates the commitment or rollback of transactions running across domains.
- *Multiple messaging models*—gateways support the following ATMI messaging models, without any need to change existing BEA Tuxedo applications:
  - *Request/response model*—ATMI applications using the BEA Tuxedo system can request services from applications running in other domains.
  - *Conversational model*—ATMI applications can establish conversations with programs running in other domains.
  - *Queuing model*—ATMI applications using the BEA Tuxedo system can store data on queues in other domains.
- *Typed buffer support*—gateways can perform encoding and decoding operations for all the types of buffers defined by BEA Tuxedo ATMI applications.

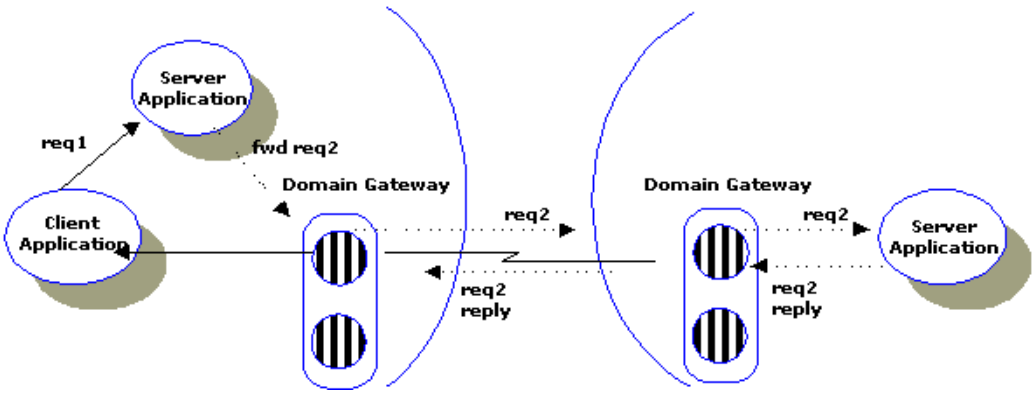
## Request/Response Communication Between Local and Remote Domains

Domain gateways provide support for the request/response model of communication defined by the ATMI interface. Except for the following BEA Tuxedo ATMI functions, which are logically limited to use within a single application and are not supported across domains, a BEA Tuxedo application can request remote services exactly as if they were offered locally:

- `tpinit(3c)`/`tpterm(3c)`—BEA Tuxedo applications do not attach to the environment of a remote domain; they use domain gateways to access a remote domain. Therefore, an extra `tpinit()`/`tpterm()` sequence is not needed for remote applications.
- `tpadvertise(3c)` and `tpunadvertise(3c)`—Domains does not support these functions because domain gateways do not support dynamic service advertisements across domains.
- `tpnotify(3c)` and `tpbroadcast(3c)`—Domains does not support the unsolicited communication paradigm provided by these functions.
- Event posting (`tppost(3c)`) and notification of events (`tpsubscribe(3c)`)—Domains does not support these functions across domains.

Support for `tpforward(3c)` is provided to preserve application portability. Forwarded requests are interpreted by domain gateways as simple service requests. This process is shown in the following figure, which illustrates the simple scenario of a service using `tpforward` to send a request to a remote service.

**Figure 1-3 Using tpforward to Send a Request to a Remote Service**




For more information about the BEA Tuxedo request/response model, see "Request/Response Communication" on page 2-10 in *Introducing BEA Tuxedo ATMI*.

# Conversational Communication Between Local and Remote Domains

Domain gateways provide support for the conversational model of communication defined by the ATMI interface. The ATMI is a connection-oriented interface that enables clients to establish and maintain conversations with services programmed in the conversational model.

BEA Tuxedo applications use `tpconnect(3c)` to open a conversation with a remote service, `tpsend(3c)` and `tprecv(3c)` to communicate with this service, and `tpdiscon(3c)` to end the conversation. Domain gateways maintain the conversation with the remote service, and support the same semantics for returns (that is, `tpreturn` with `TPSUCCESS` or `TPFAIL`) and disconnects that are defined for BEA Tuxedo conversational services.

**Note:** The ATMI connection-oriented functions provide half-duplex conversations; `tpforward(3c)` is not allowed within a conversational service.

For more information about the BEA Tuxedo conversational model, see "Conversational Communication" on page 2-11 in *Introducing BEA Tuxedo ATMI*.

# Queuing Messages on Remote Domains

Domain gateways provide support for the queuing model of communication defined by the ATMI interface. Any client or server can store messages or service requests in a queue in a remote domain. All stored requests are sent through the transaction protocol to ensure safe storage.

The BEA Tuxedo system enables messages to be queued to persistent storage (disk) or to non-persistent storage (memory) for later processing or retrieval. ATMI provides primitives that allow messages to be added (that is, `tpenqueue`) or read (that is, `tpdequeue`) from queues. Reply messages and error messages can be queued for later return to clients. An administrative command interpreter (that is, `qmadmin`) is provided for creating, listing, and modifying queues. Servers are provided to accept requests to enqueue and dequeue messages (that is, `TMQUEUE` server), to forward messages from the queue for processing (that is, `TMQFORWARD` server), and to manage the transactions that involve queues (that is, `TMS_QM` server).

For more information about the BEA Tuxedo queueing model, see "Message Queuing Communication" on page 2-12 in *Introducing BEA Tuxedo ATMI*.

# Encoding and Decoding Operations for Domains

Domain gateways support all predefined types of *typed buffers* supported by the release of BEA Tuxedo system software in which the domain gateway server processes are running. BEA Tuxedo supports 11 predefined buffer types.

Each buffer type supported by a BEA Tuxedo release has its own set of routines that can be called automatically *to initialize, send and receive messages, and encode and decode data* without programmer intervention. The set of routines is called a *typed buffer switch*.

In BEA Tuxedo ATMI applications, typed buffers are used to send data—service requests and replies—between clients and servers. Typed buffers, which by definition contain information

about themselves (metadata), allow application programmers to transfer data without needing to know which data representation scheme is used by the machines on which the application's clients and servers are running.

A domain gateway can receive and process service requests sent from workstations, from local BEA Tuxedo machines, and from remote domains. Using the appropriate typed buffer switch, a domain gateway will decode any service request that it receives encoded for the following reasons:

- Data-dependent routing depends upon matching specified criteria to fields within data. Therefore, a domain gateway must decode the encoded data in order to route that data to the appropriate remote domain for the service requested.
- Different data formats may be used within different domains, depending on the networking protocols implemented or used in a domain. Therefore, a domain gateway must decode the encoded data to determine which data format is being used.

OSI terminology provides a useful distinction between abstract syntax (that is, the structure of the data) and transfer syntax (that is, the particular encoding used to transfer the data). Each typed buffer implicitly defines a particular data structure (that is, its abstract syntax) and the encoding rules (or typed buffer operations) required to map the data structure to a particular transfer syntax (for example, XDR). For the predefined buffer types that support encoding/decoding, the BEA Tuxedo system provides the encoding rules required to map these types to the XDR transfer syntax.

For more information about typed buffers and encoding and decoding operations, see "What Are Typed Buffers?" on page 2-22 in *Introducing BEA Tuxedo ATMI*.

# BEA Tuxedo Domains Architecture

The BEA Tuxedo Domains architecture consists of four major parts:

- Domains configuration file
- Domain gateway servers
- Domains administrative servers
- Domains administrative tools

## Domains Configuration File

A Domains configuration is a set of two or more domains (applications) that can communicate and share services via the BEA Tuxedo Domains component. How multiple domains are

connected and which services they make accessible to one another are defined in *Domains configuration files*. Each BEA Tuxedo domain involved in a Domains configuration requires its own Domains configuration file.

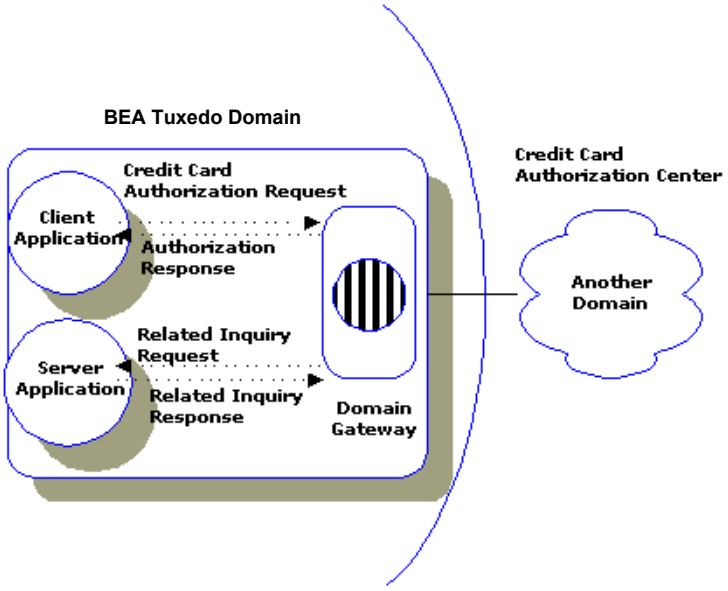
The text version of the Domains configuration file is known as the DMCONFIG file, although it may have any name as long as the content of the file conforms to the format described on reference page DMCONFIG(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*. The binary version of the Domains configuration file is known as BDMCONFIG. For a detailed description of the DMCONFIG file, see "Understanding the Domains Configuration File" on page 1-15.

## Domain Gateway Servers

The BEA Tuxedo Domains component achieves multiple-domain interoperability through a highly asynchronous, multitasking, multithreaded domain gateway process, which is a BEA Tuxedo supplied server that makes access to services across domains transparent to both the application programmer and the application user.

The following figure illustrates how one BEA Tuxedo domain communicates with another domain via a domain gateway.

**Figure 1-4 Two-Way Communication Through a Gateway**

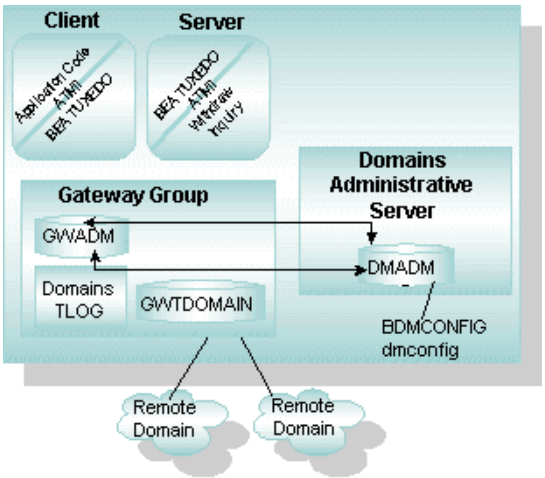



In the figure, the domain gateway handles outgoing credit card authorization requests to another domain. It also handles incoming authorization responses.

# Domains Administrative Servers

The following figure shows the BEA Tuxedo Domains administrative servers used to administer a Domains configuration.

**Figure 1-5 Domains Administrative Servers**




A domain *gateway group*, as shown in the previous figure, consists of a *gateway administrative server* (GWADM), a domain gateway server (for example, GWTDOMAIN), and (optional) a Domains transaction log (TLOG). The GWADM server enables run-time administration of the domain gateway. A BEA Tuxedo domain can communicate with one or more remote domains through a domain gateway group.

Associated with all domain gateway groups running in a BEA Tuxedo domain is a *Domains administrative server* (DMADM), which enables run-time administration of the BEA Tuxedo Domains configuration file (BDMCONFIG).

## GWADM Server

The GWADM(5) server registers with the DMADM server to obtain the configuration information used by the corresponding gateway group. GWADM accepts requests from the DMADMIN service, which is a generic administrative service advertised by the DMADM server, for run-time statistics or changes in the run-time options of the specified gateway group. Periodically, GWADM sends an "I-am-alive" message to the DMADM server. If no reply is received from DMADM, GWADM registers again. This process ensures the GWADM server always has the current information about the Domains configuration for its gateway group.

For more information about GWADM, see "Administering Domains" on page 4-1.

### DMADM Server

the DMADM(5) server provides a registration service for gateway groups. This service is requested by GWADM servers as part of their initialization procedure. The registration service downloads the configuration information required by the requesting gateway group. The DMADM server maintains a list of registered gateway groups, and propagates to these groups any changes made to the Domains configuration file (BDMCONFIG).

For more information about DMADM, see "Administering Domains" on page 4-1.

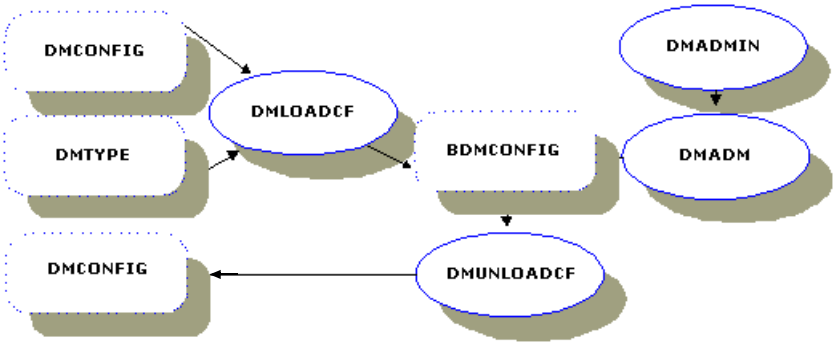
## Domains Administrative Tools

The following Domains administrative tools are provided by the BEA Tuxedo system for setting up and maintaining a Domains configuration:

- dmloadcf(1)—reads the DMCONFIG file, checks the syntax, and loads the binary BDMCONFIG configuration file.
- dmunloadcf(1)—translates the BDMCONFIG configuration file from binary to text format.
- dmadmin(1)—allows a BEA Tuxedo administrator to update the BDMCONFIG file when the Tuxedo domain is running.

The following figure shows the relationships between the Domains administrative tools and the Domains text and binary configuration files. Administration using the dmadmin utility is through the DMADMIN service, which is advertised by the DMADM server.

**Figure 1-6 Relationships Between Domains Administrative Tools and Files**



## dmloadcf Command

The dmloadcf(1) command parses the DMCONFIG file and loads the information into BDMCONFIG. The command uses the environment variable BDMCONFIG to point to the device or system filename in which the configuration should be stored.

The dmloadcf command, through the -c option, also provides an estimate of the interprocess communications (IPC) resources needed for each local domain specified in the configuration.

The dmloadcf command checks the DMTYPE file (%TUXDIR%\udataobj\DMTYPE for Windows or $TUXDIR/udataobj/DMTYPE for UNIX) to verify that the domain gateway types specified in the Domains configuration file are valid. Each type of domain gateway has a domain type designator (TDOMAIN, SNAX, OSITP, OSITPX), which is used as a tag in the DMTYPE file. Each line in this file has the following format:

*dmtype:access_module_lib:comm_libs:tm_typesw_lib:gw_typesw_lib*

The file has the following entry for the TDomain gateway:

```
TDOMAIN:-lgwt:-lnwi -lnws -lnwi::
```

For more information about dmloadcf, see reference page dmloadcf(1) in *BEA Tuxedo Command Reference*.

## dmunloadcf Command

The dmunloadcf(1) command converts the BDMCONFIG configuration file from binary to text format and prints the output to standard output. For more information about dmunloadcf, see reference page dmunloadcf(1) in *BEA Tuxedo Command Reference*.

## dmadmin Command

The dmadmin(1) command allows a BEA Tuxedo administrator to configure, monitor, and tune domain gateways when the Tuxedo domain is running. It acts as an administrative command interpreter that translates administrative commands and sends requests to the DMADMIN service, a generic administrative service advertised by the DMADM server. DMADMIN invokes functions that validate, retrieve, or update information in the BDMCONFIG file.

You invoke dmadmin with the -c option to dynamically update the BDMCONFIG file. Depending on the configuration being changed, some updates will take place immediately, while others will take place only for new occurrences of whatever is affected by the update.

For more information about dmadmin, see "Administering Domains" on page 4-1.

# Understanding the Domains Configuration File

Each BEA Tuxedo domain involved in a Domains configuration has a configuration file in which the interdomain parameters are defined. The text version of the configuration file is referred to as DMCONFIG, although the configuration file may have any name, as long as the content of the file conforms to the format described on reference page DMCONFIG(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*. Typical configuration filenames begin with the string dm, followed by a mnemonic string, such as config in the filename dmconfig.

As the administrator for the Domains configuration, you need to create a separate DMCONFIG file for each BEA Tuxedo domain participating in the configuration. You can create and edit a DMCONFIG file with any text editor.

## Location of DMCONFIG File

For a BEA Tuxedo domain involved in a Domains configuration, the DMCONFIG file resides on the machine on which the Domains administrative server DMADM is to run, as specified in the UBBCONFIG file for the Tuxedo domain. The DMADM server may run on any machine (master machine, non-master machine) in a Tuxedo domain.

**Note:** The master machine for a BEA Tuxedo domain contains the domain's UBBCONFIG file, and is designated as the master machine in the RESOURCES section of the UBBCONFIG file. Starting, stopping, and administering a Tuxedo domain is done through the master machine.

## Binary Version of DMCONFIG File

The BDMCONFIG file is a binary version of the DMCONFIG file. It is created by running the dmloadcf command, which parses DMCONFIG and loads the binary BDMCONFIG file to the location referenced by the BDMCONFIG environment variable. As with DMCONFIG, the BDMCONFIG file may be given any name; the actual name is the device or system filename specified in the BDMCONFIG environment variable. The BDMCONFIG environment variable must be set to an absolute pathname ending with the device or system filename where BDMCONFIG is to be loaded.

Unlike the TUXCONFIG file, which is the binary version of UBBCONFIG, the BDMCONFIG file is *not* propagated to any other machine in a Tuxedo domain when the Tuxedo application is booted. For the BDMCONFIG file to reside on any other machine in a Tuxedo domain, the administrator for that domain must manually place it there.

# Descriptions of Sections of the DMCONFIG File

The DMCONFIG file is made up of specification sections. Lines beginning with an asterisk (*) indicate the beginning of a specification section. Each such line contains the name of the section immediately following the *. The asterisk is required when specifying a section name.

Allowable section names are:

- DM_LOCAL (also known as DM_LOCAL_DOMAINS)
- DM_REMOTE (also known as DM_REMOTE_DOMAINS)
- DM_EXPORT (also known as DM_LOCAL_SERVICES)
- DM_IMPORT (also known as DM_REMOTE_SERVICES)
- DM_RESOURCES
- DM_ROUTING
- DM_ACCESS_CONTROL
- DM_*domtype*, where *domtype* is TDOMAIN, OSITP, OSITPX, or SNACRM + SNALINKS + SNASTACKS

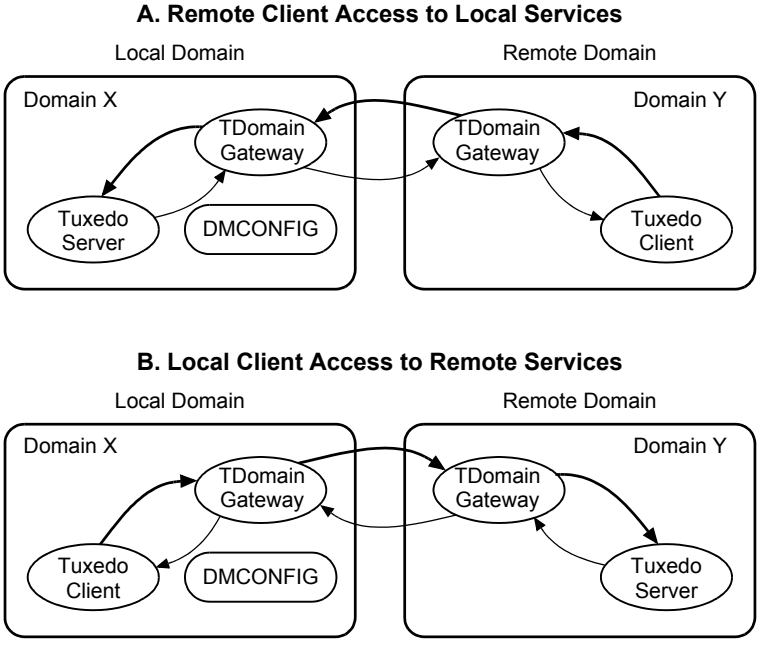
**Note:** The DM_LOCAL section must precede the DM_REMOTE section.

As the administrator for the Domains configuration, you use these sections to:

- Define local domain access points through which application clients on a remote domain can access services on the local domain.
- Specify the local services available through each local domain access point.
- Define remote domain access points through which application clients on the local domain can access services on a remote domain.
- Specify the remote services available through each remote domain access point.
- Map local domain access points and remote domain access points to specific domain gateway groups (TDOMAIN, ...) and network addresses.

The following figure is a simple example of what you are trying to accomplish.

**Figure 1-7 Establishing What Services Are Shared Between Two BEA Tuxedo Domains—Example**




In the example, you must also create a DMCONFIG file for Domain Y that complements the DMCONFIG file created for Domain X. That is, a local domain access point in the Domain X DMCONFIG file would be a remote domain access point in the Domain Y DMCONFIG file, and a remote domain access point in the Domain X DMCONFIG file would be a local domain access point in the Domain Y DMCONFIG file. The example demonstrates the use of the TDomain gateway server.

The following table provides a description of each section in the DMCONFIG file.

**Table 1-1 DMCONFIG File Sections (Sheet 1 of 4)**

| Section | Purpose |
|---|---|
| DM_LOCAL (also known as DM_LOCAL_DOMAINS) | Defines one or more local domain access point identifiers (also known as local domains, or LDOMs). For each local domain access point (logical name) that you define, you specify a domain gateway group (TDOMAIN, ...) for the access point in this section, and you specify in the DM_EXPORT section the local services available through the access point. The local services available through the local domain access point will be available to clients in one or more remote domains.<br><br>You can define multiple local domain access points in this section, one for each gateway group (TDOMAIN, SNAX, OSITP, OSITPX) used by this BEA Tuxedo domain to communicate with a remote domain.<br><br>One *and only one* local domain access point is allowed per gateway group. A domain gateway group consists of a GWADM server process and a domain gateway server process (for example, GWTDOMAIN, the TDomain gateway server).<br><br>Example of a local domain access point entry:<br>`*DM_LOCAL`<br>`LOCAL1 GWGRP=GWTGROUP`<br>`TYPE=TDOMAIN`<br>`ACCESSPOINTID="BA.CENTRAL01"`<br><br>**Note:** You may substitute DOMAINID for the ACCESSPOINTID parameter. |
| DM_REMOTE (also known as DM_REMOTE_DOMAINS) | Defines one or more remote domain access point identifiers (also known as remote domains, or RDOMs). For each remote domain access point (logical name) that you define, you specify a domain gateway group (TDOMAIN, ...) for the access point in this section, and you specify in the DM_IMPORT section the remote services available through the access point. The remote services available through the remote domain access point will be available to clients in the local domain.<br><br>You can define multiple remote domain access points in this section, one or more for each gateway group (TDOMAIN, SNAX, OSITP, OSITPX) used by this BEA Tuxedo domain to communicate with a remote domain.<br><br>Example of remote domain access point entries:<br>`*DM_REMOTE`<br>`REMOT1 TYPE=TDOMAIN`<br>`ACCESSPOINTID="BA.BANK01"`<br>`REMOT2 TYPE=TDOMAIN`<br>`ACCESSPOINTID="BA.BANK02"`<br><br>**Note:** You may substitute DOMAINID for the ACCESSPOINTID parameter. |

**Table 1-1 DMCONFIG File Sections (Sheet 2 of 4)**

| Section | Purpose |
|---|---|
| DM_EXPORT (also known as DM_LOCAL_SERVICES) | Specifies the local services exported to one or more remote domains through a local domain access point defined in the DM_LOCAL section. Only the services specified for a local domain access point are available to clients on one or more remote domains, meaning that specifying services in this section is a way to restrict remote client access to local services. If the DM_EXPORT section is absent, or is present but empty, all services advertised by the local domain are available to the remote domains.<br><br>A local service made available to remote domains inherits many of its properties from the SERVICES section of the local UBBCONFIG file. Some of the properties that may be inherited are LOAD, PRIO, AUTOTRAN, ROUTING, BUFTYPE, and TRANTIME.<br><br>Example of a local service made available to remote domains:<br>`*DM_EXPORT`<br>`LTOLOWER LACCESSPOINT=LOCAL1`<br>`CONV=N`<br>`RNAME="TOLOWER"`<br>`ACL=branch`<br><br>**Note:** You may substitute LDOM for the LACCESSPOINT parameter. |
| DM_IMPORT (also known as DM_REMOTE_SERVICES) | Specifies the remote services imported through one or more remote domain access points defined in the DM_REMOTE section and made available to the local domain through one or more local domain access points. If the DM_IMPORT section is absent, or is present but empty, no remote services are available to the local domain.<br><br>A remote BEA Tuxedo service made available to the local domain inherits many of its properties from the SERVICES section of the remote UBBCONFIG file. Some of the properties that may be inherited are LOAD, PRIO, AUTOTRAN, ROUTING, BUFTYPE, and TRANTIME.<br><br>Example of a remote service made available to the local domain:<br>`*DM_IMPORT`<br>`RTOUPPER AUTOTRAN=N`<br>`RACCESSPOINT=REMOT1`<br>`LACCESSPOINT=LOCAL1`<br>`CONV=N`<br>`RNAME="TOUPPER"`<br><br>**Note:** You may substitute RDOM for the RACCESSPOINT parameter, and LDOM for the LACCESSPOINT parameter. |

**Table 1-1 DMCONFIG File Sections (Sheet 3 of 4)**

| Section | Purpose |
|---|---|
| DM_RESOURCES | Specifies global Domains configuration information, specifically a user-supplied configuration version string. The only parameter in this section is VERSION=*string*, where *string* is a field in which users can enter a version number for the current DMCONFIG file. This field is not checked by the software. |
| DM_ROUTING | Specifies data-dependent routing criteria for routing local service requests to one of several remote domains offering the same service. For an example, see "Specifying Domains Data-Dependent Routing" on page 1-23. |
| DM_ACCESS_CONTROL | Specifies one or more access control list (ACL) names and associates one or more remote domain access points with each specified ACL name. You can use the ACL parameter in the DM_EXPORT section by setting ACL=*ACL_NAME* to restrict access to a local service exported through a particular local domain access point to just those remote domain access points associated with the *ACL_NAME*.<br>Example of an ACL entry:<br>`*DM_ACCESS_CONTROL`<br>`branch ACLIST=REMOT1` |

**Table 1-1 DMCONFIG File Sections (Sheet 4 of 4)**

| Section | Purpose |
|---|---|
| DM_*domtype* | Defines the parameters required for a particular Domains configuration. Currently, the value of *domtype* can be TDOMAIN, OSITP, OSITPX, or SNACRM + SNALINKS + SNASTACKS. Each domain type must be specified in a separate section.<br><br>In a DM_TDOMAIN section, you define the TDomain-specific network configuration for a local or remote domain access point. You can also define the network configuration for one or more remote domain access points associated with one or more WebLogic Server applications, to combine Tuxedo ATMI servers and WebLogic Server Enterprise JavaBean (EJB) servers in an application; for details, see *BEA Tuxedo Interoperability*.<br><br>The DM_TDOMAIN section should have an entry per local domain access point if requests from remote domains to local services are accepted through that access point. For each local domain access point specified in this section, you must specify the network address to be used for listening for incoming connections.<br><br>The DM_TDOMAIN section should have an entry per remote domain access point if requests from the local domain to remote services are accepted through that access point. For each remote domain access point specified in this section, you must specify the destination network address to be used when connecting to that remote domain access point.<br><br>Beginning with Tuxedo release 9.0, the DM_TDOMAIN section can have an entry per TDomain session between specific local and remote domain access points. For each TDomain session specified in this section, you must specify the destination network address to use when connecting to that TDomain session.<br><br>When Domains link-level failover is in use, you can specify more than one destination network address for a remote domain access point or TDomain session to implement the *mirrored* gateway capability. For an example of a mirrored gateway, see "How to Configure Domains Link-Level Failover" on page 1-40.<br><br>For information about the DM_OSITP, DM_OSITPX, DM_SNACRM, DM_SNALINKS, and DM_SNASTACKS sections, see the *BEA eLink Documentation* page at http://e-docs.bea.com/elink/mainfram/mainfram.htm. |

For a detailed description of the DMCONFIG file, see reference pages DMCONFIG(5) and DM_MIB(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*.

# Terminology Improvements for DMCONFIG File

For BEA Tuxedo release 7.1 or later, the Domains MIB uses improved class and attribute terminology to describe the interaction between local and remote domains. The improved terminology has been applied to the DMCONFIG(5) reference page, section names, parameter names, and error messages, and to the DM_MIB(5) reference page, classes, and error messages.

For backwards compatibility, aliases are provided between the DMCONFIG terminology used prior to BEA Tuxedo 7.1 and the improved Domains MIB terminology. For BEA Tuxedo release 7.1 or later, both versions of DMCONFIG terminology are accepted. The following table shows the mapping of the previous and improved terminology for the DMCONFIG file.

| Previous Terminology | | Improved Terminology | |
|---|---|---|---|
| **Section Name** | **Parameter Name** | **Section Name** | **Parameter Name** |
| DM_LOCAL_DOMAINS | | DM_LOCAL | |
| DM_REMOTE_DOMAINS | | DM_REMOTE | |
| | DOMAINID | | ACCESSPOINTID |
| | MAXRDOM | | MAXACCESSPOINT |
| | MAXRDTRAN | | MAXRAPTRAN |
| DM_LOCAL_SERVICES | | DM_EXPORT | |
| DM_REMOTE_SERVICES | | DM_IMPORT | |
| | LDOM | | LACCESSPOINT |
| | RDOM | | RACCESSPOINT |

For BEA Tuxedo release 7.1 or later, the dmunloadcf command generates by default a DMCONFIG file that uses the improved domains terminology. Use the -c option to print a DMCONFIG file that uses the previous domains terminology. For example:

```
prompt> dmunloadcf -c > dmconfig_prev
```

# Specifying Domains Data-Dependent Routing

You can specify data-dependent routing criteria for the routing of local service requests to remote domains in the DM_ROUTING section of the DMCONFIG file for any of the following buffer types:

- FML
- FML32
- VIEW
- VIEW32
- X_C_TYPE
- X_COMMON
- XML

In the following example, the remote service TOUPPER is available through two different remote domain access points named REMOT1 and REMOT2, and the data-dependent routing criteria for TOUPPER is defined in the routing criteria table named ACCOUNT. In the example, RTOUPPER1 and RTOUPPER2 are alias service names for TOUPPER, which is the actual service name expected by the remote domains.

```
*DM_IMPORT
RTOUPPER1   AUTOTRAN=N
            RACCESSPOINT=REMOT1
            LACCESSPOINT=LOCAL1
            CONV=N
            RNAME="TOUPPER"
            ROUTING=ACCOUNT
RTOUPPER2   AUTOTRAN=N
            RACCESSPOINT=REMOT2
            LACCESSPOINT=LOCAL1
            CONV=N
            RNAME="TOUPPER"
            ROUTING=ACCOUNT

*DM_ROUTING
ACCOUNT     FIELD=branchid
            BUFTYPE="VIEW:account"
            RANGES="MIN-1000:REMOT1,1001-3000:REMOT2"
```

For the ACCOUNT routing table, VIEW and account are the type and subtype of data buffers for which this routing table is valid, and branchid is the name of the field in the VIEW data buffer to which routing is applied. The allowed values for the branchid field are as follows:

- For the REMOT1 access point, the allowed values range from the minimum value allowed on the machine associated with REMOT1 to less than or equal to 1000.
- For the REMOT2 access point, the allowed values range from the 1001 to less than or equal to 3000.

If the value of the branchid field for a TOUPPER service request is within the range MIN-1000, the service request is routed through the REMOT1 access point. If the value of the branchid field for a TOUPPER service request is within the range 1001-3000, the service request is routed through the REMOT2 access point.

# Specifying Domains Transaction and Blocking Timeouts

The BEA Tuxedo system provides two timeout mechanisms: a *transaction timeout* mechanism and a *blocking timeout* mechanism. The transaction timeout is used to define the duration of an ATMI transaction, which may involve several service requests. The timeout value is defined when the transaction is started. The blocking timeout is used to define the duration of individual service requests, that is, how long the ATMI application is willing to wait for a reply to a service request.

If a process is *not* in transaction mode, the BEA Tuxedo system performs blocking timeouts. If a process is in transaction mode, the BEA Tuxedo system performs transaction timeouts but not blocking timeouts. The latter statement is true for *intradomain* transactions (that is, transactions handled within a single BEA Tuxedo domain) but is not true for *interdomain* transactions. For *interdomain* transactions, if a process is in transaction mode, the Domains software performs both transaction timeouts and blocking timeouts.

## How the Domains Component Handles Transaction Timeouts

The BEA Tuxedo transaction timeout mechanism is used unchanged in the Domains component. Use of the same transaction timeout mechanism is necessary because domain gateways implement the transaction manager server (TMS) functionality and therefore are required to handle the TMS timeout messages generated by the BEA Tuxedo Bulletin Board Liaison (BBL) administrative process.

A local service made available to remote domains in the DM_EXPORT section of the DMCONFIG file inherits the following transaction-related properties from the SERVICES section of the local UBBCONFIG file:

- AUTOTRAN—When AUTOTRAN is turned on for a service and a service request is received for the service that is not already within a transaction, the local BEA Tuxedo system automatically starts a transaction for the service.
- TRANTIME—Transaction timeout value in seconds for transactions automatically started for the service. If this timeout value is exceeded for a transaction, the BEA Tuxedo nodes (machines) infected with the transaction generate a TMS timeout message.

Similarly, a remote BEA Tuxedo service made available to the local domain in the DM_IMPORT section of the DMCONFIG file inherits the AUTOTRAN and TRANTIME properties from the SERVICES section of the remote UBBCONFIG file. If the TRANTIME timeout value is exceeded for a transaction, the BEA Tuxedo nodes infected with the transaction generate a TMS timeout message.

A service advertised on a machine running BEA Tuxedo release 8.1 or later inherits an additional transaction-timeout property named MAXTRANTIME 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. MAXTRANTIME has no effect on a transaction started on a machine running BEA Tuxedo 8.0 or earlier, except that when a machine running BEA Tuxedo 8.1 or later is infected by the transaction, the transaction timeout value is capped—reduced if necessary—to the MAXTRANTIME value configured for that node.

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

- If an interdomain transaction infects a node that does not understand the 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 BEA Tuxedo nodes infected with the transaction—including the node that started the transaction—generate a TMS timeout message.
- If an interdomain transaction infects a node that understands the 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).

# How the Domains Component Handles Blocking Timeouts

The BEA Tuxedo blocking timeout mechanism uses information stored in the registry slot assigned to each BEA Tuxedo client or server process—one registry slot per process—running on the local machine. Information in the registry slot is used by the local BBL to detect requesters that have been blocked for a time greater than BLOCKTIME. Because a domain gateway process is a multitasking server that can process several service requests at a time (which would require multiple registry slots), domain gateways cannot use the registry slot mechanism. When a blocking timeout condition arises in a Domains environment, a domain gateway sends an error/failure reply message to the requester and *cleans* any context associated with the service request.

In the DM_LOCAL section of the DMCONFIG file, you can set the blocking timeout for a local domain access point using the BLOCKTIME parameter. For example:

```
*DM_LOCAL
LOCAL1   GWGRP=GWTGROUP
         TYPE=TDOMAIN
         ACCESSPOINTID="BA.CENTRAL01"
         BLOCKTIME=30
```

The BLOCKTIME parameter specifies the maximum wait time a blocking ATMI call will block before timing out. A blocking timeout condition implies that the affected service request has failed.

The blocking timeout value is a multiplier of the SCANUNIT parameter specified in the RESOURCES section of the UBBCONFIG file. The value SCANUNIT * BLOCKTIME must be greater than or equal to SCANUNIT and less than or equal to 32,767 seconds.

If BLOCKTIME is not specified in the DMCONFIG file, the default is set to the value of the BLOCKTIME parameter specified in the RESOURCES section of the UBBCONFIG file. If the BLOCKTIME parameter is not specified in the UBBCONFIG file, the default is set so that (SCANUNIT * BLOCKTIME) is approximately 60 seconds.

Be aware that *interdomain* transactions generate blocking timeout conditions when transaction duration exceeds BLOCKTIME. That is, for an interdomain transaction, if the BLOCKTIME value is

less than (a) the TRANTIME timeout value specified in the SERVICES section of the UBBCONFIG file or (b) the timeout value passed in the tpbegin() call to start the transaction, the timeout for the transaction is reduced to the BLOCKTIME value. In contrast, for *intradomain* transactions (that is, transactions handled within a single BEA Tuxedo domain), the BLOCKTIME value specified in the RESOURCES section of the TUXCONFIG file has *no* effect on the timeout of an intradomain transaction.

# Specifying Domains Connection Policies

You can specify the conditions under which a local domain gateway tries to establish a connection to one or more remote domains by selecting one of the following connection policies:

- ON_DEMAND (default)—Connect when requested by either (1) a client request to a remote service or (2) an administrative "connect" command. Under this connection policy, a connection can be established in any of the following ways:
  - Client request
  - Manually through the dmadmin(1) connect command
  - Through an incoming connection
- ON_STARTUP—Connect at gateway server initialization (boot) time. Under this connection policy, a connection can be established in any of the following ways:
  - Automatically when the BEA Tuxedo application boots
  - Manual through the dmadmin(1) connect command
  - Through an incoming connection
- INCOMING_ONLY—Accept incoming connections but do not initiate a connection automatically. Under this connection policy, a connection can be established in any of the following ways:
  - Manually through the dmadmin(1) connect command
  - Through an incoming connection

Connection policy applies only to TDomain gateways.

## How To Configure Your Connection Policy

In the DM_LOCAL section of the DMCONFIG file, you set the connection policy for a local domain access point using the CONNECTION_POLICY parameter. For example:

```
*DM_LOCAL
LOCAL1   GWGRP=GWTGROUP
         TYPE=TDOMAIN
         ACCESSPOINTID="BA.CENTRAL01"
         BLOCKTIME=30
         CONNECTION_POLICY=ON_STARTUP
```

If you do not specify a connection policy for a local domain access point, the connection policy for that access point defaults to ON_DEMAND.

## Per Local/Remote Domain Connection Policy

For TDomain gateways running BEA Tuxedo release 8.1 or later software, you can set the connection policy on a per local or per remote domain basis in the DM_TDOMAIN section of the DMCONFIG file. For example:

```
*DM_LOCAL
LOCAL1   GWGRP=GWTGROUP
         TYPE=TDOMAIN
         ACCESSPOINTID="BA.CENTRAL01"
         BLOCKTIME=30

*DM_REMOTE
REMOT1   TYPE=TDOMAIN
         DOMAINID="REMOT1"

REMOT2   TYPE=TDOMAIN
         DOMAINID="REMOT2"

*DM_TDOMAIN
LOCAL1   NWADDR="//albany.acme.com:4051"
         CONNECTION_POLICY=ON_STARTUP
REMOT1   NWADDR="//newyork.acme.com:65431"
         CONNECTION_POLICY=ON_DEMAND
REMOT2   NWADDR="//philly.acme.com:65431"
```

The connection policy specified for a remote domain access point takes precedence over the connection policy specified for a local domain access point. So, in the preceding example, the connection policy configurations will be:

LOCAL1 to REMOT1 — ON_DEMAND
LOCAL1 to REMOT2 — ON_STARTUP

For BEA Tuxedo 8.1 or later, you can specify any of the following connection policy values for a *local domain access point* in the DM_TDOMAIN section of the DMCONFIG file:

- ON_DEMAND
- ON_STARTUP
- INCOMING_ONLY

Specifying no connection policy for a local domain access point defaults to the global connection policy specified in the DM_LOCAL section of the DMCONFIG file. Specifying a global connection policy in the DM_TDOMAIN section takes precedence over the global connection policy specified in the DM_LOCAL section.

**Note:** If you choose to specify a global connection policy in the DM_TDOMAIN section, do not specify a global connection policy in the DM_LOCAL section.

For BEA Tuxedo 8.1 or later, you can also specify any of the following connection policy values for a *remote domain access point* in the DM_TDOMAIN section of the DMCONFIG file:

- LOCAL (default)
- ON_DEMAND
- ON_STARTUP
- INCOMING_ONLY

Specifying LOCAL or no connection policy for a remote domain access point defaults to the global connection policy.

Without the remote-domain connection policy capability, a global connection policy of ON_STARTUP means that the local TDomain gateway will try to connect to *all* remote domains at boot time, even if some of the remote domains will not be used initially. With a large number of remote domains, the boot time could be substantial. With the remote-domain connection policy capability, you can select which remote domain connections *not* to automatically establish at boot time for a global connection policy of ON_STARTUP.

## Per TDomain Session Connection Policy

Beginning with BEA Tuxedo 9.0, you can set the connection policy for TDomain gateways on a per TDomain session basis in the DM_TDOMAIN section of the DMCONFIG file.

In order to initiate a per TDomain session connection policy you must do the following:

- Establish a TDomain session between specific local and remote domain gateway access points. This gives you the ability to restrict session connection accessibility to one or more local and remote domain gateways.

You can create one or more record entries to describe the parameters and attributes you want to use per TDomain session.

- Specify the `connection_policy` parameter attributes you want to use. If you do not specify a connection policy for a TDomain session, the connection policy attribute for that session point defaults to `LOCAL`.

## Creating A TDomain Session

Two parameters in the `DM_TDOMAIN` section of the `DMCONFIG` file are used to create a TDomain session:

- `FAILOVERSEQ`: Specifies a TDomain session failover sequence and primary record.
- `LACCESSPOINT` **(also known as `LDOM`)**: Specifies the name of a local domain access point listed in the `DM_LOCAL` section.

You can specify other TDomain session parameters and attributes, for example `SECURITY` and `DMKEEPALIVE`. For more information on `FAILOVERSEQ` and `LACCESSPOINT`, as well as other TDomain parameters and attributes, see Optional parameters for the DM_TDOMAIN section in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*.

## Creating A Per TDomain Session Connection Policy

The `FAILOVERSEQ`, `LACCESSPOINT` and `CONNECTION_POLICY` parameters are used to establish a per TDomain session policy and are specified in the following example:

**Listing 1-1 Per TDOMAIN Session Connection Policy Example**

```
*DM_LOCAL
LOCAL1   GWGRP=GWTGROUP
         TYPE=TDOMAIN
         ACCESSPOINTID="BA.CENTRAL01"
         BLOCKTIME=30

LOCAL2   GWGRP=GWTGROUP
         TYPE=TDOMAIN
         ACCESSPOINTID="BA.CENTRAL02"
         BLOCKTIME=30

*DM_REMOTE
REMOT1   TYPE=TDOMAIN
         DOMAINID="REMOT1"
```

```
REMOT2   TYPE=TDOMAIN
         DOMAINID="REMOT2"

*DM_TDOMAIN
LOCAL1   NWADDR="//albany.acme.com:4051"
LOCAL2   NWADDR="//chicago.acme.com:4032"
LOCAL1   NWADDR="//albany.acme.com:4052"
REMOT1   NWADDR="//newyork.acme.com:65431" LACCESSPOINT=LOCAL1
         CONNECTION_POLICY=ON_STARTUP
         MINENCRYPTBITS=128 MAXENCRYPTBITS=128
         FAILOVERSEQ=100
REMOT1   NWADDR="//newyork.acme.com:65432" LACCESSPOINT=LOCAL2
         CONNECTION_POLICY=INCOMING_ONLY
         FAILOVERSEQ=110
REMOT2   NWADDR="//philly.acme.com:65431" LACCESSPOINT=LOCAL2
         CONNECTION_POLICY=ON_DEMAND
         FAILOVERSEQ=120
REMOT1   NWADDR="//detroit.acme.com:65431" LACCESSPOINT=LOCAL1
         CONNECTION_POLICY=INCOMING_ONLY
         MINENCRYPTBITS=40 MAXENCRYPTBITS=40
         FAILOVERSEQ=130
```

The DM_TDOMAIN section consists of seven records that include three TDomain sessions.

**Table 1-2 TDomain Sessions**

| Record | Domain Session | Connection Policy | Connection/ Failover Sequence | Session Hierarchy |
|---|---|---|---|---|
| 4 | [LOCAL1,REMOT1] | ON_STARTUP | 1st | Primary |
| 7 | [LOCAL1,REMOT1] | Ignored | 2nd | Secondary |
| 5 | [LOCAL2,REMOT1] | INCOMING_ONLY | 1st | Primary |
| 6 | [LOCAL2,REMOT2] | ON_DEMAND | 1st | Primary |

- Record 4 is the primary record for TDomain session LOCAL1,REMOT1 since its FAILOVERSEQ number is smaller than record 7. The connection policy for this TDomain session is ON_STARTUP, and it requires 128 bits Link-Level Encryption security policy. If

connection to this record fails, then a connection attempt is made to its secondary/backup record, which is record 7.

- Record 7 is the secondary or backup record for TDomain session LOCAL1,REMOT1 since its FAILOVERSEQ number is larger than record 4.

  The connection and security policies for record 7 are ignored because record 4 is the primary record for this session. Record 7 has no secondary/backup failover record. If connection to record 7 fails, then a connection to record 4 is retried as determined by RETRY_INTERVAL until MAXRETRY is exhausted. For more information about RETRY_INTERVAL, see "How To Use Connection Retry Processing" on page 1-37.

- Record 5 is the primary record for TDomain session LOCAL2,REMOT1. The connection policy for this TDomain session is INCOMING_ONLY. There is no secondary/backup failover record for this session.

- Record 6 is the primary record for TDomain session LOCAL2,REMOT2. The connection policy for this session is ON_DEMAND. There is no secondary/backup failover record for this session. Local access point LOCAL2 connects with two TDomain sessions: one with REMOT1 in record 5, and another with REMOT2.

If two or more records for the same TDomain session have the same FAILOVERSEQ value, the first record entered will be the primary record. The failover sequence for the remaining records is determined based on record-entry order.

To initiate your per TDomain session policy, perform the following steps:

- Use `dmloadcf -y` to compile and convert DMCONFIG file into BDMCONFIG file.
- Use `tmboot -y` to boot the server

## Using Regular Expressions with TDomain Sessions

To make the DMCONFIG file smaller and easier to work with, LACCESSPOINT can contain regular expressions to describe multiple local domain access points.

**Note:** DM_TDOMAIN is the only section in the DMCONFIG file that allows LACCESSPOINT to contain regular expressions.

When the DMCONFIG file is compiled, regular expressions are expanded to their full local domain names in the output binary BDMCONFIG file. The size of the BDMCONFIG file is increased accordingly as shown in the following example:

**Listing 1-2 DMCONFIG File with Regular Expressions**

```
*DM_LOCAL
ALPHA1    . . .
ALPHA2    . . .
ALPHA3    . . .
ALPHA10    . . .
ALPHA11    . . .
ALPHA24    . . .
ALPHA36    . . .
BETA2    . . .
BETA3    . . .
BETA15    . . .
BETA20    . . .
*DM_REMOTE
REMOT1    . . .
REMOT2    . . .
REMOT3    . . .
*DM_TDOMAIN
REMOT1  NWADDR="//philly.acme.com:65431" LACCESSPOINT=ALPHA1
        CONNECTION_POLICY=INCOMING_ONLY
        FAILOVERSEQ=100
REMOT1  NWADDR="//philly.acme.com:65432" LACCESSPOINT=BETA2
        CONNECTION_POLICY=ON_DEMAND
        FAILOVERSEQ=110
REMOT1  NWADDR="//philly.acme.com:65433" LACCESSPOINT="ALPHA[1-2][0-9]"
        CONNECTION_POLICY=ON_STARTUP
        FAILOVERSEQ=120
REMOT1  NWADDR="//philly.acme.com:65434" LACCESSPOINT="BETA[1-2][0-9]*"
        CONNECTION_POLICY=ON_STARTUP
        FAILOVERSEQ=130
```

TDomain session records three and four use regular expressions to define local access points. When `dmloadcf` parses this `DMCONFIG` file, the `BDMCONFIG` file output is as follows.

**Listing 1-3 Compiled BDMCONFIG File for DMCONFIG File with Regular Expressions**

```
REMOT1  NWADDR="//philly.acme.com:65431" LACCESSPOINT=ALPHA1
        CONNECTION_POLICY=INCOMING_ONLY
```

```
        FAILOVERSEQ=100
REMOT1  NWADDR="//philly.acme.com:65432" LACCESSPOINT=BETA2
        CONNECTION_POLICY=ON_DEMAND
        FAILOVERSEQ=110
REMOT1  NWADDR="//philly.acme.com:65433" LACCESSPOINT="ALPHA10"
        CONNECTION_POLICY=ON_STARTUP
        FAILOVERSEQ=120
REMOT1  NWADDR="//philly.acme.com:65433" LACCESSPOINT="ALPHA11"
        CONNECTION_POLICY=ON_STARTUP
        FAILOVERSEQ=120
REMOT1  NWADDR="//philly.acme.com:65433" LACCESSPOINT="ALPHA24"
        CONNECTION_POLICY=ON_STARTUP
        FAILOVERSEQ=120
REMOT1  NWADDR="//philly.acme.com:65434" LACCESSPOINT="BETA2"
        CONNECTION_POLICY=ON_STARTUP
        FAILOVERSEQ=130
REMOT1  NWADDR="//philly.acme.com:65434" LACCESSPOINT="BETA15"
        CONNECTION_POLICY=ON_STARTUP
        FAILOVERSEQ=130
REMOT1  NWADDR="//philly.acme.com:65434" LACCESSPOINT="BETA20"
        CONNECTION_POLICY=ON_STARTUP
        FAILOVERSEQ=130
```

## Using DM_MIB to Specify or Request TDomain Session Information

Using DM_MIB to specify and request TDomain session information directly modifies the BDMCONFIG file. The original DMCONFIG file is unmodified. For more information about DM_MIB, see DM_MIB(5) in *Section 5 - File Formats, Data Descriptions, MIBs, and System Processes Reference*.

**Note:** You can use dmunloadcf >DMCONFIG to parse the BDMCONFIG file to update its changes in the DMCONFIG file. For more information about dmunloadcf, see,"Domains Administrative Tools" on page 1-13.

DM_MIB uses three T_DM_TDOMAIN Class Definition attributes to create a per TDomain session connection policy in the BDMCONFIG file:

- TA_DMFAILOVERSEQ: Specifies and requests the session connection failover sequences and primary records for a TDomain session record in the BDMCONFIG file.
- TA_DMLACCESSPOINT: Specifies and requests a local domain access point found in the DM_LOCAL section for a TDomain session record in the BDMCONFIG file.
- TA_DMCONNECTION_POLICY: Specifies a TDomain connection policy

You can also specify and request other T_DM_TDOMAIN Class Definition attributes, for example security and keepalive. For more T_DM_TDOMAIN Class Definition attribute information, see T_DM_TDOMAIN Class Definition in *Section 5 - File Formats, Data Descriptions, MIBs, and System Processes Reference*.

You can use DM_MIB to add, delete, or retrieve TDomain session records in the BDMCONFIG file. All applicable T_DM_TDOMAIN Class Definition key fields must be used to add, delete, or retrieve requests for TDomain session record information.

For example:

Example 1: DM_MIB request used to add a new TDomain session and connection policy record.

```
TA_OPERATION                SET
TA_CLASS                    T_DM_TDOMAIN
TA_DMACCESSPOINT            RDOM1
TA_DMNWADDR                 //philly.acme.com:65431
TA_STATE                    NEW
TA_DMLACCESSPOINT           LDOM3
TA_DMFAILOVERSEQ            50
TA_DMCONNECTION_POLICY      ON_STARTUP
```

This will add the following TDomain session record in BDMCONFIG file:

```
RDOM1   NWADDR="//philly.acme.com:65431" LACCESSPOINT=LDOM3
        FAILOVERSEQ=50
        CONNECTION_POLICY=ON_STARTUP
```

Example 2: DM_MIB request used to delete an existing TDomain session connection policy record.

The requested record is marked “invalid” in the BDMCONFIG file and is not included in the TDomain session.

```
TA_OPERATION                SET
TA_CLASS                    T_DM_TDOMAIN
TA_DMACCESSPOINT            RDOM1
TA_DMNWADDR                 //philly.acme.com:65431
TA_STATE                    INV
TA_DMLACCESSPOINT           LDOM3
TA_DMFAILOVERSEQ            50
TA_DMCONNECTION_POLICY      ON_STARTUP
```

Example 3: DM_MIB request used to retrieve an existing TDomain session connection policy record.

```
TA_OPERATION                GET
TA_CLASS                    T_DM_TDOMAIN
TA_DMACCESSPOINT            RDOM1
TA_DMNWADDR                 //philly.acme.com:65431
```

```
TA_STATE                    INV
TA_DMLACCESSPOINT           LDOM3
TA_DMFAILOVERSEQ            50
TA_DMCONNECTION_POLICY      ON_STARTUP
```

## Using DMADMIN to Specify or Request TDomain Session Information

You can use Tuxedo's command line interface, DMADMIN, to specify and request TDomain session information. For a general description of DMADMIN, see"Domains Administrative Tools" on page 1-13.

Using DMADMIN to specify and request TDomain information works similarly to using DM_MIB. That is to say, using DMADMIN modifies the BDMCONFIG file and leaves the original DMCONFIG file unmodified.

DMADMIN uses three field indentifiers to add a per TDomain connection policy record in the BDMCONFIG file:

- TA_DMFAILOVERSEQ: Specifies and requests the session connection failover sequences and primary records for a TDomain session record in the BDMCONFIG file.
- TA_LDOM: Specifies and requests a local domain access point found in the DM_LOCAL section for a TDomain session record in the BDMCONFIG file.
- TA_CONNECTION_POLICY: Specifies a TDomain connection policy

For more information about TA_DMFAILOVERSEQ, TA_LDOM, TA_CONNECTION_POLICY and other field indentifiers, see dmadmin(1) in *BEA Tuxedo Command Reference*.

You can use DMADMIN to add, delete, or retrieve TDomain session records. The following example illustrates how DMADMIN is used to add a TDomain session connection policy record in the BDMCONFIG file:

```
TA_CMPLIMIT                 2147483647
TA_MINENCRYPTBITS           0
TA_MAXENCRYPTBITS           128
TA_DMNWADDR                 //philly.acme.com:65431
TA_LDOM                     LDOM3
TA_DMFAILOVERSEQ            50
TA_RDOM                     RDOM1
TA_CONNECTION_POLICY        ON_STARTUP
```

### TDomain Session Interoperability with Older Tuxedo Releases

Tuxedo 9.x TDomain gateways can communicate with older Tuxedo release TDomain gateways. However, if you want to use the TDomain session feature running Tuxedo 9.x and 8.1 in a *mixed application environment*, please note the following limitations:

- You must use Tuxedo 9.x `DMADM` server and `dmloadcf` when you want to create a TDomain session.
- You should not limit Tuxedo 8.1 local domain gateway access to remote domains. If you do, you run the risk of getting a routing failure error. Message routing in Tuxedo 8.1 assumes local domain gateway is capable of connecting to *all* remote domains.

The TDomain session feature *does not* work with Tuxedo releases older than 8.1 in a *mixed application environment*.

## How To Use Connection Retry Processing

When `CONNECTION_POLICY` is set to `ON_STARTUP`, you can configure two other parameters to determine how many times the local domain gateway attempts to establish a connection to the remote domains. By default, the local domain gateway retries failed connections every 60 seconds, but you can specify a different value for this interval using parameters `MAXRETRY` and `RETRY_INTERVAL`.

You use the `MAXRETRY` parameter to specify the number of times that a domain gateway tries to establish connections to remote domains. The minimum value is 0, and the maximum value is 2147483647. The default setting is 2147483647. Setting this parameter to 0 turns off connection retry processing.

You use the `RETRY_INTERVAL` parameter to specify the number of seconds between automatic attempts to establish a connection to remote domains. The minimum value is 0, and the maximum value is 2147483647. The default setting is 60. If the `MAXRETRY` parameter is set to 0, setting `RETRY_INTERVAL` is not allowed.

Example 1:

```
*DM_LOCAL
LOCAL1  GWGRP=GWTGROUP
        TYPE=TDOMAIN
        ACCESSPOINTID="BA.CENTRAL01"
        BLOCKTIME=30
        CONNECTION_POLICY=ON_STARTUP
        MAXRETRY=5
        RETRY_INTERVAL=100
```

Example 2 (Only possible for TDomain gateways running BEA Tuxedo release 8.1 or later software):

```
*DM_LOCAL
LOCAL1  GWGRP=GWTGROUP
        TYPE=TDOMAIN
        ACCESSPOINTID="BA.CENTRAL01"
        BLOCKTIME=30

*DM_TDOMAIN
LOCAL1  NWADDR="//albany.acme.com:4051"
        CONNECTION_POLICY=ON_STARTUP
        MAXRETRY=5
        RETRY_INTERVAL=100
REMOT1  NWADDR="//newyork.acme.com:65431"
        CONNECTION_POLICY=ON_STARTUP
        MAXRETRY=10
        RETRY_INTERVAL=40
```

In the second example, the MAXRETRY and RETRY_INTERVAL values 10 and 40 will be the automatic connection retry criteria used by the local TDomain gateway to establish a connection to the remote domain access point named REMOT1.

# How Connection Policy Determines Availability of Remote Services

The connection policy that you specify determines how services imported from a remote domain are advertised in the BEA Tuxedo bulletin board by the domain gateway:

- For ON_DEMAND, the local domain gateway continually advertises services imported from a remote domain.
- For ON_STARTUP, the local domain gateway advertises services imported from a remote domain as long as a connection exists to the remote domain.
- For INCOMING_ONLY, the local domain gateway advertises services imported from a remote domain when the gateway receives an incoming connection or when a `dmadmin connect` command is issued.

When the connection policy is ON_STARTUP or INCOMING_ONLY (but not ON_DEMAND), *Dynamic Status*, a TDomain gateway feature, checks the status of remote services. The status of a remote service depends on the status of the network connection between the local and remote domain gateways. Remote services are advertised and available on the local domain whenever a

connection is successfully established to the domain on which they reside. Remote services are suspended and unavailable whenever the connection is not established to the domain on which they reside.

For each service, the domain gateway keeps track not only of the remote domains from which the service is imported, but also of which remote domains are available. In this way, the gateway provides intelligent load balancing of requests to remote domains. If all the remote domains from which a service is imported become unreachable, the domain gateway suspends the service in the BEA Tuxedo bulletin board.

For example, suppose a service named RSVC is imported from two remote domains, as specified by the following entries in the DM_IMPORT section of the DMCONFIG file:

```
*DM_IMPORT
RSVC   AUTOTRAN=N
       RACCESSPOINT=REMOT1
       LACCESSPOINT=LOCAL1
RSVC   AUTOTRAN=N
       RACCESSPOINT=REMOT2
       LACCESSPOINT=LOCAL1
```

When connections to both REMOT1 and REMOT2 are up, the domain gateway load balances requests for the RSVC service. If the connection to REMOT1 goes down, the gateway sends all requests for RSVC to REMOT2. If both connections go down, the gateway suspends RSVC in the bulletin board. Subsequent requests for RSVC are either routed to a local service or fail with TPENOENT.

## See Also

- Optional parameters for the DM_TDOMAIN section
- T_DM_TDOMAIN Class Definition
- “Setting Up Connections in a Domains Configuration” on page 2-55
- “Controlling Connections in a Domains Configuration” on page 2-62

# Specifying Domains Failover and Failback

In the DM_IMPORT section of the DMCONFIG file, you can set up the Domains-level failover and failback functionality for your Domains configuration. In the DM_TDOMAIN section of the DMCONFIG file, you can set up the Domains link-level failover functionality for your Domains configuration.

# How to Configure Domains-Level Failover and Failback

Domains-level failover is a mechanism that transfers requests to alternate remote domains when a failure is detected with a primary remote domain. It also provides failback to the primary remote domain when that domain is restored.

To support Domains-level failover and failback, you specify a list of the remote domain access points through which a particular service can be executed. For example:

```
*DM_IMPORT
TOUPPER RACCESSPOINT="REMOT1,REMOT2,REMOT3"
```

In this example, the TOUPPER service can be executed through any of three remote domain access points: REMOT1 (primary), REMOT2, and REMOT3. When REMOT1 is unavailable, REMOT2 is used for failover. When REMOT1 and REMOT2 are both unavailable, REMOT3 is used for failover.

You must specify ON_STARTUP or INCOMING_ONLY as the value of the CONNECTION_POLICY parameter if you want to configure alternate remote domains for a service. If you specify ON_DEMAND as your connection policy, your servers cannot "fail over" to the alternate remote domains that you have specified in the RACCESSPOINT parameter.

Domains-level failback occurs when a network connection to the primary remote domain is re-established for any of the following reasons:

- Automatic connection retries (ON_STARTUP only)
- Incoming connections
- Manual dmadmin connect command

# How to Configure Domains Link-Level Failover

Domains link-level failover is a mechanism that ensures that a secondary network link becomes active when a primary network link fails. However, it does not provide failback to the primary link when that link is restored, meaning that when the primary link is restored, you must manually bring down the secondary link to force traffic back onto the primary link.

To configure Domains link-level failover, you specify multiple entries for a remote domain access point in the DM_TDOMAIN section of the DMCONFIG file. For example:

```
*DM_TDOMAIN
REMOT1  NWADDR="//newyork.acme.com:65431"
REMOT1  NWADDR="//trenton.acme.com:65431"
```

The first entry is considered to be the primary address, which means its NWADDR is the first network address tried when a connection is being attempted to the remote domain access point. The second entry is considered to be the secondary address, which means its NWADDR is the second network address tried when a connection cannot be established using the primary address.

The second entry points to a secondary remote gateway that must reside in a different BEA Tuxedo domain than the BEA Tuxedo domain in which the primary remote gateway resides. The secondary and primary remote gateways must have the same ACCESSPOINTID defined in the DM_LOCAL section of their associated DMCONFIG files; this arrangement is often referred to as a *mirrored* gateway. This feature is not recommended for use with transactions or conversations. In addition, the mirrored gateway is not recommended for use when the primary remote gateway is available.

### How to Configure TDomain Session Link-Level Failover

The FAILOVERSEQ parameter in the DM_TDOMAIN section of the DMCONFIG file is used to configure TDomain session link-level failover. For more information about specifying FAILOVERSEQ in a TDomain session, see "Per TDomain Session Connection Policy" on page 1-29.

You can also use the TA_DMFAILOVERSEQ attribute in DM_MIB to configure TDomain session link-level failover. For more information, see "Using DM_MIB to Specify or Request TDomain Session Information" on page 1-34.

## Specifying Domains Keepalive

Domains keepalive, available for TDomain gateways running BEA Tuxedo release 8.1 or later software, allows you to enable and configure a keepalive protocol at the TCP level and/or application level for each TDomain gateway connection. TCP-level keepalive and application-level keepalive are not mutually exclusive, meaning that you can configure a Domains connection using both options.

The following table provides some key information about Domains keepalive.

**Table 1-3 About Domains Keepalive**

| Level | Interoperate With Earlier Tuxedo Release? | Individual Timer? | Quicker Connection Failure Detection? | Keepalive Event With Firewall? |
|---|---|---|---|---|
| TCP-Level Keepalive | Yes | No | Yes * | Yes |
| Application-Level Keepalive | No | Yes | Yes | Yes |

* For TCP-level keepalive to quickly detect a TDomain gateway connection failure, it must be set to a small time interval. Doing so may flood the network with TCP packets.

Most BEA Tuxedo Domains configurations span across firewalls, and firewalls typically time out idle connections. Not only will Domains keepalive keep BEA Tuxedo interdomain connections open during periods of inactivity, but it will also enable TDomain gateways to quickly detect Domains connection failures. Currently, a TDomain gateway learns of a Domains connection failure through the underlying TCP stack, which may report the failure 15 minutes or more (depending on the local operating system configuration) after the failure occurs.

## What is TCP-Level Keepalive?

Although the keepalive functionality is not part of the TCP specification, most operating systems provide a TCP keepalive timer. The TCP keepalive timer allows the server machine at one end of a TCP connection to detect whether the client machine at the other end of the connection is reachable.

Every message received by the server machine over the TCP connection resets the TCP keepalive timer. If the keepalive timer detects no activity on the TCP connection for a predefined period of time (typically two hours), the timer expires, and the server machine sends a *probe segment* packet to the client machine. If the connection is still open and the client machine is still alive, the client machine responds by sending an acknowledgement to the server machine. If the server machine does not receive an acknowledgement within a fixed period of time of sending the probe segment packet, the server machine assumes that the connection is broken and releases any resources associated with the connection.

Besides determining whether the connection is open and the client machine is alive, TCP-level keepalive is a way of keeping idle connections open through firewalls. Automatically sending a

probe segment packet after a predefined period of connection inactivity resets the firewall's idle-connection timer *before it times out*, which allows the connection to stay open.

The interval for an operating system's TCP keepalive timer is typically set to two hours. This interval can be changed, but changing it affects all TCP connections for a machine. An operating system's TCP keepalive interval is a system-wide value.

## How to Configure TCP-Level Keepalive for Domains

The BEA Tuxedo TCP-level keepalive option for Domains is named TCPKEEPALIVE, which has been added as an optional parameter in the DM_TDOMAIN section of the DMCONFIG file. You can use this parameter to enable the Domains TCP-level keepalive option on a per local or per remote domain basis.

The allowed values for TCPKEEPALIVE are:

- LOCAL (relevant only to remote domain access points)
- NO (keepalive disabled)
- YES (keepalive enabled)

By default, the Domains TCP-level keepalive option is disabled. When you enable TCP-level keepalive for a Domains connection, the keepalive interval used for the connection is the system-wide value configured for the operating system's TCP keepalive timer.

To clarify the use of TCPKEEPALIVE, consider the following Domains TCP-level keepalive configuration:

```
*DM_TDOMAIN
LOCAL1   NWADDR="//albany.acme.com:4051"
         TCPKEEPALIVE=Y
REMOT1   NWADDR="//newyork.acme.com:65431"
REMOT2   NWADDR="//philly.acme.com:65431"
         TCPKEEPALIVE=NO
```

The TCP-level keepalive configuration specified for a remote domain access point takes precedence over the TCP-level keepalive configuration specified for the local domain access point. So, in the preceding example, the TCP-level keepalive configurations will be:

LOCAL1 to REMOT1 — TCP-level keepalive enabled
LOCAL1 to REMOT2 — TCP-level keepalive disabled

For a local domain access point, you can specify any of the following values for the TCPKEEPALIVE parameter:

- NO (default)
- YES

For a remote domain access point, you can specify any of the following values for the TCPKEEPALIVE parameter:

- LOCAL (default)
- NO
- YES

Specifying LOCAL or no configuration for a remote domain access point defaults to the local TCP-level keepalive configuration.

**Note:** You can enable each of two interoperating BEA Tuxedo domains with TCP-level keepalive, assuming that both domains are running BEA Tuxedo 8.1 or later software.

If the connection policy for a Domains connection is ON_STARTUP and the TCP connection is closed due to a TCP-level keepalive failure, automatic connection retry attempts. If the connection retry is not successful, you must use the dmadmin connect command to re-establish the connection. For information about the dmadmin connect command, see "How to Establish Connections Between Domains" on page 2-62.

## What is Application-Level Keepalive?

Some people argue against using the operating system's TCP keepalive, citing that the probe segment packets consume unnecessary bandwidth and waste money on internet connections where users pay on a per packet basis. Some people also believe that keepalive belongs in the application layer or link layer, not in the transport (TCP) layer, citing that the application layer should:

- Decide whether the application has been waiting an excessively long time to receive incoming messages.
- Decide what actions to take to determine whether the TCP connection is still open and that the machine and application at the other end of the connection are still running.

Regardless of who thinks what, one advantage of application-level keepalive over TCP-level keepalive is that the interval for the keepalive timer can be set on a per connection basis. With TCP-level keepalive, the timer interval must be set on a per machine basis.

Using application-level keepalive, the server application sends an application-specific keepalive message whenever the application keepalive timer times out. (Typically, the keepalive message

consists of just header information, meaning that the message has no associated data.) The client application responds by sending an acknowledgement to the server application. If the server application does not receive an acknowledgement within a predefined period of time of sending the keepalive message, the server application assumes that the connection is broken and releases any resources associated with the connection.

Besides determining whether the connection is open and the client application is running, application-level keepalive is a way of keeping idle connections open through firewalls. Automatically sending a keepalive message after a predefined period of connection inactivity resets the firewall's idle-connection timer *before it times out*, which allows the connection to stay open.

## How to Configure Application-Level Keepalive for Domains

The BEA Tuxedo application-level keepalive option for Domains is named KEEPALIVE. This parameter and a companion parameter named KEEPALIVEWAIT have been added as optional parameters in the DM_TDOMAIN section of the DMCONFIG file. You can use these parameters to configure the Domains application-level keepalive option on a per local or per remote domain basis.

You use the DMKEEPALIVE parameter to specify the maximum time that the local TDomain gateway will wait without receiving any traffic on the Domains connection; if the maximum time is exceeded, the gateway sends an application-level keepalive request message. The allowed values for DMKEEPALIVE are:

- -1 (relevant only to remote domain access points)
- 0 (keepalive disabled)
- 1 <= *value* <= 2147483647 (keepalive enabled), in milliseconds, currently rounded up to the nearest second by the Domains software

The DMKEEPALIVE default setting is 0.

You use the DMKEEPALIVEWAIT parameter to specify the maximum time that the local TDomain gateway will wait without receiving an acknowledgement to a sent keepalive message. If the maximum time is exceeded, the gateway assumes that the connection to the remote TDomain gateway is broken and releases any resources associated with the connection. The minimum value for DMKEEPALIVEWAIT is 0, and the maximum value is 2147483647 milliseconds, currently rounded up to the nearest second by the Domains software. The DMKEEPALIVEWAIT default setting is 0.

- If DMKEEPALIVE is 0 (keepalive disabled), setting DMKEEPALIVEWAIT has no effect.
- If DMKEEPALIVE is enabled and DMKEEPALIVEWAIT is set to a value greater than DMKEEPALIVE, the local TDomain gateway will send more than one application-level keepalive message before the DMKEEPALIVEWAIT timer expires. This combination of settings is allowed.
- If DMKEEPALIVE is enabled and DMKEEPALIVEWAIT is set to 0, receiving an acknowledgement to a sent keepalive message is unimportant: any such acknowledgement is ignored by the local TDomain gateway. The local TDomain gateway continues to send keepalive messages every time the DMKEEPALIVE timer times out. *Use this combination of settings to keep an idle connection open through a firewall.*

To clarify the use of DMKEEPALIVE and DMKEEPALIVEWAIT, consider the following Domains application-level keepalive configuration:

```
*DM_TDOMAIN
LOCAL1  NWADDR="//albany.acme.com:4051"
        DMKEEPALIVE=1010
        DMKEEPALIVEWAIT=20
REMOT1  NWADDR="//newyork.acme.com:65431"
        DMKEEPALIVE=4000
        DMKEEPALIVEWAIT=3000
REMOT2  NWADDR="//philly.acme.com:65431"
        DMKEEPALIVE=-1
```

The keepalive configuration specified for a remote domain access point takes precedence over the keepalive configuration specified for the local domain access point. So, in the preceding example, the application-level keepalive configurations will be:

LOCAL1 to REMOT1 — Keepalive timer = 4 seconds, and wait timer = 3 seconds
LOCAL1 to REMOT2 — Keepalive timer = 2 seconds, and wait timer = 1 second

For a local domain access point, you can specify any of the following values for the DMKEEPALIVE parameter:

- 0 (default)
- 1 <= *value* <= 2147483647 in milliseconds, currently rounded up to the nearest second by the Domains software

For a remote domain access point, you can specify any of the following values for the DMKEEPALIVE parameter:

- -1 (default)

- 0
- 1 <= *value* <= 2147483647 in milliseconds, currently rounded up to the nearest second by the Domains software

Specifying -1 or no keepalive configuration for a remote domain access point defaults to the local application-level keepalive configuration.

**Note:** You can configure each of two interoperating BEA Tuxedo domains with application-level keepalive, using the same or different wait intervals, assuming that both domains are running BEA Tuxedo 8.1 or later software.

If the connection policy for a Domains connection is `ON_STARTUP` and the connection experiences an application-level keepalive failure, automatic connection retry processing attempts to re-establish the connection. For more information about connection retry processing, see "How To Use Connection Retry Processing" on page 1-37.

## Keepalive Compatibility with Earlier BEA Tuxedo Releases

Domains TCP-level keepalive is compatible with BEA Tuxedo 8.0 or earlier software. The BEA Tuxedo software running at the other end of the TCP connection may be any release of BEA Tuxedo because Domains TCP-level keepalive is executed at the network transport (TCP) layer.

Domains application-level keepalive is not compatible with BEA Tuxedo 8.0 or earlier software. The BEA Tuxedo software running at the other end of the TCP connection must be BEA Tuxedo 8.1 or later to be able to understand an application-level keepalive message. When connected to a TDomain gateway running an earlier release of BEA Tuxedo software, the TDomain gateway does *not* send an application-level keepalive message; instead, it logs a warning message in the local user log (ULOG) stating that the remote domain is running an earlier release of BEA Tuxedo software and does not support Domains application-level keepalive.

# Configuring a Domains Environment

The following list summarizes the tasks that you must complete to configure a Domains environment for the TDomain gateway type:

1. Edit the `UBBCONFIG` file with any text editor and configure the Domains administrative servers and the TDomain gateway server. For example:

```
*GROUPS
DMADMGRP  LMID=SITE1 GRPNO=1
GWTGROUP  LMID=SITE2 GRPNO=2
```

```
*SERVERS
DMADM      SRVGRP=DMADMGRP
           SRVID=1001
           REPLYQ=N
           RESTART=Y
           GRACE=0
GWADM      SRVGRP=GWTGROUP
           SRVID=1002
           REPLYQ=N
           RESTART=Y
           GRACE=0
GWTDOMAIN  SRVGRP=GWTGROUP
           SRVID=1003
           RQADDR="GWTGROUP"
           REPLYQ=N
           RESTART=Y
           GRACE=0
```

**Note:** In the previous example, REPLYQ=N is specified for the DMADM, GWADM, and GWTDOMAIN servers. This setting is not required: you can, if you prefer, designate a reply queue for any of these servers by specifying REPLYQ=Y. When REPLYQ is set to N, however, performance may be improved.

The TDomain gateway server and its associated GWADM server must run on the same machine in a BEA Tuxedo domain. The DMADM server may run on any machine—master machine, non-master machine—in the BEA Tuxedo domain.

2. Load the BEA Tuxedo configuration by running tmloadcf(1). The tmloadcf command parses UBBCONFIG and loads the binary TUXCONFIG file to the location referenced by the TUXCONFIG variable.

3. Edit the DMCONFIG file with any text editor and configure the Domains environment for the TDomain gateway server. For example:

```
*DM_LOCAL
LOCAL1     GWGRP=GWTGROUP
           TYPE=TDOMAIN
           ACCESSPOINTID="BA.CENTRAL01"
           BLOCKTIME=30
           CONNECTION_POLICY=ON_STARTUP
           MAXRETRY=5
           RETRY_INTERVAL=100

*DM_REMOTE
REMOT1     TYPE=TDOMAIN
           ACCESSPOINTID="BA.BANK01"
```

```
REMOT2     TYPE=TDOMAIN
           ACCESSPOINTID="BA.BANK02"

*DM_EXPORT
LTOLOWER   LACCESSPOINT=LOCAL1
           CONV=N
           RNAME="TOLOWER"

*DM_IMPORT
RTOUPPER   AUTOTRAN=N
           RACCESSPOINT=REMOT1
           LACCESSPOINT=LOCAL1
           CONV=N
           RNAME="TOUPPER"

*DM_TDOMAIN
LOCAL1   NWADDR="//albany.acme.com:4051"
REMOT1   NWADDR="//newyork.acme.com:65431"
REMOT2   NWADDR="//philly.acme.com:65431"
```

The DMCONFIG file must reside on the same machine as the DMADM server.

4. Load the Domains configuration by running dmloadcf(1). The dmloadcf command parses DMCONFIG and loads the binary BDMCONFIG file to the location referenced by the BDMCONFIG variable.

5. Start the BEA Tuxedo application servers by running tmboot(1). The tmboot command executes all administrative processes and all servers listed in the SERVERS section of the TUXCONFIG file named by the TUXCONFIG and TUXOFFSET environment variables. It starts the servers in the order that they are listed in the SERVERS section: DMADM, then GWADM, and then GWTDOMAIN. *The Domains servers must be started in this order.* In addition, the Domains servers must be started before the application servers.

For a detailed example of configuring a Domains ATMI environment, see "Planning and Configuring ATMI Domains" on page 2-1. For a detailed example of configuring a Domains CORBA environment, see "Planning and Configuring CORBA Domains" on page 3-1.

# Configuring a Domains Environment for Migration

The following sample UBBCONFIG and DMCONFIG files give you an idea of how to configure a BEA Tuxedo application for Domains migration. The entries of particular importance to the Domains migration are highlighted in bold.

**Listing 1-4 Sample UBBCONFIG File Configured for Domains Migration**

```
*RESOURCES
IPCKEY    76666
MASTER    SITE1,SITE2
OPTIONS   LAN,MIGRATE
MODEL     MP
#
*MACHINES
mach1     LMID=SITE1
          TUXDIR="/home/rsmith/tuxroot"
          APPDIR="/home/rsmith/bankapp"
          TUXCONFIG="/home/rsmith/bankapp/tuxconfig"
mach2     LMID=SITE2
          TUXDIR="/home/rsmith/tuxroot"
          APPDIR="/home/rsmith/bankapp"
          TUXCONFIG="/home/rsmith/bankapp/tuxconfig"
mach3     LMID=SITE3
          TUXDIR="/home/rsmith/tuxroot"
          APPDIR="/home/rsmith/bankapp"
          TUXCONFIG="/home/rsmith/bankapp/tuxconfig"
#
*GROUPS
DMADMGRP  LMID="SITE1,SITE3" GRPNO=1
GWTGROUP  LMID="SITE2,SITE3" GRPNO=2
.
.
.
*NETWORK
SITE1     NADDR="//albany.acme.com:4065"
          NLSADDR="//albany.acme.com:4068"
SITE2     NADDR="//auburn.acme.com:4065"
          NLSADDR="//auburn.acme.com:4068"
SITE3     NADDR="//boston.acme.com:4065"
          NLSADDR="//boston.acme.com:4068"

#
*SERVERS
```

```
DMADM       SRVGRP=DMADMGRP
            SRVID=1001
            REPLYQ=N
            RESTART=Y
            GRACE=0
GWADM       SRVGRP=GWTGROUP
            SRVID=1002
            REPLYQ=N
            RESTART=Y
            GRACE=0
GWTDOMAIN   SRVGRP=GWTGROUP
            SRVID=1003
            RQADDR="GWTGROUP"
            REPLYQ=N
            RESTART=Y
            GRACE=0
.
.
.
```

**Note:** In the previous example, `REPLYQ=N` is specified for the `DMADM`, `GWADM`, and `GWTDOMAIN` servers. This setting is not required: you can, if you prefer, designate a reply queue for any of these servers by specifying `REPLYQ=Y`. When `REPLYQ` is set to `N`, however, performance may be improved.

**Listing 1-5 Sample DMCONFIG File Configured for Domains Migration**

```
*DM_LOCAL
LOCAL1      GWGRP=GWTGROUP
            TYPE=TDOMAIN
            ACCESSPOINTID="BA.CENTRAL01"
            BLOCKTIME=30
            CONNECTION_POLICY=ON_STARTUP
            MAXRETRY=5
            RETRY_INTERVAL=100
```

```
*DM_REMOTE
REMOT1    TYPE=TDOMAIN
          ACCESSPOINTID="BA.BANK01"
REMOT2    TYPE=TDOMAIN
          ACCESSPOINTID="BA.BANK02"

*DM_EXPORT
LTOLOWER  LACCESSPOINT=LOCAL1
          CONV=N
          RNAME="TOLOWER"

*DM_IMPORT
RTOUPPER  AUTOTRAN=N
          RACCESSPOINT=REMOT1
          LACCESSPOINT=LOCAL1
          CONV=N
          RNAME="TOUPPER"

*DM_TDOMAIN
LOCAL1  NWADDR="//albany.acme.com:4051"
LOCAL1  NWADDR="//boston.acme.com:4051"
REMOT1  NWADDR="//newyork.acme.com:65431"
REMOT2  NWADDR="//philly.acme.com:65431"
```

---

In the sample configuration files, the DMADM server and the TDomain gateway group servers are configured to migrate to the SITE3 machine. For the DMADM migration, an administrator will activate a DMADM server process on the SITE3 machine after completing the following tasks:

- Setting the BDMCONFIG environment variable on the SITE3 machine.
- Running the dmloadcf(1) command to load the BDMCONFIG file on the SITE3 machine.

For the TDomain gateway group migration, an administrator will activate GWADM and GWTDOMAIN server processes on the SITE3 machine. At that point, the configurations and responsibilities associated with the LOCAL1 access point will be handled by the new GWTDOMAIN server process listening for incoming connection requests on network address boston.acme.com:4051.

**Note:** The DMADM and domain gateway group(s) do not have to be migrated to the same machine.

## How to Migrate the DMADM Server

To migrate DMADM to a new machine, follow these steps.

1. Copy DMCONFIG to the new machine and run dmloadcf.
2. Activate the DMADM server process on the new machine. For details, see "Methods for Activating Individual Server Processes" on page 1-53.
3. Optional: Restart all domain gateway groups for the BEA Tuxedo application. For details, see "Methods for Activating Individual Server Processes" on page 1-53.

   If you do not restart the domain gateway groups, they will continue to function, but after DMADM has been migrated, all MIB requests for them will fail.

## How to Migrate a TDomain Gateway Group

When transactions are being used in a Domains configuration, the TDomain gateway group can be migrated only across machines of the same type.

To migrate a TDomain gateway group, follow these steps.

1. In the DMCONFIG file, add multiple listening addresses, in the following format, to the DM_TDOMAIN section:

   ```
   *DM_TDOMAIN
   LOCAL1 NWADDR="//primary:port"
   LOCAL1 NWADDR="//backup:port"
   ```

2. If you are using transactions, you must copy the Domains transaction log manually to the backup machine.
3. The DMCONFIG files for the remote domains should include both network addresses specified in step 1.
4. Activate the GWADM and GWTDOMAIN server processes on the new machine. For details, see the following section.

## Methods for Activating Individual Server Processes

You can use any of the following methods to activate individual BEA Tuxedo server processes:

- BEA Tuxedo Administration Console
- Command tmboot(1) with the -s command line option

- MIB (`TM_MIB(5)`) API

For information about performing application migration tasks, see "Migrating Your Application" in *Administering a BEA Tuxedo Application at Run Time*.

CHAPTER 2

# Planning and Configuring ATMI Domains

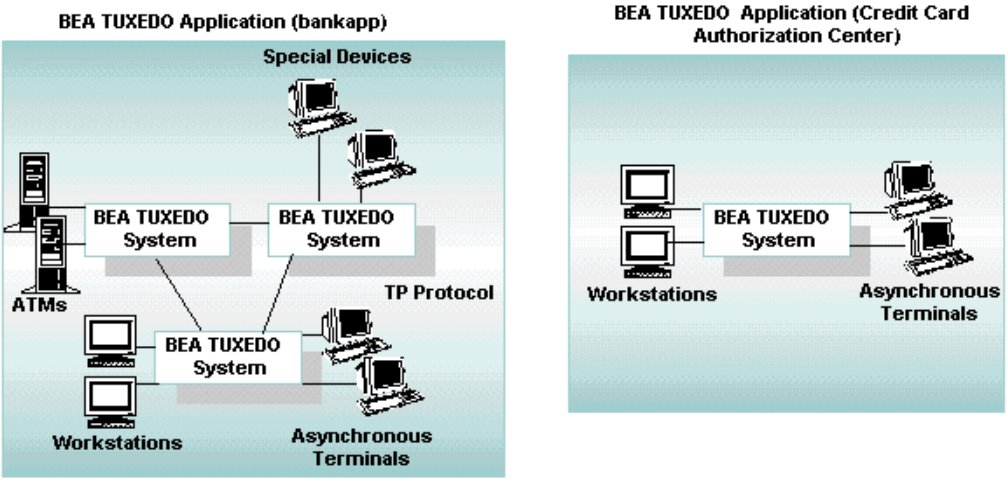
The following sections explain how to plan and configure a domain for a BEA Tuxedo ATMI Domains environment:

- Planning to Build Domains from Multiple BEA Tuxedo Applications
- Examining the creditapp Domains Configuration
- Setting Up a Domains Configuration
- Setting Up Security in a Domains Configuration
- Setting Up Connections in a Domains Configuration
- Controlling Connections in a Domains Configuration
- Configuring Domains Link-Level Failover and Keepalive

## Planning to Build Domains from Multiple BEA Tuxedo Applications

The following figure shows two BEA Tuxedo applications: the `bankapp` application and a credit card authorization application.

**Figure 2-1 Two BEA Tuxedo Applications**




The `bankapp` application connects ATMs at various bank branches to the central bank office. The credit card authorization application processes customer requests for credit cards. Over time, the bank managers realize that their customers would be better served if the `bankapp` application could communicate directly with the credit card authorization application. With direct communication, the bank could offer instant credit cards to anyone opening a new account.

The `bankapp` application is a sample application included with the BEA Tuxedo distribution, and the credit card authorization application is a hypothetical extension of `bankapp`. The `bankapp` application files reside at the following location:

- *`tux_prod_dir`*`\samples\atmi\bankapp` (Windows)
- *`tux_prod_dir`*`/samples/atmi/bankapp` (UNIX)

Where *`tux_prod_dir`* represents the directory in which the BEA Tuxedo distribution is installed.

The following listing shows the content of a file named `ubbmp`, which is the `UBBCONFIG` file for the multiple-machine version of the `bankapp` application.

**Listing 2-1 ubbmp Configuration File for the bankapp Application**

```
.
.
.
*RESOURCES
IPCKEY            80952
UID               <user id from id(1)>
GID               <group id from id(1)>
PERM              0660
MAXACCESSERS      40
MAXSERVERS        35
MAXSERVICES       75
MAXCONV           10
MAXGTT            20
MASTER            SITE1,SITE2
SCANUNIT          10
SANITYSCAN        12
BBLQUERY          30
BLOCKTIME         30
DBBLWAIT          6
OPTIONS           LAN,MIGRATE
MODEL             MP
LDBAL             Y
##SECURITY        ACL
##AUTHSVC         "..AUTHSVC"
#
*MACHINES
<SITE1's uname>   LMID=SITE1
                  TUXDIR="<TUXDIR1>"
                  APPDIR="<APPDIR1>"
                  ENVFILE="<APPDIR1>/ENVFILE"
                  TLOGDEVICE="<APPDIR1>/TLOG"
                  TLOGNAME=TLOG
                  TUXCONFIG="<APPDIR1>/tuxconfig"
                  TYPE="<machine type1>"
                  ULOGPFX="<APPDIR1>/ULOG"
<SITE2's uname>   LMID=SITE2
                  TUXDIR="<TUXDIR2>"
                  APPDIR="<APPDIR2>"
                  ENVFILE="<APPDIR2>/ENVFILE"
                  TLOGDEVICE="<APPDIR2>/TLOG"
                  TLOGNAME=TLOG
                  TUXCONFIG="<APPDIR2>/tuxconfig"
                  TYPE="<machine type2>"
                  ULOGPFX="<APPDIR2>/ULOG"
#
*GROUPS
```

```
#
# Group for Authentication Servers
#
##AUTHGRP         LMID=SITE1   GRPNO=101

#
# Group for Application Queue (/Q) Servers
#
##QGRP1           LMID=SITE1   GRPNO=102
##                TMSNAME=TMS_QM  TMSCOUNT=2
##                OPENINFO="TUXEDO/QM:<APPDIR1>/qdevice:QSP_BANKAPP"

#
# Group for Application Manager's Servers
#
##MGRGRP1         LMID=SITE1   GRPNO=103

#
# Group for EventBroker Servers
#
##EVBGRP1         LMID=SITE1   GRPNO=104

DEFAULT:          TMSNAME=TMS_SQL   TMSCOUNT=2
BANKB1            LMID=SITE1   GRPNO=1
                  OPENINFO="TUXEDO/SQL:<APPDIR1>/bankdl1:bankdb:readwrite"
BANKB2            LMID=SITE2   GRPNO=2
                  OPENINFO="TUXEDO/SQL:<APPDIR2>/bankdl2:bankdb:readwrite"

*NETWORK
SITE1             NADDR="<network address of SITE1>"
                  BRIDGE="<device of provider1>"
                  NLSADDR="<network listener address of SITE1>"
SITE2             NADDR="<network address of SITE2>"
                  BRIDGE="<device of provider2>"
                  NLSADDR="<network listener address of SITE2>"
*SERVERS
#
# TUXEDO System /T server providing application specific authentication.
# Ref. AUTHSVR(5).
#
##AUTHSVR         SRVGRP=AUTHGRP  SRVID=1  RESTART=Y  GRACE=0  MAXGEN=2
##                CLOPT="-A"

#
# TUXEDO System /T Message Queue Manager. It is a server that enqueues and
# dequeues messages on behalf of programs calling tpenqueue(3) and
# tpdequeue(3) respectively. Ref. TMQUEUE(5).
#
##TMQUEUE         SRVGRP=QGRP1    SRVID=1  CONV=N  GRACE=0
##                CLOPT="-s QSP_BANKAPP:TMQUEUE"
```

```
#
# TUXEDO System /T Message Forwarding Server that forwards messages that have
# been stored using tpenqueue(3) for later processing. Ref. TMQFORWARD(5).
#
##TMQFORWARD     SRVGRP=QGRP1    SRVID=2  CONV=N  REPLYQ=N  GRACE=0
##               CLOPT="-- -e -n -d -q Q_OPENACCT_LOG"

#
# TUXEDO System /T User Event Broker that manages user events by notifying
# subscribers when those events are posted. Ref. TMUSREVT(5).
#
##TMUSREVT       SRVGRP=EVBGRP1  SRVID=1  GRACE=3600
##               ENVFILE="<APPDIR1>/TMUSREVT.ENV"
##               CLOPT="-e tmusrevt.out -o tmusrevt.out -A --
##                    -f <APPDIR1>/tmusrevt.dat"
##               SEQUENCE=11

#
# TUXEDO Application Server that subscribes to certain events.
#
##ACCTMGR SRVGRP=MGRGRP1         SRVID=1
##               CLOPT="-A -o ACCTMGR.LOG -- -w 1000.00"
##               SEQUENCE=12

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

TLR              SRVGRP=BANKB1   SRVID=1  RQADDR=tlr1
                 CLOPT="-A -- -T 100 -e 1000.00"
TLR              SRVGRP=BANKB1   SRVID=2  RQADDR=tlr1
                 CLOPT="-A -- -T 200 -e 1000.00"
TLR              SRVGRP=BANKB2   SRVID=3  RQADDR=tlr2
                 CLOPT="-A -- -T 600 -e 1000.00"
TLR              SRVGRP=BANKB2   SRVID=4  RQADDR=tlr2
                 CLOPT="-A -- -T 700 -e 1000.00"
XFER             SRVGRP=BANKB1   SRVID=5
XFER             SRVGRP=BANKB2   SRVID=6
ACCT             SRVGRP=BANKB1   SRVID=7
ACCT             SRVGRP=BANKB2   SRVID=8
BAL              SRVGRP=BANKB1   SRVID=9
BAL              SRVGRP=BANKB2   SRVID=10
BTADD            SRVGRP=BANKB1   SRVID=11
BTADD            SRVGRP=BANKB2   SRVID=12
AUDITC           SRVGRP=BANKB1   SRVID=13  CONV=Y  MIN=1 MAX=10 RQADDR="auditc
"
BALC             SRVGRP=BANKB1   SRVID=24
BALC             SRVGRP=BANKB2   SRVID=25
#
*SERVICES
DEFAULT:         LOAD=50  AUTOTRAN=Y  TRANTIME=30
WITHDRAWAL       PRIO=50  ROUTING=ACCOUNT_ID
```

```
DEPOSIT          PRIO=50  ROUTING=ACCOUNT_ID
TRANSFER         PRIO=50  ROUTING=ACCOUNT_ID
INQUIRY          PRIO=50  ROUTING=ACCOUNT_ID
CLOSE_ACCT       PRIO=40  ROUTING=ACCOUNT_ID
OPEN_ACCT        PRIO=40  ROUTING=BRANCH_ID
BR_ADD           PRIO=20  ROUTING=BRANCH_ID
TLR_ADD          PRIO=20  ROUTING=BRANCH_ID
ABAL             PRIO=30  ROUTING=b_id
TBAL             PRIO=30  ROUTING=b_id
ABAL_BID         PRIO=30  ROUTING=b_id
TBAL_BID         PRIO=30  ROUTING=b_id
ABALC_BID        PRIO=30  ROUTING=b_id
TBALC_BID        PRIO=30  ROUTING=b_id
#
*ROUTING
ACCOUNT_ID       FIELD=ACCOUNT_ID
                 BUFTYPE="FML"
                 RANGES="10000-59999:BANKB1,
                         60000-109999:BANKB2"
BRANCH_ID        FIELD=BRANCH_ID
                 BUFTYPE="FML"
                 RANGES="1-5:BANKB1,
                         6-10:BANKB2"
b_id             FIELD=b_id
                 BUFTYPE="VIEW:aud"
                 RANGES="1-5:BANKB1,
                         6-10:BANKB2"
```

---

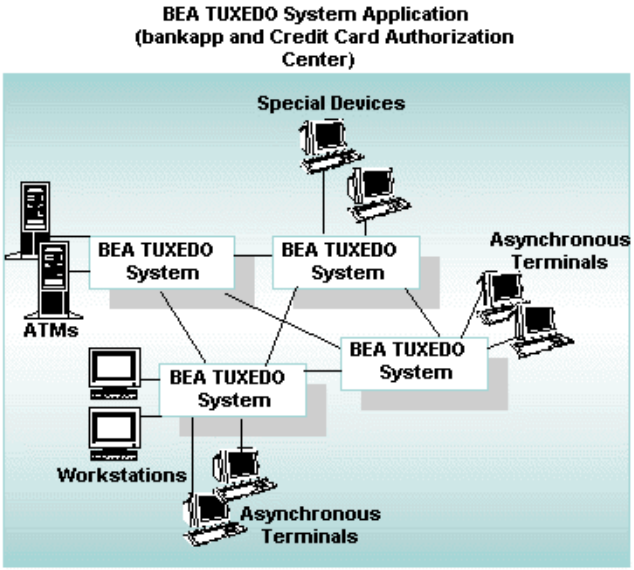
The following sections demonstrate two different ways of reconfiguring the `bankapp` application and the credit card authorization application so that they can communicate directly with one another:

- "Option 1: Reconfigure the Applications as a Single BEA Tuxedo Domain" on page 2-6
- "Option 2: Reconfigure the Applications as a Domains Configuration" on page 2-12

# Option 1: Reconfigure the Applications as a Single BEA Tuxedo Domain

One solution is to combine the `bankapp` application and the credit card authorization application into one BEA Tuxedo application, or domain, as shown in the following figure.

**Figure 2-2 Combining Two BEA Tuxedo System Applications**




## Creating the UBBCONFIG File for the Combined Application

To create the UBBCONFIG file for the combined application, take the following information from the UBBCONFIG file for the credit card authorization application and add it to the UBBCONFIG file for the bankapp application:

- Add machine, network, and group entries for the credit card authorization application to the UBBCONFIG file.

- Add Server entries for the credit card authorization application to the UBBCONFIG file.
- Add Service entries for the credit card authorization to the UBBCONFIG file.

The following listing shows a possible UBBCONFIG file for the combined application.

**Listing 2-2 Sample UBBCONFIG File for the Combined Application**

```
*RESOURCES
IPCKEY          76666
UID             0000
GID             000
PERM            0660
MAXACCESSERS    40
MAXSERVERS      35
MAXSERVICES     75
MAXCONV         10
MAXGTT          100
MASTER          SITE1,SITE2
SCANUNIT        10
SANITYSCAN      5
BBLQUERY        50
BLOCKTIME       2
DBBLWAIT        6
OPTIONS         LAN,MIGRATE
MODEL           MP
LDBAL           Y
#
*MACHINES
#
# Machines for the bankapp part
mach1           LMID=SITE1
                TUXDIR="/home/rsmith/tuxroot"
                APPDIR="/home/rsmith/bankapp"
                ENVFILE="/home/rsmith/bankapp/ENVFILE"
                TLOGDEVICE="/home/rsmith/bankapp/TLOG"
                TLOGNAME=TLOG
                TUXCONFIG="/home/rsmith/bankapp/tuxconfig"
                TYPE="type1"
                ULOGPFX="/home/rsmith/bankapp/ULOG"
mach2           LMID=SITE2
                TUXDIR="/home/rsmith/tuxroot"
                APPDIR="/home/rsmith/bankapp"
                ENVFILE="/home/rsmith/bankapp/ENVFILE"
                TLOGDEVICE="/home/rsmith/bankapp/TLOG"
                TLOGNAME=TLOG
                TUXCONFIG="/home/rsmith/bankapp/tuxconfig"
```

```
                TYPE="type2"
                ULOGPFX="/home/rsmith/bankapp/ULOG"
mach3           LMID=SITE3
                TUXDIR="/home/rsmith/tuxroot"
                APPDIR="/home/rsmith/bankapp"
                ENVFILE="/home/rsmith/bankapp/ENVFILE"
                TLOGDEVICE="/home/rsmith/bankapp/TLOG"
                TLOGNAME=TLOG
                TUXCONFIG="/home/rsmith/bankapp/tuxconfig"
                TYPE="type2"
                ULOGPFX="/home/rsmith/bankapp/ULOG"
#
# Machine for the credit card authorization part
sfexpz          LMID=SITE4
                TUXDIR="/home/rsmith/tuxroot"
                APPDIR="/home/rsmith/bankapp"
                ENVFILE="/home/rsmith/bankapp/ENVFILE"
                TLOGDEVICE="/home/rsmith/bankapp/TLOG"
                TLOGNAME=TLOG
                TUXCONFIG="/home/rsmith/bankapp/tuxconfig"
                TYPE="type1"
                ULOGPFX="/home/rsmith/bankapp/ULOG"
#
*GROUPS
DEFAULT:  TMSNAME=TMS_SQL    TMSCOUNT=2
#
# Groups for the bankapp part
BANKB1    LMID=SITE1         GRPNO=1
          OPENINFO="TUXEDO/SQL:/home/rsmith/bankapp/bankdl1:bankdb:readwrite"
BANKB2    LMID=SITE2         GRPNO=2
          OPENINFO="TUXEDO/SQL:/home/rsmith/bankapp/bankdl2:bankdb:readwrite"
BANKB3    LMID=SITE3         GRPNO=3
          OPENINFO="TUXEDO/SQL:/home/rsmith/bankapp/bankdl3:bankdb:readwrite"
#
# Group for the credit card authorization part
CREDIT    LMID=SITE4         GRPNO=4
          OPENINFO="TUXEDO/SQL:/home/rsmith/bankapp/crdtdl1:bankdb:readwrite"
#
*NETWORK
#
# Network connections for the bankapp part
SITE1     NADDR="<network address of SITE1>"
          BRIDGE="<device of provider1>"
          NLSADDR="<network listener address of SITE1>"
SITE2     NADDR="<network address of SITE2>"
          BRIDGE="<device of provider2>"
          NLSADDR="<network listener address of SITE2>"
SITE3     NADDR="<network address of SITE3>"
          BRIDGE="<device of provider3>"
```

```
          NLSADDR="<network listener address of SITE3>"
#
# Network connections for the credit card authorization part
SITE4     NADDR="<network address of SITE4>"
          BRIDGE="<device of provider4>"
          NLSADDR="<network listener address of SITE4>"
#
*SERVERS
DEFAULT:  RESTART=Y       MAXGEN=5      REPLYQ=Y   CLOPT="-A"
#
# Servers for the bankapp part
TLR       SRVGRP=BANKB1   SRVID=1       RQADDR=tlr1
          CLOPT="-A -- -T 100 -e 1000.00"
TLR       SRVGRP=BANKB1   SRVID=2       RQADDR=tlr1
          CLOPT="-A -- -T 200 -e 1000.00"
TLR       SRVGRP=BANKB2   SRVID=3       RQADDR=tlr2
          CLOPT="-A -- -T 600 -e 1000.00"
TLR       SRVGRP=BANKB2   SRVID=4       RQADDR=tlr2
          CLOPT="-A -- -T 700 -e 1000.00"
TLR       SRVGRP=BANKB3   SRVID=5       RQADDR=tlr3
          CLOPT="-A -- -T 800 -e 1000.00"
TLR       SRVGRP=BANKB3   SRVID=6       RQADDR=tlr3
          CLOPT="-A -- -T 900" -e 1000.00
XFER      SRVGRP=BANKB1   SRVID=7
XFER      SRVGRP=BANKB2   SRVID=8
XFER      SRVGRP=BANKB3   SRVID=9
ACCT      SRVGRP=BANKB1   SRVID=10
ACCT      SRVGRP=BANKB2   SRVID=11
ACCT      SRVGRP=BANKB3   SRVID=12
BAL       SRVGRP=BANKB1   SRVID=13
BAL       SRVGRP=BANKB2   SRVID=14
BAL       SRVGRP=BANKB3   SRVID=15
BTADD     SRVGRP=BANKB1   SRVID=16
BTADD     SRVGRP=BANKB2   SRVID=17
BTADD     SRVGRP=BANKB3   SRVID=18
AUDITC    SRVGRP=BANKB1   SRVID=19      CONV=Y  MIN=1  MAX=10  RQADDR="auditc"
BALC      SRVGRP=BANKB1   SRVID=20
BALC      SRVGRP=BANKB2   SRVID=21
BALC      SRVGRP=BANKB3   SRVID=22
#
# Servers for the credit card authorization part
TLRA      SRVGRP=CREDIT   SRVID=26
          CLOPT="-A -- -T 300"
ACCTA     SRVGRP=CREDIT   SRVID=27
CRDT      SRVGRP=CREDIT   SRVID=35
#
*SERVICES
DEFAULT:         LOAD=50  AUTOTRAN=Y  TRANTIME=30
#
```

```
# Services for the bankapp part
WITHDRAWAL       PRIO=50  ROUTING=ACCOUNT_ID
DEPOSIT          PRIO=50  ROUTING=ACCOUNT_ID
TRANSFER         PRIO=50  ROUTING=ACCOUNT_ID
INQUIRY          PRIO=50  ROUTING=ACCOUNT_ID
CLOSE_ACCT       PRIO=40  ROUTING=ACCOUNT_ID
OPEN_ACCT        PRIO=40  ROUTING=BRANCH_ID
BR_ADD           PRIO=20  ROUTING=BRANCH_ID
TLR_ADD          PRIO=20  ROUTING=BRANCH_ID
ABAL             PRIO=30  ROUTING=b_id
TBAL             PRIO=30  ROUTING=b_id
ABAL_BID         PRIO=30  ROUTING=b_id
TBAL_BID         PRIO=30  ROUTING=b_id
ABALC_BID        PRIO=30  ROUTING=b_id
TBALC_BID        PRIO=30  ROUTING=b_id
#
# Services for the credit card authorization part
WITHDRAWALA      PRIO=50
INQUIRYA         PRIO=50
OPENCA           PRIO=40
CLOSECA          PRIO=40
DEPOSITA         PRIO=50
OPEN_ACCT2       PRIO=40
OPENC            PRIO=40
#
*ROUTING
ACCOUNT_ID       FIELD=ACCOUNT_ID
                 BUFTYPE="FML"
                 RANGES="10000-39999:BANKB1,
                         40000-69999:BANKB2,
                         70000-109999:BANKB3,
                         *:*"
BRANCH_ID        FIELD=BRANCH_ID
                 BUFTYPE="FML"
                 RANGES="1-5:BANKB1,
                         6-10:BANKB2,
                         11-15:BANKB3"
b_id             FIELD=b_id
                 BUFTYPE="VIEW:aud"
                 RANGES="1-5:BANKB1,
                         6-10:BANKB2,
                         11-15:BANKB3"
```

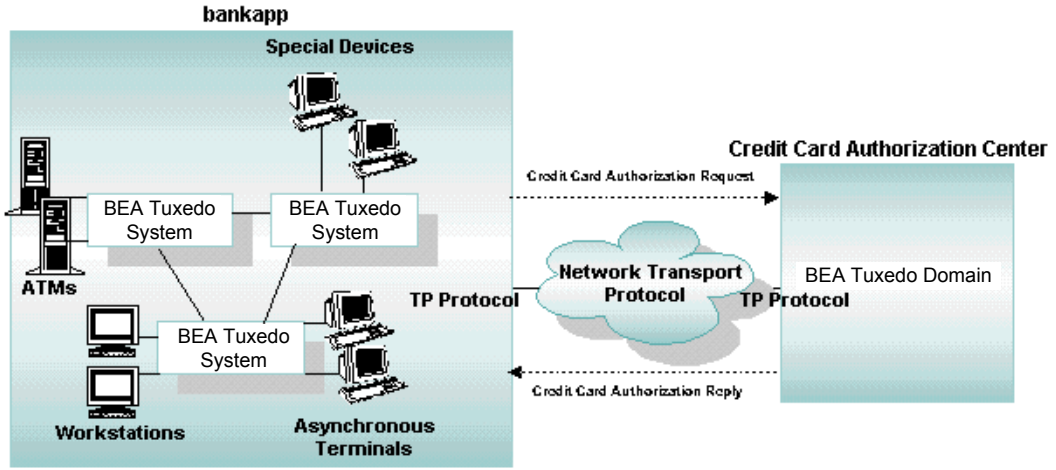
## Limitations of Option 1

- Administering a single large application can be more cumbersome than administering two smaller ones; each smaller one has its own UBBCONFIG file and hence its own administrative interface.
- Booting a networked application can be more costly because of the time required to boot each server and because of the need to propagate bulletin boards across the network. Smaller, separate applications can be booted simultaneously.

# Option 2: Reconfigure the Applications as a Domains Configuration

Another solution is to reconfigure the bankapp application and the credit card authorization application as a Domains configuration, as shown in the following figure. The two domains interoperate through two TDomain gateway server processes, one running in each domain.

**Figure 2-3 Domains Configuration**




To create the Domains configuration for the bankapp and credit card authorization applications, you need to create two UBBCONFIG files, one for each of the BEA Tuxedo applications, and two DMCONFIG files, one for each of the BEA Tuxedo applications.

## Creating the UBBCONFIG File for the bankapp Application in the Domains Environment

To create the UBBCONFIG file for the bankapp application in the Domains environment, start with a copy of the UBBCONFIG file shown in "Sample UBBCONFIG File for the Combined Application" on page 2-8 and make the following changes:

- In the MACHINES section, remove the machine entry for the credit card authorization application.
- In the NETWORK section, remove the network entry for the credit card authorization application.
- In the GROUPS section, do the following:
  - Remove the group entry for the credit card authorization application.
  - Add a group entry for the DMADM server and a different group entry for the GWADM and GWTDOMAIN servers.
- In the SERVERS section, do the following:
  - Remove the server entries for the credit card authorization application.
  - Add server entries for the DMADM, GWADM, and GWTDOMAIN servers.
- In the SERVICES section, remove the service entries for the credit card authorization application.

The following listing shows a possible UBBCONFIG file for the bankapp application in the Domains environment.

**Listing 2-3 Sample UBBCONFIG File for the bankapp Application in the Domains Environment**

```
*RESOURCES
IPCKEY          76666
UID             0000
GID             000
PERM            0660
MAXACCESSERS    40
MAXSERVERS      35
MAXSERVICES     75
MAXCONV         10
MAXGTT          100
MASTER          SITE1,SITE2
SCANUNIT        10
```

```
SANITYSCAN       5
BBLQUERY         50
BLOCKTIME        2
DBBLWAIT         6
OPTIONS          LAN,MIGRATE
MODEL            MP
LDBAL            Y
MAXBUFTYPE       16
#
*MACHINES
mach1            LMID=SITE1
                 TUXDIR="/home/rsmith/tuxroot"
                 APPDIR="/home/rsmith/bankapp"
                 ENVFILE="/home/rsmith/bankapp/ENVFILE"
                 TLOGDEVICE="/home/rsmith/bankapp/TLOG"
                 TLOGNAME=TLOG
                 TUXCONFIG="/home/rsmith/bankapp/tuxconfig"
                 TYPE="type1"
                 ULOGPFX="/home/rsmith/bankapp/ULOG"
mach2            LMID=SITE2
                 TUXDIR="/home/rsmith/tuxroot"
                 APPDIR="/home/rsmith/bankapp"
                 ENVFILE="/home/rsmith/bankapp/ENVFILE"
                 TLOGDEVICE="/home/rsmith/bankapp/TLOG"
                 TLOGNAME=TLOG
                 TUXCONFIG="/home/rsmith/bankapp/tuxconfig"
                 TYPE="type2"
                 ULOGPFX="/home/rsmith/bankapp/ULOG"
mach3            LMID=SITE3
                 TUXDIR="/home/rsmith/tuxroot"
                 APPDIR="/home/rsmith/bankapp"
                 ENVFILE="/home/rsmith/bankapp/ENVFILE"
                 TLOGDEVICE="/home/rsmith/bankapp/TLOG"
                 TLOGNAME=TLOG
                 TUXCONFIG="/home/rsmith/bankapp/tuxconfig"
                 TYPE="type2"
                 ULOGPFX="/home/rsmith/bankapp/ULOG"
#
*GROUPS
DEFAULT:  TMSNAME=TMS_SQL  TMSCOUNT=2
#
# Groups for bankapp
BANKB1    LMID=SITE1       GRPNO=1
          OPENINFO="TUXEDO/SQL:/home/rsmith/bankapp/bankdl1:bankdb:readwrite"
BANKB2    LMID=SITE2       GRPNO=2
          OPENINFO="TUXEDO/SQL:/home/rsmith/bankapp/bankdl2:bankdb:readwrite"
BANKB3    LMID=SITE3       GRPNO=3
          OPENINFO="TUXEDO/SQL:/home/rsmith/bankapp/bankdl3:bankdb:readwrite"
#
```

```
# Groups for Domains
DMADMGRP  LMID=SITE1       GRPNO=4
GWTGROUP  LMID=SITE2       GRPNO=5
#
*NETWORK
SITE1     NADDR="<network address of SITE1>"
          BRIDGE="<device of provider1>"
          NLSADDR="<network listener address of SITE1>"
SITE2     NADDR="<network address of SITE2>"
          BRIDGE="<device of provider2>"
          NLSADDR="<network listener address of SITE2>"
SITE3     NADDR="<network address of SITE3>"
          BRIDGE="<device of provider3>"
          NLSADDR="<network listener address of SITE3>"
#
*SERVERS
DEFAULT:  RESTART=Y       MAXGEN=5 REPLYQ=Y CLOPT="-A"
#
# Servers for Domains
DMADM     SRVGRP=DMADMGRP
          SRVID=1001
          REPLYQ=N
          RESTART=Y
          GRACE=0
GWADM     SRVGRP=GWTGROUP
          SRVID=1002
          REPLYQ=N
          RESTART=Y
          GRACE=0
GWTDOMAIN SRVGRP=GWTGROUP
          SRVID=1003
          RQADDR="GWTGROUP"
          REPLYQ=N
          RESTART=Y
          GRACE=0
#
# Servers for bankapp
TLR       SRVGRP=BANKB1   SRVID=1       RQADDR=tlr1
          CLOPT="-A -- -T 100 -e 1000.00"
TLR       SRVGRP=BANKB1   SRVID=2       RQADDR=tlr1
          CLOPT="-A -- -T 200 -e 1000.00"
TLR       SRVGRP=BANKB2   SRVID=3       RQADDR=tlr2
          CLOPT="-A -- -T 600 -e 1000.00"
TLR       SRVGRP=BANKB2   SRVID=4       RQADDR=tlr2
          CLOPT="-A -- -T 700 -e 1000.00"
TLR       SRVGRP=BANKB3   SRVID=5       RQADDR=tlr3
          CLOPT="-A -- -T 800 -e 1000.00"
TLR       SRVGRP=BANKB3   SRVID=6       RQADDR=tlr3
          CLOPT="-A -- -T 900" -e 1000.00
```

```
XFER      SRVGRP=BANKB1   SRVID=7
XFER      SRVGRP=BANKB2   SRVID=8
XFER      SRVGRP=BANKB3   SRVID=9
ACCT      SRVGRP=BANKB1   SRVID=10
ACCT      SRVGRP=BANKB2   SRVID=11
ACCT      SRVGRP=BANKB3   SRVID=12
BAL       SRVGRP=BANKB1   SRVID=13
BAL       SRVGRP=BANKB2   SRVID=14
BAL       SRVGRP=BANKB3   SRVID=15
BTADD     SRVGRP=BANKB1   SRVID=16
BTADD     SRVGRP=BANKB2   SRVID=17
BTADD     SRVGRP=BANKB3   SRVID=18
AUDITC    SRVGRP=BANKB1   SRVID=19      CONV=Y  MIN=1  MAX=10  RQADDR="auditc"
BALC      SRVGRP=BANKB1   SRVID=20
BALC      SRVGRP=BANKB2   SRVID=21
BALC      SRVGRP=BANKB3   SRVID=22
#
*SERVICES
DEFAULT:          LOAD=50  AUTOTRAN=Y  TRANTIME=30
WITHDRAWAL        PRIO=50  ROUTING=ACCOUNT_ID
DEPOSIT           PRIO=50  ROUTING=ACCOUNT_ID
TRANSFER          PRIO=50  ROUTING=ACCOUNT_ID
INQUIRY           PRIO=50  ROUTING=ACCOUNT_ID
CLOSE_ACCT        PRIO=40  ROUTING=ACCOUNT_ID
OPEN_ACCT         PRIO=40  ROUTING=BRANCH_ID
BR_ADD            PRIO=20  ROUTING=BRANCH_ID
TLR_ADD           PRIO=20  ROUTING=BRANCH_ID
ABAL              PRIO=30  ROUTING=b_id
TBAL              PRIO=30  ROUTING=b_id
ABAL_BID          PRIO=30  ROUTING=b_id
TBAL_BID          PRIO=30  ROUTING=b_id
ABALC_BID         PRIO=30  ROUTING=b_id
TBALC_BID         PRIO=30  ROUTING=b_id
#
*ROUTING
ACCOUNT_ID        FIELD=ACCOUNT_ID
                  BUFTYPE="FML"
                  RANGES="10000-39999:BANKB1,
                          40000-69999:BANKB2,
                          70000-109999:BANKB3,
                          *:*"
BRANCH_ID         FIELD=BRANCH_ID
                  BUFTYPE="FML"
                  RANGES="1-5:BANKB1,
                          6-10:BANKB2,
                          11-15:BANKB3"
b_id              FIELD=b_id
                  BUFTYPE="VIEW:aud"
                  RANGES="1-5:BANKB1,
```

```
                        6-10:BANKB2,
                        11-15:BANKB3"
```

**Note:** In the previous example, REPLYQ=N is specified for the DMADM, GWADM, and GWTDOMAIN servers. This setting is not required: you can, if you prefer, designate a reply queue for any of these servers by specifying REPLYQ=Y. When REPLYQ is set to N, however, performance may be improved.

## Creating a DMCONFIG File for the bankapp Application

You also need to create a DMCONFIG file for the bankapp application, an example of which is shown in the following listing. The binary version of the a DMCONFIG file (BDMCONFIG) must reside on the same machine as the DMADM server.

**Listing 2-4 Sample DMCONFIG File for the bankapp Application**

```
*DM_LOCAL
LOCAL1          GWGRP=GWTGROUP
                TYPE=TDOMAIN
                ACCESSPOINTID="BANK"
                BLOCKTIME=10
                CONNECTION_POLICY=ON_STARTUP
                DMTLOGDEV="/home/rsmith/bankapp/DMTLOG"
                AUDITLOG="/home/rsmith/bankapp/AUDITLOG"
#
*DM_REMOTE
REMOT1          TYPE=TDOMAIN
                ACCESSPOINTID="CREDIT.CARD"
#
# If the DM_EXPORT section is absent, as in this sample DMCONFIG
# file, all services advertised by the local domain are available
# to the remote domains. Thus, the following bankapp services are
# available to the credit card authorization application:
#
# WITHDRAWAL
# DEPOSIT
# TRANSFER
```

```
# INQUIRY
# CLOSE_ACCT
# OPEN_ACCT
# BR_ADD
# TLR_ADD
# ABAL
# TBAL
# ABAL_BID
# TBAL_BID
# ABALC_BID
# TBALC_BID
#
*DM_IMPORT
WITHDRAWALA  RACCESSPOINT=REMOT1
             LACCESSPOINT=LOCAL1
INQUIRYA     RACCESSPOINT=REMOT1
             LACCESSPOINT=LOCAL1
OPENCA       RACCESSPOINT=REMOT1
             LACCESSPOINT=LOCAL1
CLOSECA      RACCESSPOINT=REMOT1
             LACCESSPOINT=LOCAL1
DEPOSITA     RACCESSPOINT=REMOT1
             LACCESSPOINT=LOCAL1
OPEN_ACCT2   RACCESSPOINT=REMOT1
             LACCESSPOINT=LOCAL1
OPENC        RACCESSPOINT=REMOT1
             LACCESSPOINT=LOCAL1
#
*DM_TDOMAIN
LOCAL1       NWADDR="albany.acme.com:4051"
REMOT1       NWADDR="newyork.acme.com:65431"
```

## Creating the UBBCONFIG File for the Credit Card Authorization Application in the Domains Environment

To create the UBBCONFIG file for the credit card authorization application in the Domains environment, make the following changes to the UBBCONFIG file for the credit card authorization application:

- In the GROUPS section, add a group entry for the DMADM server and a different group entry for the GWADM and GWTDOMAIN servers.
- In the SERVERS section, add server entries for the DMADM, GWADM, and GWTDOMAIN servers.

The following listing shows a possible UBBCONFIG file for the credit card authorization application in the Domains environment.

**Listing 2-5 Sample UBBCONFIG File for the Credit Card Authorization Application in the Domains Environment**

```
*RESOURCES
IPCKEY          76666
UID             0000
GID             000
PERM            0660
MAXACCESSERS    40
MAXSERVERS      35
MAXSERVICES     75
MAXCONV         10
MAXGTT          100
MASTER          SITE1
SCANUNIT        10
MODEL           SHM
LDBAL           Y
#
*MACHINES
sfexpz          LMID=SITE1
                TUXDIR="/home/rsmith/tuxroot"
                APPDIR="/home/rsmith/creditapp"
                ENVFILE="/home/rsmith/creditapp/ENVFILE"
                TLOGDEVICE="/home/rsmith/creditapp/TLOG"
                TLOGNAME=TLOG
                TUXCONFIG="/home/rsmith/creditapp/tuxconfig"
                TYPE="type1"
                ULOGPFX="/home/rsmith/creditapp/ULOG"
#
*GROUPS
DEFAULT:  TMSNAME=TMS_SQL  TMSCOUNT=2
#
```

```
# Group for credit card authorization
CREDIT    LMID=SITE1      GRPNO=1
          OPENINFO="TUXEDO/SQL:/home/rsmith/creditapp/crdtdl1:bankdb:readwrite"
#
# Groups for Domains
DMADMGRP  LMID=SITE1      GRPNO=2
GWTGROUP  LMID=SITE1      GRPNO=3
#
*SERVERS
DEFAULT:  RESTART=Y        MAXGEN=5  REPLYQ=Y  CLOPT="-A"
#
# Servers for Domains
DMADM     SRVGRP=DMADMGRP
          SRVID=50
          REPLYQ=N
          RESTART=Y
          GRACE=0
GWADM     SRVGRP=GWTGROUP
          SRVID=60
          REPLYQ=N
          RESTART=Y
          GRACE=0
GWTDOMAIN SRVGRP=GWTGROUP
          SRVID=70
          RQADDR="GWTGROUP"
          REPLYQ=N
          RESTART=Y
          GRACE=0
#
# Servers for credit card authorization
TLRA      SRVGRP=CREDIT   SRVID=1
          CLOPT="-A -- -T 600"
ACCTA     SRVGRP=CREDIT   SRVID=2
CRDT      SRVGRP=CREDIT   SRVID=3
#
*SERVICES
DEFAULT:         LOAD=50  AUTOTRAN=Y  TRANTIME=30
# Services for credit card authorization
WITHDRAWALA      PRIO=50
INQUIRYA         PRIO=50
OPENCA           PRIO=40
CLOSECA          PRIO=40
DEPOSITA         PRIO=50
OPEN_ACCT2       PRIO=40
OPENC            PRIO=40
```

**Note:** In the previous example, REPLYQ=N is specified for the DMADM, GWADM, and GWTDOMAIN servers. This setting is not required: you can, if you prefer, designate a reply queue for any of these servers by specifying REPLYQ=Y. When REPLYQ is set to N, however, performance may be improved.

## Creating a DMCONFIG File for the Credit Card Authorization Application

You also need to create a DMCONFIG file for the credit card authorization application, an example of which is shown in the following listing.

**Listing 2-6 Sample DMCONFIG File for the Credit Card Authorization Application**

```
*DM_LOCAL
LOCAL1          GWGRP=GWTGROUP
                TYPE=TDOMAIN
                ACCESSPOINTID="CREDIT.CARD"
                BLOCKTIME=8
                DMTLOGDEV="/home/rsmith/creditapp/DMTLOG"
                AUDITLOG="/home/rsmith/creditapp/AUDITLOG"
#
*DM_REMOTE
REMOT1          TYPE=TDOMAIN
                ACCESSPOINTID="BANK"
#
# If the DM_EXPORT section is absent, as in this sample DMCONFIG
# file, all services advertised by the local domain are available
# to the remote domains. Thus, the following credit card
# authorization services are available to the bankapp application:
#
# WITHDRAWALA
# INQUIRYA
# OPENCA
# CLOSECA
# DEPOSITA
# OPEN_ACCT2
# OPENC
#
*DM_IMPORT
```

```
WITHDRAWAL    RACCESSPOINT=REMOT1
              LACCESSPOINT=LOCAL1
DEPOSIT       RACCESSPOINT=REMOT1
              LACCESSPOINT=LOCAL1
TRANSFER      RACCESSPOINT=REMOT1
              LACCESSPOINT=LOCAL1
INQUIRY       RACCESSPOINT=REMOT1
              LACCESSPOINT=LOCAL1
CLOSE_ACCT    RACCESSPOINT=REMOT1
              LACCESSPOINT=LOCAL1
OPEN_ACCT     RACCESSPOINT=REMOT1
              LACCESSPOINT=LOCAL1
BR_ADD        RACCESSPOINT=REMOT1
              LACCESSPOINT=LOCAL1
TLR_ADD       RACCESSPOINT=REMOT1
              LACCESSPOINT=LOCAL1
ABAL          RACCESSPOINT=REMOT1
              LACCESSPOINT=LOCAL1
TBAL          RACCESSPOINT=REMOT1
              LACCESSPOINT=LOCAL1
ABALC_BID     RACCESSPOINT=REMOT1
              LACCESSPOINT=LOCAL1
TBALC_BID     RACCESSPOINT=REMOT1
              LACCESSPOINT=LOCAL1
#
*DM_TDOMAIN
LOCAL1        NWADDR="newyork.acme.com:65431"
REMOT1        NWADDR="albany.acme.com:4051"
```

# Examining the creditapp Domains Configuration

The `creditapp` application is a sample Domains configuration that spans four machines. In effect, the `creditapp` application is yet another solution to reconfiguring the `bankapp` application and the credit card authorization application—as described in "Planning to Build Domains from Multiple BEA Tuxedo Applications" on page 2-1—so that the two applications can communicate directly with one another. In this solution, the `bankapp` and credit card

authorization applications are reconfigured as four BEA Tuxedo domains, one domain per machine, that interoperate using TDomain gateway server processes.

The `creditapp` application is included with the BEA Tuxedo distribution. Its files reside at the following location:

- *`tux_prod_dir`*`\samples\atmi\creditapp` (Windows)
- *`tux_prod_dir`*`/samples/atmi/creditapp` (UNIX)

Where *`tux_prod_dir`* represents the directory in which the BEA Tuxedo distribution is installed.

The Domains configuration for the `creditapp` application requires four `UBBCONFIG` files, one for each of the BEA Tuxedo domains, and four `DMCONFIG` files, one for each of the BEA Tuxedo domains. The four `UBBCONFIG` files are named `ubbdom1` through `ubbdom4`, and the four `DMCONFIG` files are named `domcon1` through `domcon4`. The files reside in the `creditapp` directory.

The following listing shows the content of the `ubbdom1` configuration file. Notice in the `SERVERS` section that this domain is configured for three TDomain gateway groups, to be used by this domain to communicate with the three other domains in the Domains configuration.

**Listing 2-7 ubbdom1 Configuration File for the creditapp Application**

```
.
.
.
*RESOURCES
IPCKEY          80952
UID             <user id from id(1)>
GID             <group id from id(1)>
PERM            0660
MAXACCESSERS    40
MAXSERVERS      35
MAXSERVICES     75
MAXCONV         10
MASTER          SITE1
MODEL           SHM
LDBAL           Y
MAXGTT          100
MAXBUFTYPE      16
SCANUNIT        10
SANITYSCAN      5
DBBLWAIT        6
```

```
BBLQUERY          50
BLOCKTIME         2
#
#
*MACHINES
<SITE1's uname>   LMID=SITE1
                  TUXDIR="<TUXDIR1>"
                  APPDIR="<APPDIR1>"
                  ENVFILE="<APPDIR1>/ENVFILE"
                  TLOGDEVICE="<APPDIR1>/TLOG"
                  TLOGNAME=TLOG
                  TUXCONFIG="<APPDIR1>/tuxconfig"
                  ULOGPFX="<APPDIR1>/ULOG"
                  TYPE="<machine type1>"

#
#
*GROUPS
DEFAULT:          LMID=SITE1
BANKB1            GRPNO=1     TMSNAME=TMS_SQLTMSCOUNT=2
                  OPENINFO="TUXEDO/SQL:<APPDIR1>/crdtdl1:bankdb:readwrite"
BANKB2            GRPNO=2
BANKB3            GRPNO=3
BANKB4            GRPNO=4
#
#
*SERVERS
#
DEFAULT:          RESTART=Y       MAXGEN=5   REPLYQ=Y  CLOPT="-A"
DMADM             SRVGRP=BANKB2   SRVID=32
GWADM             SRVGRP=BANKB2   SRVID=30
GWTDOMAIN         SRVGRP=BANKB2   SRVID=31
GWADM             SRVGRP=BANKB3   SRVID=24
GWTDOMAIN         SRVGRP=BANKB3   SRVID=25
GWADM             SRVGRP=BANKB4   SRVID=20
GWTDOMAIN         SRVGRP=BANKB4   SRVID=21
TLRA              SRVGRP=BANKB1   SRVID=2
                  CLOPT="-A -- -T 100"
BTADD             SRVGRP=BANKB1   SRVID=3
ACCTA             SRVGRP=BANKB1   SRVID=4
CRDT              SRVGRP=BANKB1   SRVID=5
CRDTA             SRVGRP=BANKB1   SRVID=6
#
*SERVICES
DEFAULT:          LOAD=50
INQUIRYA          PRIO=50
WITHDRAWALA       PRIO=50
OPEN_ACCT2        PRIO=40
OPENC             PRIO=40
```

```
OPENCA              PRIO=40
CLOSECA             PRIO=40
BR_ADD              PRIO=20
TLR_ADD             PRIO=20
```

The following listing shows the content of the `domcon1` Domains configuration file. Notice in the `DM_LOCAL` section (also known as the `DM_LOCAL_DOMAINS` section) that this domain is configured for three TDomain gateway groups, to be used by this domain to communicate with the three other domains in the Domains configuration. The `domcon1` content shown here has been updated with the improved Domains terminology described in "Terminology Improvements for DMCONFIG File" on page 1-22.

**Listing 2-8 domcon1 Domains Configuration File for the creditapp Application**

```
.
.
.
*DM_RESOURCES
#
VERSION=U22
#

#
#
*DM_LOCAL
#
QDOM1           GWGRP=BANKB2
                TYPE=TDOMAIN
                ACCESSPOINTID="QDOM1"
                BLOCKTIME=10
                MAXACCESSPOINT=89
                DMTLOGDEV="<APPDIR1>/DMTLOG"
                AUDITLOG="<APPDIR1>/AUDITLOG"
                DMTLOGNAME="DMTLOG_TDOM1"

QDOM2           GWGRP=BANKB3
                TYPE=TDOMAIN
                ACCESSPOINTID="QDOM2"
```

```
                BLOCKTIME=10
                MAXACCESSPOINT=89
                DMTLOGDEV="<APPDIR1>/DMTLOG"
                AUDITLOG="<APPDIR1>/AUDITLOG"
                DMTLOGNAME="DMTLOG_TDOM2"

QDOM3           GWGRP=BANKB4
                TYPE=TDOMAIN
                ACCESSPOINTID="QDOM3"
                BLOCKTIME=10
                MAXACCESSPOINT=89
                DMTLOGDEV="<APPDIR1>/DMTLOG"
                AUDITLOG="<APPDIR1>/AUDITLOG"
                DMTLOGNAME="DMTLOG_TDOM3"

#
#
*DM_REMOTE
#
TDOM1           TYPE=TDOMAIN
                ACCESSPOINTID="TDOM1"

TDOM2           TYPE=TDOMAIN
                ACCESSPOINTID="TDOM2"

TDOM3           TYPE=TDOMAIN
                ACCESSPOINTID="TDOM3"

#
#
*DM_TDOMAIN
#
TDOM1            NWADDR="<network address of SITE2>"
                 NWDEVICE="<device of provider2>

TDOM2            NWADDR="<network address of SITE3>"
                 NWDEVICE="<device of provider3>

TDOM3            NWADDR="<network address of SITE4>"
                 NWDEVICE="<device of provider4>

QDOM1            NWADDR="<network address of SITE1>"
                 NWDEVICE="<device of provider1>
```

```
QDOM2           NWADDR="<network address of SITE1A>"
                NWDEVICE="<device of provider1>
QDOM3           NWADDR="<network address of SITE1B>"
                NWDEVICE="<device of provider1>
#
#
*DM_EXPORT
#
WITHDRAWALA
INQUIRYA
OPENCA
CLOSECA
```

---

If you decide to run the `creditapp` application, start by reading the `README` file in the `creditapp` directory. The `README` file explains how to use a UNIX shell script named `RUNME.sh` to run the `creditapp` application. If you want to run the `creditapp` application on a Windows system, read the `README` file to learn the basic setup information and then execute the comparable tasks in the Windows environment. For details on using BEA Tuxedo on Windows, see, *Using BEA Tuxedo ATMI on Windows*.

# Setting Up a Domains Configuration

To configure a Domains environment, you as the Domains administrator must specify all the information that a BEA Tuxedo domain needs to know about the other domains—the remote domains—involved in the Domains configuration. This information includes local services exported to the remote domains, services imported from the remote domains, and addressing and security parameters for contacting the remote domains. This information is defined in the `UBBCONFIG` and `DMCONFIG` configuration files for each domain involved in the Domains configuration.

The Domains example described in the following sections is based on the `simpapp` application, which is a sample application included with the BEA Tuxedo distribution at the following location:

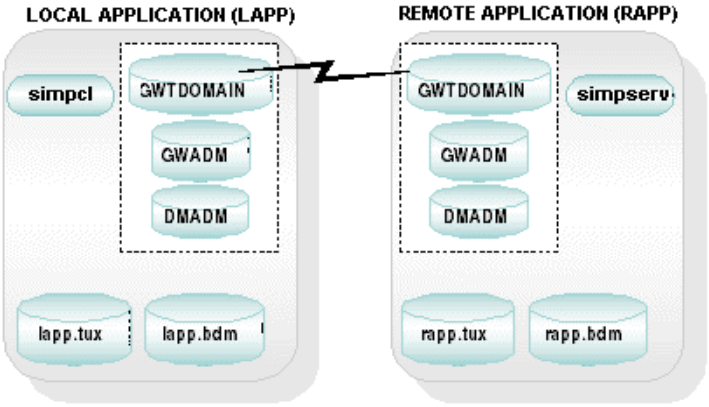
- *tux_prod_dir*`\samples\atmi\simpapp` (Windows)
- *tux_prod_dir*`/samples/atmi/simpapp` (UNIX)

Where *tux_prod_dir* represents the directory in which the BEA Tuxedo distribution is installed.

# Configuring a Sample Domains Application (simpapp)

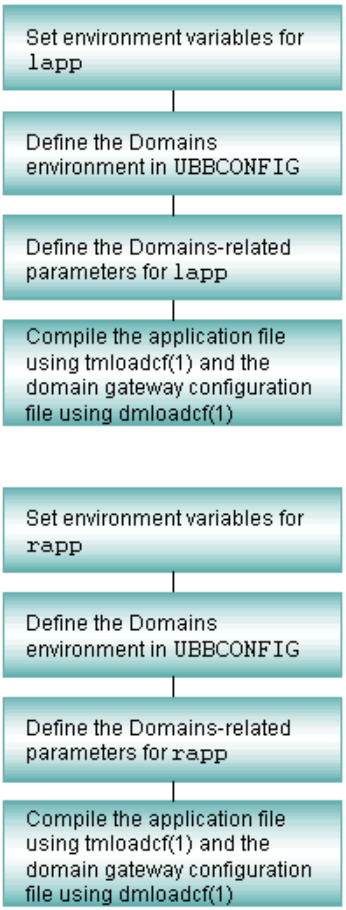
The Domains example, illustrated in the following figure, consists of two BEA Tuxedo domains: `lapp`, a *local application* based on `simpapp`, and `rapp`, a *remote application* based on `simpapp`. The `lapp` application is configured to allow its clients to access a service called `TOUPPER` that is available in the `rapp` application.

**Figure 2-4 Local and Remote Applications in simpapp**




# Configuration Tasks

The following tasks are required to configure the `lapp` and `rapp` applications.


Set environment variables for lapp
Define the Domains environment in UBBCONFIG
Define the Domains-related parameters for lapp
Compile the application file using tmloadcf(1) and the domain gateway configuration file using dmloadcf(1)
Set environment variables for rapp
Define the Domains environment in UBBCONFIG
Define the Domains-related parameters for rapp
Compile the application file using tmloadcf(1) and the domain gateway configuration file using dmloadcf(1)

# How to Set Environment Variables for lapp

You need to set the following environment variables for the `lapp` application to be configured successfully:

- `TUXDIR`—Absolute pathname to the BEA Tuxedo system root directory on this machine; sometimes represented as *`tux_prod_dir`*.
- `APPDIR`—Absolute pathname to the `lapp` application root directory on this machine.
- `TUXCONFIG`—Absolute pathname of the device or filename where the application binary configuration file for `lapp` is found on this machine.
- `BDMCONFIG`—Absolute pathname of the device or filename where the Domains binary configuration file for `lapp` is found on this machine.
- `PATH`—must include `%TUXDIR%\bin` (Windows) or `$TUXDIR/bin` (UNIX).
- `LD_LIBRARY_PATH` (UNIX only)—list of dynamically loadable libraries that must be loaded on this machine (must include `$TUXDIR/lib`); on HP-UX on the HP 9000, use `SHLIB_PATH` instead of `LD_LIBRARY_PATH`.

## Windows Example

```
prompt> set TUXDIR=C:\bea\tuxedo
prompt> set APPDIR=C:\home\lapp
prompt> set TUXCONFIG=C:\home\lapp\lapp.tux
prompt> set BDMCONFIG=C:\home\lapp\lapp.bdm
prompt> set PATH=%APPDIR%;%TUXDIR%\bin;%PATH%
```

**Note:** Windows accesses the required dynamically loadable library files through its `PATH` variable setting.

## UNIX Example

```
prompt> TUXDIR=/home/rsmith/bea/tuxedo
prompt> APPDIR=/home/rsmith/lapp
prompt> TUXCONFIG=/home/rsmith/lapp/lapp.tux
prompt> BDMCONFIG=/home/rsmith/lapp/lapp.bdm
prompt> PATH=$APPDIR:$TUXDIR/bin:/bin:$PATH
prompt> LD_LIBRARY_PATH=$APPDIR:$TUXDIR/lib:/lib:/usr/lib:
          $LD_LIBRARY_PATH
prompt> export TUXDIR APPDIR TUXCONFIG BDMCONFIG PATH LD_LIBRARY_PATH
```

# How to Define the Domains Environment for lapp in the UBBCONFIG File

In `lapp.ubb`, the text version of the `lapp` application configuration file, only the required parameters are defined. Default settings are used for the other parameters. The following listing shows the content of `lapp.ubb`.

**Listing 2-9 lapp.ubb Configuration File**

```
# lapp.ubb
#
*RESOURCES
IPCKEY          111111
MASTER          LAPP
MODEL           SHM

*MACHINES
giselle
                LMID=LAPP
                TUXDIR="/home/rsmith/tuxedo"
                APPDIR="/home/rsmith/lapp"
                TUXCONFIG="/home/rsmith/lapp/lapp.tux"
*GROUPS
LDMGRP          GRPNO=1 LMID=LAPP
LGWGRP          GRPNO=2 LMID=LAPP
.
.
.
*SERVERS
DMADM           SRVGRP=LDMGRP SRVID=1
GWADM           SRVGRP=LGWGRP SRVID=1
GWTDOMAIN       SRVGRP=LGWGRP SRVID=2 REPLYQ=N
.
.
.
*SERVICES
.
.
.
```

**Note:** In the previous `UBBCONFIG` file listing, `REPLYQ=N` is specified for the `DMADM`, `GWADM`, and `GWTDOMAIN` servers. This setting is not required: you can, if you prefer, designate a reply

queue for any of these servers by specifying REPLYQ=Y. When REPLYQ is set to N, however, performance may be improved.

### Server Group Definitions

The following server groups are defined in lapp.ubb:

- LDMGRP—contains the Domains administrative server (DMADM).
- LGWGRP—contains the gateway administrative server (GWADM) and the TDomain gateway server (GWTDOMAIN).

### Server Definitions

- DMADM—the Domains administrative server enables run-time modification of the Domains configuration information in the binary Domains configuration file (BDMCONFIG). DMADM supports a list of registered gateway groups. Only one instance of DMADM may be running in a BEA Tuxedo domain involved in a Domains configuration.
- GWADM—the gateway administrative server enables run-time administration of a particular domain gateway group. This server gets Domains configuration information from the DMADM server. It also provides administrative functionality and transaction logging for the gateway group.
- GWTDOMAIN—the TDomain gateway server enables access to and from remote BEA Tuxedo domains, allowing interoperability of two or more BEA Tuxedo domains. Information about the local and remote services that the TDomain gateway exports and imports is included in the Domains configuration file (DMCONFIG).

## How to Define Domains Parameters for lapp in the DMCONFIG File

In lapp.dom, the text version of the lapp Domains configuration file, only the required parameters are defined. Default settings are used for optional parameters. The following listing shows the content of the lapp.dom file.

**Listing 2-10 lapp.dom Domains Configuration File**

```
#
# lapp.dom
#
```

```
*DM_LOCAL
LAPP          GWGRP=LGWGRP
              TYPE=TDOMAIN
              ACCESSPOINTID="111111"

*DM_REMOTE
RAPP          TYPE=TDOMAIN
              ACCESSPOINTID="222222"

*DM_EXPORT

*DM_IMPORT
TOUPPER

*DM_TDOMAIN
LAPP          NWADDR="//giselle:5000"
RAPP          NWADDR="//juliet:5000"
```

## DM_LOCAL Section Definitions

The DM_LOCAL section identifies the local domain access points, their associated domain gateway groups, and their characteristics. There is one and only one local domain access point per domain gateway group.

The lapp.dom file specifies only one local domain access point, LAPP, and defines the following properties for the LAPP access point:

- GWGRP value is LGWGRP, the name of the domain gateway server group specified in the lapp.ubb file.
- TYPE of TDOMAIN indicates that the lapp application will be communicating with the rapp application through the local TDomain gateway server. This parameter indicates the protocol used by the gateways. Other TYPE values include IDOMAIN (BEA eLink Adapter for Mainframe gateway), SNAX (BEA eLink Adapter for Mainframe SNA gateway), and OSITP/OSITPX (BEA eLink Adapter for Mainframe OSI TP gateway).
- ACCESSPOINTID identifies the name of the local domain access point; this identifier must be unique across all domains involved in the Domains configuration.

## DM_REMOTE Section Definitions

The DM_REMOTE section identifies the remote domain access points and their characteristics. There may be one or more remote domain access points per domain gateway group.

The lapp.dom file specifies only one remote domain access point, RAPP, and defines the following properties for the RAPP access point:

- TYPE of TDOMAIN indicates that the lapp application will be communicating with the rapp application through the local TDomain gateway server.
- ACCESSPOINTID identifies the name of the remote domain access point; this identifier must be unique across all domains involved in the Domains configuration.

## DM_EXPORT Section Definitions

The DM_EXPORT section provides information about the services that are exported to one or more remote domains through a local domain access point. If this section is absent, or is present but empty, all services advertised by the local domain are available to the remote domains associated with the access points defined in the DM_REMOTE section.

As specified in the lapp.dom file, no lapp services are available to the rapp application through the LAPP access point.

## DM_IMPORT Section Definitions

The DM_IMPORT section provides information about the services that are imported through one or more remote domain access points and made available to the local domain through one or more local domain access points. If this section is absent, or is present but empty, no remote services are available to the local domain.

As specified in the lapp.dom file, the rapp service named TOUPPER is available to the lapp application.

## DM_TDOMAIN Section Definitions

The DM_TDOMAIN section defines the addressing information required by the BEA Tuxedo Domains component. Each domain access point specified in the LOCAL and REMOTE sections of the configuration file appears as an entry in the in the DM_TDOMAIN section.

Associated with each local domain access point entry is a NWADDR value, which specifies the network address at which the local domain will accept connections from one or more remote domains.

Associated with each remote domain access point entry is a NWADDR value, which specifies the network address at which the local domain will make a connection to a remote domain.

As specified in the lapp.dom file, the lapp application will listen for incoming connection requests on the network address giselle:5000, where giselle is the name of the machine on which the lapp application is running, and 5000 is the listening port. Also specified in lapp.dom is that when the lapp application attempts to make a connection to the rapp application, it will use the network address juliet:5000, where juliet is the name of the machine on which the rapp application is running, and 5000 is the destination port.

# How to Compile Application and Domains Gateway Configuration Files for lapp

The lapp.ubb application configuration file contains the information necessary to boot the lapp application. You compile this file into a binary data file by running tmloadcf(1).

The lapp.dom Domains configuration file contains the information used by the local lapp TDomain gateway to communicate with the remote rapp TDomain gateway. You compile this file into a binary data file by running dmloadcf(1).

To compile both configuration files, use the following sample session as a guide.

Windows:

```
prompt> cd C:\home\lapp
prompt> set TUXCONFIG=C:\home\lapp\lapp.tux
prompt> tmloadcf -y lapp.ubb
prompt> set BDMCONFIG=C:\home\lapp\lapp.bdm
prompt> dmloadcf -y lapp.dom
```

UNIX:

```
prompt> cd /home/rsmith/lapp
prompt> TUXCONFIG=/home/rsmith/lapp/lapp.tux
prompt> export TUXCONFIG
prompt> tmloadcf -y lapp.ubb
prompt> BDMCONFIG=/home/rsmith/lapp/lapp.bdm
prompt> export BDMCONFIG
prompt> dmloadcf -y lapp.dom
```

Once you build both the lapp and rapp applications, you boot the applications on their respective machines by executing the tmboot(1) command:

```
prompt> tmboot -y
```

The order in which the two applications are booted does not matter. Monitor the applications with dmadmin(1), as described in "Administering Domains" on page 4-1. Once both applications are booted, a client in the lapp application can call the TOUPPER service provided by the rapp application.

# How to Set Environment Variables for rapp

You need to set the following environment variables for the rapp application to be configured successfully:

- TUXDIR—Absolute pathname to the BEA Tuxedo system root directory on this machine; sometimes represented as *tux_prod_dir*.
- APPDIR—Absolute pathname to the rapp application root directory on this machine.
- TUXCONFIG—Absolute pathname of the device or filename where the application binary configuration file for rapp is found on this machine.
- BDMCONFIG—Absolute pathname of the device or filename where the Domains binary configuration file for rapp is found on this machine.
- PATH—must include %TUXDIR%\bin (Windows) or $TUXDIR/bin (UNIX).
- LD_LIBRARY_PATH (UNIX only)—list of dynamically loadable libraries that must be loaded on this machine (must include $TUXDIR/lib); on HP-UX on the HP 9000, use SHLIB_PATH instead of LD_LIBRARY_PATH.

## Windows Example

```
prompt> set TUXDIR=C:\bea\tuxedo
prompt> set APPDIR=C:\home\rapp
prompt> set TUXCONFIG=C:\home\rapp\rapp.tux
prompt> set BDMCONFIG=C:\home\rapp\rapp.bdm
prompt> set PATH=%APPDIR%;%TUXDIR%\bin;%PATH%
```

**Note:** Windows accesses the required dynamically loadable library files through its PATH variable setting.

## UNIX Example

```
prompt> TUXDIR=/home/rsmith/bea/tuxedo
prompt> APPDIR=/home/rsmith/rapp
prompt> TUXCONFIG=/home/rsmith/rapp/rapp.tux
```

```
prompt> BDMCONFIG=/home/rsmith/rapp/rapp.bdm
prompt> PATH=$APPDIR:$TUXDIR/bin:/bin:$PATH
prompt> LD_LIBRARY_PATH=$APPDIR:$TUXDIR/lib:/lib:/usr/lib:
          $LD_LIBRARY_PATH
prompt> export TUXDIR APPDIR TUXCONFIG BDMCONFIG PATH LD_LIBRARY_PATH
```

# How to Define the Domains Environment for rapp in the UBBCONFIG File

In `rapp.ubb`, the text version of the `rapp` application configuration file, only the required parameters are defined. Default settings are used for the other parameters. The following listing shows the content of the `rapp.ubb` file.

**Listing 2-11 rapp.ubb Application Configuration File**

```
# rapp.ubb
#
*RESOURCES
IPCKEY          222222
MASTER          RAPP
MODEL           SHM

*MACHINES
juliet
                LMID=RAPP
                TUXDIR="/home/rsmith/bea/tuxedo"
                APPDIR="/home/rsmith/rapp"
                TUXCONFIG="/home/rsmith/rapp/rapp.tux"

*GROUPS
RDMGRP          GRPNO=1 LMID=RAPP
RGWGRP          GRPNO=2 LMID=RAPP
APPGRP          GRPNO=3 LMID=RAPP
.
.
.
*SERVERS
DMADM           SRVGRP=RDMGRP SRVID=1
GWADM           SRVGRP=RGWGRP SRVID=1
GWTDOMAIN       SRVGRP=RGWGRP SRVID=2 REPLYQ=N
simpserv        SRVGRP=APPGRP SRVID=1
.
.
.
*SERVICES
```

```
TOUPPER
.
.
.
```

**Note:** In the previous UBBCONFIG file listing, REPLYQ=N is specified for the DMADM, GWADM, and GWTDOMAIN servers. This setting is not required: you can, if you prefer, designate a reply queue for any of these servers by specifying REPLYQ=Y. When REPLYQ is set to N, however, performance may be improved.

The following server groups are defined in rapp.ubb:

- RDMGRP—contains the Domains server DMADM.
- RGWGRP—contains the Domains servers GWADM and GWTDOMAIN.
- APPGRP—contains the application server simpserv.

The simpserv server advertises the TOUPPER service, which converts strings from lowercase to uppercase characters.

# How to Define Domains Parameters for rapp in the DMCONFIG File

In rapp.dom, the text version of the rapp Domains configuration file, only the required parameters are defined. Default settings are used for the other parameters. The following listing shows the content of the rapp.dom file.

**Listing 2-12 rapp.dom Domains Configuration File**

```
# rapp.dom
#
*DM_LOCAL
RAPP        GWGRP=RGWGRP
            TYPE=TDOMAIN
            ACCESSPOINTID="222222"

*DM_REMOTE
LAPP        TYPE=TDOMAIN
            ACCESSPOINTID="111111"
```

```
*DM_EXPORT
TOUPPER

*DM_IMPORT

*DM_TDOMAIN
RAPP        NWADDR="//juliet:5000"
LAPP        NWADDR="//giselle:5000"
```

The `rapp.dom` Domains configuration file is similar to the `lapp.dom` Domains configuration file, except that the two files list different services to be exported and imported. Specifically, the `rapp.dom` file defines the following Domains configurations for the `rapp` application:

- Specifies a local domain access point named `RAPP`, and a remote domain access point named `LAPP`. Both access points are associated with the TDomain gateway server group named `RGWGRP`.
- Specifies that the `rapp` service named `TOUPPER` is available to the `lapp` application.
- Specifies that no `lapp` services are available to the `rapp` application.
- Specifies that the `rapp` application will listen for incoming connection requests on network address `juliet:5000`, where `juliet` is the name of the machine on which the `rapp` application is running, and `5000` is the listening port.
- Specifies that if the `rapp` application attempts to make a connection to the `lapp` application, it will use the network address `giselle:5000`, where `giselle` is the name of the machine on which the `lapp` application is running, and `5000` is the destination port.

# How to Compile Application and Domain Gateway Configuration Files for rapp

The `rapp.ubb` application configuration file contains the information necessary to boot the `rapp` application. You compile this file into a binary data file by running `tmloadcf(1)`.

The `rapp.dom` Domains configuration file contains the information used by the local `rapp` TDomain gateway to communicate with the remote `lapp` TDomain gateway. You compile this file into a binary data file by running `dmloadcf(1)`.

To compile both configuration files, use the following sample session as a guide.

Windows:

```
prompt> cd C:\home\rapp
prompt> set TUXCONFIG=C:\home\rapp\rapp.tux
prompt> tmloadcf -y rapp.ubb
prompt> set BDMCONFIG=C:\home\rapp\rapp.bdm
prompt> dmloadcf -y rapp.dom
```

UNIX:

```
prompt> cd /home/rsmith/rapp
prompt> TUXCONFIG=/home/rsmith/rapp/rapp.tux
prompt> export TUXCONFIG
prompt> tmloadcf -y rapp.ubb
prompt> BDMCONFIG=/home/rsmith/rapp/rapp.bdm
prompt> export BDMCONFIG
prompt> dmloadcf -y rapp.dom
```

Once you build both the `rapp` and `lapp` applications, you boot the applications on their respective machines by executing the `tmboot(1)` command:

```
prompt> tmboot -y
```

The order in which the two applications are booted does not matter. Monitor the applications with `dmadmin(1)`, as described in "Administering Domains" on page 4-1. Once both applications are booted, a client in the `lapp` application can call the `TOUPPER` service provided by the `rapp` application.

# See Also

- "Understanding the Domains Configuration File" on page 1-15
- "How to Compress Data Between Domains" on page 2-41
- "How to Route Service Requests to Remote Domains" on page 2-41
- `UBBCONFIG(5)` in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*
- `DMCONFIG(5)` in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*

## How to Compress Data Between Domains

Data sent between domains can be compressed for faster performance. To configure compression, set the CMPLIMIT parameter in the DM_TDOMAIN section of the DMCONFIG file. This parameter, which is only relevant to remote domain access points, specifies the compression threshold to be used when sending data to a remote domain. The minimum value is 0, and the maximum value is 2147483647. The default setting is 2147483647. Application buffers larger than the specified size will be compressed.

For more information about setting the CMPLIMIT parameter, see "Compressing Data Over a Network" in *Administering a BEA Tuxedo Application at Run Time*.

## How to Route Service Requests to Remote Domains

Data-dependent routing information used by domain gateways to send service requests to specific remote domains is provided in the DM_ROUTING section of the DMCONFIG file. The FML, FML32, VIEW, VIEW32, X_C_TYPE, X_COMMON, and XML typed buffers are supported.

To create a routing table for a domain involved in a Domains configuration, you specify the following information in the DM_ROUTING section of the DMCONFIG file:

- Buffer type for which the routing entry is valid
- Name of the routing entry and field
- Ranges and associated remote domain names of the routing field.

For an example of a Domains data-dependent routing configuration, see "Specifying Domains Data-Dependent Routing" on page 1-23. For a detailed description of Domains data-dependent routing, see the DM_ROUTING section on reference page DMCONFIG(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*.

# Setting Up Security in a Domains Configuration

The BEA Tuxedo ATMI environment provides the following basic security capabilities for Domains configurations:

- *Authentication*—Verifies the identities of the local domain and a remote domain when attempting to establish a connection to one another
- *Authorization*—Restricts remote client access to local services via access control lists (ACLs)

- *Link-level encryption*—Keeps interdomain communications private

The security capabilities available to Domains configurations and those available to individual BEA Tuxedo applications are relatively independent but compatible. For information about the security capabilities available to BEA Tuxedo applications, see *Using Security in ATMI Applications*.

# Domains Security Mechanisms

The BEA Tuxedo Domains component provides the following security mechanisms:

- *Domains authentication*—Supplies the means by which the local domain and a remote domain can mutually authenticate one another when attempting to connect to one another. You specify identities, or principal names, for the local domain and each remote domain via the CONNECTION_PRINCIPAL_NAME parameter in the DM_LOCAL and DM_REMOTE sections of the DMCONFIG file.

  In addition, the local domain and a remote domain can use any of three levels of password security when attempting to connect to one another. You configure the level of password security on a local domain basis by setting the SECURITY parameter in the DM_LOCAL section of the DMCONFIG file.

- *Domains local domain access*—Restricts local services to remote domains. If a service is not exported to remote domains, it is unavailable to them. You export a service by placing an entry for the service in the DM_EXPORT section of the DMCONFIG file.

- *Domains access control lists (ACLs)*—Restricts the availability of services in a local domain to only certain remote domains. You create ACL names in the DM_ACCESS_CONTROL section of the DMCONFIG file and apply the ACL names to services in the EXPORT section of the DMCONFIG file.

- *Domains ACL policy*—Controls the ACL policy for remote domains. You configure a local or global ACL policy for a remote domain via the ACL_POLICY parameter in the DM_REMOTE section of the DMCONFIG file.

- *Domains link-level encryption*—Ensures data privacy between communicating domain gateways. For TDomain gateways, you configure link-level encryption by setting the MINENCRYPTBITS and MAXENCRYPTBITS parameters in the DM_TDOMAIN section of the DMCONFIG file.

# How to Configure Principal Names for Domains Authentication

As described in "Establishing a Link Between Domains" on page 2-24 in *Using Security in ATMI Applications*, a local TDomain gateway needs an identity, or principal name, that both the local domain and a remote domain know about so that the remote domain can authenticate the local domain when the domains are attempting to connect to one another. Similarly, the remote TDomain gateway needs an identity, or principal name, that both the remote domain and the local domain know about so that the local domain can authenticate the remote domain when the domains are attempting to establish a connection to one another. In addition, the local TDomain gateway uses its assigned principal name to acquire a set of security credentials needed when setting up the connection.

The local TDomain gateway needs a second principle name to acquire a set of security credentials required to enforce the local access control list (ACL) policy described in "How to Configure ACL Policy for a Remote Domain" on page 2-53.

As the administrator, you use the following configuration parameters to specify the principal names for the TDomain gateways running in your Release 7.1 or later BEA Tuxedo applications:

- SEC_PRINCIPAL_NAME (string) in UBBCONFIG

  Specifies the security principal name identification string to be used for authentication purposes by an application running BEA Tuxedo 7.1 or later software. This parameter may contain a maximum of 511 characters (excluding the terminating NULL character). The principal name specified for this parameter becomes the identity of one or more system processes—including TDomain gateway (GWTDOMAIN) processes—running in this application.

  During application booting, each TDomain gateway process in the application calls the authentication plug-in to acquire security credentials for the security principal name specified in SEC_PRINCIPAL_NAME. A TDomain gateway acquires these credentials for the principal name specified in the SEC_PRINCIPAL_NAME parameter.

- CONNECTION_PRINCIPAL_NAME (string) in DM_LOCAL section of DMCONFIG

  Specifies the connection principal name identifier, which is the principal name for verifying the identity of the domain gateway associated with this local domain access point when establishing a connection to a remote domain. This parameter applies only to domain gateways of type TDOMAIN running BEA Tuxedo 7.1 or later software.

  The CONNECTION_PRINCIPAL_NAME parameter may contain a maximum of 511 characters (excluding the terminating NULL character). If this parameter is not specified, the connection principal name defaults to the ACCESSPOINTID string for this local domain access point.

  For default authentication plug-ins, if a value is assigned to the CONNECTION_PRINCIPAL_NAME parameter for this local domain access point, it must be the same as the value assigned to the ACCESSPOINTID parameter for this local domain access point. If these values do not match, the local TDomain gateway process will not boot, and the system will generate the following userlog(3c) message: ERROR: Unable to acquire credentials.

- CONNECTION_PRINCIPAL_NAME (string) in DM_REMOTE section of DMCONFIG

  Specifies the connection principal name identifier, which is the principal name for verifying the identity of this remote domain access point when establishing a connection to the local domain. This parameter applies only to domain gateways of type TDOMAIN running BEA Tuxedo 7.1 or later software.

  The CONNECTION_PRINCIPAL_NAME parameter may contain a maximum of 511 characters (excluding the terminating NULL character). If this parameter is not specified, the connection principal name defaults to the ACCESSPOINTID string for this remote domain access point.

For default authentication plug-ins, if a value is assigned to the CONNECTION_PRINCIPAL_NAME parameter for this remote domain access point, it must be the same as the value assigned to the ACCESSPOINTID parameter for this remote domain access point. If these values do not match, any attempt to set up a connection between the local TDomain gateway and the remote TDomain gateway will fail, and the system will generate the following userlog(3c) message: ERROR: Unable to initialize administration key for domain *domain_name*.

In the following example, the CONNECTION_PRINCIPAL_NAME identities in the DMCONFIG file are used when establishing a connection through the LOCAL1 access point and the REMOT1 access point.

```
*DM_LOCAL
LOCAL1    GWGRP=bankg1
          TYPE=TDOMAIN
          ACCESSPOINTID="BA.CENTRAL01"
          CONNECTION_PRINCIPAL_NAME="BA.CENTRAL01"

*DM_REMOTE
REMOT1    TYPE=TDOMAIN
          ACCESSPOINTID="BA.BANK01"
          CONNECTION_PRINCIPAL_NAME="BA.BANK01"
```

# How to Configure Domains Password Security

Domain gateways can be made to authenticate incoming connections requested by remote domains. Application administrators can define when security should be enforced for incoming connections from remote domains.

As the administrator, you can specify the level of security used by a particular local domain by setting the SECURITY parameter in the DM_LOCAL section of the DMCONFIG file. There are three levels of password security:

- *No security* (using the NONE option)—Incoming connections from remote domains are not authenticated.
- *Application password* (using the APP_PW option)—Incoming connections from remote domains are authenticated using the application password defined in the TUXCONFIG file. (The application password is *not* included in the UBBCONFIG file.) The BEA Tuxedo application password is administered with tmloadcf(1), which prompts for the password when the SECURITY option is enabled in the TUXCONFIG file. The password is

automatically propagated with the TUXCONFIG file to the other machines in the configuration. You can update the password dynamically using the tmadmin command.

- *Domains password* (using the DM_PW option*)*—Connections between the local and remote domains are authenticated using passwords defined in the DM_PASSWORDS section of the BDMCONFIG file. (The DM_PASSWORDS section is *not* included in the DMCONFIG file.) These passwords are added to the binary configuration file after dmloadcf has been run, using DM_MIB(5) or the passwd subcommand of the dmadmin(1) command. Each entry contains the password used by a remote domain to access the local domain, and the password required by the local domain to access a remote domain.

If in the TUXCONFIG file the SECURITY parameter is set to NONE or is not set, the Domains configuration can still require the TDomain gateways to enforce security at the DM_PW level. If the DM_PW option is selected, each remote domain must have a password defined in the DM_PASSWORDS section of the BDMCONFIG file. In other words, incoming connections without a password are rejected by the TDomain gateway.

## Using the DM_MIB(5) to Set Domains Passwords (DM_PW)

You can use the DM_MIB to set Domains passwords (DM_PW). The T_DM_PASSWORDS class in the DM_MIB represents configuration information for interdomain authentication through local and remote access points of type TDOMAIN. The T_DM_PASSWORDS class contains the following entries for each remote domain.

- TA_DMLACCESSPOINT—Name of the local domain access point to which the password applies.
- TA_DMRACCESSPOINT—Name of the remote domain access point to which the password applies.
- TA_DMLPWD—Local password used to authenticate connections between the local domain access point (identified by TA_DMLACCESSPOINT) and the remote domain access point (identified by TA_DMRACCESSPOINT).
- TA_DMRPWD—Remote password used to authenticate connections between the remote domain access point (identified by TA_DMRACCESSPOINT) and the local domain access point (identified by TA_DMLACCESSPOINT).

  **Note:** Passwords are stored securely in encrypted format.

For information about formatting MIB administrative requests and interpreting MIB administrative replies, see reference page DM_MIB(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*.

## Using the dmadmin Command to Set Domains Passwords (DM_PW)

You can also use the dmadmin command to set Domains passwords (DM_PW):

```
prompt> dmadmin
passwd [-r] local_domain_access_point_name
   remote_domain_access_point_name
```

The dmadmin command prompts you for new passwords for the specified local and remote domain access points. For more information about dmadmin(1), see "Administering Domains" on page 4-1.

## Examples of Coding Password Security Between Domains

The SECURITY parameter in the DM_LOCAL section of the DMCONFIG file specifies the security type of a local domain. If authentication is required, it is done every time a connection is established between the local domain and a remote domain. If the security types of the two domains are incompatible, or if the passwords do not match, the connection fails.

### Example 1: Setting Security to NONE

If SECURITY is set to NONE for a local domain, incoming connection attempts are not authenticated. Even with SECURITY set to NONE, a local domain can still connect to a remote domain that has SECURITY set to DM_PW, but before such a connection can be established, you must define the passwords on both sides by using DM_MIB(5) or the dmadmin passwd command.

**Listing 2-13 Setting Security to NONE for Both Application and Domains**

```
LOCAL1: SECURITY in UBBCONFIG set to NONE
        SECURITY in DMCONFIG set to NONE

REMOT1: SECURITY in UBBCONFIG set to NONE
        SECURITY in DMCONFIG set to DM_PW
```

In this example, LOCAL1 is not enforcing any security but REMOT1 is enforcing DM_PW security. On the initiator (LOCAL1) side, the pertinent attributes in UBBCONFIG and DMCONFIG are set as follows:

```
UBBCONFIG
 *RESOURCES
 SECURITY  NONE

DMCONFIG
 *DM_LOCAL
 LOCAL1    GWGRP=bankg1
           TYPE=TDOMAIN
           ACCESSPOINTID="BA.CENTRAL01"
           CONNECTION_PRINCIPAL_NAME="BA.CENTRAL01"
           SECURITY=NONE

*DM_REMOTE
 REMOT1    TYPE=TDOMAIN
           ACCESSPOINTID="BA.BANK01"
           CONNECTION_PRINCIPAL_NAME="BA.BANK01"
```

On the responder (REMOT1) side, the pertinent attributes in UBBCONFIG and DMCONFIG are set as follows:

```
UBBCONFIG
 *RESOURCES
 SECURITY  NONE

DMCONFIG
 *DM_LOCAL
 REMOT1    GWGRP=bankg2
           TYPE=TDOMAIN
           ACCESSPOINTID="BA.BANK01"
           CONNECTION_PRINCIPAL_NAME="BA.BANK01"
           SECURITY=DM_PW

*DM_REMOTE
 LOCAL1    TYPE=TDOMAIN
           ACCESSPOINTID="BA.CENTRAL01"
           CONNECTION_PRINCIPAL_NAME="BA.CENTRAL01"
```

After the required attributes have been set in the TUXCONFIG and BDMCONFIG files, boot the applications on LOCAL1 and REMOT1.

On LOCAL1:

```
           dmadmin
           passwd LOCAL1 REMOT1
```

```
Enter Local Domain Password:foo1
Reenter Local Domain Password:foo1
Enter Remote Domain Password:foo2
Reenter Remote Domain Password:foo2
```

On REMOT1:

```
dmadmin
passwd REMOT1 LOCAL1
Enter Local Domain Password:foo2
Reenter Local Domain Password:foo2
Enter Remote Domain Password:foo1
Reenter Remote Domain Password:foo1
```

Once passwords have been created on both domains, a connection can be established and services can be invoked on the remote domain.

### Listing 2-14 Setting Application Security to NONE and Domains Security to DM_PW

On the initiator (LOCAL1) side, the pertinent attributes in UBBCONFIG and DMCONFIG are set as follows:

```
UBBCONFIG
 *RESOURCES
 SECURITY  NONE

DMCONFIG
 *DM_LOCAL
 LOCAL1    GWGRP=bankg1
           TYPE=TDOMAIN
           ACCESSPOINTID="BA.CENTRAL01"
           CONNECTION_PRINCIPAL_NAME="BA.CENTRAL01"
           SECURITY=DM_PW

*DM_REMOTE
 REMOT1    TYPE=TDOMAIN
           ACCESSPOINTID="BA.BANK01"
           CONNECTION_PRINCIPAL_NAME="BA.BANK01"
```

On the responder (REMOT1) side, the pertinent attributes in UBBCONFIG and DMCONFIG are set as follows:

```
UBBCONFIG
 *RESOURCES
 SECURITY  NONE

DMCONFIG
 *DM_LOCAL
 REMOT1    GWGRP=bankg2
           TYPE=TDOMAIN
           ACCESSPOINTID="BA.BANK01"
           CONNECTION_PRINCIPAL_NAME="BA.BANK01"
           SECURITY=DM_PW

*DM_REMOTE
 LOCAL1    TYPE=TDOMAIN
           ACCESSPOINTID="BA.CENTRAL01"
           CONNECTION_PRINCIPAL_NAME="BA.CENTRAL01"
```

After the required attributes have been set in the TUXCONFIG and BDMCONFIG files, boot the applications on LOCAL1 and REMOT1:

On LOCAL1:

```
dmadmin
passwd LOCAL1 REMOT1
Enter Local Domain Password:foo1
Reenter Local Domain Password:foo1
Enter Remote Domain Password:foo2
Reenter Remote Domain Password:foo2
```

On REMOT1:

```
dmadmin
passwd REMOT1 LOCAL1
Enter Local Domain Password:foo2
Reenter Local Domain Password:foo2
Enter Remote Domain Password:foo1
Reenter Remote Domain Password:foo1
```

---

Once passwords have been created on both domains, a connection can be established and services can be invoked on the remote domain.

## Example 2: Setting Security to APP_PW

If the SECURITY parameter in the UBBCONFIG is set to APP_PW or higher, then SECURITY in the DMCONFIG can be set to NONE, APP_PW, or DM_PW. Because you can define multiple views of a domain in one DMCONFIG file (one view per local domain definition), you can assign a different type of security mechanism to each of those views.

**Note:** If SECURITY is set to APP_PW for a local domain access point in the DMCONFIG, then SECURITY in the UBBCONFIG must be set to APP_PW or higher.

**Listing 2-15 Setting Security to APP_PW for Both Application and Domains**

```
LOCAL1: SECURITY in UBBCONFIG set to APP_PW
        SECURITY in DMCONFIG set to APP_PW

REMOT1: SECURITY in UBBCONFIG set to APP_PW
        SECURITY in DMCONFIG set to APP_PW
```

In this example, both LOCAL1 and REMOT1 enforce APP_PW security.

On the initiator (LOCAL1) side, the pertinent attributes in UBBCONFIG and DMCONFIG are set as follows:

```
UBBCONFIG
 *RESOURCES
 SECURITY  APP_PW

DMCONFIG
 *DM_LOCAL
 LOCAL1    GWGRP=bankg1
           TYPE=TDOMAIN
           ACCESSPOINTID="BA.CENTRAL01"
           CONNECTION_PRINCIPAL_NAME="BA.CENTRAL01"
           SECURITY=APP_PW

*DM_REMOTE
 REMOT1    TYPE=TDOMAIN
           ACCESSPOINTID="BA.BANK01"
           CONNECTION_PRINCIPAL_NAME="BA.BANK01"
```

On the responder (REMOT1) side, the pertinent attributes in UBBCONFIG and DMCONFIG are set as follows.

```
UBBCONFIG
 *RESOURCES
 SECURITY  APP_PW

DMCONFIG
 *DM_LOCAL
 REMOT1    GWGRP=bankg2
           TYPE=TDOMAIN
           ACCESSPOINTID="BA.BANK01"
           CONNECTION_PRINCIPAL_NAME="BA.BANK01"
           SECURITY=APP_PW

*DM_REMOTE
 LOCAL1    TYPE=TDOMAIN
           ACCESSPOINTID="BA.CENTRAL01"
           CONNECTION_PRINCIPAL_NAME="BA.CENTRAL01"
```

After the TUXCONFIG and BDMCONFIG files have been created, boot the applications on LOCAL1 and REMOT1.

# How to Configure Domains Access Control Lists

To set up a Domains access control list (ACL) in the DM_ACCESS_CONTROL section of the DMCONFIG file, you specify the name of the ACL and the remote domain access points associated with the ACL name. The following table clarifies the procedure.

| Domain ACL Field | Description |
|---|---|
| Domains ACL name | The name of this ACL.<br>A valid name consists of a string of 1-30 characters, inclusive. It must be printable and it may not include a colon, a pound sign, or a newline character.<br>Example: ACLGRP1 |
| Remote Domains list | The list of remote domains that are granted access in this access control list.<br>A valid value in this field is a set of one or more comma-separated remote domain names.<br>Examples: REMDOM1 , REMDOM2 , REMDOM3 |

Upon creating an ACL, you use the ACL parameter in the DM_EXPORT section of the DMCONFIG file to restrict access to a local service exported through a particular local domain access point to just those remote domain access points associated with the ACL name (for example, ACL=ACLGRP1).

# How to Configure ACL Policy for a Remote Domain

As the administrator, you use the following configuration parameters to set and control the ACL policy for remote domains running BEA Tuxedo release 7.1 or later software. You set these parameters in the DM_REMOTE section of the DMCONFIG file.

- ACL_POLICY (LOCAL | GLOBAL)

  Specifies the access control list (ACL) policy for this remote domain access point. This parameter applies only to domain gateways of type TDOMAIN running BEA Tuxedo 7.1 or later software and domain gateways of type OSITPX running BEA Tuxedo 8.0 or later software.

  LOCAL means that the local domain *replaces* the credential (identity) of any service request received from the remote domain *with* the principal name specified in the LOCAL_PRINCIPAL_NAME parameter for this remote domain access point. GLOBAL means that the local domain does not replace the credential received with a remote service request; if no credential is received with a remote service request, the local domain forwards the service request to the local service *as is* (which usually fails). If not specified, the default is LOCAL.

- LOCAL_PRINCIPAL_NAME (string)

  The local principal name identifier (credential) assigned by the local domain to service requests received from the remote domain when the ACL_POLICY parameter for this remote domain access point is set (or defaulted) to LOCAL. This parameter applies only to domain gateways of type TDOMAIN running BEA Tuxedo 7.1 or later software and domain gateways of type OSITPX running BEA Tuxedo 8.0 or later software.

  The LOCAL_PRINCIPAL_NAME parameter may contain a maximum of 511 characters (excluding the terminating NULL character). If this parameter is not specified, the local principal name defaults to the ACCESSPOINTID string for this remote domain access point.

- CREDENTIAL_POLICY (LOCAL | GLOBAL)

  Specifies the credential policy for this remote domain access point. This parameter applies only to domain gateways of type TDOMAIN running BEA Tuxedo 8.0 or later software.

  LOCAL means that the local domain removes the credential (identity) from a local service request destined for this remote domain access point. GLOBAL means that the local domain does not remove the credential from a local service request destined for this remote domain access point. If not specified, the default is LOCAL.

  Note that the CREDENTIAL_POLICY parameter controls whether or not the local domain removes the credential from a local service request before sending the request to a remote domain. The ACL_POLICY parameter controls whether or not the local domain replaces the credential of a service request received from a remote domain with the principal name specified in the LOCAL_PRINCIPAL_NAME parameter.

In the following example, the connection for the REMOT1 access point is configured for global ACL in the DMCONFIG file, meaning that the domain gateway for the LOCAL1 access point passes client requests *from* the REMOT1 access point without change. For global ACL, the LOCAL_PRINCIPAL_NAME entry for the REMOT1 access point is ignored. Also, because CREDENTIAL_POLICY=GLOBAL, the domain gateway for the LOCAL1 access point does *not* remove the credential from any local service request destined for the REMOT1 access point.

```
*DM_LOCAL
LOCAL1    GWGRP=bankg1
          TYPE=TDOMAIN
          ACCESSPOINTID="BA.CENTRAL01"
          CONNECTION_PRINCIPAL_NAME="BA.CENTRAL01"
          SECURITY=DM_PW

*DM_REMOTE
REMOT1    TYPE=TDOMAIN
```

```
ACCESSPOINTID="BA.BANK01"
CONNECTION_PRINCIPAL_NAME="BA.BANK01"
ACL_POLICY=GLOBAL
CREDENTIAL_POLICY=GLOBAL
LOCAL_PRINCIPAL_NAME="BA.BANK01.BOB"
```

## How to Configure Domains Link-Level Encryption

Domains link-level encryption (LLE) establishes data privacy for messages moving over the network links that connect the local domain gateway to the remote domain gateway. There are three levels of link-level encryption security: 0-bit (no encryption), 56-bit (International), and 128-bit (United States and Canada).

To set up Domains link-level encryption on domain gateway links, follow these steps.

1. Open the DMCONFIG file with a text editor and add the following lines to the DM_TDOMAIN section.

```
*DM_TDOMAIN
LOCAL1    NWADDR="newyork.acme.com:65431"
          MINENCRYPTBITS=min
          MAXENCRYPTBITS=max

REMOT1    NWADDR="albany.acme.com:4051"
          MINENCRYPTBITS=min
          MAXENCRYPTBITS=max
```

2. Load the configuration by running dmloadcf(1). The dmloadcf command parses DMCONFIG and loads the binary BDMCONFIG file to the location referenced by the BDMCONFIG variable.

In the preceding example, when tmboot(1) starts the application, each domain gateway reads the BDMCONFIG file to access various parameters, including MINENCRYPTBITS and MAXENCRYPTBITS, and propagates those parameters to its local and remote domains. When the local domain is establishing a network link with a remote domain, the two domains negotiate the key size until they agree on the largest key size supported by both.

# Setting Up Connections in a Domains Configuration

You can specify the conditions under which a local domain gateway tries to establish a connection to a remote domain. To specify these conditions, assign a value to the CONNECTION_POLICY parameter in the DM_LOCAL section of the DMCONFIG file. You can select any of the following connection policies:

- Connect when a local client program requests a remote service (ON_DEMAND)
- Connect at boot time (ON_STARTUP)
- Accept incoming connections but do not initiate a connection automatically (INCOMING_ONLY)

For BEA Tuxedo release 8.1 or later, you can also define the connection policy on a per remote domain basis in the DM_TDOMAIN section of the DMCONFIG file. For details, see "How To Configure Your Connection Policy" on page 1-27.

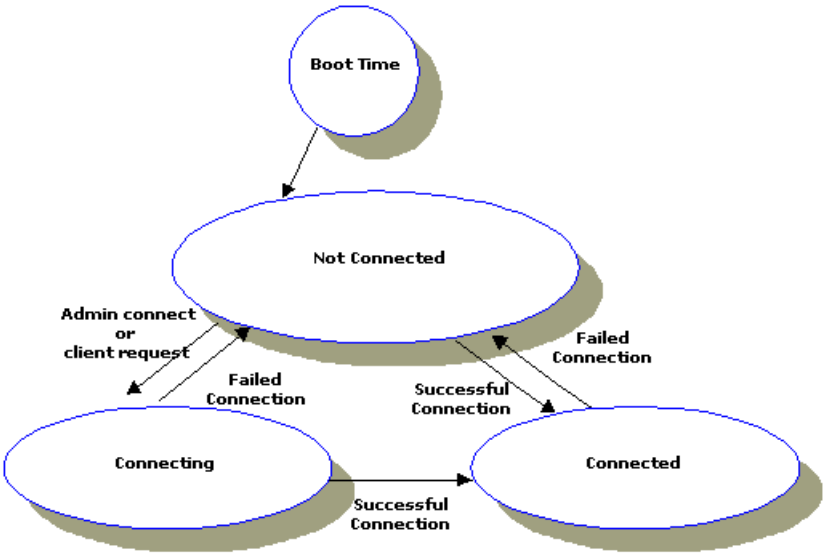
For connection policies of ON_STARTUP and INCOMING_ONLY, Dynamic Status is invoked. Dynamic Status, described in "How Connection Policy Determines Availability of Remote Services" on page 1-38, is a BEA Tuxedo Domains capability that checks and reports the status of remote services.

## How to Request Connections for Client Demands (ON_DEMAND Policy)

A connection policy of ON_DEMAND (CONNECTION_POLICY=ON_DEMAND) means that a connection is attempted only when either a local client requests a remote service or an administrative `dmadmin connect` command is run. ON_DEMAND is the default connection policy setting.

The following diagram shows how connections are attempted and made by a domain gateway for which the connection policy is ON_DEMAND.

**Figure 2-5 Connections Made with an ON_DEMAND Policy**

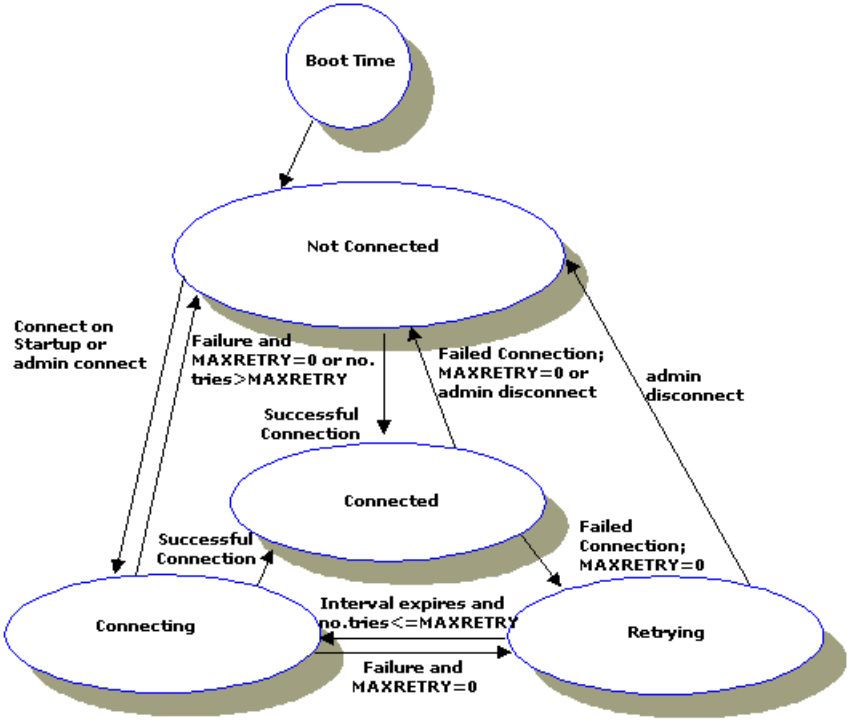



# How to Request Connections at Boot Time (ON_STARTUP Policy)

A connection policy of `ON_STARTUP` (`CONNECTION_POLICY=ON_STARTUP`) means that a domain gateway attempts to establish a connection with its remote domains when the domain gateway server is initialized. By default, the `ON_STARTUP` connection policy retries failed connections every 60 seconds, but you can specify a different value for this interval, as explained in "How to Configure the Connection Retry Interval for ON_STARTUP Only" on page 2-59.

The following diagram shows how connections are attempted and made by a domain gateway for which the connection policy is `ON_STARTUP`.

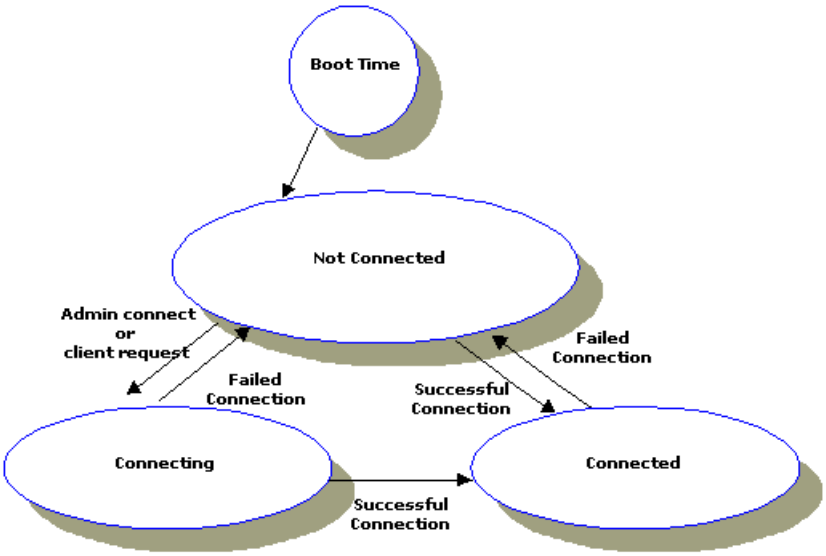
**Figure 2-6 Connections Made with an ON_STARTUP Policy**




# How to Limit Connections to Incoming Messages Only (INCOMING_ONLY Policy)

A connection policy of `INCOMING_ONLY` (`CONNECTION_POLICY=INCOMING_ONLY`) means that a domain gateway does not try to establish a connection to remote domains upon starting. The following diagram shows how connections are attempted and made by a domain gateway for which the connection policy is `INCOMING_ONLY`.

**Figure 2-7 Connections Made with an INCOMING_ONLY Policy (Accept Incoming Connections)**




# How to Configure the Connection Retry Interval for ON_STARTUP Only

When the CONNECTION_POLICY parameter is set to ON_STARTUP, automatic connection retry processing is available. Connection retry processing enables a domain gateway to retry, automatically, a failed attempt to connect to a remote domain. As the administrator, you can control the frequency of automatic connection attempts. To do so, specify the length (in seconds) of the interval during which the gateway should wait before trying, again, to establish a connection. You can specify the retry interval by setting the RETRY_INTERVAL parameter as follows:

RETRY_INTERVAL=*number_of_seconds*

You can specify between 0 and 2147483647 seconds. If the connection policy is ON_STARTUP and you do not specify a value for the RETRY_INTERVAL parameter, a default of 60 seconds is used.

The RETRY_INTERVAL parameter is valid only when the connection policy is ON_STARTUP. For the other connection policies (ON_DEMAND and INCOMING_ONLY), connection retry processing is not available.

## How to Configure the Maximum Retry Number

You indicate the number of times that a domain gateway tries to establish connections to remote domains before quitting by assigning a value to the MAXRETRY parameter: the minimum value is 0; the default and maximum value is the value of the MAXLONG parameter (2147483647).

- If you set MAXRETRY=0, connection retry processing is turned off. The local domain gateway does not attempt to connect to the remote domain gateway(s) automatically.
- If you set MAXRETRY=*number*, the gateway tries to establish a connection the specified number of times before quitting.
- If you set MAXRETRY=MAXLONG, the default setting, connection retry processing is repeated up to 2147483647 times or until a connection is established.

The MAXRETRY parameter is valid only when the connection policy is ON_STARTUP. For the other connection policies (ON_DEMAND and INCOMING_ONLY), connection retry processing is not available.

The following table presents examples of how MAXRETRY and RETRY_INTERVAL affect automatic connection retry processing.

**Table 2-1 Example Settings of the MAXRETRY and RETRY_INTERVAL Parameters**

| If You Set... | Then... |
|---|---|
| CONNECTION_POLICY=ON_STARTUP<br>MAXRETRY=3<br>RETRY_INTERVAL=30 | The local domain gateway makes three attempts to establish a connection, at 30 seconds intervals, before quitting. |
| CONNECTION_POLICY=ON_STARTUP<br>MAXRETRY=0 | The local domain gateway attempts to establish a connection at initialization time but does not retry if the first attempt fails. |
| CONNECTION_POLICY=ON_STARTUP<br>RETRY_INTERVAL=30 | The domain gateway attempts to establish a connection every 30 seconds until a connection is established. |

# Example of Coding Connection Policies Between Domains

Because domains involved in a Domains configuration work independently of one another, any combination of connection policies is allowed in a Domains configuration. However, not every connection policy combination is practical. In most cases, for example, configuring each of two interoperating domains with a connection policy of ON_STARTUP does not make much sense.

The following configuration example is a practical connection policy combination. In this example, LOCAL1 is configured for ON_STARTUP in the local DMCONFIG file, and REMOT1 is configured for INCOMING_ONLY in the remote DMCONFIG file.

In local DMCONFIG file:

```
*DM_LOCAL
LOCAL1      GWGRP=bankg1
            TYPE=TDOMAIN
            CONNECTION_POLICY=ON_STARTUP
            MAXRETRY=5
            RETRY_INTERVAL=100

*DM_REMOTE
REMOT1      TYPE=TDOMAIN
            ACCESSPOINTID="BA.BANK01"
```

In remote DMCONFIG file:

```
*DM_LOCAL
REMOT1      GWGRP=bankg2
            TYPE=TDOMAIN
```

```
            ACCESSPOINTID="BA.BANK01"
            CONNECTION_POLICY=INCOMING_ONLY

*DM_REMOTE
LOCAL1      TYPE=TDOMAIN
            ACCESSPOINTID="BA.CENTRAL01"
            CONNECTION_PRINCIPAL_NAME="BA.CENTRAL01"
```

# Controlling Connections in a Domains Configuration

As the administrator, you can control the number of connections you want to establish between domains. You can also break the connections between local and remote domains.

## How to Establish Connections Between Domains

To establish a connection between a local domain gateway and a remote domain, run the `dmadmin` command with the `connect` (`co`) subcommand:

prompt> `dmadmin co -d` *`local_domain_access_point_name`*

By default, connections are established between the local domain you have specified and all remote domains configured for the local gateway. If you want to establish a connection to only one remote domain, specify that domain on the command line with the `-R` option:

prompt> `dmadmin co -d` *`local_domain_access_point_name`*
`-R` *`remote_domain_access_point_name`*

If a connection attempt fails and the connection policy is `ON_STARTUP` with connection retry processing turned on, repeated attempts to connect (via connection retry processing) are made.

## How to Break Connections Between Domains

To break a connection between a local gateway and a remote domain (making sure that the gateway does not try to re-establish the connection through automatic connection retry processing), run the `dmadmin` command with the `disconnect` (`dco`) subcommand:

prompt> `dmadmin dco -d` *`local_domain_access_point_name`*

By default, all remote domains configured for the local gateway are disconnected. If you want to end the connection to only one remote domain, specify that domain on the command line with the `-R` option:

prompt> `dmadmin dco -d` *`local_domain_access_point_name`*
`-R` *`remote_domain_access_point_name`*

Automatic connection retry processing is stopped by this command, regardless of whether there are any active connections when the command is run.

## How to Report on Connection Status

Using the `dmadmin printdomain` command, you can generate a report on connection status and the connections being retried. The `connect` command reports whether a connection attempt has succeeded. The `printdomain` command prints information about the specified local domain, including a list of remote domains, a list of remote domains to which it is connected, and a list of remote domains to which it is trying to establish connections.

The following example shows a `dmadmin` session in which the `printdomain` command is issued (in its abbreviated form, `pd`) for a local domain access point named `LOCAL1`.

```
prompt> dmadmin
dmadmin - Copyright ...
.
.
.
pd -d LOCAL1
Local domain :LOCAL1
  Connected domains:
  Domainid:  REMOT1
  Disconnected domains being retried:
  Domainid: REMOT2

dco -d LOCAL1 -R REMOT1
Operation completed successfully. Use printdomain(pd) to obtain results.

dco -d LOCAL1 -R REMOT2
Operation completed successfully. Use printdomain(pd) to obtain results.

co -d LOCAL1 -R REMOT1
Operation completed successfully. Use printdomain(pd) to obtain results.

pd -d LOCAL1
Local domain :LOCAL1
  Connected domains:
  Domainid: REMOT1
```

In this example, the remote domain access point names (REMOT1, REMOT2) and their DOMAINID—ACCESSPOINTID—names (REMOT1, REMOT2) are the same, as defined in the DM_REMOTE section of the DMCONFIG file, to keep the example simple.

# How to Initiate Domain Connection Events

Domain connection events are generated by default when configuration or connection status changes occur between two or more domains; however, you must subscribe to a domain connection event in order to display/output warning or error messages.

Tuxedo generates the following four domain connection events:

- .SysConnectionSuccess - Connection successfully established
- .SysConnectionConfig - Connection configuration has changed. The Connection configuration changed event may happen when the following configuration parameters change between two domains:
  - CONNECTION_POLICY
  - CMPLIMIT
  - MINENCRYPTBITS
  - MAXENCRYPTBITS
  - RETRY_INTERVAL
  - MAXRETRY
  - DMKEEPALIVE
  - DMKEEPALIVEWAIT
  - TCPKEEPALIVE

  When several parameters are changed in one operation (DMMIB or dmadmin), only one event is generated.
- .SysConnectionDropped - Connection has dropped. The .SysConnectionDropped event must also indicate the reason for the drop. There are three specific reasons why a connection drop can occur and each of them must be appended to the INFO message. They are:
  - ·LDOM issued disconnect
  - ·RDOM issued disconnect

  - ·Unknown connection loss

- `.SysConnectionFailed` - Connection is unsuccessful. The `.SysConnectionFailed` event also indicates the reason for failure. There can be several reasons for why a failure and all must be appended to the INFO message:
  - ·Socket Failure
  - ·Authentication Failure
  - ·Unconfigured RDOM

# Configuring Domains Link-Level Failover and Keepalive

Domains link-level failover is a mechanism that ensures that an alternate network link becomes active when a primary link fails. Domains keepalive is a mechanism that keeps interdomain connections open through firewalls during periods of inactivity and enables quick detection of connection failures. Domains keepalive is available in BEA Tuxedo release 8.1 or later.

For a description of Domains link-level failover, see "How to Configure Domains Link-Level Failover" on page 1-40. For a description of Domains keepalive, see "Specifying Domains Keepalive" on page 1-41.

CHAPTER 3

# Planning and Configuring CORBA Domains

The following sections explain how to plan and configure a domain for a BEA Tuxedo CORBA Domains environment:

- Overview of the CORBA Domains Environment
- Single-Domain Versus Multiple-Domain Communication
- Elements of a CORBA Domains Configuration
- Understanding and Using the Configuration Files
- Specifying Unique Factory Object Identifiers in the factory_finder.ini File
- Processing the factory_finder.ini File
- Types of CORBA Domains Configurations
- Examples of CORBA Domains Configurations

## Overview of the CORBA Domains Environment

A BEA Tuxedo Domains configuration is an extension of the core ATMI *domain* environment, as explained in "What Is the BEA Tuxedo Domains Component?" on page 1-1. A BEA Tuxedo domain, or business application, is a construct that is entirely administrative. There are no programming interfaces that refer to domains. Only an administrator is aware of domains.

In a BEA Tuxedo Domains configuration, an administrator can configure which services of a domain are available to other domains in the configuration. So, from a CORBA perspective, the

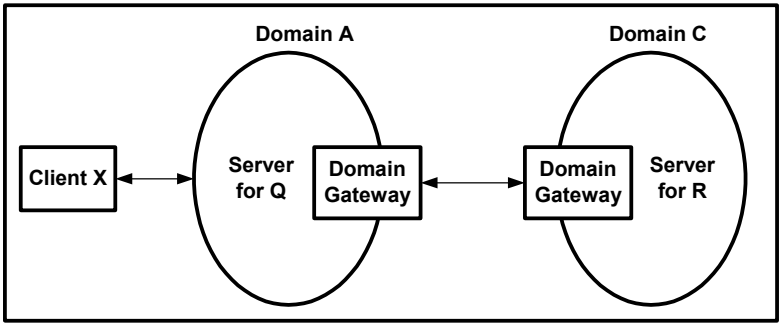
BEA Tuxedo Domains component is simply the means for BEA Tuxedo CORBA applications to interoperate with one another and share resources. The CORBA clients and the participating applications themselves do not need to know anything about the Domains configuration. All they need to know is what factory objects are available and how to access those objects.

This transparency between domains allows administrators to configure services in individual domains and to spread resources across multiple domains. If applications were to include information about domains, changing configurations would require that the applications be rewritten as well.

# Single-Domain Versus Multiple-Domain Communication

The following figure shows a simple Domains configuration consisting of two BEA Tuxedo CORBA applications.

**Figure 3-1 Domains Configuration Consisting of Two CORBA Applications**




The single-domain and multiple-domain discussions that follow are based on this simple Domains configuration.

## Single-Domain Communication

The following steps describe single-domain communication between CORBA Client X and Domain A in the simple Domains configuration:

1. Client X connects to Domain A using the Bootstrap object. The client application uses the Bootstrap object to locate a FactoryFinder and then uses the FactoryFinder to ask for a factory for objects of type Q. (The FactoryFinder call is itself an invocation on Domain A.)

2. When the FactoryFinder returns a factory, the client invokes that factory in Domain A.
3. The factory returns a reference to an object of type Q, called Q1.
4. The client then invokes on object Q1 in Domain A.

Throughout these steps, the client does not know where any of the objects are, or which domains they are in.

The administrative actions for connecting a client to Domain A are relatively simple for a client because the client is a simple machine and has very little infrastructure; it stands alone for the most part. Indeed, the connection to a BEA Tuxedo domain is the primary administration for a client. The actual administrative chore is setting the address of the ISL that is in Domain A.

## Multiple-Domain Communication

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

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

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

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

# Elements of a CORBA Domains Configuration

The following elements work together to accomplish a BEA Tuxedo Domains configuration for CORBA:

- BEA Tuxedo configuration file

  This text file, known as the UBBCONFIG file, names a domain and identifies the group and server entry for a domain gateway server. No attributes of domain gateways are specified in the UBBCONFIG file; all such attributes are in the *Domains configuration file* (explained next).

  Note that the BEA Tuxedo configuration file may have any name as long as the content of the file conforms to the format described on reference page UBBCONFIG(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*.

- Domains configuration file

  This text file, known as the DMCONFIG file, describes the remote domains that are connected to this domain—the local domain. One DMCONFIG file is required for each domain participating in a Domains configuration. If a domain is not connecting to another domain, the DMCONFIG file is not needed.

  Note that the Domains configuration file may have any name as long as the content of the file conforms to the format described on reference page DMCONFIG(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*.

- FactoryFinder Domains configuration file

  This text file, known as factory_finder.ini, specifies which factories can be searched for or found across domain boundaries. One factory_finder.ini file is required for each domain participating in a CORBA Domains configuration. If a domain is not connecting to another domain, the factory_finder.ini file is not needed.

  You must carefully coordinate the factory_finder.ini file with the DMCONFIG so that they both have information about the same connected domains and provide the same connectivity.

  Note that the FactoryFinder Domains configuration file may have any name as long as the content of the file conforms to the format described on reference page factory_finder.ini(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*.

- Invocation of an object in a remote domain

From a CORBA perspective, the whole point of the BEA Tuxedo Domains component is for an application in one CORBA domain to be able to make an invocation on an object in another CORBA domain, without either the client or server applications being aware that domains are a factor. Configuration information is intended to allow such invocations to cross domain boundaries and to hide those boundaries from applications.

- References to objects in a remote domain

  Any object reference may specify a local domain or a remote domain. A reference to a remote domain typically happens when a FactoryFinder returns a reference to a factory in a remote domain. It also happens when that factory, in turn, creates and returns a reference to an object in that remote domain (although, of course, the reference is local to the domain of the factory).

**Note:** Applications are not aware of the domain of an object reference. Applications cannot find out what domain an object reference refers to.

- FactoryFinders

  For a server in a local domain to obtain an object reference to an object in another domain, the application uses the same FactoryFinder pattern as it does for objects in the local domain. The application uses the same pattern because it is not aware that the FactoryFinder returns a reference to a factory in another domain. The configuration files hide this fact.

  Once an object reference has been obtained via a FactoryFinder or factory, the object reference can be passed anywhere; that is, passed to objects in the local domain, returned to a client, or passed to another domain.

# Understanding and Using the Configuration Files

You use the following three configuration files to set up a CORBA Domains configuration:

- `UBBCONFIG`, the BEA Tuxedo configuration file
- `DMCONFIG`, the Domains configuration file
- `factory_finder.ini`, the FactoryFinder Domains configuration file

Each domain in a CORBA Domains configuration requires a set of these three files. As the administrator, you must coordinate the configurations within each set of configuration files and between sets of configuration files. As the number of domains grows in a Domains configuration, your effort to coordinate the configurations also grows.

# The UBBCONFIG File

You must specify the following parameters in the UBBCONFIG file to configure multiple domains:

- Domain name
- Gateway group
- Gateway server

## Domain Name

Though not required for single BEA Tuxedo domains (that is, standalone domains), a domain that is connected to another domain must have a DOMAINID. You specify this parameter in the RESOURCES section of the UBBCONFIG file as follows:

```
DOMAINID domain_name
```

The *domain_name* must be 1 to13 characters long. For example:

```
DOMAINID headquarters
```

*domain_name* is the name that will be referenced in the DM_EXPORT and DM_IMPORT sections of the related DMCONFIG file. In that file, the *domain_name* will be referenced as:

```
"//domain_name"
```

The quotes are part of the reference. The slashes (//) mean that the name applies to BEA Tuxedo CORBA domains, rather than to BEA Tuxedo ATMI domains. For example:

```
"//headquarters"
```

Every domain in an enterprise must have a unique *domain_name*.

## Gateway Group and Server Names

As with every other BEA Tuxedo system server, there must be a group and a server name specified for a gateway. For example, the GROUPS section in the UBBCONFIG file might contain:

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

In this example, LGWGRP is a name chosen by a user (perhaps an abbreviation for "Local Gateway Group").

The server name for a BEA Tuxedo domain gateway—the TDomain gateway—is GWTDOMAIN and must be associated, like every other group, with a server group and a server ID. You specify

the GWTDOMAIN name in the SERVERS section associated with the server group name chosen. For example:

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

This entry tells the BEA Tuxedo CORBA application that a TDomain gateway is to be used and that additional information is found in the DMCONFIG file.

# The DMCONFIG File

Each BEA Tuxedo domain participating in a Domains configuration requires its own DMCONFIG file. A DMCONFIG file describes the relationship between the local domain (the domain in which the DMCONFIG file resides) and one or more remote domains (the domains with which the local domain will interoperate).

In most documentation for the DMCONFIG file, the focus is on the configuring of BEA Tuxedo ATMI domains to share *services*, a concept not applicable to BEA Tuxedo CORBA environments. For a BEA Tuxedo CORBA Domains environment, a "service" is simply the name of a BEA Tuxedo domain that can service BEA Tuxedo CORBA requests.

The following seven sections of the DMCONFIG file apply to a CORBA Domains environment:

- DM_LOCAL (also known as DM_LOCAL_DOMAINS)
- DM_REMOTE (also known as DM_REMOTE_DOMAINS)
- DM_EXPORT (also known as DM_LOCAL_SERVICES)
- DM_IMPORT (also known as DM_REMOTE_SERVICES)
- DM_RESOURCES
- DM_ACCESS_CONTROL
- DM_TDOMAIN

**Note:** The DM_LOCAL section must precede the DM_REMOTE section.

Many of the of the parameters in these seven sections are not relevant to configuring a CORBA Domains environment because they are ATMI-specific parameters.

The discussions that follow are based on the sample DMCONFIG file shown in the following listing.

**Listing 3-1 Sample DMCONFIG File for a BEA Tuxedo CORBA Domains Environment**

```
#
# BEA Tuxedo CORBA Domains Configuration File
#
*DM_RESOURCES
VERSION=Experimental8.9

*DM_LOCAL
LDOM   GWGRP=LGWGRP  TYPE=TDOMAIN  ACCESSPOINTID="MUTT"

*DM_REMOTE
TDOM1  TYPE=TDOMAIN  ACCESSPOINTID="JEFF"

*DM_EXPORT
"//MUTT"

*DM_IMPORT
"//JEFF"    RACCESSPOINT=TDOM1

*DM_TDOMAIN
LDOM   NWADDR="//sanfran.kmart.com:2507"
TDOM1  NWADDR="//sanhose.kmart.com:3186"
```

**Note:** The ACCESSPOINTID parameter in this listing may be replaced with the DOMAINID parameter, and the RACCESSPOINT parameter may be replaced with the RDOM parameter. This listing uses the improved DMCONFIG terminology.

## DM_RESOURCES

The DM_RESOURCES section specifies global Domains configuration information, specifically a user-supplied configuration version string. The only parameter in this section is VERSION=*string*, where *string* is a field in which users can enter a version number for the current DMCONFIG file. This field is not checked by the software.

In the sample DMCONFIG file, the VERSION parameter is set to Experimental8.9:

```
*DM_RESOURCES
VERSION=Experimental8.9
```

## DM_LOCAL

The DM_LOCAL section, also known as the DM_LOCAL_DOMAINS section, defines one or more *local domain access points* (logical names). For each local domain access point that you define, you specify a domain gateway group (TDOMAIN, ...) for the access point in this section, and—for the CORBA environment—you specify in the DM_EXPORT section the *domain_name* of the local BEA Tuxedo domain available through the access point. The local domain will be available through the local domain access point to CORBA clients in one or more remote BEA Tuxedo domains.

The DM_LOCAL section must have one *and only one* entry for each domain gateway group defined in the UBBCONFIG file. Each entry specifies the parameters required for the domain gateway processes running in that group.

Entries in the DM_LOCAL section have the form:

*LocalAccessPoint required_parameters* [*optional_parameters*]

where *LocalAccessPoint* is the local domain access point identifier (logical name) that you choose to represent a gateway group defined in the UBBCONFIG file. Note that the local domain access point is not the same name as the *domain_name* or the gateway group that is specified in the UBBCONFIG file. Rather, a local domain access point is a name used only within the DMCONFIG file to provide an extra level of insulation from potential changes in the UBBCONFIG file (changes in UBBCONFIG will affect only the defined parameters for the local domain access point, not the logical name of the local domain access point used throughout the DMCONFIG file).

The following parameters are required parameters:

GWGRP = *identifier*

This parameter specifies the name of a domain gateway server group (the name provided in the GROUPS section of the UBBCONFIG file) associated with this local domain access point.

TYPE = TDOMAIN

The TYPE parameter is required to specify the use of TDomain gateways for BEA Tuxedo CORBA environments.

ACCESSPOINTID = *string*

The ACCESSPOINTID parameter, also known as DOMAINID, is used to identify the gateway group associated with this local domain access point for purposes of security when setting up connections to remote domains. The gateway server group specified in the GWGRP parameter uses this string during any security checks. It has no required relationship to the *domain_name* found in the RESOURCES section of the UBBCONFIG file. ACCESSPOINTID

must be unique across both local and remote domains. The value of *string* can be a sequence of characters (for example, "BA.CENTRAL01"), or a sequence of hexadecimal digits preceded by 0x (for example, "0x0002FF98C0000B9D6"). ACCESSPOINTID must be 32 octets or fewer in length. If the value is a string, it must be 32 characters or fewer (counting the trailing NULL).

For example, the lines

```
*DM_LOCAL
LDOM  GWGRP=LGWGRP  TYPE=TDOMAIN  ACCESSPOINTID="MUTT"
```

identify LDOM as the local domain access point associated with the local TDomain gateway group having server group name LGWGRP (as specified in the UBBCONFIG file). If the domain gateway is ever involved in a domain-to-domain security check, it goes by the name MUTT.

**Note:** If the domain gateway is ever involved in a domain-to-domain security check *and* the CONNECTION_PRINCIPAL_NAME parameter is specified for the local domain access point, the gateway goes by the name specified in that parameter.

Optional parameters in the DM_LOCAL section describe resources and limits used in the operation of domain gateways. For a description of these parameters, see reference page DMCONFIG(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*.

## DM_REMOTE

The DM_REMOTE section, also known as the DM_REMOTE_DOMAINS section, defines one or more *remote domain access points* (logical names). For each remote domain access point that you define, you specify a domain gateway group (TDOMAIN, ...) for the access point in this section, and—for the CORBA environment—you specify in the DM_IMPORT section the *domain_name* of the remote BEA Tuxedo domain available through the access point. The remote domain will be available through the remote domain access point to CORBA clients in the local domain.

You can define multiple remote domain access points in this section, one or more for each domain gateway group used by this BEA Tuxedo domain to communicate with a remote domain.

Entries in the DM_REMOTE section have the form:

```
RemoteAccessPoint required_parameters
```

where *RemoteAccessPoint* is a remote domain access point identifier (logical name) that you choose for a particular remote domain to be accessed by a particular gateway group defined in the UBBCONFIG file. Note that a remote domain access point is not the same name as the *domain_name* or the gateway group that is specified in the local or remote domain's UBBCONFIG file. Rather, a remote domain access point is a name used only within the DMCONFIG to provide

an extra level of insulation from potential changes in UBBCONFIG (changes in UBBCONFIG will affect only the defined parameters for the remote domain access point, not the logical name of the remote domain access point used throughout the DMCONFIG file).

The required parameters are:

TYPE = TDOMAIN
: The TYPE parameter is required to specify the use of TDomain gateways for BEA Tuxedo CORBA environments.

ACCESSPOINTID = *string*
: The ACCESSPOINTID parameter, also known as DOMAINID, is used to identify the remote domain associated with this remote domain access point for purposes of security when setting up a connection to the remote domain. The gateway uses this string during any security checks. ACCESSPOINTID has no required relationship to the *domain_name* found in the RESOURCES section of the UBBCONFIG file. ACCESSPOINTID must be unique across both local and remote domains. The value of *string* can be a sequence of characters (for example, "BA.BANK01"), or a sequence of hexadecimal digits preceded by 0x (for example, "0x0002FF98C0000B9D6"). ACCESSPOINTID must be 32 octets or fewer in length. If the value is a string, it must be 32 characters or fewer (counting the trailing NULL).

For example, the lines

```
*DM_REMOTE
TDOM1   TYPE=TDOMAIN   ACCESSPOINTID="JEFF"
```

identify TDOM1 as a remote domain access point name associated with a local TDomain gateway group. If the domain gateway is ever involved in a domain-to-domain security check with a partner gateway, the gateway expects that partner to go by the name JEFF.

**Note:** If the domain gateway is ever involved in a domain-to-domain security check *and* the CONNECTION_PRINCIPAL_NAME parameter is specified for the remote domain access point, the gateway expects the partner to go by the name specified in that parameter.

## DM_EXPORT

The DM_EXPORT section, also known as the DM_LOCAL_SERVICES section, specifies in a CORBA environment the *domain_name* of the BEA Tuxedo domain to be exported through a local domain access point defined in the DM_LOCAL section. The BEA Tuxedo domain specified for a local domain access point is available to CORBA clients on one or more remote BEA Tuxedo domains. The DM_EXPORT section is required for a CORBA Domains configuration.

Entries in the DM_EXPORT section have the form:

```
service  [LACCESSPOINT=local access point name]
         [ACL=...]
```

where *service* is of the form:

```
"//domain_name"
```

This *domain_name* is the name assigned to the DOMAINID parameter in the RESOURCES section of the local UBBCONFIG file. Entering this name in the DM_EXPORT section means that the local domain accepts CORBA requests from remote domains. Also possible is to specify a *service* entry that accepts requests for a domain name other than the domain name of the local domain, in the case where the local domain acts as a pass-through for routing purposes.

The optional parameter, ACL, specifies the name of the access control list (ACL) to be used by the local domain to restrict requests made to the local domain by remote BEA Tuxedo CORBA domains. The name of the ACL is defined in the DM_ACCESS_CONTROL section of the DMCONFIG file. If this parameter is not specified, access control is not performed for remote requests to the local domain.

For example, the lines:

```
*DM_EXPORT
"//MUTT"
```

mean that the local domain with name MUTT accepts remote CORBA requests through any remote domain access point defined in the DM_REMOTE section.

## DM_IMPORT

The DM_IMPORT section, also known as the DM_REMOTE_SERVICES section, specifies in a CORBA environment the *domain_name* of the BEA Tuxedo domain to be imported through a remote domain access point defined in the DM_REMOTE section. The BEA Tuxedo domain specified for a remote domain access point is available to CORBA clients on the local domain. The DM_IMPORT section is required for a CORBA Domains configuration.

Entries in the DM_IMPORT section have the form:

```
service  [RACCESSPOINT=remote domain access point]
         [LACCESSPOINT=local domain access point]
         [TRANTIME=...]
```

where *service* is of the form:

```
"//domain_name"
```

This *domain_name* is the name assigned to the DOMAINID parameter in the RESOURCES section of the remote UBBCONFIG file. Entering this name in the DM_IMPORT section means that the remote domain accepts CORBA requests from the local domain. Also possible is to specify a *service* entry that accepts requests for a domain name other than the domain name of the remote domain, in the case where the remote domain acts as a pass-through for routing purposes.

For example, the lines:

```
*DM_IMPORT
  "//JEFF"     RACCESSPOINT=TDOM1
```

mean that the remote domain with name JEFF and associated with remote domain access point TDOM1 accepts CORBA requests through any local domain access point defined in the DM_LOCAL section.

## DM_ACCESS_CONTROL

The DM_ACCESS_CONTROL section specifies one or more access control list (ACL) names and associates one or more remote domain access points with each specified ACL name. You can use the ACL parameter in the DM_EXPORT section by setting ACL=*ACL_NAME* to restrict access to a local domain exported through a particular local domain access point to just those remote domain access points associated with the *ACL_NAME*.

Entries in the DM_ACCESS_CONTROL section have the form:

*ACL_NAME required_parameters*

where *ACL_NAME* is an *identifier* used to specify an access control list; it may contain no more than 15 characters.

The only required parameter is:

ACLIST = *identifier* [,*identifier*]

where an ACLIST is composed of one or more remote domain access point names separated by commas. The wildcard character (*) can be used to specify that all the remote domain access points defined in the DM_REMOTE section can access a local domain.

## DM_TDOMAIN

The DM_TDOMAIN section defines the network addressing information for the TDomain gateways implementing the BEA Tuxedo CORBA domains. The DM_TDOMAIN section should have:

- One entry per local domain access point if CORBA requests from remote domains are accepted through that access point

- One entry per remote domain access point if CORBA requests from the local domain to the remote domain are accepted through that access point

In the DM_TDOMAIN section, you can also define the configuration for one or more remote domain access points associated with one or more WebLogic Server applications, to combine Tuxedo CORBA servers and WebLogic Server Enterprise JavaBean (EJB) servers in an application. For details, see "Interoperability with BEA WebLogic Server" on page 2-1 in *BEA Tuxedo Interoperability*.

Entries in the DM_TDOMAIN section have the form:

```
AccessPoint required_parameters [optional_parameters]
```

where *AccessPoint* is an identifier value used to identify either (1) a local domain access point in the DM_LOCAL section or (2) a remote domain access point in the DM_REMOTE section.

The following parameter is required:

NWADDR = *string*

This parameter specifies the network address associated with a local domain access point or a remote domain access point. If the association is with a local domain access point, the network address is used by the local domain gateway to listen for connection requests from remote domains. If the association is with a remote domain access point, the network address is used by the local domain gateway to initiate a connection to the remote domain.

If *string* has the form "0x*hex-digits*" or "\\x*hex-digits*", it must contain an even number of valid hex digits. These forms are translated internally into a character array containing TCP/IP addresses. The addresses may also be in either of the following two forms:

"//*hostname*:*port_number*"
"//#.#.#.#:*port_number*"

In the first of these formats, *hostname* is resolved to a TCP/IP host address at the time the address is bound, using the locally configured name resolution facilities accessed via gethostbyname(3c). The "#.#.#.#" is the dotted decimal format, where each # represents a decimal number in the range 0 to 255.

*Port_number* is a decimal number in the range 0 to 65535 (the hexadecimal representations of the string specified).

For example, the lines:

```
*DM_TDOMAIN
  LDOM  NWADDR="//sanfran.kmart.com:2507"
  TDOM1 NWADDR="//sanhose.kmart.com:3186"
```

mean that the TDomain gateway belonging to gateway group LGWGRP—as stated in the DM_LOCAL section for the LDOM access point—is configured to listen at address "//sanfran.kmart.com:2507" for connection requests from remote domains. The TDomain gateway is also configured to initiate a connection to "//sanhose.kmart.com:3186" when sending requests to the remote domain associated with the TDOM1 access point.

For a description of the optional parameters for the DM_TDOMAIN section, see reference page DMCONFIG(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*.

# The factory_finder.ini File

The factory_finder.ini file identifies the remote factory objects that can be used in the local domain. It also identifies the local factory objects that can be used in remote domains.

The factory_finder.ini file contains two sections, DM_REMOTE_FACTORIES and DM_LOCAL_FACTORIES. As clarified in the following display, the format of the factory_finder.ini file is modeled after the syntax used in the DMCONFIG file:

```
*DM_REMOTE_FACTORIES
    "local_factory_id.factory_kind"
    DOMAINID="domain_id"
    RNAME="remote_factory_id.factory_kind"
        ...
*DM_LOCAL_FACTORIES
    "factory_id.factory_kind"
    ...
```

The following display demonstrates the syntax for CORBA factory objects:

```
*DM_REMOTE_FACTORIES
    "AccountFactory.FactoryKind"
    DOMAINID="MyAccountFactoryDomain"
    RNAME="MyAccountFactory.FactoryKind
```

where AccountFactory is the name used to register the factory in the local domain's FactoryFinder, MyAccountFactoryDomain is the name of the remote domain, and MyAccountFactory is the name used to register the factory in the remote domain's FactoryFinder.

**Note:** No two CORBA domains participating in a Domains configuration are allowed to have factory objects with the same *factory_id.factory_kind* identifier. For details, see

"Specifying Unique Factory Object Identifiers in the factory_finder.ini File" on page 3-19.

## DM_REMOTE_FACTORIES

The DM_REMOTE_FACTORIES section specifies which factory objects in remote domains are available (imported) to the local domain. Identifiers for remote factory objects are listed in this section. The identifier, under which the object is registered, including a *kind* value of FactoryInterface, must be listed in this section. For example, the entry for a remote factory object to be registered by the TP Framework with the identifier Teller in domain Norwest would be specified as:

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

If the RNAME is not specified, the *factory_kind* must be specified in the factory name, and the factory name must be enclosed in quotation marks; otherwise, the NameManager is not able to locate the appropriate factory. An entry that does not contain a *factory_kind* value is not defaulted with a value of FactoryInterface.

The following example shows a factory object to be registered with the identifier Teller in domain Norwest. Note the absence of the RNAME specification, the specification of the *factory_kind* value, and the quotation marks around the factory name.

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

Because the identities of factories in a Domains configuration may collide, the factory identifier and the RNAME parameters allow you to specify alternative identities, or "aliases," in the local domain for remote factories. The following listing shows two examples of a remote factory that is registered by the TP Framework with the identifier BankTeller in domain Norwest. In both examples, the factory is made available in the local domain with an alias of Teller.

**Listing 3-2 Assigning an Alias to a Remote Factory**

```
#EXAMPLE 1:
*DM_REMOTE_FACTORIES
   Teller
```

```
     DOMAINID="Norwest"
     RNAME=”BankTeller.FactoryInterface”
#EXAMPLE 2:
*DM_REMOTE_FACTORIES
   "Teller.FactoryInterface"
     DOMAINID="Norwest"
     RNAME="BankTeller.FactoryInterface"
```

You can also assign multiple aliases to the same remote factory. In the example shown in the following listing, the remote factory will be registered in the local domain with two aliases: `Teller` and `BankTeller`.

**Listing 3-3 Assigning Multiple Aliases to a Remote Factory**

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

## DM_LOCAL_FACTORIES

The `DM_LOCAL_FACTORIES` section specifies which factory objects in the local domain are available (exported) to remote domains. This section can be used in the following ways:

- If the `DM_LOCAL_FACTORIES` section is not present in a `factory_finder.ini`, or is present but empty, all factory objects in the local domain are available to remote domains. This software behavior allows administrators an easy means to make local factory objects available to remote domains without having to provide an entry for every factory object in the local domain.
- If the `DM_LOCAL_FACTORIES` section is present in a `factory_finder.ini` file but contains the reserved keyword `NONE`, none of the factory objects in the local domain are available to remote domains. Using the `NONE` keyword allows administrators to restrict access without having to provide an entry for every factory object in the local domain.

The identifier, or name, under which the factory object is registered, including a kind value of FactoryInterface, must be listed in the DM_LOCAL_FACTORIES section. For example, the entry for a factory object to be registered by the TP Framework with the identifier Teller would be specified as:

```
*DM_LOCAL_FACTORIES
  "Teller.FactoryInterface"
```

The *factory_kind* must be specified for the NameManager to locate the appropriate factory object. An entry that does not contain a *factory_kind* value is not defaulted with a value of FactoryInterface. This software behavior allows for the use of the CORBA NamingService.

An entry into the file for Domain A might be:

```
*DM_REMOTE_FACTORIES
fA.FactoryInterface DOMAINID=B
```

This entry means that a request in Domain A to find a factory with the identifier fA can be satisfied by the FactoryFinder in Domain B. Of course, the UBBCONFIG and DMCONFIG files for the two domains must also be set up so that there are connected domain gateways between the two domains.

An alternate form of the entry is:

```
CDE.FactoryInterface DOMAINID=B RNAME=fA.FactoryInterface
```

This entry means that a request in Domain A to find a factory with the identifier CDE will be satisfied by the FactoryFinder in Domain B using the ID fA. The alternate form is sometimes called an alias.

**Note:** The factory ID must have .FactoryInterface at the end. For simplicity, in discussions about test configurations, the .FactoryInterface is left off, but it should appear in the factory_finder.ini file.

# See Also

- UBBCONFIG(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*
- DMCONFIG(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*
- factory_finder.ini(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*

# Specifying Unique Factory Object Identifiers in the factory_finder.ini File

In a single-domain configuration, multiple factory objects with the same name are allowed, to achieve load balancing. In a Domains configuration, however, no two domains are allowed to have factory objects with the same *factory_id.factory_kind* identifier. If the same identifier, or name, is used in two domains, the software's behavior varies depending on whether or not BEA WebLogic Enterprise was used to configure the CORBA Domains environment:

- In releases prior to BEA WebLogic Enterprise 5.1, the software allows the first server in a domain to register the factory without issuing an error message. If two factories with the same name are registered in a domain, the Master NameManager fails.
- In BEA WebLogic Enterprise 5.1 or later and BEA Tuxedo 8.0 or later, the software generates an error and writes it to the ULOG.

There are two ways to ensure that your identifiers, or names, are unique across domains and thus avoid this problem:

- Use unique identifiers throughout the enterprise. Choosing this method may mean keeping a master list of all identifiers.
- In the `factory_finder.ini` file, use the `RNAME` parameter so that an alias is used by the local NameManager. Choosing this method means that you must also modify local clients to use the alias to access the remote factory object. The listing "Assigning an Alias to a Remote Factory" on page 3-16 shows an example of a `factory_finder.ini` file that uses the `RNAME` parameter to create an alias.

# Processing the factory_finder.ini File

When starting up, the Master NameManager reads the `factory_finder.ini` file. The condition under which the Master NameManager is started determines whether the Master NameManager reads all or just some of the `factory_finder.ini` file:

- If the Master NameManager process is started as part of booting the CORBA application (the initialization mode), it reads the entire content of the `factory_finder.ini` file. Thus, any new factory objects added to the `DM_REMOTE_FACTORIES` section of the `factory_finder.ini` file are made known to the local BEA Tuxedo application.
- If the Master NameManager process is restarted as a result of process failure, it reads only the `DM_LOCAL_FACTORIES` section of the `factory_finder.ini` file. Thus, any new

factory objects added to the DM_REMOTE_FACTORIES section are *not* made known to the local BEA Tuxedo application.

When adding a new domain with factory objects to the DM_REMOTE_FACTORIES section of the factory_finder.ini file, you must shut down and restart the Master NameManager. For more information about NameManager, see TMFFNAME(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*.

# Types of CORBA Domains Configurations

When using the BEA Tuxedo Domains component to connect multiple BEA Tuxedo CORBA domains, you can configure two types of configurations: directly connected domains and indirectly connected domains. You, as the administrator, configure both types using the DMCONFIG file.

## Directly Connected Domains

Every domain in a Domains configuration can have a gateway connection—a direct connection—to every other domain in the Domains configuration. With directly connected domains, a request goes directly to the target domain.

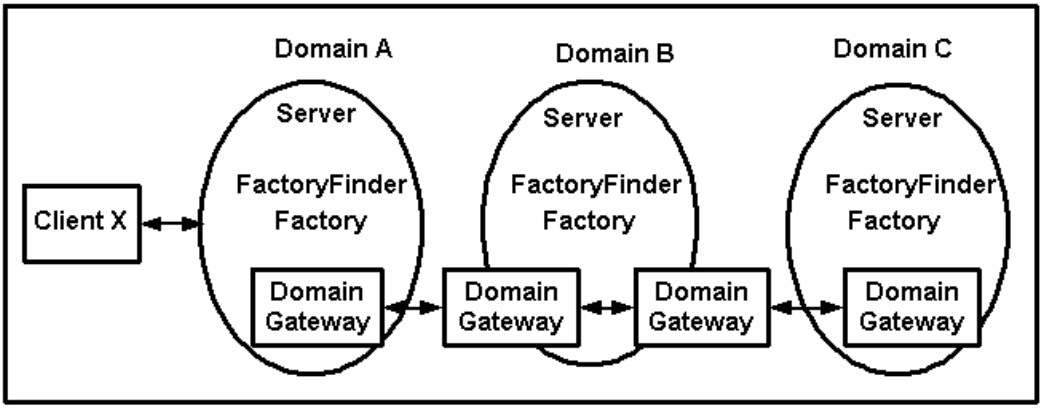
A directly connected Domains configuration, or "*n*-way" configuration, is reasonable when the number of domains is small, but each new domain added to the configuration requires two, four, ... or more new gateways. At some point, you may consider giving up speed of delivery for ease of management of domain connections by configuring indirectly connected domains.

## Indirectly Connected Domains

You should consider what the likely traffic patterns are. Domains that have only occasional interactions are candidates for gateway removal. Since there will still be interactions, it must still be possible to reach the other domain. The technique used is to route the request through an intermediate domain that does have direct access to the target domain.

For example, consider the three domains, A, B, and C, shown in the following figure.

**Figure 3-2 Indirectly Connected Domains**




Domains A and B are directly connected, and Domains B and C are directly connected, but A and C are not directly connected. For Domains A and C to communicate, they must use Domain B as the intermediary. Therefore, the DMCONFIG file for Domain A must state that it is possible to connect to Domain C by going through Domain B (and vice versa). That is, the connectivity is:

```
Domains      A       <->       B      <->     C
Gateways           GAB   GBA         GBC   GCB
```

Domain A has a gateway process, GAB (the gateway from A to B), that connects to Domain B. The Domain A DMCONFIG file states that GAB acts as a gateway to two domains, Domains B and C. The DMCONFIG file for Domain C has a similar configuration, stating that GCB is connected to B and A. The DMCONFIG file for Domain B has two gateway processes, one which connects to A (GBA) and one which connects to C (GBC). This configuration is called an indirect connection.

Given this indirect connection, when a server in A invokes a request on an object in C, BEA Tuxedo CORBA server knows that it can send the request to gateway GAB. The BEA Tuxedo gateway does not know that its partner gateway in B cannot service the request itself, but that is acceptable. Once the request is in Domain B, it is routed through GBC to C, which can service the request. Thus, the request is serviced with one extra hop.

It is even possible for the two gateways in Domain B to be a single gateway, so that there is not an extra hop within B. In effect, the same processing occurs in Domain B, but it all occurs within a single gateway process.

# Examples of CORBA Domains Configurations

The following examples show how to configure directly connected CORBA domains. If you want to use these examples, you will need to change the APPDIR, TUXCONFIG, and TUXDIR variables to match your environment. Also, you will have to substitute appropriate information wherever text is enclosed by left (<) and right (>) angle brackets (for example, *<App Server Name>*) and delete the angle brackets.

## Sample UBBCONFIG Files

The following three listings show the UBBCONFIG files for three directly connected domains: Here, There, and Yonder. To use these files, you must replace *host* with the name of the local machine.

**Listing 3-4 UBBCONFIG File for the Here Domain**

```
#
#     Copyright (c) 1999 BEA Systems, Inc.
#     All rights reserved
#
#
#
# RESOURCES
#
*RESOURCES
    IPCKEY    123312
    DOMAINID  HereD
    MASTER    LAPP
    MODEL     SHM
    LDBAL     N#
# MACHINES
#
*MACHINES
    <host>
           LMID=LAPP
           APPDIR="/tst1/wle4.2/test_dom/t07:
                    /tst1/wle4.2/dec_unix/wlemdomai"
           TUXCONFIG="/tst1/wle4.2/test_dom/tuxconfig"
```

```
          TUXDIR="/lclobb/lc"
          MAXWSCLIENTS=10
#
# GROUPS
#
*GROUPS
    DEFAULT:   LMID=LAPP
    ICEGRP     GRPNO=11 OPENINFO=NONE
    GROUP1     GRPNO=21 OPENINFO=NONE
    LDMGRP     GRPNO=3
    LGWGRP     GRPNO=4
#
# SERVERS
#
*SERVERS
    DEFAULT:   CLOPT="-A"
    DMADM      SRVGRP=LDMGRP SRVID=1
    GWADM      SRVGRP=LGWGRP SRVID=1
    GWTDOMAIN  SRVGRP=LGWGRP SRVID=2
    TMSYSEVT   SRVGRP=ICEGRP SRVID=1
    TMFFNAME   SRVGRP=ICEGRP SRVID=2
                   CLOPT="-A -- -N -M -f <FF ini file for Here>"
    TMFFNAME   SRVGRP=ICEGRP SRVID=3 CLOPT="-A -- -N"
    TMFFNAME   SRVGRP=ICEGRP SRVID=4 CLOPT="-A -- -F"
    <App Server Name>    SRVGRP=GROUP1 SRVID=2
    ISL        SRVGRP=GROUP1 SRVID=1
                    CLOPT="-A -- -d /dev/tcp -n //<host>:<port>"
#
# SERVICES
#
*SERVICES
```

UBBCONFIG File for the "There" Domain

```
#
#      Copyright (c) 1999 BEA Systems, Inc.
#      All rights reserved
```

```
#
# RESOURCES
#
*RESOURCES
    IPCKEY     133445
    DOMAINID   ThereD
    MASTER     LAPP1
    MODEL      SHM
    LDBAL  N
#
# MACHINES
#
*MACHINES
    <host>
         LMID=LAPP1
          APPDIR="D:\test_dom\t07;D:\Iceberg\qa\orb\bld\wlemdomain"
         TUXCONFIG="D:\test_dom\tuxconfig"
         TUXDIR="D:\Iceberg"
         MAXWSCLIENTS=10
#
# GROUPS
#
*GROUPS
    DEFAULT    LMID=LAPP1
    ICEGRP     GRPNO=11  OPENINFO=NONE
    GROUP1     GRPNO=21  OPENINFO=NONE
    LDMGRP     GRPNO=3
    LGWGRP     GRPNO=4
#
# SERVERS
#
*SERVERS
    DEFAULT:   CLOPT="-A"
    DMADM      SRVGRP=LDMGRP SRVID=1
    GWADM      SRVGRP=LGWGRP SRVID=1
    GWTDOMAIN  SRVGRP=LGWGRP SRVID=2
    TMSYSEV    SRVGRP=ICEGRP SRVID=1
    TMFFNAME   SRVGRP=ICEGRP SRVID=2
```

```
                CLOPT="-A -- -N -M -f <FF ini file for There>"
    TMFFNAME    SRVGRP=ICEGRP SRVID=3 CLOPT="-A -- -N"
    TMFFNAME    SRVGRP=ICEGRP SRVID=4 CLOPT="-A -- -F"
    <App Server Name>     SRVGRP=GROUP1 SRVID=2
    ISL         SRVGRP=GROUP1 SRVID=1
                   CLOPT="-A -- -d /dev/tcp -n //<host>:<port>"
#
# SERVICES
#
*SERVICES
```

**Listing 3-5 UBBCONFIG File for the Yonder Domain**

```
#       Copyright (c) 1999 BEA Systems, Inc.
#       All rights reserved
#
# RESOURCES
#
*RESOURCES
    IPCKEY    123334
    DOMAINID  YonderD
    MASTER    LAPP
    MODEL     SHM
    LDBAL     N
#
# MACHINES
#
*MACHINES
    <host>
          LMID=LAPP
          APPDIR="/tst1/wle4.2/test_dom/t07p:
                             /tst1/wle4.2/<host3>/wlemdomain"
          TUXCONFIG="/tst1/wle4.2/test_dom/<host3>/tuxconfig"
          TUXDIR="/lclobb/lc"
          MAXWSCLIENTS=10
#
```

```
# GROUPS
#
*GROUPS
    DEFAULT:    LMID=LAPP
    ICEGRP      GRPNO=11 OPENINFO=NONE
    GROUP1      GRPNO=21 OPENINFO=NONE
    LDMGRP      GRPNO=3
    LGWGRP      GRPNO=4
#
# SERVERS
#
*SERVERS
    DEFAULT:    CLOPT="-A"
    DMADM       SRVGRP=LDMGRP SRVID=1
    GWADM       SRVGRP=LGWGRP SRVID=1
    GWTDOMAIN   SRVGRP=LGWGRP SRVID=2
    TMSYSEVT    SRVGRP=ICEGRP SRVID=1
    TMFFNAME    SRVGRP=ICEGRP SRVID=2
                   CLOPT="-A -- -N -M"
    TMFFNAME    SRVGRP=ICEGRP SRVID=3 CLOPT="-A -- -N"
    TMFFNAME    SRVGRP=ICEGRP SRVID=4 CLOPT="-A -- -F"
    <App Server Name>  SRVGRP=GROUP1 SRVID=2
    ISL         SRVGRP=GROUP1 SRVID=1
                   CLOPT="-A -- -d /dev/tcp -n //<host>:<port>"
#
# SERVICES
#
*SERVICES
```

# Sample DMCONFIG File

The following three listings show the DMCONFIG files for three directly connected domains: Here, There, and Yonder. To use these listings in a Domains configuration, you must replace *host1* with the name of the local machine for the Here domain, replace *host2* with the name of the local machine for the There domain, and replace *host3* with the name of the local machine for the Yonder domain.

**Listing 3-6 DMCONFIG File for the Local Machine in the Here Domain in a Three-Domain Configuration**

```
#
# Copyright (c) 1999 BEA Systems, Inc.
#       All rights reserved
#
#
# Tuxedo Domains Configuration File
#
*DM_RESOURCES
    VERSION=U22
#
# DM_LOCAL
#
*DM_LOCAL
    LDOM1   GWGRP=LGWGRP  TYPE=TDOMAIN  ACCESSPOINTID="HereG"
#
# DM_REMOTE
#
*DM_REMOTE
    TDOM1   TYPE=TDOMAIN  ACCESSPOINTID="ThereG"
    TDOM2   TYPE=TDOMAIN  ACCESSPOINTID="YonderG"

#
# DM_TDOMAIN
#
*DM_TDOMAIN
    LDOM1   NWADDR="//<host1>:<tcpport>"
    TDOM1   NWADDR="//<host2>:<tcpport>"
    TDOM2   NWADDR="//<host3>:<tcpport>"
#
# DM_EXPORT
#
*DM_EXPORT
    "//HereD"
```

```
#
# DM_IMPORT
#
*DM_IMPORT

    "//ThereD"   RACCESSPOINT=TDOM1
    "//YonderD"  RACCESSPOINT=TDOM2
```

To use the following listing in a Domains configuration, you must replace *host1* with the name of the local machine for the `There` domain, replace *host2* with the name of the local machine for the `Here` domain, and replace *host3* with the name of the local machine for the `Yonder` domain.

**Listing 3-7 DMCONFIG File for the There Domain in a Three-Domain Configuration**

```
#
# Copyright (c) 1999 BEA Systems, Inc.
#      All rights reserved
#
#
# Tuxedo Domains Configuration File
#
*DM_RESOURCES

    VERSION=U22

#
# DM_LOCAL
#
*DM_LOCAL

    LDOM1   GWGRP=LGWGRP  TYPE=TDOMAIN  ACCESSPOINTID="ThereG"

#
# DM_REMOTE
#
*DM_REMOTE
```

```
    TDOM1   TYPE=TDOMAIN  ACCESSPOINTID="HereG"
    TDOM2   TYPE=TDOMAIN  ACCESSPOINTID="YonderG"

#
# DM_TDOMAIN
#
*DM_TDOMAIN
    LDOM1   NWADDR="//<host1>:<tcpport>"
    TDOM1   NWADDR="//<host2>:<tcpport>"
    TDOM2   NWADDR="//<host3>:<tcpport>"
#
# DM_EXPORT
#
*DM_EXPORT
    "//ThereD"
#
# DM_IMPORT
#
*DM_IMPORT
    "//HereD"     RACCESSPOINT=TDOM1
    "//YonderD"   RACCESSPOINT=TDOM2
```

To use the following listing in a Domains configuration, you must replace *host1* with the name of the local machine for the Yonder domain, replace *host2* with the name of the local machine for the Here domain, and replace *host3* with the name of the local machine for the There domain.

**Listing 3-8 DMCONFIG File for the Yonder Domain in a Three-Domain Configuration**

```
#
# Copyright (c) 1999 BEA Systems, Inc.
#      All rights reserved
#
#
```

```
# Tuxedo Domains Configuration File
#
*DM_RESOURCES
    VERSION=U22
#
# DM_LOCAL
#
*DM_LOCAL
    LDOM1   GWGRP=LGWGRP  TYPE=TDOMAIN  ACCESSPOINTID="YonderG"
#
# DM_REMOTE
#
*DM_REMOTE
    TDOM1   TYPE=TDOMAIN  ACCESSPOINTID="HereG"
    TDOM2   TYPE=TDOMAIN  ACCESSPOINTID="ThereG"
#
# DM_TDOMAIN
#
*DM_TDOMAIN
    LDOM1   NWADDR="//<host1>:<tcpport>"
    TDOM1   NWADDR="//<host2>:<tcpport>"
    TDOM2   NWADDR="//<host3>:<tcpport>"
#
# DM_EXPORT
#
*DM_EXPORT
    "//YonderG"
#
# DM_IMPORT
#
*DM_IMPORT
    "//HereD"   RACCESSPOINT=TDOM1
    "//ThereD"  RACCESSPOINT=TDOM2
```

# Sample factory_finder.ini File

The following two listings show the `factory_finder.ini` files for the `Here` and `There` domains. The `Yonder` domain does not require a `factory_finder.ini` file.

**Listing 3-9 factory_finder.ini File for the Here Local Domain**

```
# Copyright (c) 1999 BEA Systems, Inc.
# All rights reserved
#
# Factory Finder Initialization file for Domain "Here"
# This is the local domain.
#
# DM_LOCAL_FACTORIES
#
*DM_LOCAL_FACTORIES

    "AFactory.FactoryInterface"
#
# DM_REMOTE_FACTORIES
#
*DM_REMOTE_FACTORIES
    "AFacYonder.FactoryInterface"
       DOMAINID="YonderD"
       RNAME="AFactory.FactoryInterface"

    "BFactory.FactoryInterface"
       DOMAINID="YonderD"
```

**Listing 3-10 factory_finder.ini File for the There Remote Domain**

```
#
# Copyright (c) 1999 BEA Systems, Inc.
# All rights reserved
#
# Factory Finder Initialization file for Domain "There"
```

```
# This is a remote domain.
#
# DM_LOCAL_FACTORIES
#
*DM_LOCAL_FACTORIES
    "AFactory.FactoryInterface"
#
# DM_REMOTE_FACTORIES
#
*DM_REMOTE_FACTORIES
    "AFacYonder.FactoryInterface"
       DOMAINID="YonderD"
       RNAME="AFactory.FactoryInterface"
    "BFactory.FactoryInterface"
       DOMAINID="YonderD"
```

CHAPTER 4

# Administering Domains

The following sections explain how to administer a BEA Tuxedo Domains environment:

- Using Domains Run-Time Administrative Commands
- Using the Administrative Interface, dmadmin(1)
- Using the Domains Administrative Server, DMADM(5)
- Using the Gateway Administrative Server, GWADM(5)
- Using the Domain Gateway Server
- Managing Transactions in a Domains Environment

## Using Domains Run-Time Administrative Commands

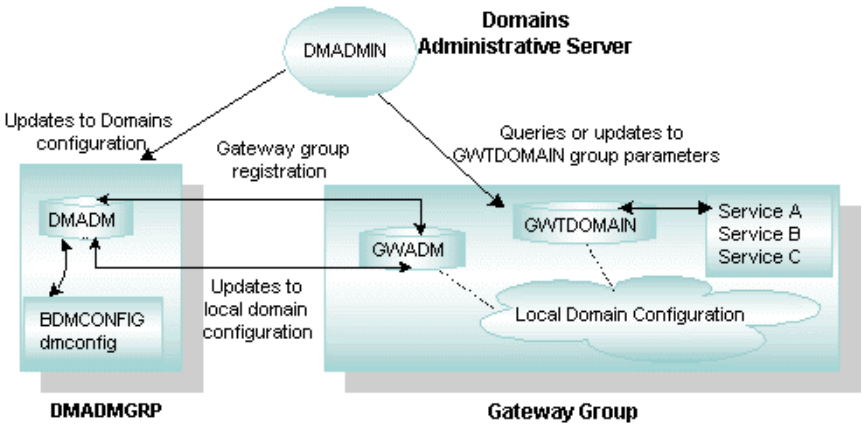
To integrate the Domains component with an existing BEA Tuxedo application, you add entries for domain gateway groups and gateway servers to the `TUXCONFIG` file. You can use either the `tmconfig(1)` or `tmadmin(1)` command to add a Domains configuration to a running BEA Tuxedo application. You can also use `tmadmin` to list the information available in the bulletin board for domain gateway groups and individual domain gateways.

Once your Domains environment is configured and integrated, you can administer it dynamically using a set of administrative tools provided by the Domains component. For example, you can specify and modify the list of services that are accessible across applications. The Domains software preserves the characteristics of the BEA Tuxedo programming interface (ATMI) and extends the scope of the ATMI so that clients can invoke services across domains. This

functionality allows programmers to expand or partition applications without changing any application code.

The following figure shows the relationship between administrative commands and servers in the Domains administrative subsystem.

**Figure 4-1 Domains Run-Time Administration**




The BEA Tuxedo Domains component offers the following administrative commands:

- dmadmin(1) command, a generic administrative service—Enables administrators to configure, monitor, and tune domain gateway groups dynamically, and to update the Domains configuration file (BDMCONFIG) while the BEA Tuxedo application is running. The command acts as a front-end process that translates administrative commands and sends service requests to the DMADMIN service, a generic administrative service advertised by the DMADM server. The DMADMIN service invokes the validation, retrieval, or update of functions provided in the DMADM server to maintain the BDMCONFIG file.

- DMADM(5), the Domains administrative server—Provides the administrative processing required for updating the Domains configuration. This server acts as a back-end to the dmadmin command. It provides a registration service to domain gateway groups. This registration service is requested by GWADM servers as part of their initialization procedure. The registration service downloads the configuration information required by the requesting domain gateway group. The DMADM server maintains a list of registered domain gateway groups, and propagates to these groups any changes made to the configuration.

- GWADM(5), the gateway administrative server—Registers with the DMADM server to obtain the configuration information used by the corresponding domain gateway group. The

GWADM accepts queries from DMADM to obtain run-time statistics or to change the run-time options of the corresponding domain gateway group. Periodically, the GWADM server sends an "I-am-alive" message to the DMADM server. If no reply is received from the DMADM server, the GWADM server registers again. This mechanism makes sure the GWADM server always has the latest copy of the Domains configuration for its group.

- GWTDOMAIN(5), the TDomain gateway server—Provides interoperability between two or more BEA Tuxedo domains. Working with the WebLogic Tuxedo Connector (WTC) gateway, a BEA WebLogic Server component, the BEA Tuxedo TDomain gateway can also provide interoperability between Tuxedo domains and WebLogic Server applications.

  **Note:** For information about domain gateway types other than GWTDOMAIN, see *BEA eLink Documentation* at http://e-docs.bea.com/elink/mainfram/mainfram.htm.

- BDMCONFIG—the binary version of the Domains configuration file, which together with the TUXCONFIG file and factory_finder.ini file (CORBA only) contain all the configuration parameters that the BEA Tuxedo software needs to create a Domains configuration.

**Note:** You can also specify gateway parameters when a domain gateway group is booted using the CLOPT parameter, when the GWADM server is defined in the SERVERS section of the TUXCONFIG file.

# Using the Administrative Interface, dmadmin(1)

dmadmin is an administrative interface to the DMADM and GWADM servers. The communication between the two servers is done via FML typed buffers. Administrators can use the dmadmin command in the following ways:

- For the interactive administration of the information stored in the BDMCONFIG file and the different domain gateway groups running within a particular BEA Tuxedo application.
- To obtain statistics or other information gathered by domain gateway groups.
- To change domain gateway group parameters.
- To add (or update) information in the BDMCONFIG file.

**Note:** You can delete information from the BDMCONFIG file at run time only if the deletions do not involve an active domain gateway group.

## See Also

- dmadmin(1) in *BEA Tuxedo Command Reference*

# Using the Domains Administrative Server, DMADM(5)

The Domains administrative server, DMADM(5), is a BEA Tuxedo-supplied server that performs the following functions:

- Supports run-time administration of the BDMCONFIG file
- Maintains the BDMCONFIG file
- Supports a list of registered domain gateway groups
- Propagates run-time configuration changes to the registered domain gateway groups

The DMADM server advertises two services:

- DMADMIN, which is used by the dmadmin command and the GWADM server.
- A service called DMADM_*svrid*, where *srvid* is the appropriate server ID for the service. Registered GWADM servers use DMADM_*svrid* for specific administrative functions (for example, to refresh the domain gateway group configuration information or to signal that a GWADM is still registered).

The DMADM server must be defined in the SERVERS section of the TUXCONFIG file as a server running within a group (for example, DMADMGRP). There should be only one instance of the DMADM server in this group.

## See Also

- DMADM(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*

# Using the Gateway Administrative Server, GWADM(5)

The gateway administrative server, GWADM(5), is a BEA Tuxedo-supplied server that provides administrative functions for a domain gateway group. The main functions of the GWADM server include:

- Getting Domains configuration information from the DMADM server, and accepting queries from dmadmin. The GWADM server gets the domain gateway group configuration information by registering with the DMADM server. The GWADM server then makes the configuration available to gateways by storing the information in shared memory.

- Providing administrative functionality for a domain gateway group, for example, by accepting queries from dmadmin for run-time statistics or by changing the run-time parameters of the domain gateway group.
- Providing transaction logging functionality for a domain gateway group. The GWADM server determines which transactions need to be logged by reading information stored in shared memory. When the GWADM server is booted, it scans the log to see whether any transactions need to be recovered, and then reconstructs the transaction information in shared memory. The gateway server scans the information in shared memory and performs recovery for the corresponding transactions. The recovery procedure is performed asynchronously with new incoming or outgoing requests received by the domain gateway group.

The GWADM server advertises a service name based on the local domain access point name (as specified in the DM_LOCAL section of the BDMCONFIG file) associated with the domain gateway group to which the GWADM server belongs. The dmadmin command uses this service to retrieve information from all active domain gateway groups or from a specific domain gateway group.

The GWADM server must be defined in the SERVERS section of the TUXCONFIG file. It should not be part of the MSSQ used by the gateways associated with the group. It must be the first server booted within the domain gateway group; that is, either (a) it must have a SEQUENCE number, or (b) it must be defined ahead of the gateway servers.

The GWADM server requires the existence of a DMADM server. Specifically, a DMADM server must be booted before that GWADM is booted.

The GWADM server must create the shared memory required by the domain gateway group to populate the configuration tables with information received from the DMADM server. The GWADM server uses IPC_PRIVATE with shmget and stores the ipckey returned in the shmid field of its registry entry in the bulletin board. Gateways can obtain the ipckey by retrieving the GWADM registry entry and checking the shmid field.

## See Also

- GWADM(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*

# Using the Domain Gateway Server

A domain gateway server provides connectivity to remote domain gateway servers, and can communicate with one or more remote gateways simultaneously. A gateway advertises the services imported to a BEA Tuxedo application and controls access to the local services exported by the application. You define your application's exported and imported services in the Domains

configuration file (DMCONFIG). Use dmadmin to dynamically configure, monitor, and tune domain gateway groups.

## See Also

- "Types of Domain Gateways" on page 1-3

## Tuning the Performance of the Domain Gateway

BEA Tuxedo 9.x improves the GWTDOMAIN gateway performance while also keeping compatibility with other types of /Domain gateways. It limits most of the performance to the threaded platforms. It also allows other types of /Domain gateways to take advantage of this feature in the enhanced Common Gateway Architecture with simple program changes.

Many factors may affect performance of applications across multiple domains. For example:

- Service processing time
- Database transactions
- Message buffer size
- Security facilities, such as Link Level Encryption
- Network transmission speed

Therefore, in order to observe achieved Domain Gateway performance improvement, the application has to minimize the above-mentioned factors. Otherwise the performance improvement for the gateway may not be significant.

Some suggestions for performance testing this feature are listed below:

- Many clients send a request to a single remote service. The size of the message is small (less than a few K bytes.)
- Many clients send a request to different remote services or use load balancing to distribute service requests. In this case, the message size can be larger.

**Note:** As a prerequisite, service processing time on the server side should *not* be time-consuming. The total response time contains both gateway processing time and service processing time. If service processing time is quite long, the performance improvement for the gateway is submerged.

# Managing Transactions in a Domains Environment

Application programmers can request the execution of remote services within a transaction. Also, users of remote domains can request local services to be executed within a transaction. Domains, therefore, coordinates the mapping of remote transactions to local transactions, and the sane termination (commitment or rollback) of these transactions.

The BEA Tuxedo system architecture uses a separate process, the Transaction Manager Server (TMS), to coordinate the commitment and recovery of transaction branches accessing a particular group. In a Domains environment, however, this architecture would require extra messages from the gateway to the TMS server to process a commitment for an incoming transaction. To simplify the Domains architecture and to reduce the number of messages, the TMS code is integrated with the gateway code. Thus, domain gateways can process the transaction protocol used by the BEA Tuxedo system. The BEA Tuxedo transaction protocol requires that the domain gateway group advertise the TMS service, which is done when the first gateway is booted. Once the TMS service is advertised, any transaction control messages directed to the domain gateway group are placed on the gateway's queue.

Domain gateway groups should be defined in the TUXCONFIG file without the TMSNAME, TMSCOUNT, OPENINFO, and CLOSEINFO parameters. These four parameters apply only to groups that use an XA-compliant resource manager, which domain gateways do not use.

The commitment protocol across domains is strictly hierarchical. It is not possible to flatten the transaction tree because the structure of the transaction tree is not fully known by every domain; a superior knows only its immediately subordinate domains. Flattening the tree would also require the root domain to be fully connected to all domains participating in the transaction.

Domain gateways provide four capabilities that you can use to manage transactions. These capabilities are described in the following sections:


- "Using the TMS Capability Across Domains" on page 4-7
- "Using GTRID Mapping in Transactions" on page 4-11
- "Using Logging to Track Transactions" on page 4-18
- "Recovering Failed Transactions" on page 4-20


## Using the TMS Capability Across Domains

In the BEA Tuxedo system, the TMS is a special server that is implicitly associated with server groups that use X/Open XA-compliant resource managers. The TMS server releases application

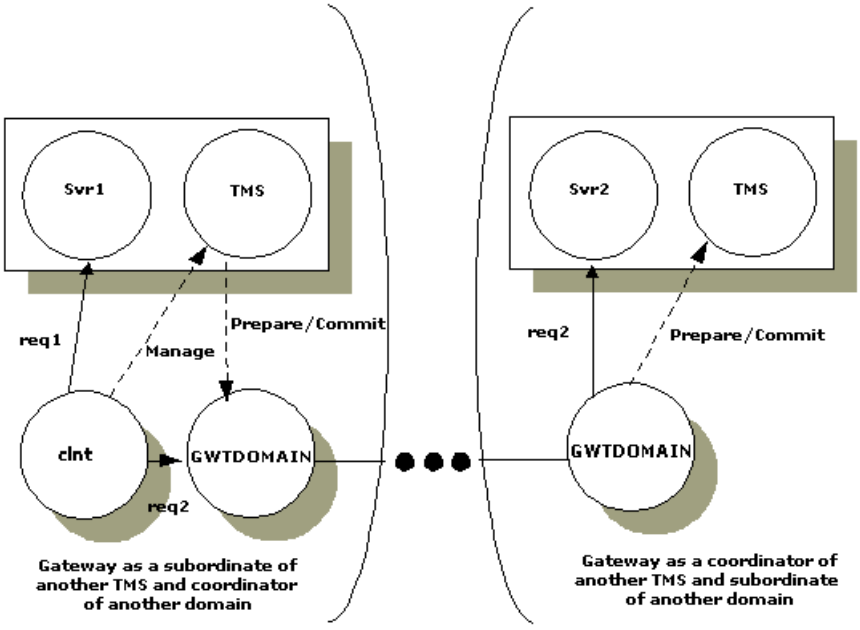
servers from the delays associated with the distributed 2-phase commitment protocol. TMS servers coordinate the commitment of a transaction via special service requests to the TMS service, which is offered by all TMS servers.

In a Domains environment, GWTDOMAIN gateways are not associated with an XA-compliant resource manager. The Transaction Processing Working Group (TPWG) of X/Open has proposed an advanced XA interface. This interface is not used in the BEA Tuxedo system because the interface does not match the highly asynchronous and non-blocking model required by the gateway. While domain gateways do not use a separate TMS server, they do offer the Transaction Manager Servers capability, which allows gateways to coordinate the 2-phase commitment of transactions executed across domains.

Domain gateways coordinate transactions across domains in the following manner:

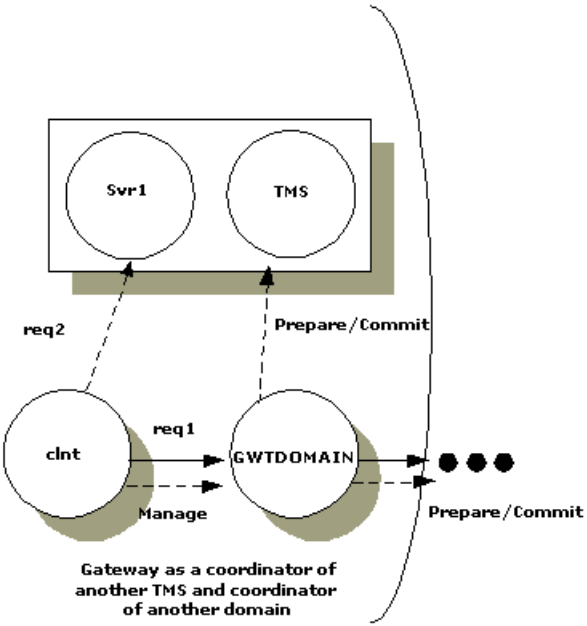
1. Domain gateways advertise the TMS service and perform all operations associated with that service. Messages sent to this service are placed on the queue used by the appropriate domain gateway group, and the gateways manage the transactions associated with the group.
2. A gateway can act as a subordinate of transactions coordinated by another group within the domain. In this case, the gateway is a superior of the transaction branches executed in other remote domains. When acting as a subordinate of a transaction coordinated by a remote domain, the gateway also acts as the coordinator for all groups in the local domain accessed by the transaction. The gateway, acting as both subordinate and coordinator, is illustrated in the following figure.

**Figure 4-2 The Domain Gateway as Subordinate/Coordinator of Another Domain Gateway Group**




3. As a coordinator of transactions within the domain, the gateway manages the commitment of a transaction for a particular client. This is illustrated in the following figure.

**Figure 4-3 Client Commit Managed by a Domain Gateway**




4. Gateways manage transaction commitment for a particular client or for a server that uses the forwarding service with the AUTOTRAN capability. When this combination is used, the last server in the forward chain (the domain gateway) issues the commit and becomes the coordinator of the transaction. (A domain gateway always acts as the last server in a forward chain.)

5. Gateways automatically start and terminate transactions for remote services specified with the AUTOTRAN capability. This capability is required when an the application administrator wants to enforce reliable network communication with remote services. Administrators can specify this capability by setting the AUTOTRAN parameter to Y in the corresponding remote service definition.

   For more information, see the DM_IMPORT section of DMCONFIG(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*.

6. Gateways map the BEA Tuxedo system transaction protocol to the networking transaction protocol used for interoperation with remote domains. How this mapping is done depends on which instantiation of domain gateway you are using: TDomain, SNA, or OSI TP.

# Using GTRID Mapping in Transactions

In the BEA Tuxedo system, a transaction tree is a 2-level tree where the root is the domain gateway group coordinating a global transaction and branches are involved in the transaction. Each group performs its part of the global transaction independently from the parts performed by other groups. Each group, therefore, implicitly defines a transaction branch. The BEA Tuxedo system, through TMS servers, coordinates the completion of the global transaction, making sure each branch is completed.

A GTRID is a Global Transaction Identifier. GTRID mapping defines how to construct a transaction tree that crosses domain boundaries. You specify GTRIDs using the MAXGTT parameter in the RESOURCES section of the BEA Tuxedo configuration file.

## Defining Tightly-coupled and Loosely-coupled Relationships

In the X/Open DTP Model, a Transaction Manager Server can construct transaction trees by defining either *tightly-coupled* or *loosely-coupled* relationships with a resource manager (RM) by the way it interprets the transaction identifiers (XIDs) used by the XA interface.

A *tightly-coupled relationship* is one in which a single transaction identifier, XID, is used by all processes participating in a single global transaction, accessing a single RM. This relationship maximizes data sharing between processes; XA-compliant RMs expect to share locks for resources used by processes having the same XID. The BEA Tuxedo system achieves the tightly-coupled relationship via the group concept; that is, all work done by a group on behalf of a given global transaction belongs to the same transaction branch; all the processes executed by the group are given the same XID.

In a *loosely-coupled relationship*, the TMS generates a transaction branch for each part of the work in support of the global transaction. The RM handles each transaction branch separately; there is no sharing of data or of locks between the transaction branches. Deadlocks between transaction branches can occur and result in the rollback of a global transaction. In the BEA Tuxedo application, when different groups participate in a single global transaction, each group defines a separate transaction branch, which results in a loosely-coupled relationship.

## Global Transactions Across Domains

There are several differences between global transactions in a single BEA Tuxedo application and global transactions across domains. The first difference is that in the Domains framework, the transaction tree cannot be flattened to a 2-level tree. There are two reasons for this:

- The transaction may involve more domains than can be known from the root domain (where the transaction is controlled), so the structure of the transaction tree cannot be fully known.
- If a transaction tree is flattened to two levels, the root domain must be connected directly to all domains in the transaction.

This means that the commitment protocol across domains must be hierarchical. Even a loopback service request defines a new branch in the transaction tree.

**Note:** A loopback request goes to another domain and then comes back to be processed in the original domain. For example, Domain A requests a service of Domain B. The service in Domain B requests another service in Domain A. The transaction tree has two branches at the network level: a branch b1 from A to B and a branch b2 from B to A. Domain A cannot commit the work done on branch b2 before receiving commit instructions from B.

The structure of a transaction tree for global transactions across domains also depends on the distributed transaction processing protocol used by a relevant domain gateway instantiation. For example, in the OSI TP protocol each *dialogue* (the OSI TP word for a service request) is associated with a different transaction branch. In the BEA Tuxedo system, the OSI TP instantiation uses a dialogue for each service request, so each service request is mapped to a separate transaction branch. The XAP-TP interface hides this mapping and provides a mechanism by which an entire OSI TP subtree can be referenced by a user-defined identifier. (In the BEA Tuxedo implementation, this identifier is the `GTRID`.) The `GTRID` is used to instruct XAP-TP how a transaction tree must be constructed, that is, which dialogues must be included within a given OSI TP transaction. Therefore, from the BEA Tuxedo perspective, a whole OSI TP subtree can be managed as a single transaction branch.

This property, however, applies only to outgoing service requests (that is, service requests sent from the root domain to subordinate domains). It cannot be applied to incoming service requests. The OSI TP instantiation consequently implements a loosely-coupled relationship; each incoming service request is mapped to a new BEA Tuxedo global transaction.

The TDomain instantiation tries to optimize `GTRID` mapping by implementing a tightly-coupled relationship. In TDomain, multiple service requests issued on behalf of the same global transaction are mapped to the same network transaction branch. Therefore, incoming service requests can be mapped to a single BEA Tuxedo transaction. However, the hierarchical structure of interdomain communication and the interdomain transaction tree must still be maintained.

The optimization that TDomain introduces applies only to a single domain. When two or more domains are involved in a transaction, the network transaction tree contains at least one branch per domain interaction. Hence, across domains, the network transaction tree remains

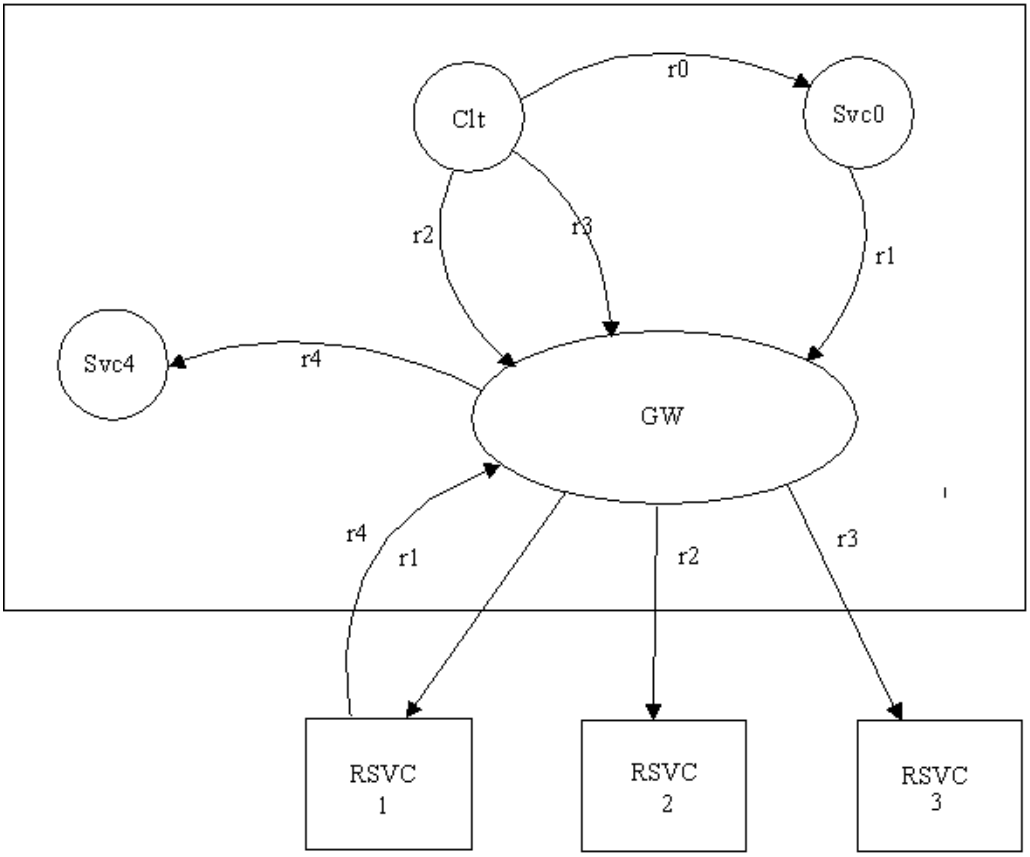
loosely-coupled. There are as many branches as there are domains involved in the transaction (even if all the branches access the same resource manager instance).

Domain gateway groups implement a loosely-coupled relationship because they generate different transaction branches for interdomain transactions.

## Example of a Service Request Graph Generating Local and Remote Requests

The following figure shows the service request graph for a client that generates three service requests: one local request (`r0`) and two remote requests (`r2` and `r3`). Request `r0` goes to a local service (`Svc0`), which generates another remote service request (`r1`). Request `r1` goes to remote service `Rsvc1`, which issues a loopback service request `r4` to local service `Svc4`. `Svc0` and `Svc4` are executed in different groups (`G0` and `G4`). The domain gateway is executed within another group (`GW`), and the remote services `Rscv1`, `Rsvc2`, and `Rsvc3` are executed in another domain (Domain B).

**Figure 4-4 Service Request Graph**




## Transaction Trees for BEA eLink OSI TP and BEA Tuxedo Domains

The following two figures show the transaction tree for BEA eLink OSI TP and the transaction tree for BEA Tuxedo domains. It is assumed, in these figures, that both Domain A and Domain B are BEA Tuxedo system applications.

BEA eLink OSI TP is loosely-coupled because of the OSI TP protocol. The transaction tree for this instantiation shows group G0 in Domain A coordinating the global transaction started by the client. Group `G0` coordinates group `GW`. Requests `r1`, `r2`, and `r4` are mapped each to an OSI TP dialogue and therefore to an OSI TP transaction branch. However, OSI TP uses the XAP-TP feature that allows an entire OSI TP transaction to be referred by a unique identifier (`T1`) and uses

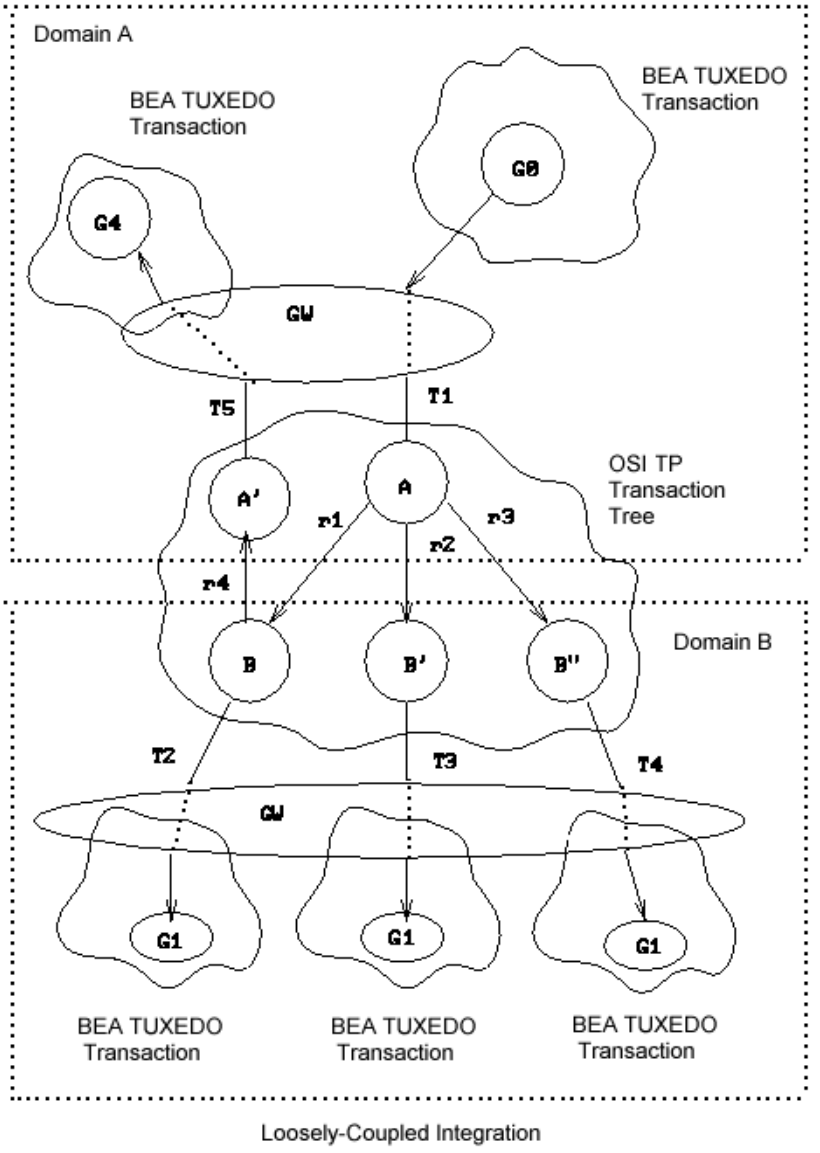
this identifier for requests r1, r2, and r3. It is up to XAP-TP to generate OSI TP transaction identifiers and to construct the corresponding OSI TP transaction tree. The only function that must be performed by the generic Domains software is the mapping of service requests r1, r2, and r3 to the T1 identifier.

In Domain B, OSI TP uses the rule that new transaction branches must be mapped to a new BEA Tuxedo transaction. Therefore, OSI TP transaction branches r1, r2, and r3 get mapped to three different BEA Tuxedo transactions (the corresponding mapping is represented by identifiers T2, T3, and T4). The graph shows the domain gateway group GW in Domain B coordinating three BEA Tuxedo transactions on group G1.

Finally, there is the loopback service request r4 that generates another branch in the transaction tree. OSI TP maps this request to identifier T2, but XAP-TP generates a new branch in its transaction tree (r4: B to A'). This is a new transaction branch on Domain A, and therefore, the gateway generates a new mapping T5 to a new BEA Tuxedo transaction. Therefore, the transaction graph shows that domain gateway group GW on Domain A coordinates group G4.

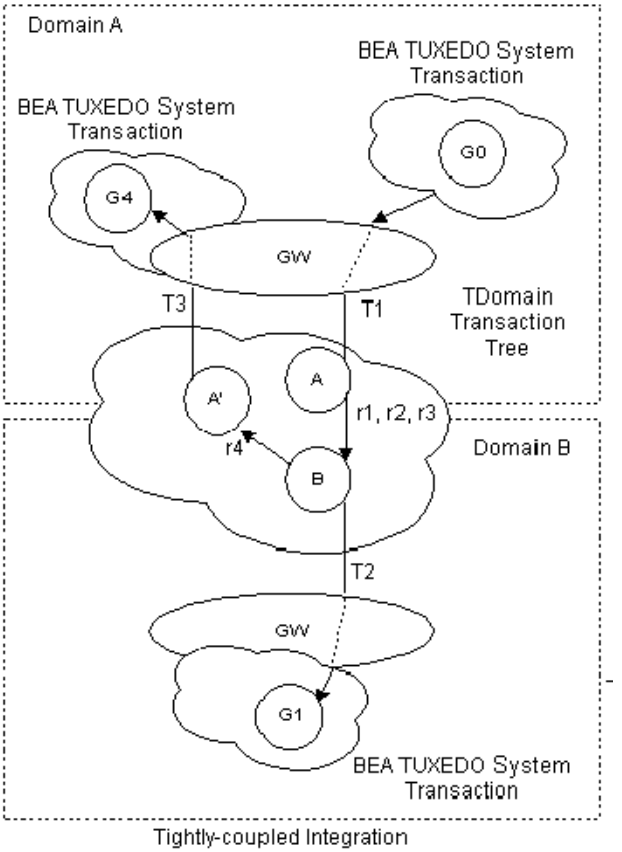
Notice that the hierarchical nature of the OSI TP protocol is fully enforced by these mappings. However, because these mappings introduce a loosely-coupled relationship, the probability of intratransaction deadlock is increased (for example, there are three BEA Tuxedo transactions accessing the RM represented by group G1).

**Figure 4-5 Transaction Tree for BEA eLink OSI TP Environment**



The TDomain instantiation provides a tightly-coupled integration that solves this deadlock problem by minimizing the number of transaction branches required in the interoperation between two domains. The corresponding transaction tree is shown in the following figure.

**Figure 4-6 Transaction Tree for TDomain Environment**




Notice that the gateway still must perform mappings between a BEA Tuxedo system transaction and a network transaction, and that the hierarchical nature of the communication between domains must be strictly enforced. The diagram shows that requests `r1`, `r2`, and `r3` are mapped to a single TDomain transaction branch. Therefore, on Domain B only one BEA Tuxedo system transaction needs to be generated; `T2` represents this mapping and the graph shows domain gateway group `GW` on Domain B coordinating group `G1`. Request `r4` is mapped to identifier `T2` on Domain B, but TDomain will generate a new branch in its transaction tree (`r4`: B to A'). Because

this is a new transaction branch on Domain A, the gateway generates a new mapping, T3, to a new BEA Tuxedo system transaction. The graph shows that domain gateway group GW on Domain A also coordinates group G4. Hence, the hierarchical nature of interdomain communication is fully enforced with this mapping: group G4 cannot commit before group G1.

### Summary of Domains Transaction Management

Domains transaction management can be summarized as follows:

- Gateways generate mappings from a BEA Tuxedo system transaction to a network transaction. A new mapping is generated for each BEA Tuxedo system transaction and each incoming network transaction branch.
- Each instantiation of domain gateway (TDomain, SNA, or OSI TP) handles its own representation of the network transaction tree. All instantiations observe the hierarchical nature of the interdomain communication.

## Using Logging to Track Transactions

Logging is used to keep track of the progress of a 2-phase commit protocol. The information stored in the log is used to make sure a transaction is completed in the event of a network failure or machine crash.

To ensure completion of transactions across domains, domain gateways log the mapping between local and remote identifiers. Along with this information, the Domains transaction management facility records the decisions made during different phases of the commitment protocol, and any information available about the remote domains involved in the transaction. In the OSI TP case, the XAP-TP interface logs the information required for the recovery of the OSI TP protocol machine. The information is referred to as a *blob* (binary large object) and is kept in the same log record as the commit information to make recovery easier.

Domains log records have a different structure from the log records stored in the BEA Tuxedo system TLOG. TLOG records are fixed in size and are stored in a single page. Domains log records vary in size; more than one page may be required to store the record. The Domains logging mechanism, DMTLOG, has the capability of storing variable-size log records.

When a TMS is the superior of a domain gateway group, the BEA Tuxedo TLOG is still required to coordinate the commitment.

Logging is performed by the GWADM administrative server. The request for a log write is made by the GWTDOMAIN process, but the actual log write is performed by the GWADM process.

You must create a log called DMTLOG for each domain gateway group. The DMTLOG files are defined in the DM_LOCAL section of the DMCONFIG file. To create a DMTLOG file, add an entry for the DMTLOGDEV parameter:

DMTLOGDEV=*string*

where *string* is the name of the log file. In addition, you cam set one or both of the two optional parameters:

- DMTLOGNAME=*identifier*
- DMTLOGSIZE=*numeric*

For more information, see DMCONFIG(5) in *BEA Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*.

Administrators also have the option of using the run-time administration utility (dmadmin) to create a DMTLOG. For more information, see dmadmin(1) in *BEA Tuxedo Command Reference*.

If a DMTLOG has not been created when a domain gateway group is booted, the gateway server automatically creates the log, based on information in the BDMCONFIG file.

Until a logging device is specified in the BDMCONFIG file, a domain gateway group cannot process requests in transaction mode and the domain gateway group cannot offer the TMS service.

To coordinate the commit protocol, domain gateways require the following two log records:

- *Ready record*—a ready record is a file created by a gateway acting as a leaf or intermediate machine in a transaction tree. It records information about the superior and subordinate remote domains involved in the transaction. A ready record indicates that all subordinates of the domain gateway group logging the record have been prepared.
- *Commit record*—a commit record documents that a transaction has been committed. A domain gateway creates a commit record as the coordinator of a particular transaction tree.

When a transaction has been committed on all machines, these logs for the transaction are removed.

When the OSI TP protocol is being used, two types of heuristic records are logged:

- *Log Heuristic record*—this record holds the details of a heuristic decision in the domain until the outcome of the relevant transaction is known by the superior.
- *Log Damage record*—this record is created to indicate one of two conditions for a transaction branch: (run with tmadmin(1)) a *heuristic hazard* (when the outcome of the transaction branch for a subordinate is unknown) or a *heuristic mix* (when the transaction subtree has a mixed outcome).

Heuristic log records persist until they are explicitly removed by the administrator. This persistence is required to provide the correct information during recovery after a crash, and to provide diagnostic information for administrators.

The administrator uses the `forgettran` command (run with `tmadmin(1)`) to remove heuristic records when they are no longer needed.

## Recovering Failed Transactions

When a domain gateway group is booted, the gateway server performs an automatic *warm-start* of the `DMTLOG`. The warm-start includes scanning the log to see if any transactions were not completed. If incomplete transactions are found, action is taken to complete them.

In OSI TP, any *blobs* stored in the `DMTLOG` with a transaction record are passed to the network access module, which uses the blobs to reconstruct its internal state and to recover any failed connections

In the case of heuristic decisions, if a domain gateway group is a subordinate of a local `TMS` and a heuristic decision has been indicated, the `TMS` generates a `TMS_STATUS` message to learn the final decision:

- If a gateway fails, then it cleans up after itself when it is restarted (this is called a *hot-start*). The gateway rolls back all undecided transactions in which it was involved.
- If a communication line failure occurs and the first phase of the commit has not been completed, the gateway rolls back the transactions associated with that connection.
- If OSI TP Domains is being used and a transaction fails in the second phase of the commit, recovery is managed by XAP-TP.