THE ENTERPRISE MIDDLEWARE SOLUTION

BEA TUXEDO

Domains Guide

BEA TUXEDO Release 6.5
Document Edition 6.5
February 1999

Copyright

Copyright © 1999 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

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

Trademarks or Service Marks

BEA, ObjectBroker, TOP END, TUXEDO, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Connect, BEA Manager, BEA MessageQ, Jolt and M3 are trademarks of BEA Systems, Inc.

All other company names may be trademarks of the respective companies with which they are associated.

BEA TUXEDO Domains Guide

Document Edition	**Date**	**Software Version**
6.5	February 1999	BEA TUXEDO Release 6.5

Contents

CHAPTER

1 Introducing Domains

What This Chapter Is About

This chapter introduces the following topics:

- How Domains Works
- Cooperating with Other Applications
- Extending the BEA TUXEDO Client-Server Model
- Requesting Local and Remote Services Using Request/Response Communication
- Enabling Conversational Services Using Conversational Communication
- Sending Data Between Clients and Servers with Typed Buffers
- Defining Transaction and Blocking Timeouts
- Ways to Establish Connections Between your Domains
- Failover and Failback in Domains

What Is a Domain?

As a business grows, application developers may need to organize the different segments of the business by sets of functionality that require administrative autonomy but allow sharing of services and data. Each functionality set defines an application that may span one or more computer machines, and that is administered independently from other applications. Such a functionally distinct application is referred to as a domain; in practice, the organization often uses the domain's functionality as part of its name so you find applications with names like the "accounting" domain or the "order entry" domain.

Interoperability Between Domains

The BEA TUXEDO system Domains feature provides a framework for interoperability between the domains of a business that continues the BEA TUXEDO enhanced client/server model. Interoperability means more than merely the capability of communicating from one domain to another. By transparently making access to services of a remote domain available to users of the local domain (or accepting local service requests from users of a remote domain), Domains, in effect, breaks down the walls between the business applications of an organization. Domains does this by defining a set of processes, called gateways, that manage the communication between domains. BEA TUXEDO system Domains is offered in an instance called TDOMAIN, which can be used to provide interoperability among two or more BEA TUXEDO applications.

Domains Terms

The following terms are used in our discussion of Domains.

Advertised
: A service is advertised when a service table entry exists for it in the BEA TUXEDO Bulletin Board. When a Domain gateway server is booted, it advertises all of the remote services that it is importing from remote domains in the Bulletin Board of the local domain (that is, the domain on which the gateway server is booted). After a remote service is advertised by the domain gateway server, it remains advertised until either an `unadvertise` command is issued or a MIB request removes the service.

Alternate Remote Domain
: A remote domain that is used when the primary remote domain is unavailable.

BEA TUXEDO Application
: A BEA TUXEDO application is bounded by the environment described in a single `TUXCONFIG` file. A BEA TUXEDO application can communicate with another BEA TUXEDO application through a domain gateway group.

BEA TUXEDO Domain
: A BEA TUXEDO application that is independently administered. A domain can be connected to other BEA TUXEDO domains through the Domains feature.

BEA TUXEDO Domains
: The BEA TUXEDO feature that allows one domain to communicate with another domain.

Bridge Failback
: The restoration of message traffic to a higher priority circuit. The Bridge process always tries to use the highest priority circuit defined for the machine. When that circuit becomes unavailable (due to circuit failure or other reasons), the Bridge transfers message traffic to a lower priority circuit, and periodically checks the availability of higher-priority circuits. When possible, the Bridge restores message traffic to a higher priority circuit.

Bridge Failover
: The transfer of message traffic to a lower priority circuit when a higher priority circuit fails.

Domain
: *See* BEA TUXEDO Domain.

Domain Gateway
: A process that relays requests and replies between one domain and others.

Gateway Group
: A collection of processes that provide communication services between one domain and other domains. It contains both a `GWADM` process and a gateway process, for example, a `GWTDOMAIN` process.

Domain Gateway Group
: *See* Gateway Group.

Failback
: *See* Bridge Failback and `TDOMAIN` Failback.

Failover
: *See* Bridge Failover and `TDOMAIN` Failover.

Incoming Connection
: A connection to the local gateway that is initiated by a remote domain gateway process on a remote domain.

Lazy Connection

A connection between a domain gateway and a remote domain that is not established until the remote domain receives a request for a remote service. A lazy connection keeps initialization overhead low for configurations involving many domains.

When a domain gateway server is booted, no connections are made to any remote domains. All remote services are assumed to be available and are advertised in the BEA TUXEDO Bulletin Board. When the first request for a service in a particular remote domain is made, the gateway server receives the request and tries to establish the connection. If the connection is made, the request flows to the remote domain and the connection remains active. If the connection fails, the client receives a failure message and the service remains advertised.

Link-level Failover in Domains

Use of an alternate network address for a particular remote domain.

Load Balancing

The practice of distributing service requests among all the servers in a given domain to achieve the most efficient handling of those requests.

When a service request is routed to a domain gateway, the gateway implements a load balancing algorithm and applies the data-dependent routing algorithm to find the proper remote domains to which the request should be routed. Load balancing and data-dependent routing algorithms are based on the remote service table entries and remote domain table entries in the gateway shared memory.

Local Domain

View of an application (that is, a subset of the application’s services) that is available to other domains. A local domain is always represented by a domain gateway group, and the terms are used interchangeably.

Local Gateway

The functionality of a specific gateway group within a local BEA TUXEDO application. Multiple gateways may be running within a single BEA TUXEDO application. Other BEA TUXEDO applications view these gateways as different remote domains.

Local Service

A service of a local domain that is available to remote domains through a domain gateway group.

Outgoing Connection
: A connection from the local gateway that has been generated as a result of one of the following: an automatic retry of the connection, an initial request to the remote domain, or a `dmadmin`(1) `connect` command issued by the administrator.

Primary Remote Domain
: The remote domain with the highest priority. It is used when it is available.

Remote Domain
: View of an application that is accessed through a domain gateway group.

Remote Gateway
: The functionality of a specific gateway group within a remote BEA TUXEDO application.

Remote Service
: A service of a remote domain that is made available to the local application through a domain gateway group.

Remote Service Fan-Out
: A configuration in which a remote service that is imported from multiple remote domains is advertised only once by a local domain gateway. A remote service can be fanned-out by a gateway server that imports the same service name from multiple remote domains. If the remote service name is imported from multiple remote domains, the local gateway advertises the service only once in the BEA TUXEDO Bulletin Board. Fan-Out is achieved through the gateway shared memory.

TDOMAIN Failback
: The restoration of message traffic to a primary remote domain. The domain gateway always tries to use the primary domain or the highest-level alternate remote domain defined for a service. When these domains become unavailable (due to circuit failure or other reasons), the gateway transfers message traffic to a lower-priority alternate remote domain, and periodically checks the availability of the primary remote domain and the highest-level alternate remote domain. When possible, the gateway restores message traffic to the primary remote domain or the highest-level remote domain.

TDOMAIN Failover
: The transfer of message traffic to an alternate remote domain when a primary remote domain fails.

Unadvertised
: A service is unadvertised when there is no service table entry for it in the BEA TUXEDO Bulletin Board.

How Domains Works

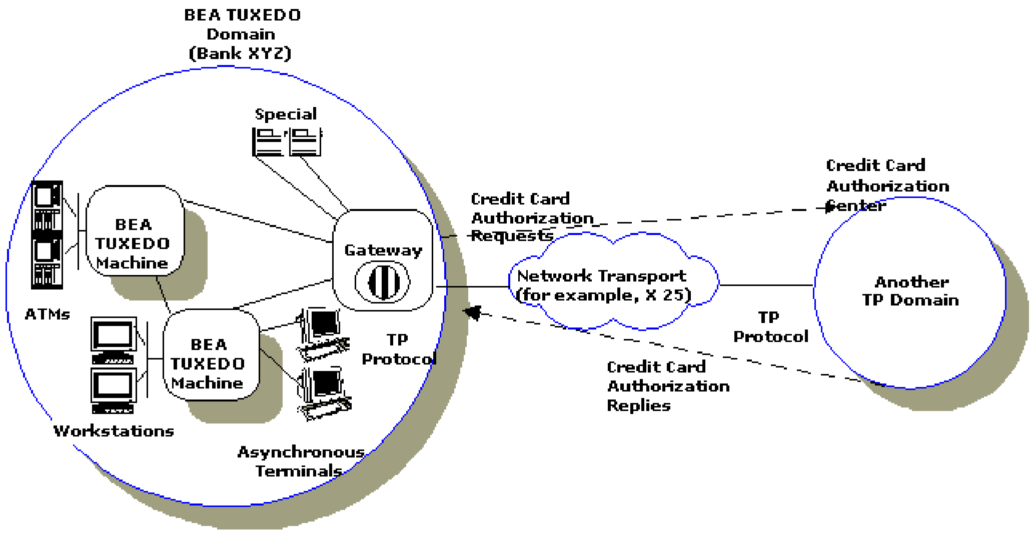
BEA TUXEDO provides an application programming framework that simplifies the development of open OLTP distributed applications by hiding the complexity associated with the distribution of application processing. The framework consists of the following:

- ♦ An extended client/server model that hides the heterogeneity of the different computers and application programs, as well as the location of the application programs.
- ♦ A centralized administration system that allows application administrators to control all cooperating machines as a single application.

It is not always possible, however, to structure applications as a single distributed application because of their nature, geographical location, confidentiality, and potential growth. Also, an enterprise may want to expand their business by cooperating with other organizations that provide OLTP services under the control of different transaction processing monitors, such as BEA’s TopEnd, Transarc’s Encina, IBM’s CICS, Bull’s TDS, Bull’s TP8, ICL’s TPMS, and so forth.

The BEA TUXEDO system provides the basic framework for domain interoperability. This framework defines a set of processes, called *gateways*, that manage the communication between domains. These gateways run within a BEA TUXEDO server group. This is illustrated in Figure 1-1, which shows a BEA TUXEDO application that requires services (in this case, credit card authorizations) from a remote domain.

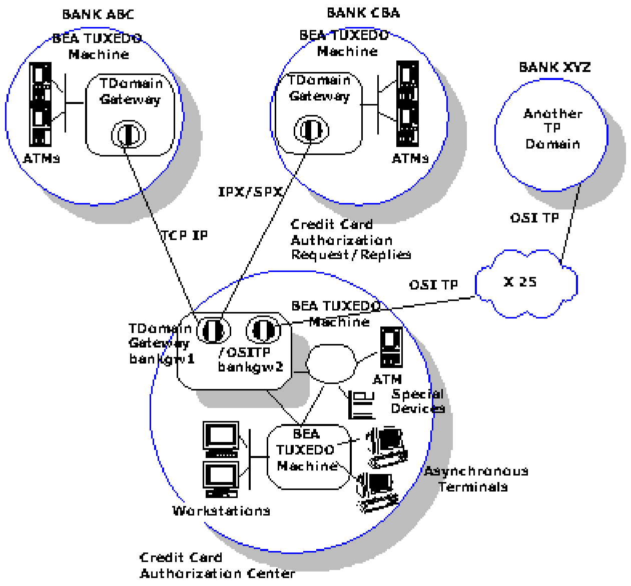
Figure 1-1 A Generalized View of Two Communicating Domains


BEA TUXEDO Domain (Bank XYZ)
Special
BEA TUXEDO Machine
Gateway
ATMs
BEA TUXEDO Machine
Workstations
TP Protocol
Asynchronous Terminals
Credit Card Authorization Requests
Network Transport (for example, X 25)
Credit Card Authorization Center
Another TP Domain
TP Protocol
Credit Card Authorization Replies

The application also accepts service requests (for example, balance inquiries) from remote domains. The gateway process provides bidirectional transaction control, and administrative tools that allow the configuration of the information required for interoperability of the local application with other domains. This configuration information includes the identification of a set of exported services, imported services, addressing, and access control. The BEA TUXEDO configuration of `TDOMAIN` is the subject of Chapter 2, “Configuring TDOMAIN.”

Figure 1-2 shows a more detailed view of the environment associated with domains.

Figure 1-2 A Look at Domains Environments



The example shows a credit card authorization center running under the control of the BEA TUXEDO system. The authorization center has two gateway groups: bankgw1 and bankgw2. `bankgw1` provides access to two remote BEA TUXEDO domain; `bankgw2` provides access to one remote domains (Bank XYZ) using the OSI TP protocol.

In this example, Bank ABC generates service requests to the credit card authorization center. These requests are received by a gateway running within group `bankgw1`. This gateway issues a service request on behalf of the remote domain to the credit card authorization service that is provided by a local server. The server processes the request and sends the reply to the gateway, and the gateway returns it to Bank ABC.

The credit card authorization center may also issue service requests, For example, the authorization center may send balance inquiries to Bank ABC. Domains makes this possible by behaving as a local server that advertises the services available on the other domains as if they were local services.

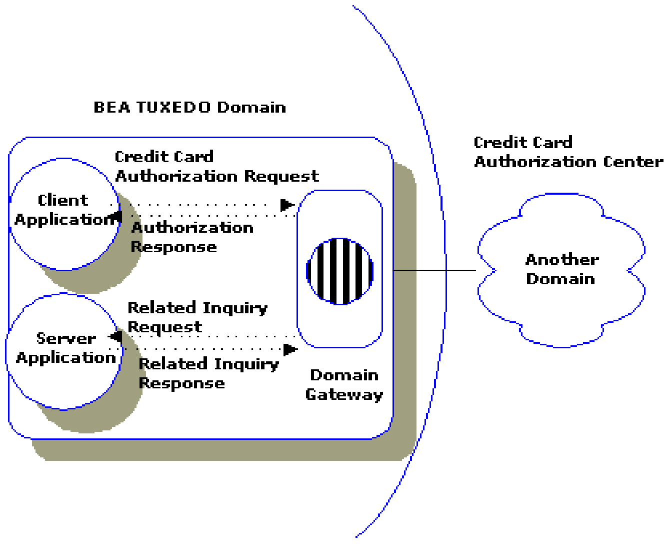
Domains provides the notion of a *local domain* to control incoming requests, and to provide a generic addressing framework for the application. Local domains help to provide partial views of an application, that is, a subset of the local services available to a set of remote domains. Each local domain is always represented by at most one gateway server group.

Cooperating with Other Applications

The main goal of the Domains feature is to allow the cooperation of BEA TUXEDO applications with dozens of applications running in other administrative domains. By meeting this goal, the BEA TUXEDO system provides a common framework for controlling very large applications that may include domains running other transaction processing systems.

Domains maintains the BEA TUXEDO client/server model by making access to services on the remote TP System (or accepting service requests from a remote TP System) transparent to both the application programmer land the end-user (see Figure 1-3). Application programmers use the ATMI interface to access the services provided by remote domains, or to define services that can be executed by a remote domain. The following section discusses the implications of these interdomain communications on the different ATMI primitives.

Figure 1-3 Two-way Communication through a Gateway



Domains provides an asynchronous multithreaded gateway able to process both requests going out to remote domains and requests coming in from remote domains. Any request can be processed within a transaction.

Security in Domains

The following four types of security are provided in Domains. These methods can be used separately or in combination with each other:

- Service export—The administrator can restrict which services are available to remote domains by defining them in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file.
- Access control—Access to local services within a local domain can be restricted so that only certain remote domains can execute these services.

- Domain passwords—`TDOMAIN` provides a way to instruct domain gateways to authenticate connections to remote domains using passwords.
- User identity mapping—SNA Connect provides a mechanism whereby user identities within a domain can be mapped to and from RACF usernames on LU6.2.

ATMI Programming Framework

The BEA TUXEDO system provides an application programming framework that simplifies the development of open OLTP applications, and hides the complexity associated with the distribution of the application. The framework is an extended client/server model that offers three programming paradigms:

- Request/response communication
- Connection-oriented conversations
- Enqueue/dequeue

These paradigms are provided through the Application Transaction Management Interface (ATMI), and are designed to meet the requirements of open OLTP systems. ATMI helps open OLTP application developers structure their code for portability, location transparency, performance, modular growth, and reliability.

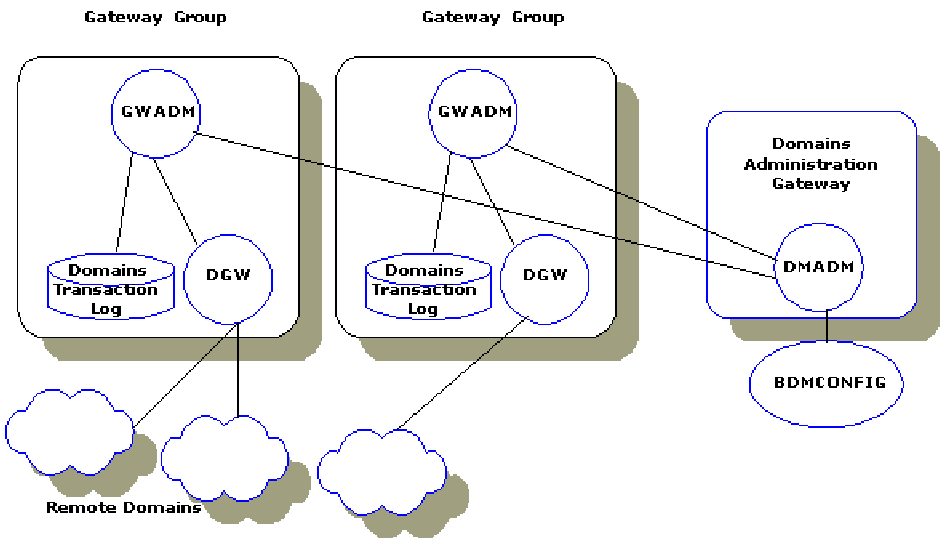
Extending the BEA TUXEDO Client-Server Model

Domains extends the BEA TUXEDO client/server model to provide transaction interoperability across TP domains. This extension preserves the model and the ATMI interface by making access to services on the remote domain (or accepting service requests from a remote domain) transparent to both the application programmer and the end user. Domains makes this possible via a highly asynchronous multitasking gateway that processes outgoing and incoming service requests to or from remote domains.

What Is a Domain Gateway?

A Domain gateway is a BEA TUXEDO-supplied server that enables access to and from remote domains (DGW in Figure 1-4). In addition, Domains provides a *gateway administrative server* (`GWADM` in Figure 1-4) that enables run-time administration of the Domain gateway group and a *Domain administrative server* (`DMADM` in Figure 1-4) that enables run-time administration of the BEA TUXEDO application-wide Domain configuration information. The application administrator enables remote domain access by specifying one or more gateway groups (each containing a domain gateway and a gateway administration server) and a domain administrative group in the `GROUPS` section of the `TUXCONFIG` file, and by adding entries for the gateway and the administrative servers in the `SERVERS` section. Configuration is covered in Chapter 3, “Running Domains.”

Figure 1-4 Domains Configuration



Domain gateways support the following functionality:

- ATMI request/response model—Application programs using the BEA TUXEDO system can request services from applications running in another domain. Also, remote applications can request services from local servers. No changes are required to the application programs.

- ATMI conversational model—Application programs can establish conversations with programs running in another domain. Remote domains can establish conversations with conversational services offered by local servers.
- Transaction management—Application programs can interoperate with other domains within a transaction. The gateway coordinates the commitment or rollback of transactions running across domains.
- Administration—Gateways can be booted or shut down exactly as any other BEA TUXEDO server. Run-time administration is provided through an administrative server, `DMADM`. Using `DMADM`, application administrators can make changes to the domain configuration file, and tune the performance of a gateway group. (The `DMADM` administrative server should be booted before gateway groups.)
- Typed buffer support—Gateways can perform encoding and decoding operations for all typed buffers defined by the application.
- Multidomain interaction—Gateways can communicate with multiple domains of the same type.
- Multinetwork support—Gateways can communicate with other domains via a variety of network protocols, such as `TCP/IP`, `IPX/SPX`, and others. However, a gateway is limited by the capabilities of the specific networking library linked with the gateway. In other words, a gateway typically supports a single type of network protocol.

The functions of the BEA TUXEDO client/server model listed above are realized through the following capabilities of Domain gateways:

- Request/Response communication between local and remote services
- Conversational communication between local and remote services
- Typed buffers to send data between clients and servers
- Mechanisms for transaction timeouts and blocking timeouts

Requesting Local and Remote Services Using Request/Response Communication

Domain gateways provide support for the request/response model of communication defined by the ATMI interface. A BEA TUXEDO application can request remote services exactly as if they were offered within the application. Also, remote domains can request services offered within the local application exactly as if they were offered within the remote application.

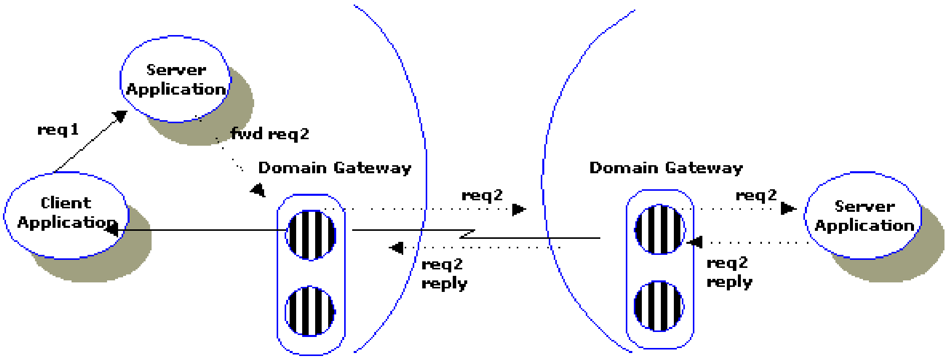
Support for ATMI Primitives

The following BEA TUXEDO ATMI primitives are logically limited to use within a single application and are not supported across domains:

- `tpinit(3c)`/`tpterm(3c)`—BEA TUXEDO applications do not attach to the environment of a remote domain; they use Domain gateways to access the remote domain. Therefore, an extra `tpinit()`/`tpterm()` is not needed for remote applications.
- `tpadvertise`(3c) and `tpunadvertise`(3c)—cannot be used across domains. 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 primitives.
- Event posting (`tppost`(3c)) and notification of events (`tpsubscribe`(3c))—are not available across domains.

Support for `tpforward`(3c) is provided to preserve the portability of applications using this primitive. Forwarded requests are interpreted by Domain gateways as simple service requests. This is illustrated in Figure 1-5 in the simple case of a service using `tpforward` to send a request to a remote service.

Figure 1-5 Using tpforward to Send a Request to a Remote Service



Enabling Conversational Services Using Conversational Communication

The ATMI interface is a connection-oriented interface that enables clients to establish and maintain a conversation with services configured and coded with the conversational paradigm (instead of the request/response paradigm).

BEA TUXEDO applications can 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 provide the same semantics for returns (that is, `tpreturn` with `TPSUCCESS` or `TPFAIL`) and disconnects as defined for BEA TUXEDO conversational services.

Note that the ATMI connection-oriented primitives provide half-duplex conversations and that `tpforward`(3c) is not allowed within a conversational service.

Application administrators indicate whether a remote service is a conversational service by specifying `CONV=Y` in the `DM_REMOTE_SERVICES` section of the `DMCONFIG` file.

Sending Data Between Clients and Servers with Typed Buffers

In the BEA TUXEDO system, applications use typed buffers to send data between clients and servers. The typed buffer mechanism allows application programmers to transfer data without knowing the data representation scheme 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, BEA TUXEDO machines, or remote domains with different machine representations. Hence, gateways must be linked with the typed buffer switch defined by the application administrator so the gateways know how, for example, to decode the data sent with the service request.

Domain gateways always try to decode any service request that is received encoded for two reasons:

- The gateway may have to apply data-dependent routing to select the remote domain that must execute the service request. (Data-dependent routing criteria for remote domains is defined in the Domains configuration file.)
- Individual domains may use different data formats as required by the networking protocols they implement or use.

Following the OSI terminology, there is a difference 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 (that is, its typed buffer operations) required to map the data structure to a particular transfer syntax (for example, `XDR`).

The BEA TUXEDO system provides a set of predefined buffer types (`STRING`, `CARRAY`, `FML`, `VIEW, X_C_TYPE, X_OCTET`, and `X_COMMON`) and the encoding rules required to map these types to the `XDR` transfer syntax.

Application programmers can supply their own buffer types by adding an instance to the `tm_typesw` array in `$TUXDIR/lib/tmtypesw.c` (see `tuxtypes`(5) and `typesw`(5)), and by supplying the routines for the new type (see `buffer`(3c)).

Defining 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 a transaction, which may involve several service requests. The timeout value is defined when the transaction is started (with `tpbegin`(3c)). The blocking timeout is used to define the duration of service requests, that is, how long the application is willing to wait for a reply to a service request. The timeout value is a global value defined in the `BLOCKTIME` field of the `RESOURCES` section of the `TUXCONFIG` file. The transaction timeout always overrides a blocking timeout value.

The BEA TUXEDO transaction timeout mechanism is used unchanged in the Domains framework. This is possible because Domain gateways implement the TMS functionality and therefore are required to handle the `TMS_TIMEOUT` messages generated by the `BBL`.

Domain gateways, however, cannot use the BEA TUXEDO blocking timeout mechanism. This is because the blocking timeout mechanism uses information stored in the registry slot assigned to each client or server. Information in the registry slot is used by the local `BBL` to detect requesters that have been blocked for a time greater than `BLOCKTIME`. Domain gateways, however, are multitasking servers that may process several service requests at a time. This means that the Domain gateways cannot successfully use the registry slot mechanism.

When a Domain blocking timeout condition arises, the Domain gateway sends an error/failure reply message to the requester, and then it *cleans* any context associated with the service request.

Ways to Establish Connections Between your Domains

You can specify *the* conditions under which a local domain gateway tries to establish a connection to a remote domain. You specify these conditions by assigning a value to the `CONNECTION_POLICY` parameter in the domains configuration file (`dmconfig`). You can select any of the following connection policies:

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

Note: For more detailed information on configuring connection policies, see Chapter 2, “Configuring TDOMAIN.”

Failover and Failback in Domains

Domains-level failover provides fail over to alternate remote domains when a failure is detected with a primary remote domain. It also provides failback to the primary remote domain when it is restored. Domains-level failover/failback defines a remote domain as being available when a network connection can be established to the remote domain, and unavailable when a network connection cannot be established to the remote domain.

Note: For more detailed information on failover and failback, see Chapter 2, “Configuring TDOMAIN.”

2 Configuring TDOMAIN

What This Chapter Is About

The chapter describes how Domains is defined in a configuration file. The chapter covers the following topics:

- Overview of a Domains Configuration
- Loading and Unloading the Domain Configuration File
- Establishing Security in Domains
- Setting Up Connections Between Your Domains
- Configuring Failover and Failback

Overview of a Domains Configuration

Domains integrates with an existing BEA TUXEDO application by adding new group and server information to the configuration file. In a manner similar to the `UBBCONFIG` and `TUXCONFIG` files of BEA TUXEDO, the configuration is defined in a text file known as the `DMCONFIG` file, and then loaded into a binary file called `BDMCONFIG`. Configuration can be done dynamically by using subcommands of the `dmadmin` command.

The architecture of the `DMCONFIG` file follows closely the architecture of the `UBBCONFIG` file. There is an ASCII file (the `DMCONFIG` file) that is compiled into a binary configuration file, `BDMCONFIG`. There is one `BDMCONFIG` file per BEA

TUXEDO application wishing to add the Domains functionality. The BDMCONFIG file can be contained within the same VTOC (set of BEA TUXEDO devices) as an existing TUXCONFIG file without requiring application shutdown.

The BEA TUXEDO Configuration File

You must add the following entries to the *SERVERS section of the TUXCONFIG file to enable Domains:

- DMADM (the Domains administrative server)
- GWADM (the Gateway administrative server)
- GWTDOMAIN (the gateway for TDOMAINS connectivity

  Note: A different executable is used here for other gateway types. Refer to the BEA Connect documentation for additional information.

Each GWADM and GWTDOMAIN pair must run in the same BEA TUXEDO group. You can add pairs of GWADM/GWTDOMAIN, as long as each pair is in a different BEA TUXEDO group. You must define a single instance of DMADM, which can be in any group on a machine containing the highest version of BEA TUXEDO for the domain.

The Domains Configuration File

All domain configuration information is stored in a binary file, the BDMCONFIG file.

The DMCONFIG file is created and edited with any text editor by the application administrator. The compiled BDMCONFIG can be updated while the system is running by using the dmadmin(1) command.

There must be one BDMCONFIG file per BEA TUXEDO application wishing to add the Domains functionality.

System access to the BDMCONFIG file is provided through the Domains administrative server, DMADM(5). When a gateway group is booted, the gateway administrative server, GWADM(5), requests from the DMADM server a copy of the configuration required by that group. The GWADM server and the DMADM server also ensure that run-time changes to the configuration are reflected in the corresponding Domains gateway groups.

DMCONFIG Sections

The `dmconfig`(5) reference page spells out the complete detail of the `DMCONFIG` file; this section touches on some of the highlights.

The `DMCONFIG` file is organized into the following sections:

`*DM_LOCAL_DOMAINS`
: This section describes the environment required for a particular domain gateway group. It assigns a logical application name, `LDOM`, to the subset of local services that can be accessed by remote domains. Multiple entries in this section are used to define multiple gateway groups within a single BEA TUXEDO application. Each gateway group can provide access to domains of different types.

`*DM_REMOTE_DOMAINS`
: This section identifies the remote domains that can be accessed by clients and servers of this Domains application.

`*DM_LOCAL_SERVICES`
: This section describes the local services provided by a `LDOM` in the `*DM_LOCAL_DOMAINS` section. Specification of services can also be used to restrict access to local services from remote domains; that is, only services specified are available to the remote domain.

`*DM_REMOTE_SERVICES`
: This section describes the services provided by remote domains. It also names the local gateway group (through the `LDOM` parameter) that provides access to the remote services.

`*DM_ROUTING`
: This section specifies criteria for data dependent routing used by gateways to route service requests to specific remote domains.

`*DM_ACCESS_CONTROL`
: This section can contain a named list, the Access Control List, which carries the names, `RDOM`s, of remote domains permitted to request local services. A parameter in the `*DM_LOCAL_SERVICES` section, `ACL =` *`list_name`*, can be used to restrict access to a particular local service to the listed set of remote domains.

*DM_<*dmtype*>

This section defines the specific parameters required for a particular Domains instantiation. At the present time, dmtype can be OSITP, SNAX, or TDOMAIN. This guide focuses only on TDOMAIN. Consult BEA Connect documentation for information about OSITP and SNAX. Each domain type must be specified in a section of its own.

In a DM_TDOMAIN section entries associated with a remote domain can be specified more than once, with different network addresses, to implement the *mirrored* gateway facility. (See dmconfig(5) for a description and an example of a mirrored gateway.)

The architecture of the DMCONFIG file is compatible with the architecture of the UBBCONFIG file.

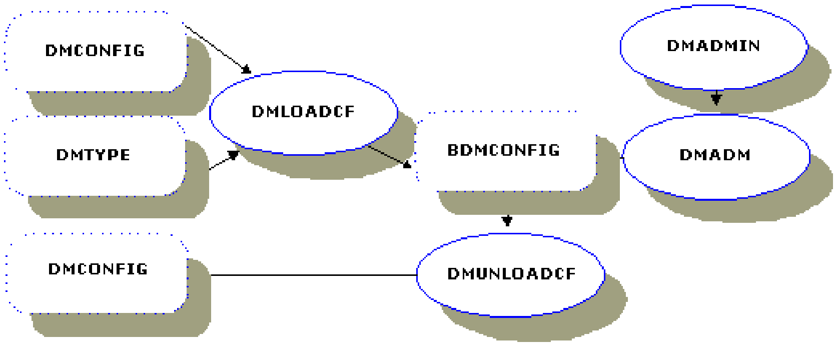
In /TDOMAIN, the binary version of the DMCONFIG file contains an extra section: *DM_PASSWORDS. The *DM_PASSWORDS section contains the passwords used by local and remote domains during the authentication procedure. The passwords are added by the Domains administrator using the dmadmin(1) command.

Loading and Unloading the Domain Configuration File

Using dmloadcf(1)

The command dmloadcf(1) parses the syntax of the DMCONFIG file, and loads the domain configuration into a BDMCONFIG binary file. The command uses the environment variable BDMCONFIG to point to the BEA TUXEDO file system where the configuration should be stored. The BDMCONFIG file can be stored on the same device as the TUXCONFIG file.

Figure 2-1 Relationships Between Configuration Commands and Files



The `dmloadcf`(1) command, through its `-c` option, also provides an estimate of the IPC resources needed for each local domain specified in the configuration.

As shown in Figure 2-1, the `dmloadcf` command uses the `$TUXDIR/udataobj/DMTYPE` file. It checks the `DMTYPE` file to verify that the domain types specified in the configuration file are valid. Each Domains instantiation has a domain `type`. The `type` is used as a tag in the file `$TUXDIR/udataobj/DMTYPE.` Each line in this file has the format:

`dmtype:access_module_lib:comm_libs:tm_typesw_lib:gw_typesw_lib`

As supplied for the initial release, the file has the following entry for /`TDOMAIN`:

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

Using dmunloadcf(1)

The `dmunloadcf`(1) command is required to unload a binary version of a Domains configuration into an ASCII file.

Establishing Security in Domains

The native BEA TUXEDO system provides authentication of clients, servers, and administrative programs. The default security mechanism provided with BEA TUXEDO is an authentication scheme that uses one password per BEA TUXEDO application. Clients are required to present this password to be authenticated and allowed to join the application. Servers are authenticated to be running as the user identified as the application administrator. Access Control Lists can be set up whereby access to services is controlled by means of lists that are automatically checked when the service is requested. The BEA TUXEDO system also provides an optional mechanism based on a server that performs per-user authentication. There are also hooks for connecting third-party vendors' authentication modules.

The BEA TUXEDO application password is administered with the following programs: `tmloadcf`(1) 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 by the `tagent` mechanism; and the password can be dynamically updated with the `tmadmin` command.

Security and the BEA TUXEDO System Domains Framework

Because the very concept of Domains implies the possibility of domains existing under diverse ownership, the native BEA TUXEDO application password scheme does not provide sufficient security. It has, therefore, been part of the Domains design to enhance security through the following constructs:

Local Domains

The local domain concept provides a first level of security. At this level a partial view of the application (that is, a subset of services) is made available to remote domains. This partial view is defined by including the corresponding services in the `*DM_LOCAL_SERVICES` section of the `DMCONFIG` file.

Access Control

The second level of security is provided by access control. As defined in the `DMCONFIG` file, access control provides another level of security in which access to local services within a local domain can be restricted in such a way that only selected remote domains can execute these services.

Domain Passwords

The partial view in a local domain and the access control concepts only serve to throttle down access to a Domains application. Therefore, additional authentication techniques are required to assure the proper *identity* of each remote domain. Passwords are one of these authentication techniques, and Domains provides a facility for the definition of passwords on a per remote domain basis.

Domain Passwords

This feature is specified as a part of the Domains generic framework but its implementation is dependent upon each domain type. The /TDOMAIN instantiation uses this feature to enforce security between BEA TUXEDO applications.

Domain passwords are implemented in a special section (called `*DM_PASSWORDS`) of the `BDMCONFIG` file. Except for a declaration of the intent to invoke password security, the password specification is not part of the /TDOMAIN configuration process; the `*DM_PASSWORDS` section that ends up in the `BDMCONFIG` file is not part of the ASCII `DMCONFIG` file; the `*DM_PASSWORDS` section is defined and updated by the Domains administrator through the `dmadmin`(1) command. Each entry contains the password used by a remote domain to access a particular local domain and the password required by the local domain, in turn, to access the remote domain.

How Domain Passwords Work

Domains gateways can be made to authenticate incoming connections requested by remote domains. The authentication mechanism is optional and compatible with the BEA TUXEDO mechanism specified in the `TUXCONFIG` file.

Application administrators can define when security should be enforced for incoming connections from remote domains. The `SECURITY` keyword of the

*DM_LOCAL_DOMAINS section specifies the level of security allowed by a particular local domain. There are three basic security levels:

No Security
: Incoming connections from remote domains will not be authenticated. This level is specified with the NONE option.

Application Password
: Incoming connections from remote domains should be authenticated using the application password defined in the TUXCONFIG file. This level is specified with the APP_PW option.

Remote Domain Password
: Incoming connections from remote domains should be authenticated using the passwords defined in the *DM_PASSWORDS section of the BDMCONFIG file. This level is specified with the DM_PW option.

Domains password security to some extent reflects the SECURITY option specified in the *RESOURCES section of the TUXCONFIG file; that is, if APP_PW is defined in the TUXCONFIG file, then local domains cannot be defined with the NONE option; they must be defined with either the APP_PW option or the DM_PW option.

Even if the SECURITY option is not set in TUXCONFIG, the Domains configuration can still require the Domains gateways to enforce security at the APP_PW or DM_PW level.

If the DM_PW option is selected, then each remote domain must have a password defined in the *DM_PASSWORDS section of the BDMCONFIG file; in other words, incoming connections from remote domains without a password are rejected by Domains gateways.

The *DM_PASSWORDS table contains an entry for each remote domain as follows:

LDOM
: The name of the local domain providing access to the remote domain.

RDOM
: The name of the remote domain

LPWD
: The password used by a local domain to authenticate with the remote domain.

RPWD
: The password used by the remote domain to authenticate with the local domain.

Note: Passwords are stored encrypted.

Examples of Security Between Domains

The SECURITY parameter in the *DM_LOCAL_DOMAINS section of the DMCONFIG file specifies the security type of a local domain.

The security protocol is used only when a remote domain first connects to a given local domain. If the security types are incompatible between the domains or if the passwords do not match, the connection establishment will fail.

If SECURITY is set to NONE for a local domain, it implies that any connections coming in will not be authenticated. Bear in mind that by setting SECURITY to NONE, a local domain can still connect to remote domains that have SECURITY set to DM_PW, but before the connection is established with a domain that has SECURITY of DM_PW, you would have to define passwords on both sides by running dmadmin(1) or by using the /AdminAPI.

If the SECURITY in UBBCONFIG is set to APP_PW or higher, then SECURITY in DMCONFIG can be NONE, APP_PW, or DM_PW. Since you can define multiple views of a domain in one DMCONFIG file, each of those views can have a different type of security mechanism.

Note that if SECURITY is set to APP_PW in DMCONFIG, then SECURITY in UBBCONFIG must be set to APP_PW or higher.

Listing 2-1 through Listing 2-3 show some common variations.

Listing 2-1 /T Security NONE, Domains Security DM_PW

On the initiator side, the pertinent attributes in UBBCONFIG and DMCONFIG look like this:

```
UBBCONFIG
 SECURITY=NONE

DMCONFIG
 *DM_LOCAL_DOMAINS
 DOM1
 DOMAINID=DOM1
  SECURITY=DM_PW

 *DM_REMOTE_DOMAINS
 DOM2 DOMAINID=“DOM2”
```

On the responder side, the pertinent attributes in UBBCONFIG and DMCONFIG look like this:

```
UBBCONFIG
 SECURITY=NONE

DMCONFIG
 *DM_LOCAL_DOMAINS
 DOM2
  DOMAINID=DOM2
  SECURITY=DM_PW

 *DM_REMOTE_DOMAINS
 DOM1 DOMAINID=“DOM1”
```

After the `TUXCONFIG` and `BDMCONFIG` files have been made, boot the applications on `DOM1` and `DOM2`.

```
On DOM1:
     dmadmin
      passwd DOM1 DOM2
      Enter Local Domain Password:foo1
      Reenter Local Domain Password:foo1
      Enter Remote Domain Password:foo2
       Reenter Remote Domain Password:foo2

On DOM2:
     dmadmin
      passwd DOM2 DOM1
      Enter Local Domain Password:foo2
      Reenter Local Domain Password:foo2
      Enter Remote Domain Password:foo1
      Reenter Remote Domain Password:foo1
```

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

Listing 2-2 /T Security NONE, Domains Security NONE

```
DOM1: SECURITY in UBBCONFIG set to NONE
       SECURITY in DMCONFIG set to NONE

DOM2: SECURITY in UBBCONFIG set to NONE
       SECURITY in DMCONFIG set to DM_PW
```

In this example, DOM1 is not enforcing any security but DOM2 is enforcing DM_PW security.

On the initiator side, the pertinent attributes in UBBCONFIG and DMCONFIG look like this:

```
UBBCONFIG
 SECURITY=NONE

DMCONFIG
 *DM_LOCAL_DOMAINS
DOM1
   DOMAINID=DOM1
   SECURITY=NONE

 *DM_REMOTE_DOMAINS
DOM2 DOMAINID=“DOM2”
```

On the responder side, the pertinent attributes in UBBCONFIG and DMCONFIG look like this:

```
UBBCONFIG
 SECURITY=NONE

DMCONFIG
 *DM_LOCAL_DOMAINS
DOM2
   DOMAINID=DOM2
   SECURITY=DM_PW

 *DM_REMOTE_DOMAINS
DOM1 DOMAINID=“DOM1”
```

After the `TUXCONFIG` and `BDMCONFIG` files have been made, boot the applications on `DOM1` and `DOM2`.

```
On DOM1:
     dmadmin
      passwd DOM1 DOM2
       Enter Local Domain Password:foo1
       Reenter Local Domain Password:foo1
       Enter Remote Domain Password:foo2
       Reenter Remote Domain Password:foo2

On DOM2:
     dmadmin
     passwd DOM2 DOM1
     Enter Local Domain Password:foo2
     Reenter Local Domain Password:foo2
     Enter Remote Domain Password:foo1
      Reenter Remote Domain Password:foo1
```

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

Listing 2-3 /T Security APP_PW, Domains Security APP_PW

```
DOM1: SECURITY in UBBCONFIG set to APP_PW
       SECURITY in DMCONFIG set to APP_PW

DOM2: SECURITY in UBBCONFIG set to APP_PW
       SECURITY in DMCONFIG set to APP_PW
```

In this example, both DOM1 and DOM2 are enforcing APP_PW security.

`On the initiator side, the pertinent attributes in UBBCONFIG and DMCONFIG look like this:`

```
UBBCONFIG
 SECURITY=APP_PW

DMCONFIG
 *DM_LOCAL_DOMAINS
DOM1
  DOMAINID=DOM1
  SECURITY=APP_PW

 *DM_REMOTE_DOMAINS
DOM2 DOMAINID="DOM2"
```

On the responder side, the pertinent attributes in UBBCONFIG and DMCONFIG look like this:

```
UBBCONFIG
 SECURITY=APP_PW

DMCONFIG
 *DM_LOCAL_DOMAINS
DOM2
  DOMAINID=DOM2
  SECURITY=APP_PW

 *DM_REMOTE_DOMAINS
DOM1 DOMAINID="DOM1"
```

After the `TUXCONFIG` and `BDMCONFIG` files have been made, boot the applications on `DOM1` and `DOM2`.

Setting Up Connections Between Your Domains

There are three policies that you can use to establish connections between domains: `ON_STARTUP`, `ON_DEMAND`, and `INCOMING_ONLY`. The following section describes each of these policies and how to configure them.

Configuring the Policy for Connections Between Domains

You can specify the conditions under which a local domain gateway tries to establish a connection to a remote domain. You specify these conditions by assigning a value to the `CONNECTION_POLICY` parameter in the domains configuration file. You can select any of the following connection policies:

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

For connection policies of `ON_STARTUP` and `INCOMING_ONLY`, Dynamic Status is invoked. Dynamic Status is a capability of BEA TUXEDO Domains that provides a system-determined status of remote services.

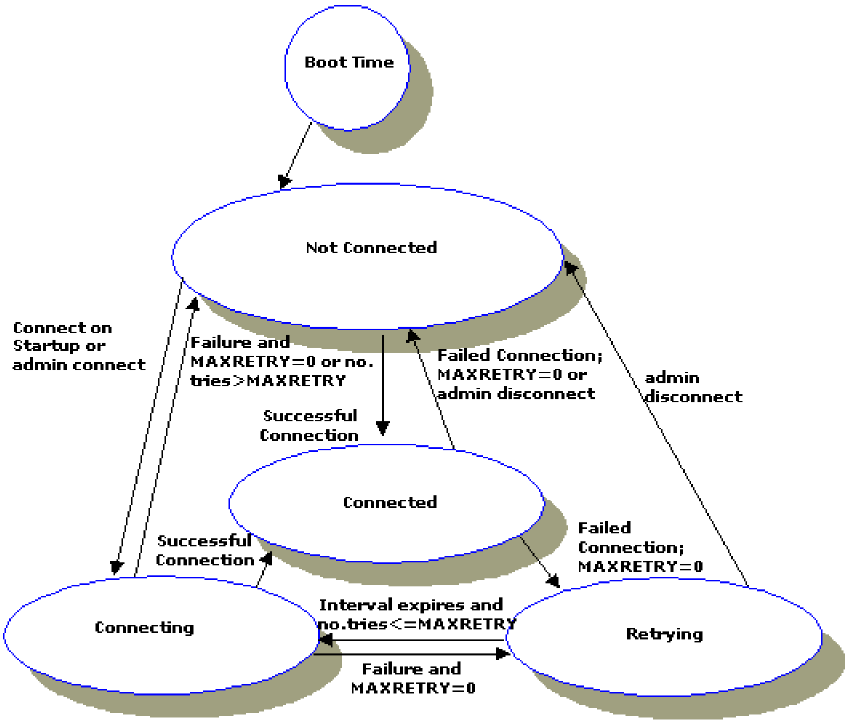
Connect at Boot Time (ON_STARTUP Policy)

A policy of `ON_STARTUP` means that a domain gateway attempts to establish a connection with its remote domains at gateway server initialization time. By default, this connection policy retries failed connections every 60 seconds, but you can specify a different value for this interval (using the `RETRY_INTERVAL` parameter).

```
CONNECTION_POLICY=ON_STARTUP
```

The following diagram shows how connections are attempted and made by a gateway for which the connection policy is `ON_STARTUP` (This policy invokes Dynamic Status.)

Figure 2-2 Connections Made with an ON_STARTUP Policy


Boot Time
Not Connected
Connect on Startup or admin connect
Failure and MAXRETRY=0 or no. tries>MAXRETRY
Successful Connection
Failed Connection; MAXRETRY=0 or admin disconnect
admin disconnect
Connected
Successful Connection
Failed Connection; MAXRETRY=0
Connecting
Interval expires and no.tries<=MAXRETRY
Retrying
Failure and MAXRETRY=0

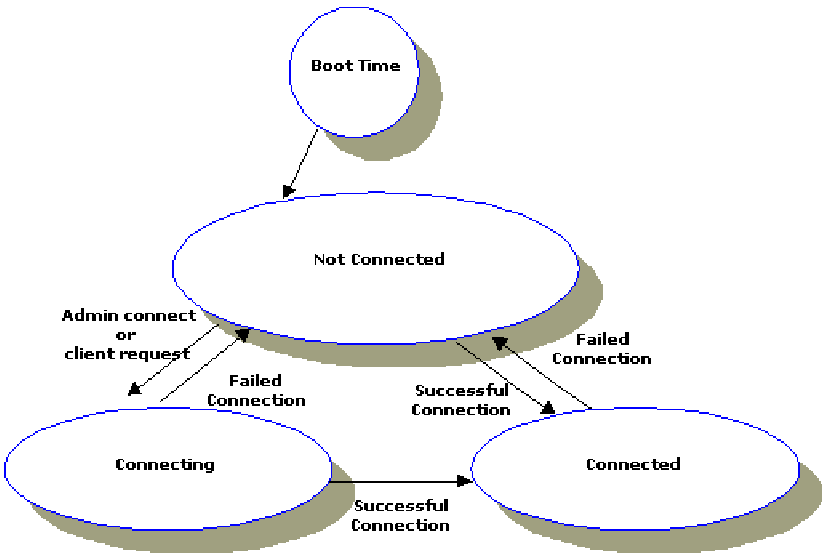
Connect When a Client Program Requests a Remote Service (ON_DEMAND Policy)

A connection policy of `ON_DEMAND` means that a connection is attempted only when requested by either a client request to a remote service or an administrative “connect” command. The default setting for `CONNECTION_POLICY` is `ON_DEMAND`. Connection retry processing is not allowed when the connection policy is `ON_DEMAND`.

```
CONNECTION_POLICY=ON_DEMAND
```

The following diagram shows how connections are attempted and made by a gateway for which the connection policy is `ON_DEMAND`. (This policy does not invoke Dynamic Status.)

Figure 2-3 Connections Made with an ON_DEMAND Policy (when a client requests a remote service)



Accept Incoming Connections but Do Not Initiate a Connection (INCOMING_ONLY Policy)

A connection policy of `INCOMING_ONLY` means that a domain gateway does not try to establish a connection to remote domains upon starting. Connection retry processing is not allowed when the connection policy is `INCOMING_ONLY`.

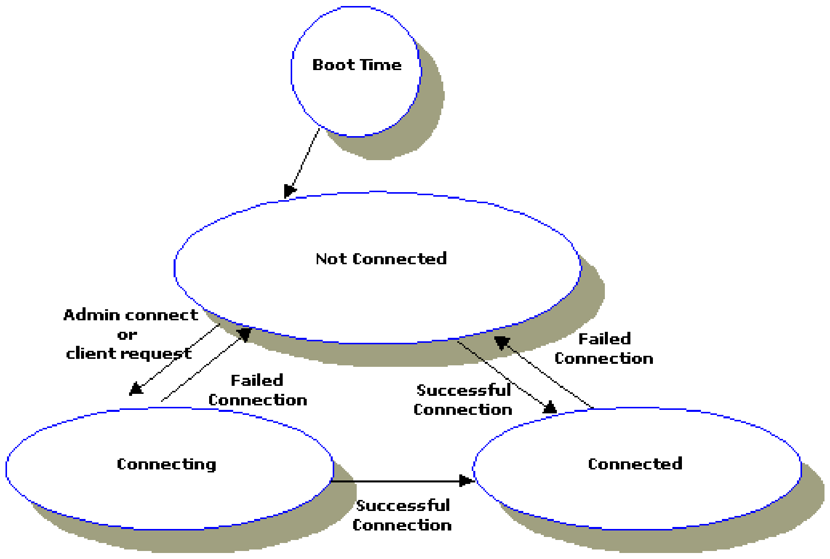
To use this policy, enter the following line in your Domains configuration file.

```
CONNECTION_POLICY=INCOMING_ONLY
```

You can also establish the connection manually using the `dmadmin connect` command.

The following diagram shows how connections are attempted and made by a gateway for which the connection policy is `INCOMING_ONLY`. (This policy invokes Dynamic Status.)

Figure 2-4 Connections Made with an INCOMING_ONLY Policy (accept incoming connections)



Configuring the Connection Retry Interval (for ON_STARTUP Only)

The Connection Retry Interval enables failed attempts at connections to remote domains to be retried automatically if the connection policy is `ON_STARTUP`. As an 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 to establish a connection again. You can specify the retry interval by setting the `RETRY_INTERVAL` parameter in the Domains configuration file:

```
RETRY_INTERVAL=[number_of_seconds]
```

If the connection policy is `ON_STARTUP` and you do not specify a value for the `RETRY_INTERVAL` parameter, a default of 60 is used.

To configure the connection retry interval, you set the following parameter in the `DM_LOCAL_DOMAINS` section of the domains configuration file:

- `RETRY_INTERVAL=[`*`seconds`*`]`—To specify the number of seconds after which an attempt is made to connect or reconnect to a remote domain. Depending on the result of the connection attempt, the remote services are marked either available or unavailable. (The minimum value is 0; the maximum value is 2147483647.)

The `RETRY_INTERVAL` parameter is valid only when the connection policy is `ON_STARTUP`. For the other connection policies (`ON_DEMAND` and `INCOMING_ONLY`), retry processing is disabled.

Configuring 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 `MAXLONG`.)

- If you set `MAXRETRY=0`, the automatic connection retry processing is turned off. The server does not attempt to connect to the remote gateways 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`, retry processing is repeated indefinitely 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`), retry processing is disabled. The `RETRY_INTERVAL` is rounded up to a multiple of `SCANUNIT`.

Table 2-1 Examples of Seting the MAXRETRY and RETRY_INTERVAL Parameters

If You Set	Then

`CONNECTION_POLICY=ON_STARTUP`
`RETRY_INTERVAL=30`
`MAXRETRY=3`
	The gateway attempts to establish a connection every 30 seconds for 3 times before quitting.

`CONNECTION_POLICY=ON_STARTUP`
`MAXRETRY=0`
	The gateway attempts to establish a connection at initialization with no retries.

`CONNECTION_POLICY=ON_STARTUP`
`RETRY_INTERVAL=30`
	The gateway attempts to establish a connection every 30 seconds until a connection is established.

The Dynamic Status Capability

The gateway process (GWTDOMAIN) advertises those services that are imported from one or more remote domains in the Bulletin Board. These services typically remain advertised regardless of whether the remote service is reachable. The Dynamic Status behavior provides intelligence to detect the availability of remote services and to react accordingly.

Dynamic Status provides a system-determined status of remote services. When Dynamic Status is in effect, the status of a remote service depends on the status of the network connection between the local and remote gateways. The remote services are considered available whenever a connection to the remote domain is available. When a network connection to a remote domain is not available, services in that domain are considered unavailable. This capability of the TDOMAIN Gateways is known as the Dynamic Status feature. This policy is invoked when the connection policy is ON_STARTUP or INCOMING_ONLY.

For each service, the gateway not only keeps track of the remote domains from which the service is imported, but also which of those remote domains are available (that is, connections are up), yielding intelligent load balancing of requests to remote domains. If all remote domains for which the service is imported become unreachable, the service is suspended in the Bulletin Board.

For example, a service is imported from two remote domains. This is accomplished by listing two entries, as in the following example, in the DM_REMOTE_SERVICES section:

```
RSVC RDOM=R1
RSVC RDOM=R2
```

When connections to both R1 and R2 are up, the gateway load balances the requests. If the connection to R1 goes down, the gateway sends all requests for services RSVC to R2. If both connections go down, the gateway suspends RSVC in the Bulletin Board. Subsequent requests for RSVC will either be routed to a local service or another gateway, or will fail with TPEVENT.

Note: When the connection policy is ON_DEMAND, advertisement of remote services begins whenever the gateway is started up and remains advertised.

How Connection Policies Affect the Dynamic Status Capability

The connection policy you have set up between your domains determines the availability or suspension of services. The following table describes how each connection policy affects the Dynamic Status capability.

Table 2-2 Connection Policies Affect on Dynamic Status

Under This Policy	Dynamic Status Is
`ON_STARTUP`	On. Services imported from a remote domain are advertised while a connection to that remote domain exists. Ways in which a connection can be established are: ♦ Automatic connect ♦ Manual `dmadmin` connect ♦ Incoming connection
`ON_DEMAND`	Off. Services imported from remote domains are always advertised. Ways in which a connection can be established are: ♦ Client request ♦ Manual `dmadmin connect` ♦ Incoming connection
`INCOMING_ONLY`	On. Remote services are initially suspended. The domain gateway is available for incoming connections from remote domains, and remote services are advertised when the local domain gateway receives an incoming connection or a manual `connect` command is issued. Ways in which a connection can be established are: ♦ Manual `dmadmin connect` ♦ Incoming connection

Controlling the Connections Between Domains

As the administrator, you can control the number of connections you want to establish between domains. You can also beak the connections between local and remote domains. This section explains how to perform these tasks.

Establishing Connections Between Domains

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

```
dmadmin co -d local_domain_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, you specify that domain on the command line with the `-R` option.

```
dmadmin co -d local_domain_name -R remote_domain_name
```

If a connection attempt fails, you have configured the domain to retry a connection, repeated attempts to connect (via automatic connection retry processing) are made.

Breaking Connections Between Domains

To break a connection between a local gateway and a remote domain, (making sure that the gateway will not try to reestablish the connection through automatic connection retry processing), run the `dmadmin` command with the `disconnect` (`dco`) subcommand, as follows.

```
dmadmin dco -d local_domain_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 as follows.

```
dmadmin dco -d local_domain_name -R remote_domain_name
```

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

Reporting on Connection Status

Using the `printdomain` command, you can report on connection status and the connections being retried. The `connect` command reports whether the connection attempt was successfully initiated. The `printdomain` command prints information about the specified local domain, including a list of connected remote domains, a list of remote domains being retried.

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

```
$ dmadmin
dmadmin - Copyright (c) 1996 BEA Systems, Inc.
Portions * Copyright 1986-1997 RSA Data Security, Inc.
All Rights Reserved.
Distributed under license by BEA Systems, Inc.
TUXEDO is a registered trademark.

pd -d LDOM
Local domain :LDOM
  Connected domains:
  Domainid:  RDOM1
  Disconnected domains being retried:
  Domainid: RDOM2

 dco -d LDOM -R RDOM1
Operation completed successfully. Use printdomain(pd) to obtain
results.

 dco -d LDOM -R RDOM2
Operation completed successfully. Use printdomain(pd) to obtain
results.

 co -d LDOM -R RDOM3
Operation completed successfully. Use printdomain(pd) to obtain
results.

pd -d LDOM
Local domain :LDOM
  Connected domains:
  Domainid: RDOM3
```

Configuring Failover and Failback

There are two types of failover:

- Link-Level Failover
- Domains Level Failover and Failback

If you want failover and failback functionality in your domain, you must configure your Domains configuration file to support this functionality.

For details about the Domains configuration file, see `DMCONFIG`(5) in the *BEA TUXEDO Reference Manual*.

Link-Level Failover

Link-level failover provides fail over to alternate network links when a failure is detected with a primary link. To use link-level failover, the primary and alternate gateway must reside on different remote domains (that is; *gateway mirroring)*. Currently, link-level failover does not support alternate links to the same gateway. The two gateways cannot reside on the same remote domain. You specify link-level failover in the `DM_TDOMAINS` section of the Domains configuration file as follows:

```
*DM_TDOMAINS
RDOM1 NWADDR=//addr1:0
RDOM1 NWADDR=//addr1:1
```

The first entry refers to the primary network link for remote domain `RDOM1`; the second entry refers to the alternate link.

Link-level failback is a manual procedure. When the primary link is restored, the administrator must bring down the alternate link manually. This may cause requests that are in progress to fail, and new traffic to be resumed over the primary link.

Note: For more detailed information on gateway mirroring, see the `DMCONFIG`(5) reference page in the *BEA TUXEDO Reference Manual*.

Domains-level Failover and Failback

Domains-level failover provides failover to alternate remote domains when a failure is detected with a primary remote domain. It also provides failback to the primary remote domain when it is restored. This level of failover/failback depends on Dynamic Status. The domain must be configured with a `CONNECTION_POLICY` of `ON_STARTUP` or `INCOMING_ONLY` to enable Domains-level failover/failback. Domains-level failover/failback defines a remote domain as being available when a network connection to the remote domain exists, and unavailable when a network connection to the remote domain does not exist.

Prerequisite to Using Domains-level Failover and Failback

You must specify `ON_STARTUP` or `INCOMING_ONLY` as the value of the `CONNECTION_POLICY` parameter to use Domains-level failback. (A connection policy of `ON_DEMAND` is unsuitable for Domains-level failback as it assumes that the remote domain is always available. If you do not specify `ON_STARTUP` or `INCOMING_ONLY` as your connection policy, your servers cannot fail over to the alternate remote domains that you have specified with the `RDOM` parameter.)

Note: A remote domain is *available* if a network connection to it exists; a remote domain is *unavailable* if a network connection to it does not exist.

Configuring Domains to Support Failover

To support failover, you must specify the following in your Domains configuration file:

- ♦ A list of remote domains responsible for executing a particular service as values of the `RDOM` parameter in the `DM_REMOTE_SERVICES` section. You can also specify alternate remote domains, as follows.

  ```
  RDOM=identifier_1, identifier_2, identifier_3
  ```

Example

Suppose the TOUPPER and TOUPPER2 services are available from three remote domains: R1 (the primary remote domain), R2, and R3. You include the following entry in your Domains configuration file.

```
*DM_REMOTE_SERVICES
DEFAULT: RDOM=R1, R2, R3
TOUPPER
TOUPPER2
```

If a value for the RDOM parameter and a routing criteria are not specified in your configuration file, then the local domain assumes that any remote domain of the same type accepts requests for a particular service.

Configuring Domains to Support Failback

Failback occurs when the network connection to the primary remote domain is reestablished. This can occur because of the following reasons:

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

Note: For automatic retries, connection retry must be on (that is, MAXRETRY>0)

CHAPTER

3 Running Domains

What This Chapter Is About

This chapter covers the following topics:

- Domains run-time administrative commands and servers
- Transaction management

Run-time Administration

Domains integrates with an existing BEA TUXEDO application by adding entries for domain gateways groups and gateway servers in the `TUXCONFIG` file. The existing `tmconfig`(1) and `tmadmin`(1) commands can be used to add this new configuration dynamically to a running BEA TUXEDO application. The `tmadmin` command can also be used to list the information contained in the Bulletin Board for Domains gateway groups and their gateways.

In addition to `tmadmin`(1), a new administrative command, `dmadmin`(1), is provided. This command provides administrators with the run-time administration capabilities to maintain the Domains configuration and active gateway groups. Figure 3-1 shows the relationship between administrative commands and servers.

Figure 3-1 Domains Run-Time Administration

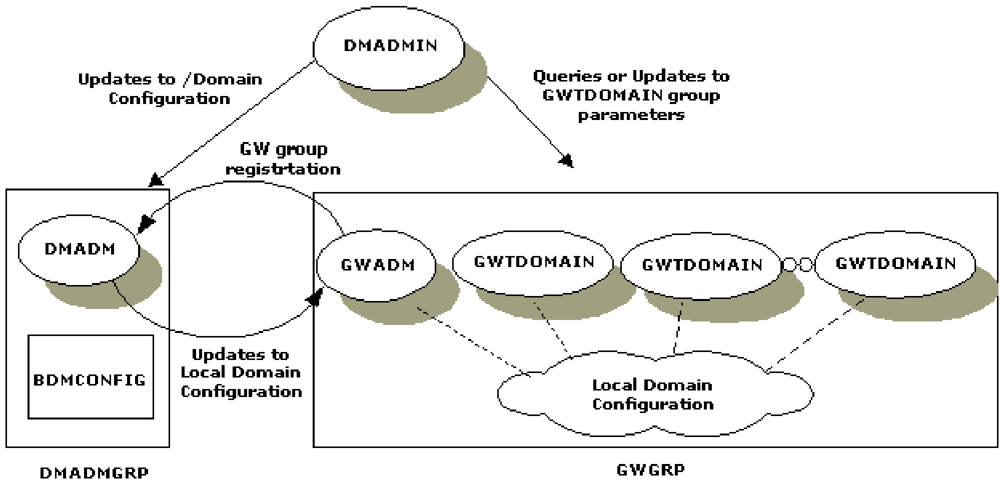

DMADMIN
Updates to /Domain Configuration
Queries or Updates to GWTDOMAIN group parameters
GW group registrtation
DMADM
BDMCONFIG
Updates to Local Domain Configuration
GWADM
GWTDOMAIN
GWTDOMAIN
GWTDOMAIN
Local Domain Configuration
DMADMGRP
GWGRP

Figure 3-1 shows the components of the Domains administrative subsystem: the `dmadmin`(1) command, the Domains administrative server, `DMADM`(5), the gateway group administrative server, `GWADM`(5), the gateway process, `GWTDOMAIN`(5), and the Domains configuration file, `BDMCONFIG`. The `dmadmin` command can be used by the administrator to update the `BDMCONFIG` file while the BEA TUXEDO application is running. The command acts as a front-end process that translates administrative commands to service requests to the `DMADMIN` service, which is a generic administrative service advertised by the `DMADM` server. The `DMADMIN` service invokes the validation, retrieval, or update functions provided within the `DMADM` server to maintain the `BDMCONFIG` file. The `DMADM` server also provides a registration service to 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 gateway group. The `DMADM` server maintains the list of registered gateway groups, and propagates to these groups any changes made to the configuration.

Note: The gateway process, `GWTDOMAIN`(5), which provides connectivity to remote gateway processes, focuses on throughout of messages between BEA TUXEDO domains. This process is not get involved in Domains administration.

Figure 3-1 also shows some of the functionality provided by the GWADM server. This server registers with the DMADM server to obtain the configuration information used by the corresponding gateway group. The GWADM accepts queries from dmadmin to obtain run-time statistics or to change the run-time options of the corresponding 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.

Another run-time feature is the capability to specify gateway parameters when a gateway group is booted. This capability requires the use of the CLOPT parameter when the GWADM server is defined in the *SERVERS section of the TUXCONFIG file.

The main capabilities of the run-time administration feature are as follows:

- A command, dmadmin(1), that allows administrators dynamically to configure, monitor, and tune domain gateway groups
- A domain administrative server, DMADM(5), that provides the administrative processing required for updating the Domains configuration. This server acts as a back-end to the dmadmin command.
- A gateway administrative server, GWADM(5), that provides the run-time administrative processing required for a specific gateway group. This server also acts as a back-end to the dmadmin command.
- The gateway process, GWTDOMAIN(5), that provides connectivity to remote gateway processes. Clients and servers send and receive messages across BEA TUXEDO domains via the GWTDOMAIN process.

The capabilities listed above are described in more detail later in this chapter.

Migration Restriction

The migration of the DMADM is possible. Also, the migration of a domain gateway group (GWADM/GWTDOMAIN) to another machine is possible. However, when using transactions, the domain gateway group can be migrated only across machines of the same type. You must complete the following procedures to migrate GWADM/GWTDOMAIN.

1. The DMCONFIG file should include multiple listening addresses in the following format in the DM_TDOMAIN section:

```
*DM_TDOMAIN
LDOM NWADDR=“//primary:port”
LDOM NWADDR=“//backup:port”
```

Note: This step is unnecessary if third party IP failover solutions are used.

2. The `DMCONFIG` source must be copied manually to the backup machine and built using `dmloadcf`.
3. If you are using transactions, the domains transaction log must also be copied manually to the backup machine.
4. The `DMCONFIG` files of the remote domains should include both network addresses as specified in Step 1.

Security Limitation

For security reasons, the domain administrative command, `dmadmin`, cannot be run from a workstation when Domains is used with an older release of BEA TUXEDO. Older releases cannot distinguish application administrators from ordinary users accessing the system from a workstation.

DMADMIN(1)

The dmadmin command is used by application administrators for the interactive administration of the information stored in the BDMCONFIG file and the different gateway groups running within a particular BEA TUXEDO application. dmadmin is used to obtain statistics or other information gathered by gateway groups, to change the gateway group parameters, and to add (or update) information to the BDMCONFIG file.

dmadmin(1) is designed for extensibility. In future releases, new commands will be integrated with only minor changes to the software.

dmadmin is implemented as a front-end to the DMADM and GWADM servers. The communication between the two servers is done via FML typed buffers.

For details, see dmadmin(1) in the *BEA TUXEDO Reference Manual*.

Run-time Deletions

Run-time deletions to the BDMCONFIG file can be performed only when the changes do not involve an active gateway group.

DMADM(5)

The domain administrative server, DMADM(5), is a BEA TUXEDO-supplied server that provides run-time administration of the BDMCONFIG file. Its main function is to maintain the BDMCONFIG file and to support a list of registered gateway groups. The DMADM propagates run-time configuration changes to the registered gateway groups (see Figure 3-1).

The DMADM server advertises two services:

- DMADMIN, which is used by dmadmin and the GWADM servers
- A service whose name is derived from the server identifier, DMADM_<*svrid*>. DMADM_<*svrid*> is used by registered GWADM servers for specific administrative functions (for example, to refresh the gateway group configuration information or to signal that a GWADM is still registered).

For details, see DMADM(5) in the *BEA TUXEDO Reference Manual*.

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 and it must be defined with no reply queue, that is, REPLYQ=N.

The administrator may define only one copy of the BDMCONFIG file per BEA TUXEDO application (that is, per domain).

GWADM(5)

The gateway administrative server, GWADM(5), is a BEA TUXEDO-supplied server that provides administrative functions for a Domains gateway group.

The main functions of the GWADM server are:

- To get Domains configuration information from the DMADM server, and to accept queries from dmadmin. The GWADM server gets the 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.
- To provide administrative functionality for a gateway group, for example, to accept queries from dmadmin for run-time statistics or to change run-time parameters of the gateway group.
- To provide transaction logging functionality for a gateway group. The GWADM server determines what transactions need to be logged by reading information stored in shared memory. When the GWADM server is booted, it performs a scan of the log to see if any transactions need to be recovered; it then reconstructs the transaction information in shared memory. The first gateway booted, called the *distinguished* gateway, 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 gateway group.

The GWADM server advertises two services:

- A generic service, GWADMIN
- A service name based on the local domain name (the value of the LDOM keyword in the BDMCONFIG)

The dmadmin command uses the services to retrieve information from all active gateway groups or from a specific 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 and it must not have a reply queue, that is, REPLYQ=N must be specified. It must be the first server booted within the 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.

For details, see GWADM(5) in the *BEA TUXEDO Reference Manual*.

The GWADM server must create the shared memory required by the 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 learn the ipckey by retrieving the GWADM registry entry and checking the shmid field.

GWTDOMAIN(5)

The GWTDOMAIN, or gateway process, provides connectivity to remote gateway processes. A GWTDOMAIN process can communicate with one or more remote gateways simultaneously.

GWTDOMAIN gateways should not be specified as members of a MSSQ set. They should not have a reply queue (REPLYQ=N should be specified). GWTDOMAIN gateways may be restartable.

Transaction Management

Application programmers can request the execution of remote services within a transaction. Also, users at remote domains can request the execution of local services within a transaction. Therefore, Domains coordinates the mapping of remote transactions to local transactions, and the sane termination (commitment or rollback) of these transactions.

BEA TUXEDO architecture uses a separate process (the `TMS`) to coordinate the commitment and the recovery of transaction branches accessing a particular group. In Domains, however, this architecture would require extra messages from the gateway to the `TMS` server to process the commitment for incoming transactions. To simplify the Domains architecture and to reduce the number of messages, the `TMS` code has been integrated with the gateway code.

Hence, Domains gateways can process the transaction protocol used by the BEA TUXEDO system. BEA TUXEDO transaction protocol requires that the gateway group advertise the `TMS` service. This advertisement is done when the first gateway is booted. Once the `TMS` service is advertised, any transaction control messages directed to the gateway group are put on the gateways' queue.

Domains gateway groups should be defined in the `TUXCONFIG` file without the parameters `TMSNAME`, `TMSCOUNT`, `OPENINFO`, and `CLOSEINFO`. These four parameters apply to groups that use an XA-compliant resource manager; Domains gateways do not use the XA interface.

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 at every domain, that is, 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.

The following sections describe the transaction management capabilities that are provided by domain gateways. These capabilities are:

- `TMS` functionality
- Mapping of `GTRIDs` (Global Transaction Identifiers)
- Logging transactions
- Recovery

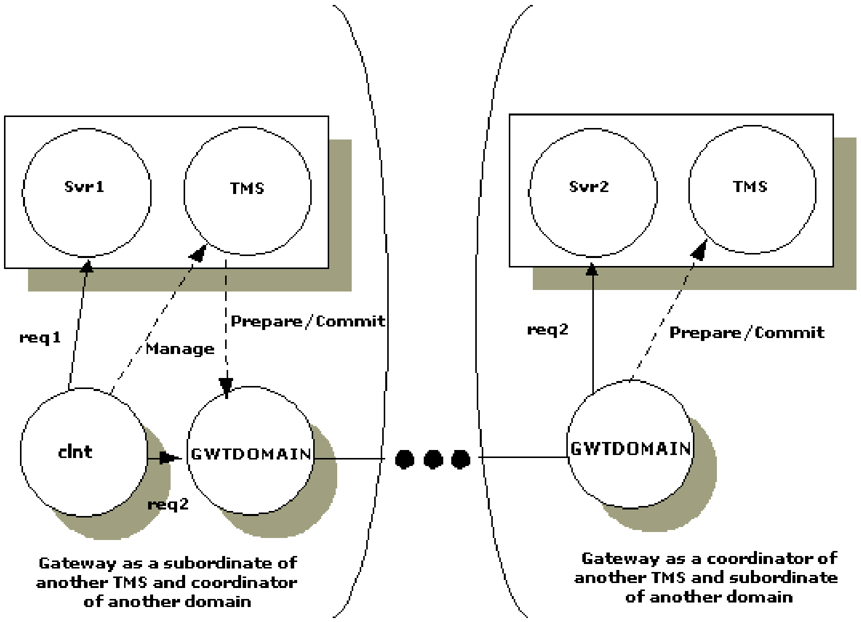
TMS Functionality

In BEA TUXEDO, 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 two-phase commitment protocol. `TMS`s coordinate the commitment of a transaction via special service requests to the `TMS` service, which is offered by all `TMS` servers.

In Domains, `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 BEA TUXEDO because the interface does not match the highly asynchronous and non-blocking model required by the gateway. While Domains gateways do not use a separate `TMS` server, they do offer the `TMS` capability, which allows gateways to coordinate the two-phase commitment of transactions executed across domains. This is accomplished as follows:

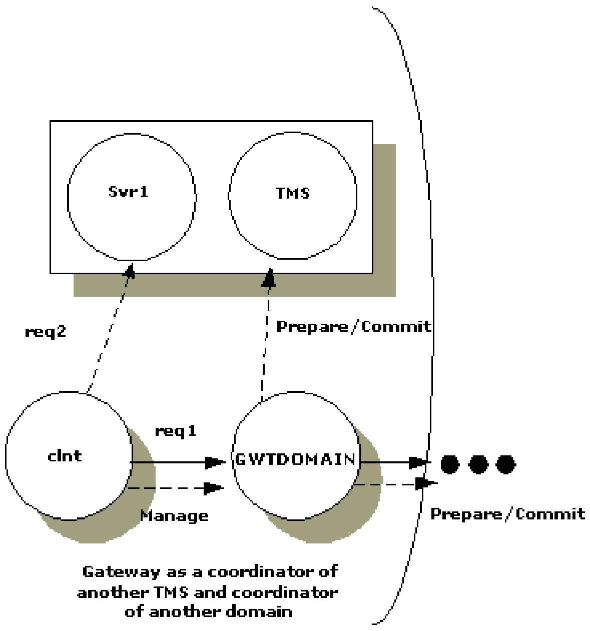
- Domains gateways advertise the `TMS` service and perform all operations associated with that service. Messages sent to this service are placed on the queue used by the gateways and the gateways are able to process transaction management functions for transactions associated with the group.
- A gateway 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. (See `GWTDOMAIN` in Figure 3-2.)
- 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. (See `GWTDOMAIN` in Figure 3-2.)

Figure 3-2 The Gateway as Subordinate/Coordinator


Svr1
TMS
req1
Manage
Prepare/Commit
clnt
GWTDOMAIN
req2
Gateway as a subordinate of another TMS and coordinator of another domain
Svr2
TMS
req2
Prepare/Commit
GWTDOMAIN
Gateway as a coordinator of another TMS and subordinate of another domain

- A gateway can act as a coordinator of transactions within the domain. In this case, the gateway manages the transaction commitment for a particular client. (See Figure 3-3.)

Figure 3-3 The Gateway Managing Commitment for a Client


Svr1
TMS
req2
Prepare/Commit
req1
clnt
GWTDOMAIN
Manage
Prepare/Commit
Gateway as a coordinator of another TMS and coordinator of another domain

- 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 (the Domains gateway) in the forward chain issues the commit and becomes the coordinator of the transaction; as a matter of fact, Domains gateways always act as the last server in a forward chain.
- Gateways automatically start/terminate transactions for remote services specified with the `AUTOTRAN` capability. This capability is required when the application administrator wants to enforce reliable network communication with remote services. Application administrators specify this capability by setting the `AUTOTRAN` parameter to `Y` in the corresponding remote service definition. (See the `*DM_REMOTE_SERVICES` section of the `dmconfig`(5) reference page in the *BEA TUXEDO Reference Manual*.)
- Gateways map the BEA TUXEDO transaction protocol to the networking transaction protocol used for interoperation with remote domains. This mapping is instantiation-dependent.

GTRID Mapping

A GTRID is a Global Transaction IDentifier. GTRID mapping defines how a transaction tree that crosses domain boundaries is constructed.

In the BEA TUXEDO system, a transaction tree is a two-level tree where the root is the group coordinating a global transaction and the machines are other groups involved in the transaction. Each group performs its part of the global transaction independently from the parts done by other groups. Each group, therefore, implicitly defines a transaction branch. The BEA TUXEDO system, through Transaction Manager Servers, TMSs, coordinates the completion of the global transaction, making sure each branch completes.

Tightly-coupled or Loosely-coupled

In the X/Open DTP Model, a Transaction Manager can construct transaction trees by defining either *tightly-coupled* or *loosely-coupled* relationships with a Resource Manager by the way it interprets the transaction identifiers (XIDs) used by the XA interface.

A *tightly-coupled relationship* is one in which the same transaction identifier, XID, is used by all processes participating in the same global transaction and accessing the same 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, work done by a group on behalf of a given global transaction belongs to the same transaction branch; all the processes are given the same XID.

In a *loosely-coupled relationship*, the TM 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. A deadlock results in the rollback of the global transaction. In the BEA TUXEDO system, when different groups participate in the same global transaction each group defines a transaction branch; this results in a loosely-coupled relationship.

Global Transactions across Domains

The first difference between a global transaction in a single BEA TUXEDO application and global transactions across domains is that in the Domains framework the transaction tree cannot be flattened to a two-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 the transaction tree were flattened to a two-level tree, the root domain would have to be directly connected to all domains in the transaction.

This means that the commitment protocol across domains must be hierarchical. Even a loop-back service request defines a new branch in the transaction tree.

Note: A loop-back request is one that 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 the transaction tree for global transactions across domains also depends on the distributed transaction processing protocol used by the particular Domains 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 sub-tree 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.

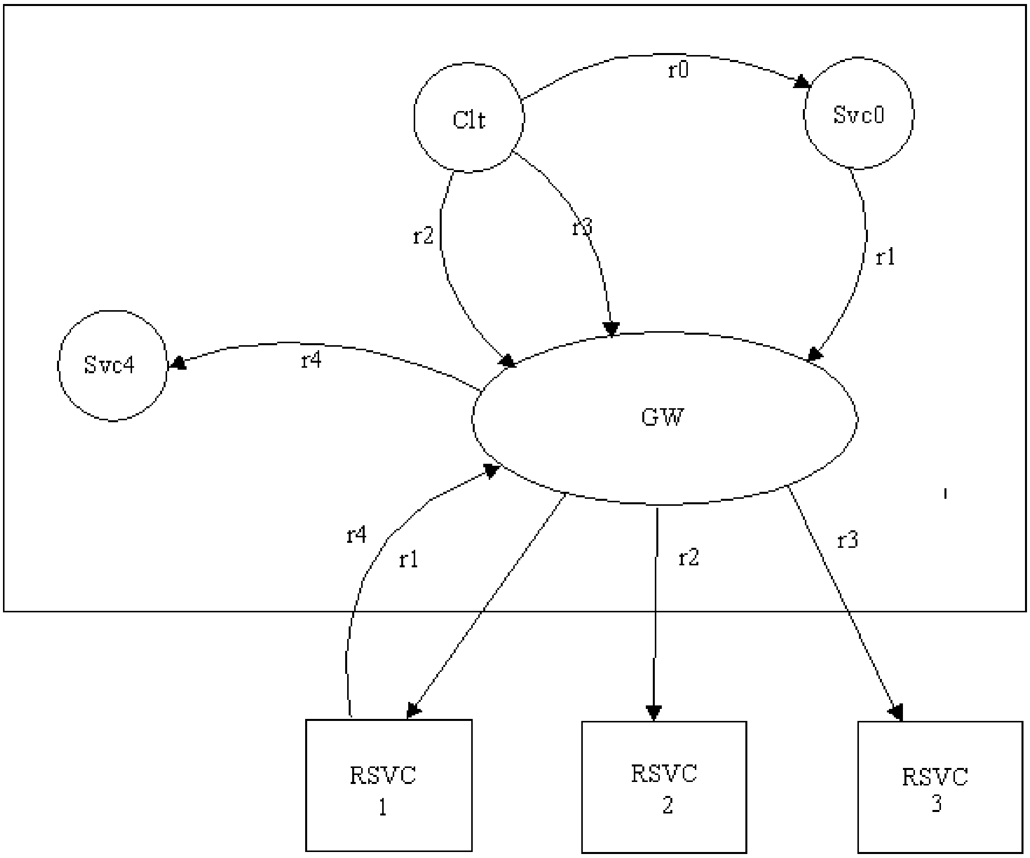
The /TDOMAIN instantiation tries to optimize GTRID mapping by implementing a tightly-coupled relationship. In /TDOMAIN, service requests issued on behalf of the same global transaction are mapped to the same network transaction branch. Therefore, incoming service requests can be mapped to a single BEA TUXEDO transaction. However, the hierarchical structure of inter-domain communication and the inter-domain 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 the transaction, the network transaction tree contains at least one branch per domain interaction. Hence, across domains, the network transaction tree remains loosely-coupled. There will be as many branches as there are domains involved in the transaction (even if all branches access the same resource manager instance).

Domains gateway groups implement a loosely-coupled relationship because they generate different transaction branches for inter-domain transactions.

Figure 3-4 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 loop-back service request `r4` to local service `Svc4`. `Svc0` and `Svc4` execute in different groups (`G0` and `G4`). The Domains gateway executes within another group (`GW`), and the remote services `Rscv1`, `Rsvc2`, and `Rsvc3` execute in another domain (domain B).

Figure 3-4 Service Request Graph

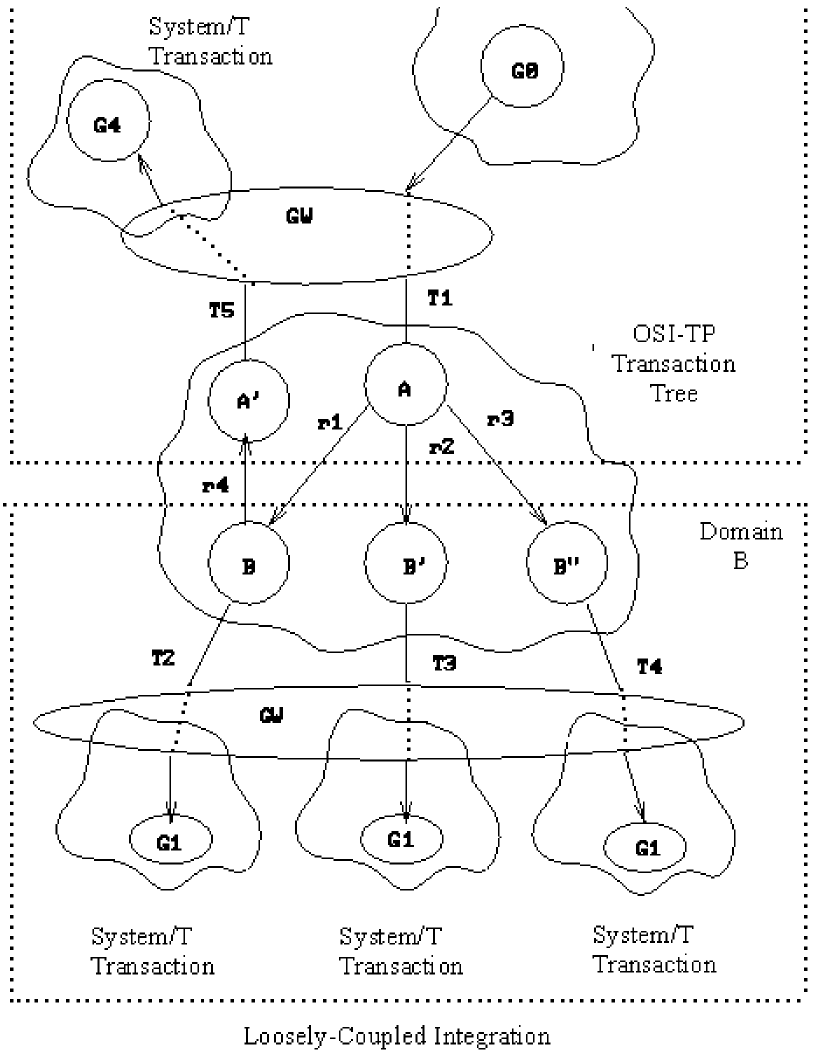


For the service request graph shown in Figure 3-4, Figure 3-5 shows the transaction tree for BEA Connect OSI TP and Figure 3-6 shows the transaction tree for /TDomain. It is assumed, in these figures, that both domains A and B are BEA TUXEDO applications. BEA Connect OSI TP is loosely-coupled because 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 generic part of

Domains only needs to map 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 gateway group `GW` in Domain B coordinating three BEA TUXEDO transactions on group `G1`. Finally, there is the loop-back 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. Hence, the transaction graph shows that gateway group GW on Domain A coordinates group `G4`.

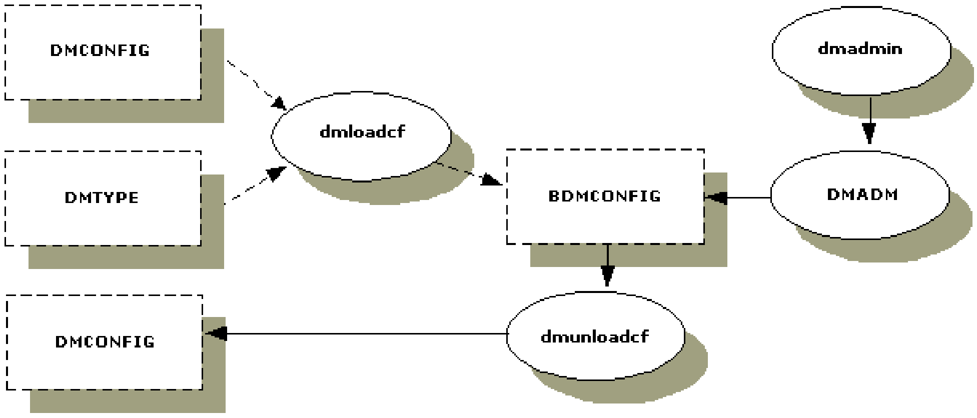
Notice that the hierarchical nature of the OSI TP protocol is fully enforced by these mappings. However, because these mappings introduce a loosely-coupled relationship, the probability of intra-transaction deadlock is increased (e.g., there are three BEA TUXEDO transactions accessing the RM represented by group `G1`).

Figure 3-5 Transaction Tree, BEA Connect OSI TP



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 Figure 3-6.

Figure 3-6 Transaction Tree, /TDOMAIN



Notice that the gateway still must perform mappings between a BEA TUXEDO transaction and a network transaction, and that the hierarchical nature of the communication between domains must be strictly enforced. The figure shows that requests `r1`, `r2`, and `r3` are mapped to a single /TDOMAIN transaction branch. Therefore, on domain B only one BEA TUXEDO transaction needs to be generated; `T2` represents this mapping and the graph shows 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'). This is a new transaction branch on Domain A, and therefore, the gateway generates a new mapping `T3` to a new BEA TUXEDO transaction. The graph shows that gateway group `GW` on domain A also coordinates group `G4`. Hence, the hierarchical nature of inter-domain communication is fully enforced with this mapping: group `G4` cannot commit before group `G1`.

Summary of Transaction Management

We can summarize Domains transaction management as follows:

- ♦ Gateways generate mappings from a BEA TUXEDO transaction to a network transaction. A new mapping is generated per BEA TUXEDO transaction or per incoming network transaction branch.
- ♦ Each instantiation handles its own representation of the network transaction tree. Instantiations observe the hierarchical nature of the inter-domain communication.

Logging

Logging is used to keep track of the progress of the two-phase commit protocol. The information stored in the log is used to make sure the transaction is completed in the event of a network failure or machine crash.

To ensure completion of transactions across domains, Domains gateways log the mapping between local and remote identifiers. Along with this information, the Domains transaction management facility records the decisions during the different phases of the commitment protocol, and the information about the different 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* (for binary large object); it 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's `TLOG`. The `TLOG` records are fixed size and are stored in a single page. Domains log records are variable size; more than one page may be required to store the record. The Domains logging mechanism, `DMTLOG`, has the ability to store variable size log records.

When a `TMS` is the superior of a Domains gateway group, the BEA TUXEDO `TLOG` is still required to coordinate the commitment.

How Logging Works

Logging is done by the GWADM administrative server. While the GWTDOMAIN process requests a log write to be performed, the GWADM performs the actual log write. Each Domains gateway group has its own log. The specification of the DMTLOG is included in the Domains configuration file, DMCONFIG, for each local domain by means of one required parameter (DMTLOGDEV=*string*) and two optional parameters (DMTLOGNAME=*identifier* and DMTLOGSIZE=*numeric*) in the *DM_LOCAL_DOMAINS section. (See dmconfig(5) in the *BEA TUXEDO Reference Manual*.) The administrator can optionally create a DMTLOG with the run-time administration facility. (See dmadmin(1) in the *BEA TUXEDO Reference Manual*.)

If the DMTLOG has not been created when a Domains gateway group is booted, the distinguished gateway automatically creates the log. The information for this automatic creation is extracted from the BDMCONFIG file.

Until the logging device is specified in the BDMCONFIG file the Domains gateway group cannot process requests in transaction mode and the gateway group cannot offer the TMS service.

To coordinate the commit protocol, Domains gateways require the following two log records:

- *Ready record*—The ready record is 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. The log record indicates that all subordinates of the domain gateway group logging the record have been prepared.
- *Commit record*—The commit record notes that a transaction has been committed. A domain gateway creates the commit record when it is the coordinator of a particular transaction tree.

When the transaction has been committed at all machines, these log records of the transaction are removed.

In the OSI TP protocol, two types of heuristic records are logged:

- *Log Heuristic record* —This record holds the details of a heuristic decision at the domain until the outcome of the transaction is known by the superior.
- *Log Damage record* —This record is created for a transaction branch to indicate either a *heuristic hazard* (the outcome of the transaction branch for a subordinate is unknown) or a *heuristic mix* (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 right information during recovery after a crash, and to provide diagnostic information for administrators.

The administrator uses the `forgettran` command of `dmadmin`(1) to remove heuristic records when they are no longer needed.

Recovery

When a Domains gateway group is booted, the distinguished gateway performs an automatic *warm-start* of the `DMTLOG`. The warm-start includes scanning the log to see if any transactions were not completed. If uncompleted transactions are found, action is taken to move them to completion.

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 connection.

In the case of heuristic decisions, if the Domains gateway group is a subordinate of a local `TMS` and the 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 commit has not been completed, the gateway rolls back the transactions associated with that connection.
- In OSI TP if a transaction was in the second phase of commit when a failure is produced, recovery is managed by the XAP-TP.

CHAPTER

4 Connecting Domains

What This Chapter Is About

This chapter describes how to hook BEA TUXEDO system applications together as communicating /TDOMAINs. We want to make it clear at the outset that while this discussion refers to sample programs delivered with the BEA TUXEDO system software (and add-ons), we are describing the steps in a process, not documenting sample programs presently delivered with the Domains software.

The Principal Sections of the Chapter

This chapter contains the following sections:

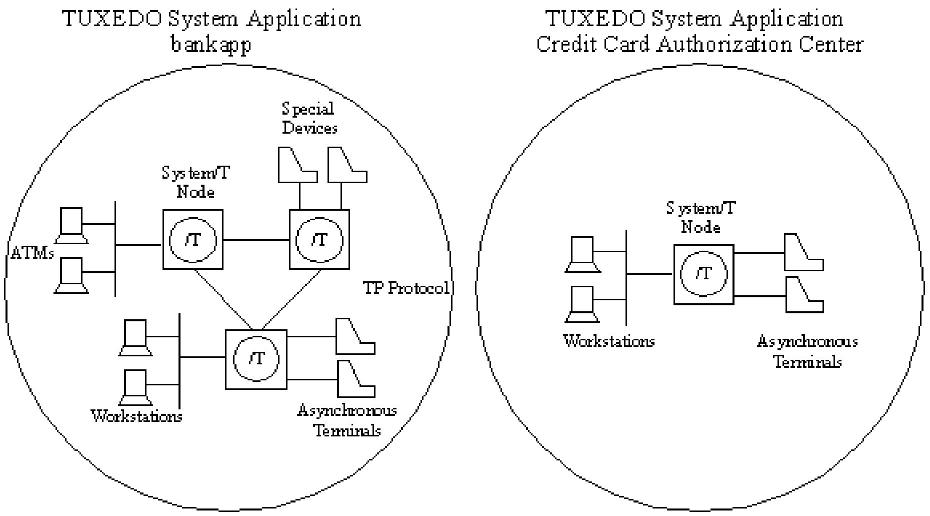
- “The Starting Point: Two BEA TUXEDO Applications”—This section describes two applications that are frequently used as examples in the documentation; it also tells where you can find out more about them.
- “The Need to Intercommunicate”—This section also describes a common problem with business applications, when two applications exist there soon arises a business need for communication between them.
- “Solution 1: Reconfigure the Applications”—One solution (to the need to intercommunicate) is to redefine the two applications as a single networked application. This section shows how to do so.

- "Solution 2: Redefine the Applications as /TDOMAINs"—Redefining the two applications as a single MP application is not the only solution; with the Domains software, you can recast them as domains.
- "Sample /TDOMAIN Application: creditapp"—The chapter ends with a look at a runnable sample application that is delivered with the software.

The Starting Point: Two BEA TUXEDO Applications

Suppose a company has developed the two BEA TUXEDO system applications shown in Figure 4-1.

Figure 4-1 Two BEA TUXEDO System Applications



bankapp and the Credit Authorization Application

The applications shown in Figure 4-1 are known as `bankapp` and the Credit Authorization application. `bankapp` is distributed as a sample application with the BEA TUXEDO software. It can be found in `$TUXDIR/apps/bankapp`, and is documented in the *BEA TUXEDO Application Development Guide*. Be sure to take a close look at the configuration file for implementing `bankapp` as a multiprocessor application: `$TUXDIR/apps/bankapp/ubbmp`.

The Credit Authorization application is described in more detail later in this chapter. For the time being, we ask you to think of it only as a hypothetical extension of `bankapp`.

The Need to Intercommunicate

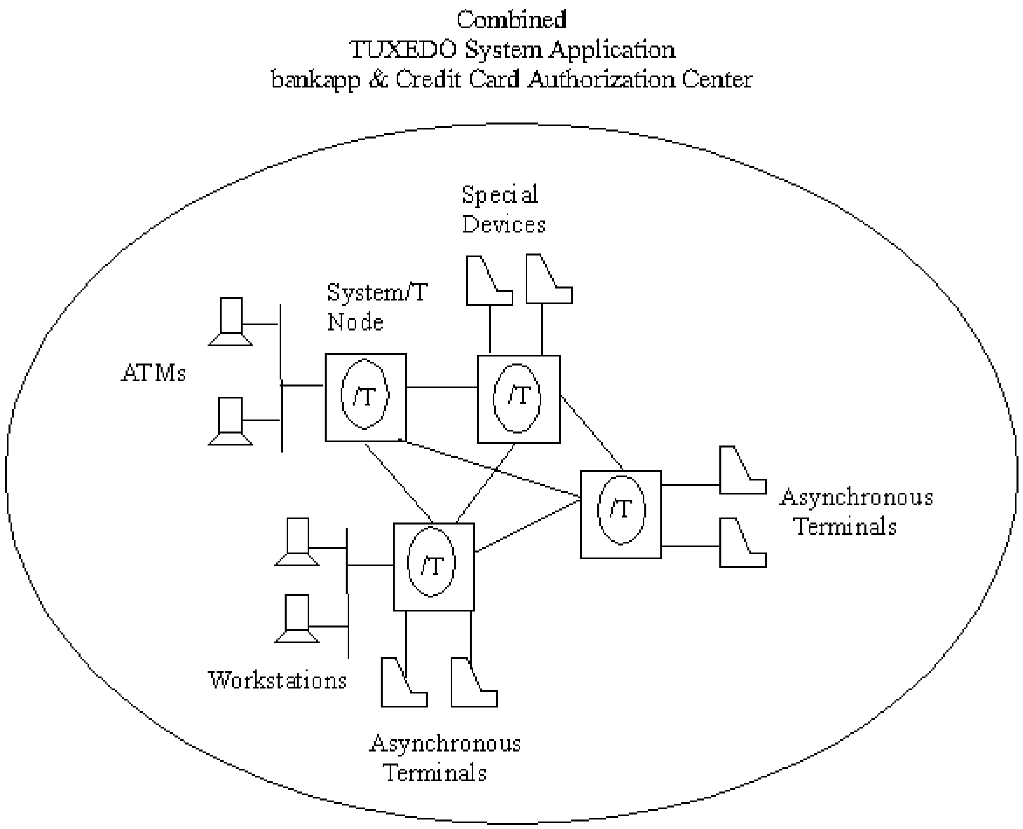
In the course of time our hypothetical business organization has come to realize that their customers would be better served if the `bankapp` application could communicate directly with the Credit Authorization application. That way they could offer instant credit cards to anyone opening a new account.

In the remainder of this chapter we are going to look at two solutions to this problem.

Solution 1: Reconfigure the Applications

The first solution is to combine two BEA TUXEDO system applications into one, as shown in Figure 4-2.

Figure 4-2 Combining Two BEA TUXEDO System Applications



In the process of combining the two applications into a single configuration, the following changes are made:

- `OPTION=LAN` is specified and a `*NETWORK` section is included.
- Server migration is enabled by specifying `OPTION=MIGRATE`; at the same time a backup master site is defined.
- The gateway server is redefined as three other servers: `TLRA`, `ACCTA`, and `CRDT`.
- Credit Authorization services are added.

Configuration File to Combine the Sample Applications

Listing 4-1 shows a possible configuration file for the combined applications. A similar config file (differing only in the network address) must be installed on each machine of the application.

Listing 4-1 Sample Configuration File for the Combined Application

```
*RESOURCES
IPCKEY          76666
UID             0000
GID             000
PERM            0660
MAXACCESSERS    40
MAXSERVERS      35
MAXSERVICES     75
MAXCONV         10
MASTER          SITE1,SITE2
SCANUNIT        10
MODEL           MP
LDBAL           Y
OPTIONS         LAN,MIGRATE
MAXGTT          100
MAXBUFTYPE      16
SCANUNIT        10
SANITYSCAN      5
DBBLWAIT        6
BBLQUERY        50
BLOCKTIME       2
#
#
*MACHINES
#
mach1 LMID=SITE1
                TUXDIR="/home/mylogin/tuxroot"
                APPDIR="/home/mylogin/bankapp"
                ENVFILE="/home/mylogin/bankapp/ENVFILE"
                TLOGDEVICE="/home/mylogin/bankapp/TLOG"
                TLOGNAME=TLOG
                TUXCONFIG="/home/mylogin/bankapp/tuxconfig"
                ULOGPFX="/home/mylogin/bankapp/ULOG"
                TYPE="type1"
#
mach2 LMID=SITE2
                TUXDIR="/home/mylogin/tuxroot"
                APPDIR="/home/mylogin/bankapp"
```

```
                 ENVFILE=“/home/mylogin/bankapp/ENVFILE”
                 TLOGDEVICE=“/home/mylogin/bankapp/TLOG”
                 TLOGNAME=TLOG
                 TUXCONFIG=“/home/mylogin/bankapp/tuxconfig”
                 ULOGPFX=“/home/mylogin/bankapp/ULOG”
                 TYPE=“type2”
#
mach3 LMID=SITE3
                 TUXDIR=“/home/mylogin/tuxroot”
                 APPDIR=“/home/mylogin/bankapp”
                 ENVFILE=“/home/mylogin/bankapp/ENVFILE”
                 TLOGDEVICE=“/home/mylogin/bankapp/TLOG”
                 TLOGNAME=TLOG
                 TUXCONFIG=“/home/mylogin/bankapp/tuxconfig”
                 ULOGPFX=“/home/mylogin/bankapp/ULOG”
                 TYPE=“type2”
#
mach4 LMID=SITE4
                 TUXDIR=“/home/mylogin/tuxroot”
                 APPDIR=“/home/mylogin/bankapp”
                 ENVFILE=“/home/mylogin/bankapp/ENVFILE”
                 TLOGDEVICE=“/home/mylogin/bankapp/TLOG”
                 TLOGNAME=TLOG
                 TUXCONFIG=“/home/mylogin/bankapp/tuxconfig”
                 ULOGPFX=“/home/mylogin/bankapp/ULOG”
                 TYPE=“type1”
#
*GROUPS
#
DEFAULT:  TMSNAME=TMS_SQL    TMSCOUNT=2
BANKB1    LMID=SITE1         GRPNO=1
OPENINFO=“TUXEDO/SQL:/home/mylogin/bankapp/bankdl1:bankdb:readwrite”
BANKB2             LMID=SITE2         GRPNO=2
OPENINFO=“TUXEDO/SQL:/home/mylogin/bankapp/bankdl2:bankdb:readwrite”
BANKB3             LMID=SITE3         GRPNO=3
OPENINFO=“TUXEDO/SQL:/home/mylogin/bankapp/bankdl3:bankdb:readwrite”
BANKB4    LMID=SITE4         GRPNO=4
OPENINFO=“TUXEDO/SQL:/home/mylogin/bankapp/bankdl4: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>”
SITE3    NADDR=“<network address of SITE3>”
```

```
         BRIDGE="<device of provider3>"
         NLSADDR="<network listener address of SITE3>"
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=2
TLR     SRVGRP=BANKB2   SRVID=3         RQADDR=tlr2       CLOPT="-A -- -T 600"
TLR     SRVGRP=BANKB3   SRVID=4
XFER    SRVGRP=BANKB1   SRVID=10
XFER    SRVGRP=BANKB2   SRVID=6
XFER    SRVGRP=BANKB3   SRVID=8
ACCT    SRVGRP=BANKB1   SRVID=11
ACCT    SRVGRP=BANKB2   SRVID=7
ACCT    SRVGRP=BANKB3   SRVID=13
BTADD   SRVGRP=BANKB1   SRVID=12
BTADD   SRVGRP=BANKB2   SRVID=14
BTADD   SRVGRP=BANKB3   SRVID=16
# Servers for the Credit Authorization Part
TLRA    SRVGRP=BANKB4   SRVID=5                            CLOPT="-A -- -T 600"
ACCTA   SRVGRP=BANKB4   SRVID=9
CRDT    SRVGRP=BANKB4   SRVID=15
#
#
*SERVICES
#
DEFAULT:        LOAD=50         AUTOTRAN=N
# Services for the bankapp part
BR_ADD          PRIO=20         ROUTING=BRANCH_ID
TLR_ADD         PRIO=20         ROUTING=BRANCH_ID
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
# Services for the Credit 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="1-3:BANKB1,
                            4-6:BANKB2,
                            7-10:BANKB3,
                            *:*"
#
```

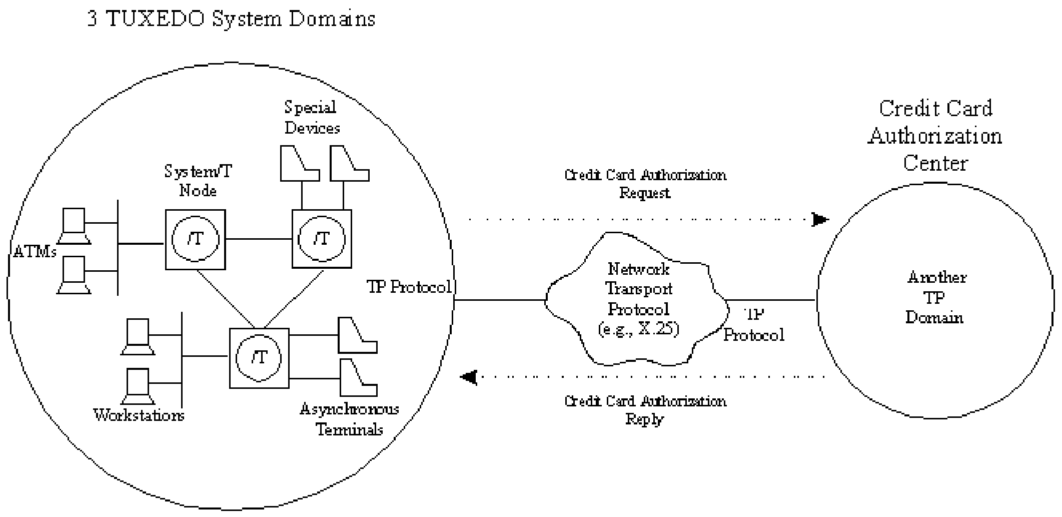
Limitations of This Solution

- Administering a single large application can be more cumbersome than administering two smaller ones; each smaller one has 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.

Solution 2: Redefine the Applications as /TDOMAINS

Figure 4-3 shows the combined application reconfigured as four /TDOMAINs. (Three of the domains are in the left-hand circle.)

Figure 4-3 Picture of the Domain Configuration


3 TUXEDO System Domains
Special Devices
System/T Node
/T
/T
/T
ATMs
Workstations
Asynchronous Terminals
TP Protocol
Credit Card Authorization Request
Network Transport Protocol (e.g., X.25)
TP Protocol
Credit Card Authorization Reply
Credit Card Authorization Center
Another TP Domain

Changes to the BEA TUXEDO System Configuration Files

To reconfigure the combined application as /TDOMAINs, make the following changes to the `UBBCONFIG` files:

- ♦ Change `MODEL` to `SHM`.
- ♦ Remove the `*NETWORK` section.
- ♦ Add domains-specific servers, for example `DMADM`, `GWADM`, and `GWTDOMAIN.`

Listing 4-2 shows a sample converted `UBBCONFIG` file.

Listing 4-2 A Converted UBBCONFIG File

```
*RESOURCES
IPCKEY          76666
UID             7901
GID             601
PERM            0660
MAXACCESSERS    40
MAXSERVERS      35
MAXSERVICES     75
MAXCONV         10
MASTER          SITE1
SCANUNIT        10
MODEL           SHM
LDBAL           Y
MAXGTT          100
MAXBUFTYPE      16
SCANUNIT        10
SANITYSCAN      5
BBLQUERY        50
BLOCKTIME       2
#
*MACHINES
sfexpz          LMID=SITE1
                TUXDIR=“/home/mylogin/tuxroot”
                APPDIR=“/home/mylogin/creditapp”
                ENVFILE=“/home/mylogin/creditapp/ENVFILE”
                TLOGDEVICE=“/home/mylogin/creditapp/TLOG”
                TLOGNAME=TLOG
                TUXCONFIG=“/home/mylogin/creditapp/tuxconfig”
                ULOGPFX=“/home/mylogin/creditapp/ULOG”
                TYPE=“type1”
#
#
#
*GROUPS
DEFAULT:        LMID=SITE1
BANKB1          GRPNO=1  TMSNAME=TMS_SQL         TMSCOUNT=2
OPENINFO=“TUXEDO/SQL:/home/mylogin/creditapp/crdtdl1:bankdb:readwrite”
BANKB2          GRPNO=2
BANKB3          GRPNO=3
BANKB4          GRPNO=4
DMADMGRP  LMID=mach1 GRPNO=5
#
#
#
*SERVERS
#
```

```
DEFAULT: RESTART=Y MAXGEN=5 REPLYQ=Y CLOPT="-A"
GWADM          SRVGRP=BANKB2    SRVID=30
               REPLYQ = Y RESTART = Y GRACE = 0
GWTDOMAIN      SRVGRP=BANKB2    SRVID=31
               REPLYQ = Y RESTART = Y GRACE = 0
GWADM          SRVGRP=BANKB3    SRVID=24
               REPLYQ = Y RESTART = Y GRACE = 0
GWTDOMAIN      SRVGRP=BANKB3    SRVID=25
               REPLYQ = Y RESTART = Y GRACE = 0
GWADM          SRVGRP=BANKB4    SRVID=20
               REPLYQ = Y RESTART = Y GRACE = 0
GWTDOMAIN      SRVGRP=BANKB4    SRVID=21
               REPLYQ = Y RESTART = Y GRACE = 0
DMADM          SRVGRP="DMADMGRP"   SRVID=50
               REPLYQ = N RESTART = Y GRACE = 0
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
```

Adding DMCONFIG Files

You also need to make four `DMCONFIG` files. A sample is shown in Listing 4-3.

Listing 4-3 Sample DMCONFIG File

```
#
#
*DM_LOCAL_DOMAINS
#
#
QDOM1   GWGRP=BANKB2
        TYPE=TDOMAIN
        DOMAINID=QDOM1
        BLOCKTIME=10
        MAXDATALEN=56
        MAXRDOM=89
        DMTLOGDEV="/home/mylogin/creditapp/DMTLOG"
        AUDITLOG="/home/mylogin/creditapp/AUDITLOG"

QDOM2   GWGRP=BANKB3
        TYPE=TDOMAIN
        DOMAINID=QDOM2
        BLOCKTIME=10
        MAXDATALEN=56
        MAXRDOM=89
        DMTLOGDEV="/home/mylogin/creditapp/DMTLOG"
        AUDITLOG="/home/mylogin/creditapp/AUDITLOG"
        DMTLOGNAME="DMTLOG_TDOM2"
QDOM3   GWGRP=BANKB4
        TYPE=TDOMAIN
        DOMAINID=QDOM3
        BLOCKTIME=10
        MAXDATALEN=56
        MAXRDOM=89
        DMTLOGDEV="/home/mylogin/creditapp/DMTLOG"
        AUDITLOG="/home/mylogin/creditapp/AUDITLOG"
        DMTLOGNAME="DMTLOG_TDOM3"
#
*DM_REMOTE_DOMAINS
#
#
TDOM1   TYPE=TDOMAIN
        DOMAINID=TDOM1

TDOM2   TYPE=TDOMAIN
        DOMAINID=TDOM2
```

```
TDOM3   TYPE=TDOMAIN
        DOMAINID=TDOM3
#
#
*DM_TDOMAIN
#
QDOM1    NWADDR=“0x0002DEEF93026927”
         NWDEVICE=“/dev/tcp”
QDOM2    NWADDR=“0x0002BEEF93026927”
         NWDEVICE=“/dev/tcp”
QDOM3    NWADDR=“0x0002CEEF93026927”
         NWDEVICE=“/dev/tcp”
TDOM1    NWADDR=“0x0002DEEF93026947”
         NWDEVICE=“/dev/null”
TDOM2    NWADDR=“0x0002BEEF9302691D”
         NWDEVICE=“/dev/tcp”
TDOM3    NWADDR=“0x0002CEEF9302690E”
         NWDEVICE=“/dev/tcp”
#
#
#
*DM_LOCAL_SERVICES
#
#
WITHDRAWALA
INQUIRYA
OPENCA
CLOSECA
```

Sample /TDOMAIN Application: creditapp

A sample application, `creditapp`, is distributed with the software. `creditapp` is a runnable version of the hypothetical application that was the basis for the discussion above.

The application is located in `$TUXDIR/apps/creditapp`. It includes the files shown in Listing 4-4.

Listing 4-4 creditapp Files

```
ACCT.ec        ACCTA.ec        AUDITC.c       BAL.ec         BALANCE.m
BALANCEA.m     BALC.ec         BTADD.ec       CBALANCE.m     CCLOSE.m
CDEPOSIT.m     CLOSE.m         COPEN.m        CRDT.ec        CRDTA.ec
CRMENU.m       CRMENU2.m       CTRANSFER.m    CWITHDRAW.m    DEPOSIT.m
DEPOSITA.m     FILES           HCBALANCE.m    HCCLOSE.m      HCLOSE.m
HCOPEN.m       HCWITHDRAW.m    HELP.m         HOPEN.m        OPEN.m
README         RUNME           RUNME.sh       SETUP.sh       TLR.ec
TLR1.ec        TLR2.ec         TLR3.ec        TLRA.ec        TRANSFER.m
WITHDRAW.m     WITHDRAWA.m     XFER.c         appinit.c      aud.h
aud.v          audit.c         auditcon.c     bank.flds      bank.flds.h
bank.h         cleanup.sh      crbank.sh      crbankdb.sh    crdt_app.mk
crdt_app2.mk   crdt_app3.mk    crdt_app4.mk   crdt_flds.h    crdtvar
crdtvar2       credit.flds     crtlog.sh      crtlog2        crtlog2.sh
domcon1        domcon2         domcon3        domcon4        driver.sh
envfile.sh     gendata.c       gentran.c      hostmk         listnr
populate.sh    run.sh          setenv         ubbdom1        ubbdom2
ubbdom3        ubbdom4         util.c
```

Listing 4-5 is a version of the `README` file from the `creditapp` directory. The `README` file documents a script that installs and runs `creditapp`. It has been edited to include a few things that are not in the original script.

Listing 4-5 README File for creditapp

```
SIMPLE BUILD PROCEDURE

The creditapp application is an enhancement of the
bankapp and hostapp applications.

The creditapp application is designed to be a four domain application,
so the software must be built on four machines. The RUNME.sh
script will lead you through the necessary steps.


Step 1: Copy the Software for creditapp.

Make a new directory under your $HOME directory and copy all of
the source files from <TUXDIR>/apps/creditapp into that directory.
TUXDIR is the root directory under which your BEA TUXEDO System
software is installed. We call the new directory
$HOME /creditapp. The rest of the steps in this procedure are
done in the directory $HOME/creditapp.


Step 2: On each of the remaining three machines:

Make a directory creditapp in a directory that can be used for the application.

We call this directory $HOME/creditapp.

Make a note of the full directory path for $HOME/creditapp and TUXDIR
for each machine. These will be needed by the RUNME.sh script.


Step 3: On the “master site” execute the “RUNME.sh” script.

The shell script “RUNME.sh” is an interactive program designed to
lead you through initialization, booting, shutdown and cleanup
of the four domain creditapp application. The shell is interactive
and requires no command line arguments. All you need in the directory
is the source from the TUXDIR/apps/creditapp directory that you
copied in Step 1.

You will be prompted to enter values for RSH and RCP
environment variables, or accept the defaults.

IT IS VERY IMPORTANT THAT VALUES FOR RSH AND RCP BE ENTERED AS THEY ARE
USED TO REMOTE COPY AND EXECUTE THE NECESSARY SCRIPTS.

The following environment variables are important. The script picks up
the values for TUXDIR and APPDIR from your environment and
prompts you (in OPTION 4) for BLKSIZE:
```

```
TUXDIR         Root directory of the BEA TUXEDO System where you have
                 installed the  software.

APPDIR          Directory in which the creditapp application resides.
                 crdtvar.dm1 initially is set to allow this to default
                 to the current working directory, which agrees with
                 our intention to use $HOME/creditapp. This is the
                 directory into which you copied the creditapp files in
                 Step 1.

BLKSIZE         Logical blocksize for the database in bytes.
                 Must be an integral multiple of the physical
               page size of the computer (for example, 512 bytes or 4096 bytes).
```

When you invoke RUNME.sh you are shown a menu with 10 options (11 counting “quit”). Here is the list of choices:

```
          1) Initialize configuration files and makefiles.
          2) Copy files to remote sites.
          3) Build crdtapp clients and servers.
          4) Create databases.
          5) Generate binary tuxconfig and bdmconfig files.
          6) Create Transaction Log file.
          7) Boot the application.
          8) Populate the database.
          9) Shutdown the application.
          10) Cleanup IPC Resources, database files and log files.
          q) Quit.
```

To go through the complete process of building and running the sample application, start with choice No. 1. When the script completes a step, the menu is displayed for your next choice.

```
OPTION 1. Initialize configuration files and makefiles.
          This option sets up makefiles, UBBCONFIG and DMCONFIG files that are
          necessary for the application.

          All questions must be answered.

          ENTER the system name: enter uname for machines you are using
                                  beginning with the current machine you are on.


          ENTER TUXDIR for each machine.

          ENTER APPDIR for each machine.


          Continue to answer all queries.
```

```
       An example of 4 hexadecimal digits may be (beef, cfff, 6774, aeef).
NOTE: EACH MACHINE MUST HAVE A UNIQUE HEX SEQUENCE.


OPTION 2. Copies the files to the other domains in the configuration.


OPTION 3. Builds clients and servers on all machines.

          NOTE: CAREFULLY CHECK THAT THE BUILDS ARE COMPLETED SUCCESSFULLY ON
                EACH SITE. IF NECESSARY YOU MAY RUN THE BUILD YOURSELF.

                ON THE SPECIFIC SITE ENTER
                nohup make -f CRDT{$MACH}.mk2

                where ${MACH} is the uname for the machine you are building on.
                For example,

                        nohup make -f CRDTtux1.mk2


OPTION 4. Builds the databases on each site.

          NOTE: ON EACH SITE MAKE SURE THE BLKSIZE VALUE IN files

                crdt${MACH}.dm1 for the primary site

                or crdt${MACH}.dm2 for the remote sites

                where ${MACH} is the uname for the machine you are building on

                ARE CORRECT FOR THAT SPECIFIC MACHINE


OPTION 5. Generates the tuxconfig and bdmconfig files.


All other options are similar to bankapp.


After OPTION 8 : Populate the database

                  Enter q to Quit the menu.

RUNNING CREDITAPP.
__________________


            On each machine a script run.sh exists.

            Execute run.sh.
            run
At the response :

               Is this machine the Credit Card Authorization Center(y/n)?

               If machine is the primary machine answer y .
               If machine is any other answer n.

On the primary machine a different menu will be seen than the other 3 machines.

All Credit accounts exist on primary machine and all machines can access any
account.


            ACCOUNTS 10000000 - 120000000

Machines 2,3,4 are the enhanced bankapp application.

            ACCOUNTS 10000 - 39999 exist on machine 2
            ACCOUNTS 40000 - 79999 exist on machine 3
            ACCOUNTS 80000 - 109999 exist on machine 4

All processing is done using the /DOMAIN software.

A tail -f of the ULOG###### will show the actual processing of the requests.

On the machine that will process the request enter :

           tail -f ULOG###### where ###### is today’s date.
```