



THE ENTERPRISE MIDDLEWARE SOLUTION

BEA Connect

SNA

User Guide

BEA Connect SNA 2.0
Document Edition 2.0
May 1998

Copyright

Copyright © 1998 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, BEA Connect, BEA Jolt, Distributed Application Framework, and Enterprise Middleware Solutions are trademarks of and are developed and licensed by BEA Systems, Inc., Sunnyvale, California. TUXEDO is a registered trademark of Novell, Inc., exclusively licensed to BEA Systems, Inc.

OpenView is a registered trademark of Hewlett-Packard Company. SunNet Manager and Solstice are trademarks of Sun Microsystems, Inc. in the United States and other countries. IBM, NetView, CICS/ESA, IMS/ESA, MVS/ESA, and VTAM are registered trademarks of International Business Machines Corporation. Oracle is a registered trademark of Oracle Corporation. Informix is a registered trademark of Informix. Cabletron Spectrum is a trademark of Cabletron Systems, Inc. UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

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

Connect SNA Users Guide

Document Edition	Part Number	Date	Software Version
2.0	825-001001-001	May 1998	BEA Connect SNA 2.0

Contents:

Preface

Purpose of This Manual.....	xiii
Scope	xiii
Who Should Read This Manual	xiii
Administrators	xiv
Operators	xiv
Recommended Reading	xiv
How This Manual Is Organized	xv
How to Use This Manual	xvi
Online Document Considerations	xvi
Opening the Manual in a Web Browser	xvi
Printing from a Web Browser	xvi
Document Conventions	xviii
Related Documentation	xix
Connect SNA Documentation	xix
Product Manuals	xix
Other Publications	xix
Contact Information.....	xx
Documentation Support.....	xx
Technical Support	xx

1. Introducing BEA Connect SNA

Features.....	1-1
Functions	1-2

2. Understanding How BEA Connect SNA Works

Overview of Domain Gateways	2-1
-----------------------------------	-----

The BEA Connect SNA Domain	2-3
The DMINIT and SNACRM Servers	2-5
Other Gateway Servers	2-7

3. Installing BEA Connect SNA

Pre-Installation Considerations	3-1
Local Environment	3-1
Remote Environment	3-2
Planning	3-3
Installation	3-4
Running install.sh	3-4
Installation Components	3-7
Executables	3-8
Application Files	3-8
Libraries	3-8
Message Queues	3-9
Post Installation Action	3-9

4. Configuring BEA Connect SNA

Local Domain Configuration	4-1
Editing the UBBCONFIG File	4-2
Editing the DMCONFIG File	4-2
Prerequisites	4-2
*DM_LOCAL_DOMAINS Section	4-2
*DM_REMOTE_DOMAINS Section	4-3
*DM_SNADOM Section	4-3
*DM_SNACRM Section	4-3
*DM_SNASTACKS Section	4-5
*DM_SNA_LINKS Section	4-6
*DM_REMOTE_SERVICES Section	4-10
Using the dmadm Command Interpreter	4-10
Configuring the *DM SNACRM SECTION	4-11
Configuring the *DM_SNASTACKS Section	4-11
Configuring the *DM_SNALINKS Section	4-12
Remote Host Domain Configuration	4-13

Establishing the VTAM Configuration	4-13
Configuring the CICS/ESA LU	4-13
Create Connections at the Remote Host	4-14
Define the Session at the Remote Host	4-14
View Connection and Session Status	4-15
Completing Cross-Platform Definitions	4-15
Setting Stack Traces	4-20

5. Administering the BEA Connect SNA Application Domain

Administration Facilities	5-1
The dmadm Command Interpreter	5-2
The SNACRM and PU 2.1 Servers	5-3
Starting the PU2.1 Server	5-3
Starting the SNACRM	5-3
Using DMINIT	5-3
Security	5-4
Where You Specify Security Parameters	5-5
UBBCONFIG File Security Parameters	5-6
DMCONFIG File Security Parameters	5-7
Security Setting Summary	5-9
Security Setting Summary Tables	5-11
How To Administer Security	5-13
Adding a Userid and Password	5-13
Mapping a Userid	5-14
Removing a Userid's Mapping	5-14
Deleting a Userid and Password	5-15
Modifying a Password	5-15
Data Translations	5-16
Buffers Going To The Remote Host Application	5-16
INBUFTYPE Parameter Definition	5-17
Data Conversion for STRING Typed Buffer	5-17
Data Conversion for X_OCTET/CARRAY Typed Buffers	5-17
Data Conversion for VIEW/VIEW32/X_C_TYPE/X_COMMON Typed Buffers	5-17
Data Conversion for FML/FML32 Typed Buffers	5-18

Buffers Coming From The Remote Host Application	5-18
OUTBUFTYPE Parameter Definition	5-18
Data Conversion for STRING Typed Buffer	5-18
Data Conversion for X_OCTET/CARRAY Typed Buffers	5-19
Data Conversion for VIEW/VIEW32/X_C_TYPE/X_COMMON Typed Buffers	5-19
Data Conversion for FML/FML32 Typed Buffers	5-19
Data Conversion For DPL Services	5-19
DPL Requests Originating From A TUXEDO Application.....	5-20
DPL Requests Originating From a CICS DPL.....	5-21
Usage Notes	5-21

6. Verifying BEA Connect SNA

Overview	6-1
Building Your Verification Tests	6-2
Building Tuxedo Executables.....	6-2
Using the buildserver Utility	6-3
Using the buildclient Utility	6-3
Modifying the UBBCONFIG file	6-3
Executing the tmloadcf Command.....	6-4
Modifying the DMCONFIG File	6-4
Executing the dmloadcf Command	6-6
Modifying the Environment Files	6-6
Building CICS/ESA Executables	6-9
Choosing the Source Code Language	6-10
Transferring the Source Code to Host.....	6-10
Translating CICS/ESA Verbs.....	6-11
Compiling the Translated Source File.....	6-12
Link-Editing the Compiled Source File	6-15
Configuring the CICS/ESA Application	6-17
Viewing Connection and Session Status	6-22
Running the Sample Application.....	6-23
Running the Application from a Tuxedo Client	6-24
Running the Application from a CICS/ESA Client.....	6-25
CICS/ESA Client with CPI-C	6-25

CICS/ESA Client with DPL.....	6-26
-------------------------------	------

7. Programming Considerations

Overview	7-1
APPC/IMS Programming	7-2
CICS/ISC Programming	7-4
Multi-Region versus Multi-Processor Operations.....	7-4
ISC Operations	7-5
Asynchronous Processing	7-6
Function Request Shipping	7-6
Transaction Routing	7-6
Distributed Program Link	7-7
Step-by-Step Description: Generic DPL Transaction	7-8
Distributed Transaction Processing	7-9
Step-by-Step Description: Generic DTP Transaction	7-10
CICS/ESA Sync-Levels	7-11
Time-out and Error Handling	7-13
Application-to-Application Programming.....	7-13
Distributed Program Link (DPL)	7-14
Step-by-Step Description: Tuxedo Client Request/Response to CICS/ESA DPL.....	7-17
Step-by-Step Description: Tuxedo Client Asynchronous Request/Response to CICS/ESA DPL	7-19
Step-by-Step Description: Tuxedo Client Asynchronous Request/Response with No Reply to CICS/ESA DPL.....	7-21
Step-by-Step Description: CICS/ESA DPL to Tuxedo Request/Response Server	7-23
Step-by-Step Description: CICS/ESA DPL to Tuxedo Request/Response Server, Service in Autonomous Transaction	7-25
Step-by-Step Description: Tuxedo Client Request/Response to CICS/ESA DPL, in Autonomous Transaction	7-27
Step-by-Step Description: Transactional Tuxedo Client Multiple Requests/Responses to CICS/ESA DPL	7-29
Step-by-Step Description: Transactional CICS/ESA DPL to Tuxedo Request/Response Server.....	7-31
Distributed Transaction Processing (DTP)	7-33

Step-by-Step Description: Tuxedo Client Request/Response to CICS/ESA DTP	7-35
Step-by-Step Description: Tuxedo Client Asynchronous Request/Response to CICS/ESA DTP	7-37
Step-by-Step Description: Tuxedo Client Asynchronous Request/Response with No Reply to CICS/ESA DTP	7-39
Step-by-Step Description: Tuxedo Conversational Client to CICS/ESA DTP, Server Gets Control	7-41
Step-by-Step Description: Tuxedo Conversational Client to CICS/ESA DTP, Client Retains Control	7-43
Step-by-Step Description: Tuxedo Conversational Client to CICS/ESA DTP, Client Grants/Gets Control	7-45
Step-by-Step Description: CICS/ESA DTP to Tuxedo Conversational Server, Client Retains Control	7-47
Step-by-Step Description: CICS/ESA DTP to Tuxedo Conversational Server, Client Relinquishes Control	7-49
Step-by-Step Description: Transactional Tuxedo Client Request/Response to CICS/ESA DTP	7-51
Step-by-Step Description: Transactional Tuxedo Conversational Client to CICS/ESA DTP, Server Gets Control	7-53
Step-by-Step Description: Transactional Tuxedo Conversational Client to CICS/ESA DTP, Client Grants/Gets Control	7-55
Step-by-Step Description: CICS/ESA DTP to Transactional Tuxedo Conversational Server, Host Client Relinquishes Control	7-57
CPI-C Programming	7-59
Step-by-Step Description: Tuxedo Client Request/Response to Host CPI-C	7-61
Step-by-Step Description: Tuxedo Client Asynchronous Request/Response to Host CPI-C	7-63
Step-by-Step Description: Tuxedo Client Asynchronous Request/Response to Host with No Reply	7-65
Step-by-Step Description: Tuxedo Conversational Client to Host CPI-C, Server Gets Control	7-67
Step-by-Step Description: Tuxedo Conversational Client to Host CPI-C, Client Retains Control	7-69
Step-by-Step Description: Tuxedo Conversational Client to Host CPI-C, Client Grants/Gets Control	7-71

Step-by-Step Description: Host CPI-C to Tuxedo Asynchronous Request/Response Server with No Reply	7-73
Step-by-Step Description: HOST CPI-C to Tuxedo Server Request/Response	7-75
Step-by-Step Description: Host CPI-C to Tuxedo Conversational Service, Client Retains Control	7-77
Step-by-Step Description: Tuxedo Conversational Service, Client Grants Control	7-79
Step-by-Step Description: Transactional Tuxedo Client Request/Response to Host CPI-C	7-81
Step-by-Step Description: Transactional Tuxedo Conversational Client to Host CPI-C, Server Gets Control	7-83
Step-by-Step Description: Transactional Tuxedo Conversational Client to Host CPI-C, Client Grants/Gets Control	7-85
Step-by-Step Description: Transactional Host CPI-C to Tuxedo Conversational Server, Client Grants Control	7-87
Where to Find Additional Information.....	7-88

A. Reference Pages

B. ATMI to CPI-C Function Mapping

C. CPI-C Parameters and Values

D. Error Messages

E. Sample VTAM Configurations

Glossary



Preface

Purpose of This Manual

This guide provides information about **CONNECT SNA**, a **TUXEDO** multi-domain connectivity product which enables client/server communications between **MVS/CICS** programs and **TUXEDO** applications via a **Systems Network Architecture (SNA)** network.

Scope

This guide explains **BEA CONNECT SNA** technical concepts and provides step-by-step procedures for:

- ◆ Installing and configuring **CONNECT SNA** platform software
- ◆ Administering the **CONNECT SNA** gateway
- ◆ Programming **TUXEDO** applications for transactional conversations with **IBM CICS** programs.

Who Should Read This Manual

The audience for this book is primarily **BEA TUXEDO** System application administrators and operators.

Administrators

As the application administrator of a BEA TUXEDO System, you will configure CONNECT SNA using the DMCONFIG file and its associated *dm* commands. You must have sufficient SNA knowledge to configure the underlying SNA *Stack* so it conforms with definitions created in VTAM and CICS for each remote domain. This document provides information to help you understand the relationship between BEA TUXEDO system configuration settings and SNA-based application configuration concepts.

Successfully linking and establishing conversations between BEA TUXEDO system applications and SNA-based programs requires special coordination. The names and characteristics of SNA resources, configured in the SNA stack, must agree with resources and characteristics defined in VTAM and CICS. This guide includes examples of these relationships.

Typically, remote VTAM and CICS resources are defined by system personnel in a data center where IBM mainframes are located. Therefore, you need to request the remote names of CONNECT SNA and CICS resources from data center systems personnel and use those names to configure the local SNA stack.

Operators

BEA TUXEDO operators use existing skills with the BEA TUXEDO domain administration facility to modify SNA domain configurations and get information about the configuration and run-time environment.

Recommended Reading

You should read the *BEA TUXEDO Product Overview*. Additionally, you should have read and understand the *BEA TUXEDO/Domain Guide* and the *BEA TUXEDO Application Development Guide*, which are essential to comprehending the material in this document.

How This Manual Is Organized

The *BEA Connect SNA User Guide* is organized as follows:

- ◆ **Introducing BEA Connect SNA**
- ◆ **Understanding How BEA Connect SNA Works**
- ◆ **Installing BEA Connect SNA**
- ◆ **Configuring BEA Connect SNA**
- ◆ **Administering the BEA Connect SNA Application Domain**
- ◆ **Verifying BEA Connect SNA**
- ◆ **Programming Considerations**

In addition, the appendixes contain information about:

- ◆ Reference Pages (formerly called man pages)
- ◆ ATMI to CPI-C Function Mapping
- ◆ CPI-C Parameters and Values
- ◆ Error Messages
- ◆ Sample VTAM Configurations
- ◆ Glossary

How to Use This Manual

Online Document Considerations

This document, *BEA Connect SNA User Guide*, is designed primarily as an online, hypertext guide. If you are reading this as a paper publication, note that to get full use from this guide you should install and access it as an online document via a Web browser that supports HTML 3.0. Netscape Navigator 2.02 or Microsoft Internet Explorer 3.0 or later are recommended. (Information on how to install the online documentation is available in the *BEA CONNECT SNA Release Notes*.)

Opening the Manual in a Web Browser

To access the online version of this document, open the following HTML file in a Web browser:

`http://(directory path to Connect SNA HTML files)/begin.htm`

Note: The online documentation requires a Web browser that supports HTML 3.0. Netscape Navigator 2.02 or Microsoft Internet Explorer 3.0 or later are recommended.

Figure 1 shows the online guide with the clickable navigation bar and table of contents.

Printing from a Web Browser

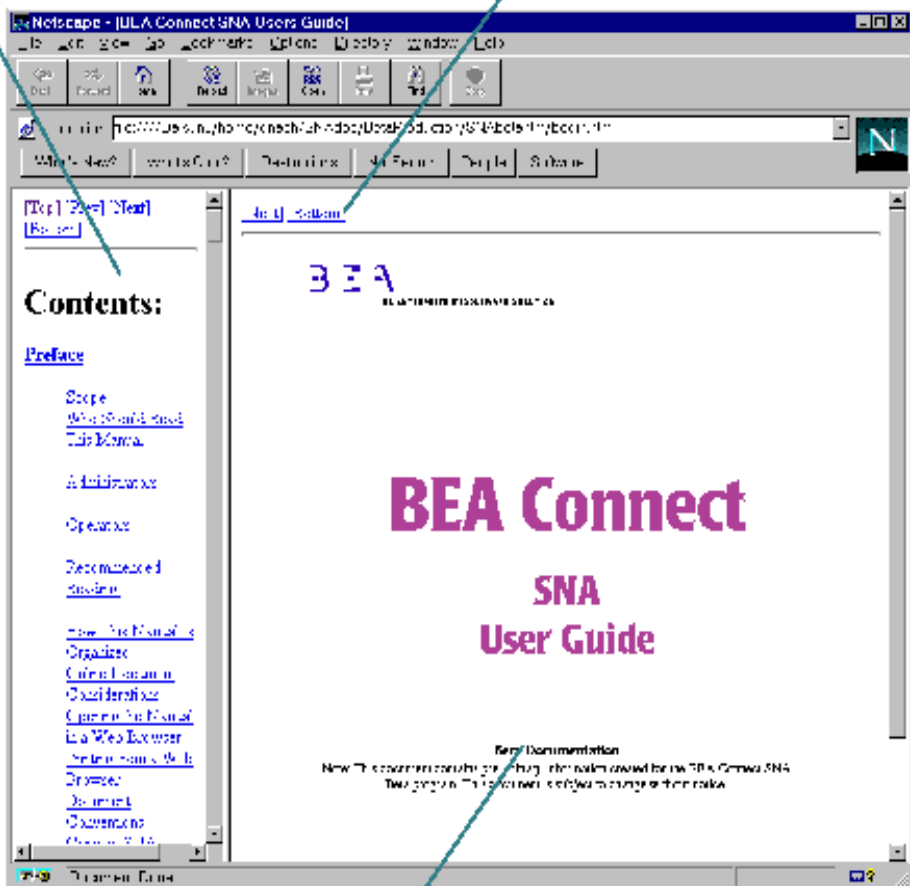
You can print a hard-copy version of this document, one file at a time, from the Web browser. Before you print, make sure that the chapter or appendix you want is displayed and *selected* in your browser. (To select a file, click anywhere inside the frame you want to print. If your browser offers a Print Preview feature, you can use it to verify which file you are about to print.)

Figure 1 *BEA Connect SNA User Guide* Displayed in Netscape Web Browser



[Click on topic to view it.](#)

Navigation Links



Document Display Area

Document Conventions

The following documentation conventions are used throughout this manual:

Item	Convention	Example
Arguments	appear in brackets and are formatted in a lowercase monospace font. Optional arguments are formatted in italic font. Predefined arguments are formatted in an uppercase font. Vertical bars separate arguments from which you may choose only one.	<code>{name, 0, value}</code> <code>{ACCTID 2 5000}</code>
Environment variables	are formatted in an initial uppercase font enclosed in angle brackets.	<code><Envfile>=\${APPDIR}</code>
Literals and parameters	are formatted in a monospace font.	<code>class extendSample</code> and the <code>LOCALLU</code> parameter
Programs, applications, files, and subsystem names	are formatted with uppercase font in text, but are in monospace uppercase or lowercase in code examples.	Use the DMCONFIG file and the CLASS LIBRARY . An <code>aix.env</code> file example is included with your product software
User input	are formatted in a monospace font.	Type <code>cd TUXDIR</code>
Glossary terms	are formatted in italics in the printed copy. Online glossary terms are in italics, color, and may be underlined, depending on your browser settings.	<i>Server1</i> is the host name of a BEA Connect server. <i>Server1</i> is the host name of a Jolt server.

Related Documentation

Connect SNA Documentation

The Connect SNA documentation consists of the following items:

BEA Connect SNA User Guide

BEA Connect SNA 2.0 Release Notes

Product Manuals

TUXEDO Product Overview

TUXEDO /Domain Guide

TUXEDO Application Development Guide

TUXEDO System 6 Reference Manual

TUXEDO System 6 Programmer's Guide, Volumes 1 and 2

Other Publications

The TUXEDO System (Andrade, Carges, Dywer, Felts)

TUXEDO: An Open Approach to OLTP (Primatesta)

Building Client/Server Applications Using TUXEDO (Hall)

Contact Information

Documentation Support

If you have questions or comments on the documentation, you can contact the BEA Information Engineering Group by e-mail at **docsupport@beasys.com** or by telephone at **+1.408.542.4193**. (For information on how to contact Technical Support, refer to the following section.)

Technical Support

If you have a question you cannot answer using this guide or the *BEA Connect SNA Release Notes*, you can contact BEA Connect SNA Technical Support.

When contacting technical support, be prepared to provide the following information:

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

In the USA, you can call BEA Connect SNA Technical Support at **+1.888.BEA.SUPT (+1.888.232.7878)** or **408.743.4070** between 5:00am and 6:00pm Pacific time. Extended Support Customers may call anytime.

You may also send questions to BEA Connect SNA Technical Support by fax to **+1.408.743.4071** or by e-mail to **support@beasys.com**.

1 Introducing BEA Connect SNA

Features

BEA Connect SNA provides a bidirectional link enabling BEA TUXEDO and IBM mainframe applications to interact as either a client or server, using a *Customer Information Control System/Extended System Architecture (CICS/ESA) Distributed Program Link (DPL)*, *Information Management System (IMS) implicit LU6.2*, or any IBM-supported *Application Program-to-Program Communication (APPC)* or *Common Programming Interface for Communications (CPI-C)* interface.

BEA Connect SNA version 2.0 offers significant enhancements to prior versions. In summary, these include:

- ◆ New Process Architecture incorporating SNA Communication Resource Manager (SNACRM) for *InterSystem Communications (ISC)* connectivity along with a Process Initialization Server
- ◆ Sync-level 2 support enabling transactions between Tuxedo and a mainframe
- ◆ Added support for CICS/ESA ISC DPL
- ◆ Additional logging and trace files

BEA Connect SNA 2.0 is supported on the following UNIX platforms:

- ◆ HP-UX 10.20 w/ HP SNAplus2 5.1
- ◆ Solaris 2.5.1 w/ SUN Link 9.1
- ◆ AIX 4.2.1 w/ IBM Comm Server 5.1

Functions

BEA Connect SNA performs the following functions:

- ◆ It supports the *Application-Transaction Monitor Interface (ATMI)* request/response model. Applications running in local Tuxedo System /T domains can request services in a remote domain. The remote domain is a MVS/CICS region. Applications in the remote domain can request services in a local TUXEDO System domain.
- ◆ It also supports the ATMI conversational model. Applications can establish a conversation between clients and servers in a Tuxedo System /T domain and an MVS/CICS region defined as a remote domain. Either side can initiate the conversation.
- ◆ It performs administration and configuration of a BEA Connect SNA gateway through the DMADM server and DMCONFIG server for a /SNA Domains instantiation.
- ◆ It supports the following typed buffers: FML, FML32, VIEW, VIEW32, X_COMMON, STRING, C_ARRAY, and X_OCTET. All buffer types except C_ARRAY and X_OCTET are converted automatically to and from EBCDIC between Tuxedo and CICS; C_ARRAY and X_OCTET are not converted.
- ◆ It maintains connectivity to multiple *System Network Architecture (SNA)* networks from the local Tuxedo System /T domain.
- ◆ It performs conversion between ASCII and EBCDIC encoding.

2 Understanding How BEA Connect SNA Works

Overview of Domain Gateways

The BEA Domains gateway architecture grew out of a need for organizations to share services and data among computer applications while enabling application and administrative autonomy within single segments of the business. For example, a bank could have a number of separate applications, such as a loan application, a retail application, and so forth. Each application could span one or more computer nodes and could be administered independently from other applications.

The *Domain*, a separately administered application environment that is interconnected with other domains by gateways, was developed to address this need. To provide a common functional framework for gateways interconnecting individual domains, BEA developed the *Domain Gateway* architecture. This architecture specifies the common functionality and administrative interfaces needed for gateways to perform inter-domain communications. You can find a more detailed discussion of the BEA Domains gateway architecture in *BEA TUXEDO Domain User Guide*.

The BEA Connect family of products provides domain-compliant gateways that permit administration of the remote *Transaction Processing (TP)* system as a foreign domain. The BEA Connect SNA and BEA Connect OSI TP gateways are operating examples.

The Domains Gateway architecture extends the scope of BEA TUXEDO to provide coordinated transaction processing across an enterprise's geographic or organizational boundaries. Within each domain, the administrator determines which local services are available to other specific domains, thus enabling client applications to request those services.

The Domains Gateway architecture is aimed at the TUXEDO application administrator, who makes services in other domains available to application programmers. The existence of applications within distinct domains is, however, totally transparent to the application programmers. They can use TUXEDO programming paradigms to request services offered in other BEA Domains exactly as if they were services offered within the local application.

The TUXEDO application administrator enables remote domains to access a subset of *Local Services*. This subset is called a *Local Domain*. The local domain helps the administrator provide secure “views” of the application.

Furthermore, the administrator can restrict access to local services through an Access Control List (ACL). The list designates which remote domains are allowed to issue requests to a particular local service.

Here are common features shared by all BEA Domains-compliant gateways:

- ◆ *X/Open XATMI*. Application programs using BEA TUXEDO can request services from applications running in another domain. Also, remote applications can request services from local servers.
- ◆ *Administration*. BEA Domains-compliant gateways can be booted or shutdown exactly like any other BEA TUXEDO server. Run-time administration is provided, and administrators can dynamically change the configuration to tune the performance of a gateway. Each compliant gateway supports a common set of administrative Application Programming Interface (API), tools, and configuration files.
- ◆ *Multi-domain interaction*. BEA Domains-compliant gateways can inter-operate with multiple domains at the same time.
- ◆ *Transparent multi-network support*. BEA Domains-compliant gateways can communicate with remote domains using different types of networks transparently to the application.

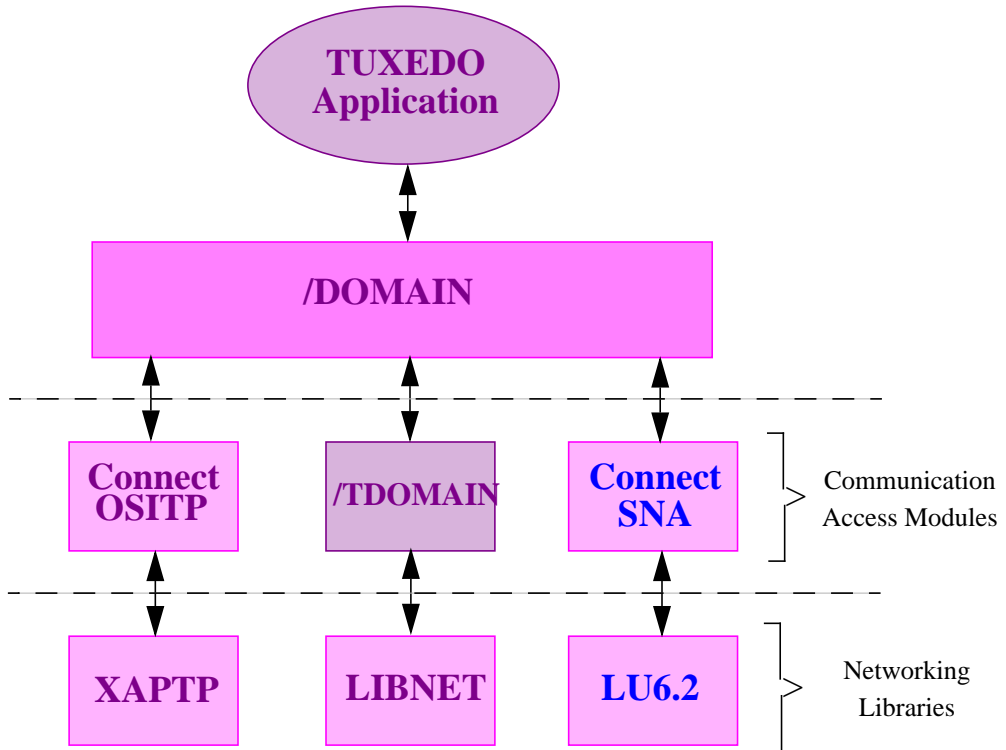
- ◆ *Service names aliases.* This permits the administrators to map service names used by the remote application to service names used by the local application. By using this feature, administrators can easily integrate applications that use different naming schemes.
- ◆ *Data dependent routing.* Administrators can use the configuration capability to define data dependent routing criteria for remote domains. This feature allows the gateway to select a remote domain according to a field value defined in the request (for example, an account number).

The BEA Connect SNA Domain

The BEA Connect SNA Domain, shown in Figure 2-1, is the gateway for communications using LU6.2 sessions and conversations to a CICS/ESA region. It adds multi-domain connectivity, bidirectional request/response, and conversations between Tuxedo and SNA-based applications. It provides access to SNA based APPC applications as well as inter-system communication with CICS/ESA. When accessing CICS/ESA systems, the SNA domain acts as a CICS/ESA region, capable of supporting the following sync level 2 functions:

- ◆ Bidirectional synchronous, asynchronous, and conversational service requests
- ◆ Peer support for CICS and support for multiple remote MVS regions
- ◆ Event monitoring and reporting
- ◆ Both static and dynamic configuration support where allowed by SNA
- ◆ Automatic Data conversion between Unix and Host formats
- ◆ Security through Access Control Lists on the local domain
- ◆ User and password security for CICS communications

Figure 2-1 BEA TUXEDO System /Domain Architecture



BEA Connect SNA allows the TUXEDO System clients and servers to interact with clients and servers in a non-TUXEDO, remote SNA domain. The remote and local domains must support SNA LU6.2 protocols in order to inter-operate. A set of SNA sessions of type LU6.2 over which the local and remote domains communicate are established. When two entities (in this case local and remote domains) communicate via an LU6.2 session, a conversation is allocated to the session. The gateway and CICS program have exclusive use of the session during the conversation. When the conversation is de-allocated, the session becomes available for use by another client/server pair.

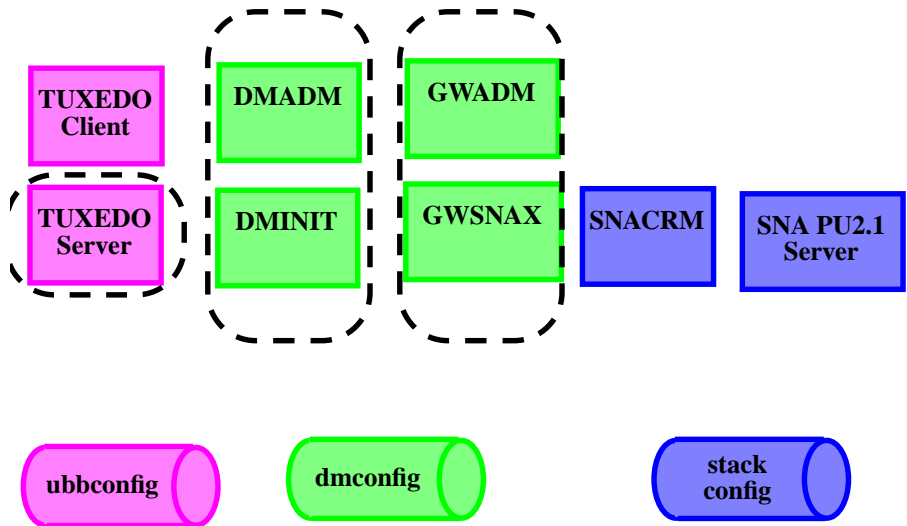
For each client/server communication model, non-conversational or conversational, parallel sessions are allocated for the LU6.2 conversation between the local and remote domains. The behavior of the applications at either end of a conversation depends on the type of client/server communication model in effect for the duration of the LU6.2 conversation. Refer to Chapter 7, “Programming Considerations” for more details about application behavior.

The CONNECT SNA domain is composed of several elements that can be configured to provide SNA solutions. For the most part, the SNA domain is much like other TUXEDO domain gateways; it uses the DMADM and GWADM servers for administration. With BEA CONNECT SNA, additional servers and processes support peer CICS/ESA connectivity and sync level 2. Figure 2-2 shows each component of the CONNECT SNA Product.

The DMINIT and SNACRM Servers

Connect SNA additionally provides the DMINIT server which can be used to start non-BEA TUXEDO servers at gateway group boot time. One such server is the SNA Communication Resource Manager (SNACRM). Both the SNACRM and the SNA PU 2.1 servers must be running, before any other servers are booted. Configuring the SNACRM server is explained in the “*DM_SNACRM Section” on page 4-3.

Figure 2-2 BEA Connect SNA Domain Components



LEGEND:

DMADM = Domain Administration Server

DMINIT = Initialization Server

GWADM = Gateway Administration Server

GWSNAX = Transactional/SNA Domains Instance

SNACRM = SNA Communications Resource Manager

SNA PU2.1 Server = Vendor-supplied SNA Stack

Other Gateway Servers

The SNA Domain Gateway (GWSNAX) uses the administrative servers provided with BEA TUXEDO software for domain and gateway configuration and administration. These are the Domain Administration Server (DMADM) and Gateway Administration Server (GWADM). Specific information about configuring the various sections of the SNA Domain Gateway can be found in the following:

- ◆ Chapter 4, “Configuring BEA Connect SNA”
- ◆ The `dmconfig` file description found in Appendix A, “Reference Pages” located in the back of this guide.
- ◆ *BEA TUXEDO/Domain Guide*

3 Installing BEA Connect SNA

Pre-Installation Considerations

The normal CONNECT SNA environment generally consists of two properly configured components, a local environment and a remote environment.

Local Environment

A local environment is a UNIX-based machine running CONNECT SNA 2.0. CONNECT SNA 2.0 is a fully bidirectional program, supporting the local system as either a client or server. This environment consists of the following components:

- ◆ Hardware, which is any workstation and network interface supported by the required software
- ◆ Operating system software supporting:
 - Solaris 2.5.1
 - HPUX 10.20
 - AIX 4.2.1

Local Environment (cont.)

- ◆ SNA protocol stacks (PU servers) supporting:
 - Sunlink 9.1
 - Brixton 4.0
 - HPSnaPlus2
 - IBM Communications Server for AIX

- ◆ Tuxedo 6.4 software

The platform on which CONNECT SNA is installed must have BEA TUXEDO 6.4 already installed. Stack installation should also be completed and verified before installing CONNECT SNA.

You should shut down all Domain administrative and gateway servers within an application domain prior to installing this product, particularly the following:

- ◆ GWTDOMAIN
- ◆ GWADM
- ◆ DMADM

Do not run the following commands until the installation is complete:

- ◆ dmadmin
- ◆ dmloadcf
- ◆ dmunloadcf

Remote Environment

A remote environment is an IBM mainframe that may or may not be on the same local network. As in the local environment, CONNECT SNA 2.0 is a fully bidirectional program, supporting the remote system as either a client or server. This environment consists of the following components:

- ◆ Hardware is any workstation and network interface supported by the required software.

Remote Environment (cont.)

- ◆ Software
 - ◆ MVS 5.22 9510 or higher, including OS/390 through 1.3
 - ◆ VTAM 4.3 or higher
 - ◆ CICS 3.3 or higher
 - ◆ IMS/ESA DC 5.01 (optional)
- ◆ Support software for a mainframe (optional)

Planning

Prior to installing BEA CONNECT SNA software, you must ensure the CICS/ESA remote domain is prepared to conduct operations with the TUXEDO local domain. This includes:

- ◆ Establishing the VTAM configuration
- ◆ Configuring the CICS/ESA Logical Unit (LU)
 - ◆ Creating connections
 - ◆ Defining sessions
- ◆ Completing cross-platform definitions
- ◆ Setting stack traces

Chapter 4, “Configuring BEA Connect SNA” gives more detailed information about these tasks.

Note: It is important to involve your mainframe system support personnel early in the process. In a large shop, separate individuals are likely to be responsible for MVS, VTAM, and CICS/ESA. Make sure everyone is involved.

Installation

When installing or upgrading BEA CONNECT SNA, you must run the `install.sh` script and enter the needed information as it progresses through the install process. This script installs all of the necessary components required to run the software.

Running `install.sh`

As you run the script, you will be asked for the following information:

- ◆ System on which to install BEA CONNECT SNA 2.0:
 - ◆ HP/HPUX 10.20
 - ◆ IBM/AIX 4.2.1
 - ◆ SUN 5.x/SOL 2.5.1
- ◆ Version of BEA CONNECT SNA to install
- ◆ Directory where BEA TUXEDO is installed (Defaults to the current directory if it is not specified)

Listing 3-1 gives an example of running this script. The values in bold are supplied by the user during installation. To accept default values at a prompt, press Enter.

Listing 3-1 install.sh Example

```

user@dalhpw1:/bea/loads/tmp/cmadm.dist4/sna20> sh ./install.sh

01) hp/hpux1020      02) ibm/aix421      03) sun5x/sol251

Install which platform's files? [01-3, q to quit, l for list]: 01

** You have chosen to install from hp/hpux1020 **

BEA Connect SNA Release 2.0 for
  BEA TUXEDO System Release 6.4.

This directory contains BEA Connect SNA for
HPUX 10.20 on HP PA-RISC.
Please refer to the product documentation for release notes and
further information.

Is this correct? [y,n,q]: y

To terminate the installation at any time
press the interrupt key,
typically <del>, <break>, or <ctrl+c>.

The following packages are available:

      1      sna20      BEA Connect SNA Release 2.0

Select the package(s) you wish to install (or 'all' to install all
packages) (default: all) [?,??,q]: 1

BEA Connect SNA
(9000) Release 2.0
Copyright (c) 1998 BEA Systems, Inc.
Portions * Copyright 1986-1998 RSA Data Security, Inc.
All Rights Reserved.
Distributed under license by BEA Systems, Inc.
TUXEDO is a registered trademark.

TUXEDO directory (TUXDIR) where files are to be installed [?,q]:
/bea/work/jdahl/tuxedo

Determining if sufficient space is available ...
48212 blocks are required
1592650 blocks are available to /bea/work/jdahl/tuxedo

```

Using /bea/work/jdahl/tuxedo as the SNA base directory

Unloading /bea/loads/tmp/cmadm.dist4/sna20/hp/hpux1020/sna/sna.Z

```
...
bin/DMINIT
bin/GWSNAX
bin/Xsnacrm
bin/CRMLLOGS
bin/xsnacrm
bin/SNACRM
connect/sna/simpapp/BEACONN.RDO
connect/sna/simpapp/BEASNA.RDO
connect/sna/simpapp/DMCONFIG
connect/sna/simpapp/MIRRDTPC.c
connect/sna/simpapp/MIRRDTPC.cbl
connect/sna/simpapp/Makefile
connect/sna/simpapp/README
connect/sna/simpapp/TOUPDPLS.c
connect/sna/simpapp/TOUPDPLS.cbl
connect/sna/simpapp/TOUPDTPS.c
connect/sna/simpapp/TOUPDTPS.cbl
connect/sna/simpapp/UBBCONFIG
connect/sna/simpapp/aix.env
connect/sna/simpapp/app.env
connect/sna/simpapp/dminit.scr
connect/sna/simpapp/hpux.env
connect/sna/simpapp/mirrorsrv.c
connect/sna/simpapp/solaris.env
connect/sna/simpapp/toupclt.c
lib/libctxmess.a
lib/libctxdebugs.a
lib/libctxos.a
lib/libctxprim.a
lib/fmb.def
lib/libgws.a
lib/libcsxappc.a
lib/libcsxxxfm.a
lib/libcsxxmw.a
lib/libcsxgpw.a
lib/libcsxcrm.a
lib/libcsxxcrm.a
lib/libcsxscrm.a
lib/libcsxhp51.a
lib/libctxmess.sl
lib/libctxdebugs.sl
```

```
lib/libctxos.sl
lib/libctxprim.sl
lib/libgws.sl
lib/libcsxappc.sl
lib/libcsxxfm.sl
lib/libcsxxmw.sl
lib/libcsxgpw.sl
lib/libcsxcrm.sl
lib/libcsxxcrm.sl
lib/libcsxscrm.sl
lib/libcsxhp51.sl
locale/C/LIBGWS_CAT
locale/C/LIBGWS.text
47580 blocks
... finished
```

```
Changing file permissions...
... finished
```

Installation of BEA Connect SNA was successful

The following packages are available:

1	sna20	BEA Connect SNA Release 2.0
2	snaDOC	BEA Connect SNA Documentation

Select the package(s) you wish to install (or 'all' to install all packages) (default: all) [?,?,q]: **q**

Please don't forget to fill out and send in your registration card

Installation Components

When you run the `install.sh` script, it installs all of the following BEA CONNECT SNA components.

Executables

This script installs the following BEA CONNECT SNA executables the `./bin` directory:

```
DMINIT  
GWSNAX  
Xsnacrm  
CRMLOGS  
xsnacrm  
SNACRM
```

Application Files

This script installs the following BEA CONNECT SNA application files into the `./connect/sna/simpapp` directory:

```
BEACONN.RDO  
BEASNA.RDO  
DMCONFIG  
MIRRDTPC.c  
MIRRDTPC.cbl  
Makefile  
README  
TOUPDPLS.c  
TOUPDPLS.cbl  
TOUPDTPS.c  
TOUPDTPS.cbl  
UBBCONFIG  
aix.env  
app.env  
dminit.scr  
hpux.env  
mirrorsrv.c  
solaris.env  
toupclt.c
```

Libraries

This script installs the following BEA CONNECT SNA libraries into the `./lib` directory:

```
libctxmess.a  
libctxdebugs.a  
libctxos.a  
libctxprim.a
```

```
fmb.def
libgws.a
libcsxappc.a
libcsxxfm.a
libcsxxmw.a
libcsxgpw.a
libcsxcrm.a
libcsxxcrm.a
libcsxscrm.a
libcsxhp51.a
libctxmess.sl
libctxdebugs.sl
libctxos.sl
libctxprim.sl
libgws.sl
libcsxappc.sl
libcsxxfm.sl
libcsxxmw.sl
libcsxgpw.sl
libcsxcrm.sl
libcsxxcrm.sl
libcsxscrm.sl
libcsxhp51.sl
```

Message Queues

This script installs the following BEA CONNECT SNA message queue files into the `./locale/C` directory:

```
LIBGWS_CAT
LIBGWS.text
```

Post Installation Action

After installing Connect SNA 2.0, modify your existing DMTYPE file, located in the `$TUXDIR/udataobj` directory, to contain the following line:

```
SNAX:-lgw -lgws:-lgp -lgpnet -lcsxcrm::
```


4 Configuring BEA Connect SNA

This chapter is divided into two parts: Local Domain Configuration and Remote Host Domain Configuration.

Local Domain Configuration

There are four aspects to configuring a BEA CONNECT SNA local domain:

- ◆ Edit the UBBCONFIG file to define the Connect SNA administrative and gateway servers to the BEA TUXEDO system. (The DMINIT server can be used to start the SNACRM process.) Use this method to establish a new gateway configuration or modify an existing one.
- ◆ Edit the DMCONFIG text file and process it with the `dmloadcf` utility. Use this method to establish a new gateway configuration.
- ◆ Use the configuration part of the `dmadmin` command. This gives you access to the binary BDMCONFIG file. Use this method to change an existing gateway configuration. Refer to Appendix A, “Reference Pages” for more information about the `dmadmin` command.
- ◆ Perform a cold start on the SNA domain after making changes to the `*DM_SNACRM`, `*DM_SNASTACKS`, or `*DM_SNA_LINKS` sections. If you do not cold start the SNA domain, an error will occur on domain start-up indicating cold start required for the configuration change.

Editing the UBBCONFIG File

Refer to the *BEA TUXEDO Reference Manual* for detailed information about this subject.

Editing the DMCONFIG File

The configuration specified in the DMCONFIG file controls much of the operation of the BEA CONNECT SNA gateway. A sample of this file is provided in the installation directory of your BEA CONNECT SNA product software.

Note: Because BEA CONNECT SNA may be installed on a variety of platforms, the procedures in this section make only general references to command entries. Many steps show UNIX command examples. Be sure to use the proper syntax for your platform when making command-line entries.

Prerequisites

The BEA CONNECT SNA product software must be installed and accessible to your text editor. You must have file permission to access the `install` directory and modify the sample DMCONFIG file. In addition, the following prerequisites must be met to successfully complete the editing procedure:

- ◆ The `$TUXDIR/udataobj/DMTYPE` file defining the valid domain types must exist (see `dmloadcf` in Appendix A, “Reference Pages”).
- ◆ The effective user identifier of the person running `dmloadcf` must match the UID in the `RESOURCES` section of the TUXCONFIG file (refer to `dmloadcf` in Appendix A, “Reference Pages”).

The following paragraphs describe the significant parameters within specific sections of the DMCONFIG file.

*DM_LOCAL_DOMAINS Section

In this section, the value for the `TYPE` parameter distinguishes this gateway from other gateway types. Currently, `SNAX` replaces the value `SNADOM` used in previous releases. The parameter entry takes the form:

```
TYPE={SNAX | OSITP | TDOMAIN}
```

Select the value `TYPE=SNAX` for this entry.

***DM_REMOTE_DOMAINS Section**

In this section, the value for the `TYPE` parameter indicates that the remote domain communicates using the SNA protocol. The parameter entry takes the form:

```
TYPE={SNAX | OSITP | TDOMAIN}
```

Select the value `TYPE=SNAX` for this entry.

***DM_SNADOM Section**

This section has been replaced with the following sections, each of which is described in subsequent paragraphs:

- ◆ `*DM_SNACRM` section
- ◆ `*DM_SNASTACKS` section
- ◆ `*DM_SNA_LINKS` section

Note: Any changes to the `*DM_SNACRM`, `*DM_SNASTACKS`, or `*DM_SNA_LINKS` sections require a cold start for the SNA domain. If you do not cold start the SNA domain, an error will occur on domain start-up indicating cold start required for the configuration change.

***DM_SNACRM Section**

The `DM_SNACRM` section provides three keywords used to identify the SNA Communications Resource Manager that will provide ATMI transaction semantics between a given domain and its partners. Entries have the general form:

```
<SNA CommunicationsResourceManagerName> parameters
```

Where `<SNA CommunicationsResourceManagerName>` is the locally known name of this `SNACRM` definition to be used when referencing this `SNACRM` in subsequent sections. This name is an ASCII string 1-30 characters in length. The parameters are the keyword/value pairs that makeup the definition. All keywords are required for a valid `SNACRM` definition. Keywords can be in any order.

LDOM <LOCALDOMAINNAME> (REQUIRED)

LDOM associates this SNACRM with a defined local domain. <LocalDomainName> is the reference to an entry in the *DM_LOCAL_DOMAINS section. This name is an ASCII string 1 to 30 characters in length. This parameter is required. This parameter has no default.

NWDEVICE <DEVICENAME>

This is the network device to be used for communication between the SNA gateway and the SNACRM. <DeviceName> should be set to

`/dev/tcp.`

SNACRMADDR <HEXSOCKETADDRESS> (REQUIRED)

SNACRMADDR provides the socket address the domain gateway will use to communicate with the SNACRM. If the SNACRM is started independently of the domain gateway, this address must be used on the SNACRM command line. This parameter is required. This parameter must contain an even number of hex characters. This parameter has no default.

<HexSocketAddress> is a TCP/IP address using the sockaddr_in format of family,port,address:

`<0xFFFFPPPPAAAAAAA> or //hostname:port_addr`

Where:

FFFF is the hex value of the protocol family, always 0x0002 for the INET family.

PPPP is the hex value of an unused TCP/IP port

AAAAAAA is the hex value of the IP address for the machine running the SNACRM

Therefore if the SNACRM was running on a machine named `myhost` with an IP address of 206.189.43.13, and you wanted to use port 6000 for the SNACRM, then SNACRMADDR would be:

`0x00021770CEBD2B0D or //myhost:6000`

***DM_SNASTACKS Section**

The `*DM_SNASTACKS` section provides five keywords which identify the third party SNA stack that should be used for connections established between a given domain and its partners. Entries have the general form:

```
<StackReference> parameters
```

Where `<StackReference>` is the locally known name of this stack definition to be used when referencing this stack in subsequent sections. This name is an ASCII string 1-30 characters in length. The parameters are the keyword/value pairs that make up the definition. Keywords can be in any order. All keywords are required for a valid stack definition.

LOCALLU <LOCALLUALIAS> (REQUIRED)

`LOCALLU` provides a reference to an LU alias defined in the third party SNA stack. `<LocalLUAlias>` is the name used to identify the local LU definition as specified by the third party SNA stack configuration. This is a name that represents the end node for an LU6.2 connection. The value for this parameter is an ASCII string, 1-8 characters in length. This parameter is required. This parameter has no default. The third party SNA stack will require a corresponding definition for a local LU.

LTPNAME <LOCALTRANSACTIONPROGRAMNAME> (REQUIRED)

`LTPNAME` identifies the inbound transaction programs which will be serviced by any `SNACRM` using this stack definition. `<LocalTransactionProgramName>` is the name used to identify inbound transaction programs for which an attach will be accepted. The only useful value is an asterisk. This indicates all inbound attaches will be accepted. This parameter is required. This parameter has no default. Partial TP names are not supported. The third party SNA stack will require a corresponding definition for inbound TP names.

SNACRM <SNACOMMUNICATIONSRESOURCEMANGERNAME> (REQUIRED)

`SNACRM` provides a name to which the associated `SNACRM` definition is referenced. `<SNACommunicationsResourceMangerName>` is the name used to associate the `DM_SNACRM` definition with this `DM_SNASTACKS` entry. The value for this parameter is an ASCII string, 1-30 characters in length. This parameter is required. This parameter has no default.

**STACKPARMS <PARAMETERS REQUIRED FOR THIRD PARTY SNA STACK>
(REQUIRED)**

STACKPARMS provides a method for the domain gateway to pass any required parameters to the third party SNA stack. The <parameters required for third party sna stack> is an ASCII string, 1-128 characters in length. Currently, the only value used is the TCP/IP `hostname` for the machine running the third party SNA stack. This parameter is required. This parameter has no default.

STACKTYPE={SUN91 | HP51 | IBM51}

This option is used to indicate which vender SNA stack is being used. It is also used to determine the names of specific CONNECT SNA system libraries. Because of this, it is essential that the value of this option be coded correctly. These values are mapped to the equivalent CONNECT SNA system library.

***DM_SNA_LINKS Section**

This section defines the SNA Link information required by domains of type SNA.

STACKREF = STRING (REQUIRED)

This required parameter defines the stack that will be used for establishment of this link. The `STACKREF` string is the tag that was used in the corresponding definition established in the `*DM_SNASTACKS` section.

RDOM=NAME

Each link defines a connection between a BEA TUXEDO System application domain and a remote system connected with an SNA network. The remote system is, in BEA TUXEDO terms, a remote domain. The `RDOM` option associates the link with a remote domain. This remote domain must have been configured with the `TYPE=SNAX` option. The `RDOM` name should match a `RDOM` value previously identified in the `*DM_REMOTE_DOMAINS` section.

LSYSID=NAME

[Reserved For Future Use.] **LSYSID** is the four-character identifier for this link. This should match the connection ID in the CICS/ESA resource definition used by a partner CICS/ESA to communicate to the **SNACRM** across this link. If you are using the macro definition, this is a four-character name on the **SYSIDNT** option of the **DFHTCT** macro.

RSYSID=NAME

[Reserved For Future Use.] **RSYSID** is the four-character remote sysid of the partner. Typically it is the sysid of a CICS/ESA region, but could also be the subsystem ID of an IMS control region. This parameter must match the actual sysid of the remote partner. This name is the **SYSIDNT** of the **DFHSIT** or the value in the CICS/ESA start-up overrides

RLUNAME=NAME (REQUIRED)

The **RLUNAME** value represents an alias known to the third party SNA stack that resolves to a VTAM **netname** for the remote application. This would most likely be the VTAM **applid** for a CICS/ESA region, however it could also be an APPC/MVS LU defined for use with IMS. The value must be unique within the SNA network. The value *name* should be 1-8 characters. This parameter is required. This parameter has no default. The third party stack configuration requires a matching definition.

MODENAME=NAME (REQUIRED)

MODENAME is VTAM mode entry defined to the third-party SNA stack. For a CICS/ESA link, this must be compatible with the session definition or profile entry for the corresponding connection. For an IMS connection, this must be compatible with the **DLOGMOD** entry on the LU definition used to access the IMS scheduler. The value *name* should be 1-8 ASCII characters. This parameter is required. This parameter must match the third-party SNA stack configuration and must be compatible with the corresponding entries defined to VTAM and/or CICS/ESA.

SECURITY={LOCAL, IDENTIFY, VERIFY, PERSISTENT, MIXIDPE}

SECURITY specifies the security setting in CICS/ESA connection resource definition. It identifies the level of security enforced under CICS/ESA by the external security manager. Legal values are LOCAL, IDENTIFY, VERIFY, PERSISTENT or MIXIDPE. The default setting is LOCAL. PERSISTENT and MIXIDPE identify the setting in the remote connection definition, but are identical to the VERIFY option in this release of CONNECT SNA.

MAXSESS=NUMBER

This number represents the maximum number of sessions that can be concurrently acquired on this link. It must be greater than or equal to four, and less than or equal to the maximum number of sessions that can be configured by the SNA stack. The actual number of concurrent sessions is determined by both system configurations to be the lowest maximum number of sessions allowed by either system.

MINWIN=NUMBER

This minimum number of contention winners. Typically, this value is half the MAXSESS value. This number added to all CICS/ESA session definition winner numbers for the connection should be equal to the MAXSESS value.

MAXSYNCLVL={0, 1, 2}

This represents the maximum sync-level conversation that can be supported on this link. The default is sync-level 2. If the installation is not licensed for sync-level 2, this parameter must be set to 0 or 1 for the link to be established. Transaction support is only available at sync-level 2.

- | | |
|--------------|--|
| Sync-level 0 | A value of zero (0) means this link is non-transactional. No synchronization is maintained. This can be used for sending and receiving messages from IMS via the APPC/MVS transparency interface. |
| Sync-level 1 | Allows Sync-level 0 capabilities as well as support for SYNCONRETURN <i>Distributed Program Link (DPL)</i> with CICS/ESA systems (outbound ATMI <code>tpcall()</code> requests with <code>TPNOTRAN</code>). |

Sync-level 2

Supports all sync-level 0 and sync-level 1 features for systems able to exchange logs and compare states. In addition, full syncpoint synchronization at sync-level 2 is supported.

Also supports:

Full transaction coordination and recovery between BEA TUXEDO and CICS/MVS systems. This includes transaction support between BEA TUXEDO services and CICS APPC applications as well as DPL.

Outbound ATMI `tpcall()` as a DPL request with full two phase commit transaction semantics using `tpcommit()`.

Outbound ATMI `tpconnect()` as APPC or CPI-C *Distributed Transaction Processing (DTP)* with full commit transaction semantics using `tpbegin()` and `tpcommit()`.

Inbound EXEC CICS LINK requests with full two phase commit transaction semantics using Prepare, Rollback, and Syncpoint verbs.

Inbound APPC or CPI-C conversations with full two phase commit transaction semantics using Prepare, Rollback, and Syncpoint verbs.

Caution: If you set `MAXSYNCLVL=2` or make no entry for this parameter (that is, accept the default) without having installed the CONNECT SNA software licensed for that level, the system configuration automatically reverts to sync-level 1 and an error message is sent to the error log. To clear that error message, you must either reset the `MAXSYNCLVL` parameter to an appropriate value or purchase and install the correct software.

STARTTYPE={WARM|COLD}

This option sets the recovery mode for transactional links. When set to `WARM`, the system restarts using configuration and link data recovered from the in-flight transaction log. When set to `COLD`, the system uses configuration data taken from the current `DMCONFIG` file and loses any in-flight link data. Changing `DMCONFIG` file parameters and performing a `WARM` start results in a message warning that changed parameters are ignored until the next cold start. To force a cold start and disregard the `STARTTYPE` setting, delete the `SNA*LOG` files in `$APPDIR`.

DM_REMOTE_SERVICES Section*FUNCTION={APPC | DPL}**

The **FUNCTION** option has been added to allow outbound Tuxedo service requests to map to APPC transaction programs or CICS/ESA programs. The default value **APPC** indicates the remote service is a transaction program that may or may not be running under CICS/ESA. The **DPL** value indicates the remote service maps to a program running under CICS/ESA.

RNAME=NAME

The **RNAME** option is the name of the host **TP_NAME**. For non-CICS/ESA systems, this name can be up to 64 characters in length. For CICS/ESA systems, this name is the transaction ID for **FUNCTION=APPC** and the program name for **FUNCTION=DPL** requests. CICS/ESA trans-id names cannot exceed four characters and CICS/ESA program names cannot exceed eight characters. The **RNAME** option must observe these requirements.

Using the dmadmin Command Interpreter

The **dmadmin** command is an interactive command interpreter for the administration of domain gateway groups defined for a particular TUXEDO System/T application. The **dmadmin** command enters configuration mode when you execute it with the **-c** option, or when you use the **config** subcommand. In this mode, **dmadmin** performs run-time updates to the **BDMCONFIG** file. You can use any text editor available on your platform to make **dmadmin -c** changes.

This subsection provides quick references to changes you can make to **CONNECT SNA**-related sections of the **DMCONFIG** file. Refer to Appendix A, “Reference Pages” for details about using the **dmadmin** command.

Note: The ***DM_SNA** section has been replaced with the ***DM_SNALINKS**, ***DM_SNASTACKS**, and ***DM_SNACRM** sections.

Configuring the *DM SNACRM SECTION

The following table lists the fields in the *DM_SNACRM section which you may need to update for your configuration.

Table 4-1 *DM_SNACRM Field Updates

Field Identifier	Field Type	Update	Notes
TA_LDOM	string	NoGW	
TA_NWDEVICE	string	NoGW	
TA_SNACRMADDR	hex-string	NoGW	

Configuring the *DM_SNASTACKS Section

The following table lists the fields in the *DM_SNASTACKS section which you may need to update for your configuration.

Table 4-2 *DM_SNASTACKS Field Updates

Field Identifier	Field Type	Update	Notes
TA_SNACRM	string	NoGW	
TA_STACKTYPE	string	Yes	sun91 hp51 ibm51
TA_LOCALLU	numeric	Yes	Local SYSID
TA_LTPNAME	string		wildcard for local tpname
TA_STACKPARMS	string		Parameters for vender stack

Configuring the *DM_SNALINKS Section

The following table lists the fields in the *DM_SNALINKS section which you may need to update for your configuration.

Table 4-3 *DM_SNALINKS Field Updates

Field Identifier	Field Type	Update	Notes
TA_STACKREF	string	NoGW	
TA_RDOM	string	Yes	Key
TA_LSYSID	numeric	Yes	Local SYSID
TA_RSYSID	string	NoGW	Remote SYSID
TA_RLUNAME	string	NoGW	as known to VTAM
TA_MODENAME	string	NoGW	for the sessions
TA_SECURITY	string	NoGW	LOCAL, IDENTIFY, VERIFY, PERSISTENT, MIXIDPE
TA_STARTTYPE	string	NoGW	Auto Cold
TA_MAXSESS	numeric		Maximum number of concurrent APPC sessions
TA_MINWIN	numeric	NoGW	Minimum contention winners
TA_MAXSYNCLVL	numeric	NoGW	Maximum conversation sync level for link

Remote Host Domain Configuration

You must ensure the CICS/ESA remote domain is prepared to conduct operations with the TUXEDO local domain. This includes:

- ◆ Establishing the VTAM configuration
- ◆ Configuring the CICS/ESA Logical Unit (LU)
 - ◆ Creating connections
 - ◆ Defining sessions
- ◆ Completing cross-platform definitions
- ◆ Setting stack traces

Note: Any samples provided illustrate a starting point for configuring your system and do not represent all possibilities. The samples represent one way a mainframe can be configured to work in an APPN LAN environment.

Establishing the VTAM Configuration

If your CONNECT SNA is used in a *Virtual Telecommunications Access Method (VTAM)* environment, you must ensure the host configuration supports it. Refer to Appendix E, “Sample VTAM Configurations” for some examples based on the requirements for BEA CONNECT SNA to be used in a VTAM environment with an Ethernet LAN and an APPN mainframe system.

Configuring the CICS/ESA LU

Before you can connect to the remote stack, the LU configuration must be established. This entails creating connection definitions, creating session definitions, and installing resource definitions.

Create Connections at the Remote Host

If it is not already in place, you must work with the mainframe support personnel to create a remote connection definition file. When placed on the remote host, the definition provides a connection with the TUXEDO local domain. It corresponds to the LSYSID in the local domain's DMCONFIG file. The following is an example of a connection definition file:

```
DEFINE CONNECTION(BEA)          GROUP(BEACONN)
    DE(CONNECT SNA EXAMPLE RDO CONNECTION)
    ACCESSMETHOD(VTAM)          PROTOCOL(APPC)
    NETNAME(**VTAM NETWORK NAME OF REMOTE SYSTEM**)
    ATTACHSEC(LOCAL)            AUTOCONNECT(NO)
```

To install the sample connection definition, put it in on the host in a separate group which does not contain existing connection. Use the CEDA INSTALL command, for example:

```
CEDA INSTALL I GROUP(BEACONN)
```

Define the Session at the Remote Host

If it is not already in place, you must work with the mainframe support personnel to create a session definition. When placed on the remote host, it defines the logical links by which the TUXEDO local domain communicates with the remote host. The following is an example of a session definition:

```
DEFINE SESSION(BEATEST)         GROUP(BEACONN)
    CONNECTION(BEA)
    DE(CONNECT SNA EXAMPLE RDO SESSION)
    PROTOCOL(APPC)               AUTOCONNECT(YES)
    MODENAME(**MODE**)           MAXIMUM(**SESSNBR**, **WINNER**)
```

AUTOCONNECT indicates how the activation of the session is to be negotiated. YES enables the CICS/ESA host to negotiate its own winner sessions when a conversation is allocated.

The MODENAME can be either a CICS/ESA-supplied mode name, such as SMSNA100, or with your own defined mode name. If another set of session definitions exist for the BEA connection, this mode name must be unique among all sets defined to the connection. The mode name corresponds to the VTAM LOGMODE name.

The MAXIMUM option defines the total number of sessions in the set and the total number of winner sessions. The total number of winner sessions must include those for the host and the remote stack. The WINNER number plus the remote WINNER number should equal the SESSNBR. The local SESSNBR must equal the remote SESSNBR,

View Connection and Session Status

Once you have installed group definitions, you can view the status of connections and sessions using the following CICS/ESA system commands:

CEMT I CONN(BEA)	**view the status of the connection
CEMT I NET(**Netname**)	**View the status of the sessions
CEMT I MODENAME(**MODE**)	**View the status of the mode

Completing Cross-Platform Definitions

It is important to communicate with the administrator of the CICS/ESA remote domain to obtain key parameters in the VTAM definition that must be included in the SNA stack configuration, as well as in other configuration files in the CONNECT SNA local domain.

Before installing CONNECT SNA software, please examine Table 4-4 for a summary of cross-platform definitions. Consult with the VTAM system administrator to obtain the value indicated in the Name column and make the corresponding entries shown in the Needed In column.

Table 4-4 Summary of Cross-Platform Definitions

Item	Name	Originates In	Needed In
1.	SNA Network ID (e.g., SNANET1) and VTAM Host ID (e.g., VTAMHOST)	VTAM configuration	SunLink SNA Stack Configuration: Example: CP NQ_CP_NAME=SNANET1.SPARC1 DLC RMTNQ_CP_NAME=SNANET1.VTAMHOST LU NQ_LU_NAME=SNANET1.L0P0024A PTNR_LU NQ_LU_NAME=SNANET1.CICSSYN SNApplus2 or IBM CS/AIX Stack Configuration: Example: fgcp_name=SNANET1.SPARC1 adj_cp_name=SNANET1VTAMHOST fgplu_name=SNANET1.CICSSYN
2.	Mode Name (e.g.,SNA62)	VTAM-MODEENT definition	CICS Sessions Definition: Example: MODENAME(SNA62) SunLink SNA Stack Configuration: Example: MODE NAME=SNA62 SNApplus2 Stack Configuration: Example: mode_name=SNA62 TUXEDO Domain Configuration: Example: *DM_SNA_LINKS MODENAME="SNA62"

Table 4-4 Summary of Cross-Platform Definitions

Item	Name	Originates In	Needed In
3.	Control Point Name CPNAME (e.g.,SPARC1)	VTAM-PU definition	<p>SunLink SNA Stack Configuration: Example: CP NAME=SPARC1 CP NQ_CP_NAME=SNANET1.SPARC1</p> <p>SNaplus2 Stack Configuration: Example: fgcp_name=SNANET1.SPARC cd_alias=SPARC1</p>
4.	Local LU Name (e.g.,L0F0024A)	VTAM-LU definition	<p>CICS CONNECTION definition: Example: NETNAME(L0F0024A)</p> <p>SunLink SNA Stack Configuration: Example: LU NAME=L0F0024A LU NQ_LU_NAME=SNANET1.L0F0024A PTNR_LU LOC_LU_NAME=L0F0024A TP LOC_LU_NAME=L0F0024A</p> <p>SNaplus2 Stack Configuration: Example: lu_name=L0F0024A lu_alias=L0F0024A</p> <p>TUXEDO Domain Configuration: Example: *DM_SNA_STACKS LOCALLU="L0F0024A"</p>

Table 4-4 Summary of Cross-Platform Definitions

Item	Name	Originates In	Needed In
5.	CICS LU Name (e.g., CICSSYN)	VTAM-LU definition	<p>SunLink SNA Stack Configuration: Example: PTNR_LU NAME=CICSSYN PTNR_LU NQ_LU_NAME=SNANET1.CICSSYN MODE PTNR_LU_NAME=CICSSYN</p> <p>SNaplus2 Stack Configuration: Example: fgplu_name=SNANET1.CICSSYN plu_alias=CICSSYN</p> <p>TUXEDO Domain Configuration: Example: *DM_SNA_LINKS RLUNAME="CICSSYN"</p>
6.	Terminal Identifier (e.g.,05DF0024)	VTAM (IDNUM+IDBLK)	<p>SunLink SNA Stack Configuration: Example: DLC TERM << ALIGN=CHAR</p> <p>SNaplus2 Stack Configuration: Example: node_id=<05000002></p>
7.	Local Network Device (e.g.,/dev/tr)	UNIX Configuration	<p>SunLink SNA Stack Configuration: Example: TRLINE DEVICE='/dev/tr' or SDLCLINE DEVICE='/dev/dcp1'</p>
8.	Local MAC Address (token ring only)	Token ring address of Host	<p>SunLink SNA Stack Configuration: Example: TRLINE Source_Address=X'080020117d7a'</p>

Table 4-4 Summary of Cross-Platform Definitions

Item	Name	Originates In	Needed In
9.	Remote MAC Address (token ring only)	Token ring address of local machine	<p>SunLink SNA Stack Configuration: Example: DLC RMTMACADDR=X'40000101000'</p> <p>SNAPLUS2 Stack Configuration: Example: mac_address=<400031720001></p>
10.	LAN Speed (e.g.,4MBs)	Speed of token ring network	<p>SunLink SNA Stack Configuration: Example: TRLIN LAN_rate=RING_4MBS</p>
11.	SDLC parameters (line protocol)	VTAM-line definition	<p>SunLink SNA Stack Configuration: Example: SDLCLINE DUPLEX=half LINE=switched NRZI=no PAUSE=1 SPEED=4800</p>
12.	Partner Definition (e.g.,TUXPART1)	CICS/ESA	<p>Mainframe Client Application: (e.g., COBOL with embedded CPI-C to route CICS to appropriate LU for BEA Connect SNA)</p> <p>COBOL Example: MOVE 'TUXPART1' TO SYM-DEST-NAME CALL 'CMINIT' USING CONVERSATION-ID SYM-DEST-NAME, CM_RETCODE END-CALL</p>
13.	Set LU definition so maximum sync-level allowed corresponds to DMCONFIG entry: *DM_SNA_LINKS MAXSYNCLVL	Stacks	<p>SunLink SNA Stack Configuration: Example: SYNC_LVL=SYNCPT</p> <p>SNAPLUS2 Stack Configuration: Example: [define local_lu] Syncpt.Support=YES</p>

Table 4-4 Summary of Cross-Platform Definitions

Item	Name	Originates In	Needed In
14.	Map all incoming conversations to BEA Connect SNA gateway (make sure TPs have all privileges available, e.g., CNOS, syncpoint if licensed, service conversations, etc.).	Stacks	SunLink SNA Stack Configuration: Example: TP_HEXNAME=x'2a' SNApplus2 Stack Configuration: Example: Sna_tps <404040...hex representation of 64 EBCDIC spaces...404040> TYPE=QUEUED TIMEOUT=-1 USERID=authorized_user_here GROUP=authroized_group_here LUALIAS=lu_name_here
15.	CICS Transaction IDs (e.g.,TOUP)	CICS/ESA	TUXEDO Domain Configuration: Example: *DM_REMOTE_SERVICES

Setting Stack Traces

Consult the vendor publications for instructions on how to set up stack tracing.

5 Administering the BEA Connect SNA Application Domain

The topics in this section cover activities an administrator performs to maintain SNA applications. More details about these activities can be found in BEA TUXEDO documentation. Before attempting the activities, it is essential that you become familiar with BEA TUXEDO configuration procedures.

Administration Facilities

The administrative interface for BEA CONNECT SNA has two components:

- ◆ The interface to the SNA stack administration and configuration, which is dependent on the stack provider and thus cannot be covered in this guide. Refer to vendor publications for the stack(s) used in your environment.
- ◆ The interface to the administration and configuration of the BEA TUXEDO System portion of a CONNECT SNA application, which is provided by the command `dmadmin`, other shell-level commands, and by screens of the BEA TUXEDO *Graphical Administrative Interface*.

The dmadmin Command Interpreter

The `dmadmin` interactive command interpreter is used for the administration of domain gateway groups defined for a BEA TUXEDO System/T application. It can operate in two modes, administration mode and configuration mode. Configuration mode allows the administrator to change a configuration while the application is running. The `dmadmin` description in Appendix A, “Reference Pages” of this guide has been edited to include material specific to CONNECT SNA and new subcommands that apply only to CONNECT SNA.

Note: The following four `dmadmin` commands are not applicable to CONNECT SNA because transactions, in the BEA TUXEDO System sense, are not supported:

- ◆ `dsdmlog`
- ◆ `forgettrans`
- ◆ `indmlog`
- ◆ `printtrans`

New subcommands have been added to `dmadmin` to support entering of userids and passwords in the domain security table. These new commands can be entered only as subcommands of `dmadmin`:

- ◆ `addumap [options]`
- ◆ `addusr (addu) [options]`
- ◆ `delumap [options]`
- ◆ `delusr (delu) [options]`

The reference pages for these subcommands and other /Domain commands have been modified for the BEA CONNECT SNA product. Use Appendix A, “Reference Pages” in preference to similar pages in the *BEA TUXEDO Reference Manual*.

The SNACRM and PU 2.1 Servers

The `SNACRM` communicates directly with the PU 2.1 server to provide SNA connectivity. These servers are not BEA TUXEDO-based and can be started manually or via the `DMINIT` server. In either case, the PU 2.1 server must always be started before the `SNACRM`. Both servers must be started before starting the associated SNA Domain gateway group. Once the `SNACRM` and PU2.1 servers are started, there is no reason to restart them, outside of a catastrophic failure.

Starting the PU2.1 Server

Please refer to `SNACRM` in Appendix A, “Reference Pages” of this guide for information about starting the PU2.1 server.

Starting the SNACRM

You can manually start `SNACRM` from the command line or during `tmboot` with the `DMINIT` server. You can either start all `SNACRM` processes with a single `DMINIT` process or individually start each one with a `DMINIT` server that is defined to each gateway server group.

When you start `SNACRM` from the UNIX command line, the `SNACRM` Command Line Console puts its prompt in this window, and if exited, shuts down all of the active links. When started from `DMINIT`, the console is redirected to the null device. In this case, you can use the `xsnacrm` command to monitor the console and enable/disable tracing of `SNACRM` and the SNA stack.

Please refer to `SNACRM` in Appendix A, “Reference Pages” of this guide.

Using DMINIT

Please refer to `SNACRM` in Appendix A, “Reference Pages” of this guide for information about using `DMINIT`.

Security

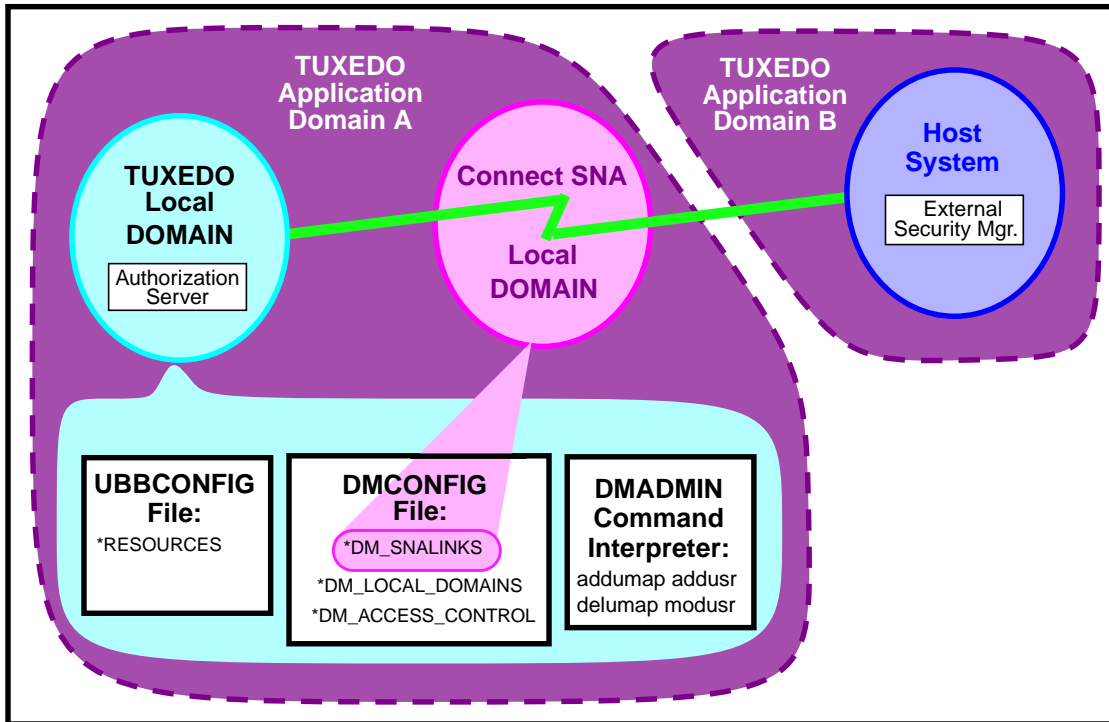
In order for any security checking to occur, the TUXEDO Local Domain and the Host System must have security mechanisms in place. For the Local Domain, this is the Authorization Server. For the Host System, this is the External Security Manager. Figure 5-1 shows these elements.

There are four sections in two BEA TUXEDO configuration files in which you specify parameters bearing on Local Domain security. The two configuration files are DMCONFIG and UBBCONFIG.

Userid and password mapping between the Local Domain and the Host System also bears on security. There are five `DMADMIN` subcommands which you use to enter userids and passwords, set up mappings, remove mappings, remove userids and passwords, and modify passwords.

Caution: Mixed environment security is more complex than depicted in Figure 5-1. A domain without an operational security mechanism in place accepts all transaction requests by treating userids as “trusted users.” Refer to TUXEDO documentation for more detailed information about domain security.

Figure 5-1 Connect SNA Security Elements



Where You Specify Security Parameters

The configuration sections where security is specified are:

- ◆ *RESOURCES section of the UBBCONFIG file
- ◆ *DM_LOCAL_DOMAINS section of the DMCONFIG file
- ◆ *DM_SNALINKS section of the DMCONFIG file
- ◆ *DM_ACCESS_CONTROL section of the DMCONFIG file

UBBCONFIG File Security Parameters

The `*RESOURCES` section in this file contains a `SECURITY` parameter which works in conjunction with the `SECURITY` parameter in the `DMCONFIG` file to establish how CONNECT SNA controls access to the TUXEDO local domain. This parameter takes the form:

`SECURITY = value`

where `value` is:

`NONE`

No security is enforced (default)

`APP_PW`

Requires password authorization for TUXEDO clients and administrative tools to connect to the local application.

`USER_AUTH`

Same as `APP_PW`, but additional authorization is required on a per-user basis.

`ACL`

Same as `USER_AUTH`, but additional access-control checks are done on service names, queue names, and event names. If no *Access Control Lists (ACL)* exists for a given name, access is granted.

`MANDATORY_ACL`

Same as `ACL`, but if no `ACL` exists for a given name, access is denied.

In most cases, the `UBBCONFIG` file has already been configured and you don't need to establish the `SECURITY` parameter settings, but examining this file enables you to ascertain how CONNECT SNA enforces security.

If this parameter is set to `NONE`, no security is enforced. If set to `APP_PW`, the local TUXEDO domain's Authorization Server prompts for the application password. If set to `USER_AUTH`, `ACL`, or `MANDATORY_ACL`, the qualified security is enforced as specified.

DMCONFIG File Security Parameters

Three sections in the DMCNFIG file contain parameters affecting CONNECT SNA control of access to the TUXEDO local domain:

- ◆ *DM_LOCAL_DOMAINS section contains a SECURITY parameter which specifies the type of security enforced for the TUXEDO local domain.
- ◆ *DM_SNADLINKS section contains a SECURITY parameter which records the security in effect for the host system.
- ◆ *DM_ACCESS_CONTROL section contains local access control lists used by the TUXEDO local domain to associate local resources with host systems permitted to have access to them.

Caution: Do not delete the DMCNFIG file. Tables of remote users, remote passwords, and remote mappings are stored in this file. If deleted, all security information must be re-entered.

*DM_LOCAL_DOMAINS SECTION

The SECURITY parameter settings in this section work in conjunction with the SECURITY parameter in the *RESOURCES section of the TUXEDO local domain's UBBNFIG file to establish how CONNECT SNA controls access to the TUXEDO local domain. The parameter takes the form:

SECURITY = *value*

where *value* is:

NONE

No security is enforced.

APP_PW

No security is enforced.

DM_USER_PW

User and password security is enforced.

If this parameter is set to NONE or APP_PW, the local domain takes no action with regard to security. If this parameter is set to DM_USR_PW, the local domain enforces security according to the setting in the UBBNFIG file (refer to "UBBNFIG File Security Parameters" on page 5-6).

*DM_SNALINKS SECTION

This section of the DMCONFIG file is dedicated to CONNECT SNA parameters. It records the security in effect for the host system. It correlates to the values set for the ATTACHSEC parameter in the connection resource definition. In the following parameter definition, *remote* refers to the Tuxedo domain and *local* refers to the host system. The parameter takes the form:

SECURITY_TYPE = *value*

where *value* is:

LOCAL

Specifies that a user identifier is not to be supplied by the remote system. LOCAL is the default value.

IDENTIFY

Specifies that a user identifier is expected on every attach request. All remote users of a system must be identified to the remote external security manager.

VERIFY

Attaches a userid and valid password to the remote region. The userid and password are controlled by the region's external security manager.

PERSISTENT

Not fully supported. Functions the same as VERIFY.

MIXIDPE

Not fully supported. Functions the same as VERIFY.

The values LOCAL and IDENTIFY are roughly equivalent to NONE and APP_PW for the SECURITY parameter in the DMCONFIG file.

*DM_ACCESS_CONTROL SECTION

This section contains local *Access Control Lists (ACL)* used by the TUXEDO local domain to restrict access by remote regions to local resources. (Refer to "Security Administration" in the *TUXEDO Administrator's Guide*.) Each entry consists of an ACL_NAME resource identifier along with a list of required parameters designating remote domains permitted to access the resource. If no entry exists for a local service, the service is accessible to all remote domains.

Security Setting Summary

The domain administrator can add or delete users from the domain security table in two ways: first, through data entry panels in the Tuxedo System administration Graphical Administrative Interface; second, by using the `dmadmin` command.

The combined settings of the `SECURITY` parameters in the `UBBCONFIG` and the `DMCONFIG` files have the following effects:

- ◆ When the `DM_LOCAL_DOMAINS` Security parameter is set to `NONE` or `APP_PW`, no action is taken by the `CONNECT SNA` gateway with regard to security.
- ◆ However, when the `UBBCONFIG` file Security parameter is set to `APP_PW`, the application password is validated by an `AUTHSVC` when clients join the application. The `AUTHSVC` is provided by the user application.

If security is to be enforced by both the local domain and the host system for each request outbound from the local domain, the following settings must be made:

- ◆ `UBBCONFIG` file `SECURITY` parameter must be set to one of `USER_AUTH`, `ACL`, or `MANDATORY_ACL`
- ◆ the `DMCONFIG` file `*DM_LOCAL_DOMAINS` section `SECURITY` parameter must be set to `DM_USER_PW`
- ◆ and the `*DM_SNALINKS` `SECURITY` parameter must be set to `IDENTIFY` or `VERIFY`

(Refer to Table 5-2 for a summary of the file settings required to achieve different security combinations for outbound requests from the local domain.)

If security is to be enforced by both the local domain and the host system for each request inbound from the host system to the local domain, the following settings must be made:

- ◆ The `UBBCONFIG` file `SECURITY` parameter must be set to one of `USER_AUTH`, `ACL`, or `MANDATORY_ACL`;
- ◆ The `DMCONFIG` file `*DM_LOCAL_DOMAINS` section `SECURITY` parameter must be set to `DM_USER_PW`
- ◆ The `*DM_SNALINKS` `SECURITY` parameter must be set to `IDENTIFY` or `VERIFY`.

(Refer to Table 5-1 for a summary of the file settings required to achieve different security combinations for inbound requests from the host system.)

For a request sent to the host system, the local principal userid is located in the domain security table and the associated remote userid, or userid and password, are put into the conversation start-up request before being sent over the LU6.2 conversation. (This occurs if `SECURITY` is set to `IDENTIFY` or `VERIFY` in the `DM_SNALINKS` section of the `DMCONFIG` file.)

For requests sent from the host system, the local domain extracts the remote userid, or userid and password, from the conversation start-up request and checks the domain security table. That table contains pairs of local principal userids and remote userids, maintained on a service-by-service basis. The remote userid is mapped to the local principal userid. The local principal userid and password are used for further Access Control List (ACL) checking, as specified in the `UBBCONFIG` file.

When a request is received from the host system, the local domain checks the `DMCONFIG` file ACL for the local service to see if requests from the remote domain are permitted. If the `DMCONFIG` file does not contain an ACL for the local service, the service is accessible to all requests.

Therefore, if the `ATTACHSEC` level for the connection definition in the host system is `Identify` or `Verify`, the `DMCONFIG SECURITY` parameter must be set to `DM_USER_PW` so that a userid and a password are sent on the conversation start-up requests.

Security Setting Summary Tables

Table 5-1 shows settings for the `SECURITY` parameters in the `UBBCONFIG` and `DMCONFIG` files required to achieve local domain and host system security combinations for inbound requests.

Note: Security setting combinations other than those shown in the tables will have unpredictable results.

Table 5-1 Security Setting Summary (Inbound Requests from Host System)

(Col. A) SECURITY COMBINATION		(Col. B) SETTINGS IN UBBCONFIG FILE	(Col.C) SETTINGS IN DM_LOCAL_DOMAINS SECTION of DMCONFIG FILE	(Col. D) SETTINGS IN DM_SNALINKS SECTION of DMCONFIG FILE	(Col.E) OUTBOUND USERID and PASSWORD ACTIVITY (UID/PW)
LOCAL DOMAIN	HOST SYSTEM				
NO	NO	SECURITY = NONE or APP_PW	SECURITY = NONE or APP_W	SECURITY= LOCAL	Not Applicable
YES	NO	SECURITY = USER_AUTH, ACL, or MANDATORY_ACL	SECURITY = NONE or APP_PW	SECURITY = LOCAL	Not Applicable
NO	YES	SECURITY = NONE or APP_PW	SECURITY = DM_USER_PW	SECURITY = IDENTIFY or VERIFY	INVALID
YES	YES	SECURITY = USER_AUTH, ACL, or MANDATORY_ACL	SECURITY = DM_USER_PW	SECURITY = IDENTIFY or VERIFY	UID or UID+PW checked by Host System

Table 5-2 shows settings for the `SECURITY` parameters in the `UBBCONFIG` and `DMCONFIG` files required to achieve local domain and host system security combinations for outbound requests.

Note: Security setting combinations other than those shown in the tables will have unpredictable results.

Table 5-2 Security Setting Summary (Outbound Requests from Local Domain)

(Col. A) SECURITY COMBINATION		(Col. B) SETTINGS IN UBBCONFIG FILE	(Col.C) SETTINGS IN DM_LOCAL_DOMAINS SECTION of DMCONFIG FILE	(Col. D) SETTINGS IN DM_SNALINKS SECTION of DMCONFIG FILE	(Col.E) INBOUND USERID and PASSWORD ACTIVITY (UID/PW)
LOCAL DOMAIN	HOST SYSTEM				
NO	NO	<code>SECURITY =</code> <code>NONE</code> or <code>APP_PW</code>	<code>SECURITY =</code> <code>NONE</code> or <code>APP_W</code>	<code>SECURITY=</code> <code>LOCAL</code>	Not Applicable
YES	NO	<code>SECURITY =</code> <code>USER_AUTH</code> , <code>ACL</code> , or <code>MANDATORY_ACL</code>	<code>SECURITY =</code> <code>NONE</code> or <code>APP_PW</code>	<code>SECURITY =</code> <code>LOCAL</code>	INVALID
NO	YES	<code>SECURITY =</code> <code>NONE</code> or <code>APP_PW</code>	<code>SECURITY =</code> <code>DM_USER_PW</code>	<code>SECURITY =</code> <code>IDENTIFY</code> or <code>VERIFY</code>	UID or UID+PW from Host System ignored
YES	YES	<code>SECURITY =</code> <code>USER_AUTH</code> , <code>ACL</code> , or <code>MANDATORY_ACL</code>	<code>SECURITY =</code> <code>DM_USER_PW</code>	<code>SECURITY =</code> <code>IDENTIFY</code> or <code>VERIFY</code>	UID or UID+PW checked by Local Domain

How To Administer Security

After setting up and/or checking the security settings for the TUXEDO domain and the host system, you must relate the security information in both domains to each other. To do this, use the `addusr` and `addumap` subcommands provided with the `dmadmin` command interpreter.

Once the user security information in both domains is mapped, you can perform administration on the affected security files in each domain. To do this, use the `delumap`, `modusr`, and `delusr` subcommands.

The following paragraphs discuss how you enter these commands. Refer to Appendix A, “Reference Pages” for detailed information about each subcommand.

Adding a Userid and Password

Use the `addusr` subcommand to add a TUXEDO local domain’s userid and password to the remote CICS/ESA domain’s user and password file. Enter the following command:

```
addusr -d local_domain_id -R remote_domain_id -u remote_userid
```

where:

`-d`

Specifies the name of the local domain with which the userid and password are associated.

`-R`

Specifies the name of the remote domain to which the userid and password are to be added.

`-u`

Specifies the user name to be added. Enter the user’s password when prompted.

Mapping a Userid

Use the `addumap` subcommand to map a local domain principal userid number to a remote domain user name. The userid must be added before it can be mapped (refer to "Adding a Userid and Password"). Enter the following command:

```
addumap -d local_domain_id -R remote_domain_id  
-p local_principal_userid -u remote_userid
```

where:

- d Specifies the name of the local domain with which the userid is associated.
- R Specifies the name of the remote domain to which the userid is mapped.
- p Specifies the local principal userid number defined in the ACL.
- u Specifies the remote user name as defined in the security application of the remote domain.

Removing a Userid's Mapping

Use the `delumap` subcommand to remove the mapping for a local domain principal userid to a remote domain user name. Enter the following command:

```
delumap -d local_domain_id -R remote_domain_id  
-p local_principal_userid -u remote_userid
```

where:

- d Specifies the name of the local domain with which the userid is associated.
- R Specifies the name of the remote domain to which the userid is mapped.
- p Specifies the local principal userid number defined in the ACL.
- u Specifies the remote user name as defined in the security application of the remote domain.

Deleting a Userid and Password

Use the `delusr` subcommand to remove a local TUXEDO domain's user ID and password from the CICS/ESA remote domain's user and password file. The mapping for a userid must be removed before the userid can be removed (refer to "Removing a Userid's Mapping"). Enter the following command:

```
delusr -d local_domain_id -R remote_domain_id -u remote_userid
```

where:

- d
Specifies the name of the local domain with which the userid and password are associated.
- R
Specifies the name of the remote domain from which the userid and password are to be deleted.
- u
Specifies the user name to be deleted. Enter the user's password when prompted.

Modifying a Password

Use the `modusr` subcommand to modify a local TUXEDO domain user's password recorded in a CICS/ESA remote domain's user and password file. Enter the following command:

```
modusr -d local_domain_id -R remote_domain_id -u remote_userid
```

where:

- d
Specifies the name of the local domain with which the userid and password are associated.
- R
Specifies the name of the remote domain in which the userid and password are to be modified.
- u
Specifies the user name to be modified. Enter the user's password when prompted.

Data Translations

Like other parts of BEA TUXEDO Systems, CONNECT SNA uses the *Typed Buffer* to transmit and receive data. When communication is between a CONNECT SNA application and a remote host region, there are some special considerations.

Since the remote host application does not understand the TUXEDO typed buffer, the TUXEDO application must communicate with the host application by using an aggregate data type known as a *record*. A record is a flat data area defined by a template that describes the data type and length of each field in the record.

The TUXEDO typed buffer must be represented to the host application as record aggregate data, and the record must be represented to the TUXEDO application as a typed buffer. In addition, string data must be in EBCDIC format for the host region and in ASCII format for the TUXEDO application.

The data may be manipulated into the correct format by the TUXEDO application or remote host application, or it may be formatted prior to transmission by CONNECT SNA. The service definitions in the `*DM_LOCAL_SERVICES` and the `*DM_REMOTE_SERVICES` section of the DMCONFIG file provide parameters to describe the typed buffer/record combination required for successful communications between TUXEDO applications and host applications. These parameters are `INBUFTYPE` and `OUTBUFTYPE`.

The parameter definitions are made from the perspective of the receiver of the buffer, the TUXEDO application which receives a typed buffer, or the remote host application which receives record data. That is, `INBUFTYPE` is used to format a typed buffer into a record for the remote host application; `OUTBUFTYPE` is used to format a host record into a typed buffer for the TUXEDO application.

Buffers Going To The Remote Host Application

The TUXEDO application buffer is a typed buffer that must be communicated to the remote host as a record containing aggregate data fields. CONNECT SNA looks at the service definition to determine what, if any, conversion must be applied to the data buffer. The service definition uses the `INBUFTYPE` in both the `*DM_LOCAL_SERVICES` and `*DM_REMOTE_SERVICES` section of the DMCONFIG file to describe the type of buffer manipulation.

INBUFTYPE Parameter Definition

`INBUFTYPE = type:subtype`

In this parameter definition, `type` must be one of the designated TUXEDO typed buffers described in the following subsections.

The `subtype` value names a view and is required for certain TUXEDO typed buffers.

Only one `type:subtype` may be entered for the `INBUFTYPE` parameter.

Data Conversion for STRING Typed Buffer

The null terminated string is converted to EBCDIC. The null character is part of the converted record.

Data Conversion for X_OCTET/CARRAY Typed Buffers

No data conversion is performed on these typed buffers. The TUXEDO application or remote host application performs all conversion of data fields in the record. This includes all numeric and EBCDIC conversions.

These typed buffers are used when a data record cannot be described or converted using one of the other *strong* typed buffers. *Strong* means that CONNECT SNA can understand all data fields and perform the required data conversions.

These typed buffers are also options when the remote service expects many styles of data aggregation (multiple record types). The `INBUFTYPE` parameter is limited to one `type:subtype`.

Data Conversion for VIEW/VIEW32/X_C_TYPE/X_COMMON Typed Buffers

A subtype is required for these typed buffers. The subtype is the name of the view that describes the remote host record. The TUXEDO buffer is converted from a C structure to the record, including EBCDIC conversion, using the compiled VIEW file. By default, the record is a COBOL structure, mapped by the remote host application using a COBOL copybook.

Data Conversion for FML/FML32 Typed Buffers

A subtype is required for these typed buffers. The subtype is the name of the view that describes the remote host record. The data in the typed buffer is *Field Manipulation Language (FML)* data. The CONNECT SNA converts the buffer to a record described by the view; this includes EBCDIC conversion.

The TUXEDO buffer is converted from an FML typed buffer to a C structure using the subtype compiled VIEW file. The C structure is then converted to the record using the same subtype compiled VIEW file. By default, the record is a COBOL structure that is mapped by the remote host application using a COBOL copybook

Buffers Coming From The Remote Host Application

An aggregate data type known as a record is received from the remote host application. The record must be converted to a TUXEDO typed buffer by CONNECT SNA before it is passed to the TUXEDO application. CONNECT SNA looks at the service definition to determine what, if any, conversion must be applied to the host data buffer. The service definition uses the OUTBUFTYPE in both the *DM_LOCAL_SERVICES and *DM_REMOTE_SERVICES section of the DMCONFIG file to describe the host record, and how to convert the record to the TUXEDO typed buffer.

OUTBUFTYPE Parameter Definition

OUTBUFTYPE=type:subtype

In this parameter definition, `type` must be one of the designated TUXEDO typed buffers described in the following subsections. The `type` not only determines the typed buffer, but also describes the host record.

The `subtype` value names a view and is required for certain TUXEDO typed buffers.

Only one `type:subtype` may be entered for the OUTBUFTYPE parameter.

Data Conversion for STRING Typed Buffer

The null terminated string is converted to ASCII. The converted string is moved to a TUXEDO STRING typed buffer.

Data Conversion for X_OCTET/CARRAY Typed Buffers

No data conversion is performed on these typed buffers. The remote host application or the TUXEDO application converts the data fields in the record. This includes all numeric and ASCII conversions.

These typed buffers are used when the data record cannot be described or converted using one of the other *strong* type buffers. *Strong* means CONNECT SNA can understand all data fields and perform the required data conversion.

These typed buffers are also options when the remote service expects many styles of data aggregation (multiple record types). The OUTBUFTYPE parameter is limited to one `type:subtype`.

Data Conversion for VIEW/VIEW32/X_C_TYPE/X_COMMON Typed Buffers

A subtype is required for these typed buffers. The subtype is the name of the view that describes the remote host record. The remote host record is converted to a TUXEDO typed buffer. The compiled VIEW file is used to perform the data and ASCII conversion on the host record. By default, the host record is a COBOL data aggregate and the converted typed buffer is mapped by the TUXEDO application using a C structure.

Data Conversion for FML/FML32 Typed Buffers

A subtype is required for these typed buffers. The subtype is the name of the view that describes the remote host record. The host record is converted to an FML buffer that is passed to the TUXEDO application.

By default, the host record is a COBOL record aggregate data type. The data is converted to a C structure, including ASCII conversion, using the compiled VIEW file. This data is then converted to an FML buffer using the field definitions associated with the VIEW.

Data Conversion For DPL Services

CONNECT SNA supports remote CICS services as *Distributed Program Link (DPL)* programs (as described in Chapter 1, “Introducing BEA Connect SNA”). The DPL support is performed as if the TUXEDO service is a peer CICS/ESA service.

In a DPL program, the application is protected from all *Distributed Transaction Processing (DTP)*. The DPL application is a request/response service, with all data communication performed by the passing of a COMMAREA.

A basic DPL request API looks like this:

```
EXEC CICS LINK  
  
      PROGRAM( )  
      DATALENGTH( )  
      LENGTH( )  
      COMMAREA( )
```

In the preceding example, the requester sends the COMMAREA of DATALENGTH size and the receiving application receives the COMMAREA data contents in a buffer the size of LENGTH. The DATALENGTH size might be smaller than the LENGTH size, but the requester expects and receives the response in the original COMMAREA buffer the size of LENGTH.

The difference between a DPL program and a TUXEDO service is that a receiving TUXEDO service can resize a reply buffer, while the DPL program expects a reply buffer of a designated size. Also, a TUXEDO requester can receive a resized buffer in a buffer different from the original reply buffer.

CONNECT SNA performs the manipulation described in the following subsections to smoothly adjust to the requirements of both types of applications.

DPL Requests Originating From A TUXEDO Application

CONNECT SNA must determine what size COMMAREA the remote DPL service is expecting because there is no corresponding LENGTH parameter on a TUXEDO request.

To determine the LENGTH value for a DPL request, CONNECT SNA uses the larger potential size of the INBUFTYPE or the OUTBUFTYPE parameter definitions, as described in Table 5-3.

The remote DPL application receives a buffer of LENGTH size and returns a buffer of LENGTH size.

Table 5-3 DPL Request LENGTH Calculation

INBUFTYPE or OUTBUFTYPE	LENGTH CALCULATION
STRING/X_OCTET/ CARRAY	For these typed buffers, only the <code>INBUFTYPE</code> parameter definition is used to determine the <code>LENGTH</code> .
VIEW/VIEW32/ X_COMMON/ X_C_TYPE	<code>LENGTH</code> is the maximum size of the <code>VIEW</code> file. This calculation takes in the potential size of both the <code>C</code> structure and the target record.
FML/FML32	The maximum size of the <code>VIEW</code> file. This calculation takes in the potential size of both the <code>C</code> structure, and the target record. The length of the <code>FML</code> buffer is not taken into account.

If no `LENGTH` can be determined, then the largest value allowed by the CICS application is allocated. Refer to the “Usage Notes” section.

DPL Requests Originating From a CICS DPL

`CONNECT SNA` receives a `LENGTH` value and `COMMAREA` data of `DATALength` size from the requesting CICS DPL. `CONNECT SNA` allocates a buffer of `LENGTH` size and moves the `COMMAREA` data into this buffer before performing the conversions.

Usage Notes

The current size limit of remote host messages is limited to approximately 32K bytes. Any conversions resulting in records larger than 32756 bytes are not supported.

6 Verifying BEA Connect SNA

After installing and configuring the BEA CONNECT SNA product software, it is important to verify the operational integrity of the environment. To do this, you run a sample application on a simple server in client/server transaction scenarios. This process employs programs available in your product software libraries.

Note: Before running the sample application, you must configure the SNA gateway environment, start the Physical Unit (PU), start the SNACRM, and start TUXEDO. Refer to Chapter 4, “Configuring BEA Connect SNA” and Chapter 5, “Administering the BEA Connect SNA Application Domain” for details.

Overview

The sample applications are located in the SNA simple server library. The simple server passes a string from the client to the server.

The CICS/ESA programs may run as either DTP or DPL processes, and as either servers or clients. In addition, the simple server may run in either transactional or nontransactional mode. In the transactional mode, these scenarios verify that the sync-level 2 protocol is established between the two application environments.

When the client runs as a TUXEDO client, the server runs as a CICS/ESA host. You enter a text string in lower-case letters with command arguments. The CICS/ESA server converts the lower-case letters to upper-case letters and re-displays the text string.

When the client runs as a CICS/ESA client, the server runs as a TUXEDO server. Again, you enter a text string in lower-case letters. The TUXEDO server converts the text string into a mirror image and displays the string as reversed letters.

Note: The verification process is intended for the CICS/ESA environment only, that is, between TUXEDO applications and CICS/ESA applications. If your TUXEDO applications operate in other environments, you must create your own verification process.

Building Your Verification Tests

You must build the verification test to run in two domains, the TUXEDO local domain and the CICS/ESA remote domain. The executables in each domain are different. The following subsections discuss how to build these executables.

Building Tuxedo Executables

Use the following checklist to complete this process:

1. Modify the name of the simple server in the UBBCONFIG file provided.
2. Execute the `tmloadcf` command to compile the UBBCONFIG file.
3. Modify the required local and remote definitions in the DMCONFIG file provided.
4. Execute the `dmloadcf` command to compile the DMCONFIG file.
5. Modify the `apps.env` and `<machine>.env` environment files (optional).
6. Use the `buildserver` utility to build a server load module.
7. Use the `buildclient` utility to build a client load module.

Using the buildserver Utility

Use the TUXEDO `buildserver` utility to build the `mirrorsrv` server load module from the source file `mirrorsrv.c` provided. The source file contains two service entries, `MIRROR` and `DOUBLEMIRROR`, which will be advertised by the `mirrorsrv` server.

When executed, the `MIRROR` service receives a text string from the client, reverses the letters, and displays a mirror image of the input text string.

When executed, the `DOUBLEMIRROR` service receives a text string from the client, reverses the letters, and concatenates the reversed string to the forward image of the string.

This is an example of a command entry to invoke the `buildserver` utility:

```
buildserver -o mirrorsrv -f mirrorsrv.c -s MIRROR,DOUBLEMIRROR
```

(Refer to the *BEA TUXEDO Reference Manual* for option descriptions.)

Using the buildclient Utility

Use the TUXEDO `buildclient` utility to build a client load module from the source file `toupclt` provided. When executed, the load module sends a lower-case text string to the server, which converts it to uppercase in several modes, causing different server scenarios to execute. (Refer to the *BEA TUXEDO Reference Manual* for option descriptions.)

For example:

```
buildclient -o toupclt -f toupclt.c
```

Modifying the UBBCONFIG file

Include the name of the TUXEDO simple server in the UBBCONFIG file. If you plan to run the transactional version of the verification process, enable the `TLOGDEVICE` comment in the Machine section. This must point to a valid DTP transaction log. The application server `GROUP3` in the Groups section must point to a valid transaction manager server.

Note: To run transaction examples, create the DTP transaction log named on the UBBCONFIG `TLOGDEVICE`. Create the log with the Tuxedo bulletin modification interpreter `tmadmin`. (Refer to the *BEA TUXEDO Reference Manual* for more details.)

For example:

```
*GROUP
GROUP3          LMID=sna GRPNO=3 TMSNAME=<transaction mgr name>
#example        TMSNAME=tstms
mirrorsrv       SRVGRP=GROUP3 SRVID=1 RQADDR=MIRR1 REPLYQ=Y
*SERVICES
MIRROR
```

Executing the `tmloadcf` Command

Execute the TUXEDO `tmloadcf` command to parse the UBBCONFIG file and create a binary version of the file. (Refer to the *BEA TUXEDO Reference Manual* for more details.)

For example:

```
tmloadcf UBBCONFIG
```

Respond to the prompts as the command executes.

Modifying the DMCONFIG File

The DMCONFIG file must contain both local and remote definitions for the simple server.

Note: A sample DMCONFIG file is included with the simple server.

Listing 6-1 Sample DMCONFIG File

```
*DM_LOCAL_SERVICES
#The Tuxedo reverse string server
MIRROR          LDOM="simpsnad"
                CONV=N
                RNAME="MIRRORSERV"
                INBUFTYPE="STRING"
                OUTBUFTYPE="STRING"
DOUBLEMIRROR
                CONV=N
                RNAME="MIRRDPLS"
                INBUFTYPE="STRING"
                OUTBUFTYPE="STRING"
*DM_REMOTE_SERVICES
#The CICS upper-case DTP and DPL servers
SIMPDDL          AUTOTRAN=N
                LDOM="simpsnad"
                RDOM=SNAG1
                CONV=N
                RNAME="TOUPDPLS"
                INBUFTYPE="STRING"
                OUTPBUFTYPE="STRING"
                FUNCTION="DPL"
SIMPDTP          AUTOTRAN=N
                LDOM="simpsnad"
                RDOM=SNAG1
                CONV=N
                RNAME="DTPS"
                INBUFTYPE="STRING"
                OUTPBUFTYPE="STRING"
                FUNCTION="APPC"
```

In the preceding DMCONFIG file example, both instances of the LDOM name correspond to the SNA domain name in the *DM_LOCAL_DOMAINS section. The server is a request/response server.

In the `*DM_LOCAL_SERVICES` section, the `RNAME="MIRRORSERV"` and `RNAME="MIRRDPLS"` values are the names passed from the CICS/ESA environment. `MIRROR` and `DOUBLEMIRROR` refer to the advertised services provided by the `mirrorsrv` server named in the `UBBCONFIG` file. The `CONV=N` definition indicates the protocol that is observed by the SNA domain, although the CICS/ESA client does not perform a Tuxedo ATMI `tpcall`.

In the `*DM_REMOTE_SERVICES` section, the `RNAME` value identifies what is invoked in the CICS/ESA domain. For the *Distributed Program Link (DPL)* request, the `RNAME` equals the name of the *program* called. For the *Distributed Transaction Processing (DTP)* request, the `RNAME` equals the name of the *transaction id*.

If you want to run transactional verification tests, you must enter a link definition `MAXSYNCLVL=2` in the `*DM_SNALINKS` section. If you want to run non-transactional DPL tests only, you must enter `MAXSYNCLVL=1`.

Executing the `dmloadcf` Command

Execute the TUXEDO `dmloadcf` command to parse the `DMCONFIG` file and create a binary version of the file. (Refer to the *BEA TUXEDO Reference Manual* for more details.)

For example:

```
dmloadcf DMCONFIG
```

Respond to the prompts as the command executes.

Modifying the Environment Files

Two types of files are provided with your CONNECT SNA product software which can be used to define the application and/or machine environments for verification testing. If their equivalents do not already exist, it is recommended you modify the files provided and make them available to your system. The files are `apps.env` and `<machine>.env`.

THE APPS.ENV FILE

Modify the `apps.env` file and include it with the `ENVFILE` parameter in the `*MACHINES` section of the `UBBCONFIG` file. The `apps.env` file looks like this:

Listing 6-2 The apps.env File

```
#=====
# apps.env
#     Environment macros for tuxedo application testing.
#
# See also
#     See $(TOP)/Makefile for more information.
#
# @(#)SNA Devel apps/simpsna app.env 1.4 98/03/03 15:42:30
# Copyright 1997, BEA Systems, Inc., all rights reserved.
#-----
APPDIR=<Your application directory here>
TUXCONFIG=<Your Tux configuration here>
BDMCONFIG=<Your Domain configuration here>
TUXDIR=<Your Tuxedo directory here>
FLDTBLDIR32=<Your Tuxedo directory here>/lib
FLDTBLS32=fmb.def
```

THE <MACHINE>.ENV FILES

Modify the <machine>.env file that is appropriate for your system:

- ◆ aix.env
- ◆ hpux.env
- ◆ solaris.env

Each of these files is executable. Once you have modified the appropriate file for your system, execute it to export the machine environment variables. The files look like this:

Listing 6-3 The aix.env File

```
#=====
# aix.env
#     Environment macros for AIX testing.
#
# See also
#     See $(TOP)/Makefile for more information.
#
```

```
# @(#)SNA Devel apps/simpsna aix.env 1.2 98/02/23 12:39:36
# Copyright 1997, BEA Systems, Inc., all rights reserved.
#-----
export APPDIR=<Your application directory bin here>
export TUXCONFIG=<Your tuxedo configuration qualified name here>
export BDMCONFIG=<Your tuxedo domain configuration qualified name
here>
export TUXDIR=<Your tuxedo product directory here>
export STACK=<Your stack product library here>
#example STACK=/opt/SUNWappc or /opt/sna/lib
export FLDTBLDIR32=$TUXDIR/lib
export FIELDTBLS32=fmb.def
export PATH=$APPDIR:$TUXDIR/bin:$PATH
export LIBPATH=$TUXDIR/lib:$LIBPATH:$STACK
```

Listing 6-4 The hpux.env File

```
#=====
# hpux.env
#      Environment macros for HP-UX testing.
#
# See also
#      See $(TOP)/Makefile for more information.
#
# @(#)SNA Devel apps/simpsna hpux.env 1.3 98/02/23 12:38:34
# Copyright 1997, BEA Systems, Inc., all rights reserved.
#-----
export APPDIR=<Your application directory bin here>
export TUXCONFIG=<Your tuxedo configuration qualified name here>
export BDMCONFIG=<Your tuxedo domain configuration qualified name
here>
export TUXDIR=<Your tuxedo product directory here>
export STACK=<Your stack product library here>
#example STACK=/opt/sna/lib
export PATH=$APPDIR:$TUXDIR/bin:$PATH
export SHLIB_PATH=$APPDIR:$TUXDIR/lib:$SHLIB_PATH:$STACK
```

Listing 6-5 The solaris.env File

```
#=====
# solaris.env
#      Environment macros for SOLARIS testing.
#
# See also
#      See $(TOP)/Makefile for more information.
#
# @(#)SNA Devel apps/simpsna solaris.env 1.3 98/02/23 12:39:05
# Copyright 1997, BEA Systems, Inc., all rights reserved.
#-----

export APPDIR=<Your application directory bin here>
export TUXCONFIG=<Your tuxedo configuration qualified name here>
export BDMCONFIG=<Your tuxedo domain configuration qualified name
here>
export TUXDIR=<Your tuxedo product directory here>
export STACK=<Your stack product library here>
#example STACK=/opt/SUNWappc
export FLDTBLDIR32=$TUXDIR/lib
export FIELDTBLS32=fmb.def
export PATH=$APPDIR:$TUXDIR/bin:$PATH
export LD_LIBRARY_PATH=$TUXDIR/lib:$LD_LIBRARY_PATH:$STACK
```

Building CICS/ESA Executables

Use the following checklist to complete this process:

1. Choose either COBOL or C source.
2. Transfer source from SNA install directory to the PDS.
3. Translate CICS/ESA verbs.
4. Compile the translated source file.
5. Link-edit the compiled source file.
6. Configure the CICS/ESA application (similar to CICS LU configuration).
7. View the connection, session, or mode status.

Choosing the Source Code Language

The CICS/ESA sample programs used for verification are unloaded during the installation of your BEA CONNECT SNA product software. These programs are available in two languages, COBOL and C. You must choose which language is used to build the CICS/ESA executable object code. (Your choice might be affected by the type of compiler available on your MVS host.)

You can identify the sample program names by their suffixes:

- ◆ `TOUPDTPS.c` is the C language program name
- ◆ `TOUPDTPS.cbl` is the COBOL language program name (requires a compiler that accepts COBOL II verbs)

Although the structures of the sample programs are different, they both perform the same function. The TUXEDO executable program that you build communicates with either.

Transferring the Source Code to Host

Transfer the source code to the host by the method you prefer, for example FTP (File Transfer Program). The destination could be sequential dataset or a PDS. Table 6-1 lists the source code files provided for UNIX and MVS platforms. The UNIX filename extensions suggest the type of destination libraries into which the source code may be transferred.

Table 6-1 Source Code Filenames

UNIX Filename	MVS Member Name
BEACONN.RDO	BEACONN
BEASNA.RDO	BEASNA
MIRRDPLC.<c> <cbl>	MIRRDPLC
MIRRDTPC.<c> <cbl>	MIRRDTPC
TOUPDPLS.<c> <cbl>	TOUPDPLS
TOUPDTPS.<c> <cbl>	TOUPDTPS

Translating CICS/ESA Verbs

This step translates the EXEC CICS verbs into program CALL statements of the form required by the selected source language. The source is read from the SYSIN dataset. The translated source program is written to the SYSPUNCH dataset. The translator listing is written to the SYSPRINT dataset.

Different translator modules are provided for different source languages. There are also language-specific parameters for the translation step. (Refer to the IBM *CICS/ESA Application Programming Guide* for additional translation options that might apply to your environment.)

The translator modules are installed in the CICS/ESA load datasets. This is indicated in the following examples by the CICSxxx.SDFHLOAD entry, where xxx is the CICS/ESA release number.

COBOL LANGUAGE TRANSLATOR EXAMPLE

The translator module name in the following example is DFHECP1\$. The parameter COBOL2 indicates that a source module containing COBOL II verbs is to be translated.

Listing 6-6 COBOL Language Translator Example

```
//TRN EXEC PGM=DFHECP1$,
//      PARM=' COBOL2,NOS,CICS',REGION=256K
//STEPLIB DD DSN=CICSXXX.SDFHLOAD,DISP=SHR
//SYSIN DD DSN=YOUR.PDS(pgmname),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSPUNCH DD DSN=&&SYSCIN,
//          DISP=(,PASS),UNIT=SYSDA,
//          DCP=BLKSIZE=400
//          SPACE=(400,(400,100))
```

C LANGUAGE TRANSLATOR EXAMPLE

The translator module name in Listing 6-7 is `DFHEDP1$`. The parameter `C` indicates that a source module containing C verbs is to be translated.

Listing 6-7 C Language Translator Example

```
//TRN EXEC PGM=DFHEDP1$,  
//      PARM='C,NOS,CICS',REGION=256K  
//STEPLIB DD DSN=CICSXXX.SDFHLOAD,DISP=SHR  
//SYSIN DD DSN=YOUR.PDS(pgmname),DISP=SHR  
//SYSPRINT DD SYSOUT=*  
//SYSPUNCH DD DSN=&&SYSCIN,  
//          DISP=(,PASS),UNIT=SYSDA,  
//          DCB=BLKSIZE=400  
//          SPACE=(400,(400,100))
```

Compiling the Translated Source File

The next step is to compile the translated source file `&&SYSCIN`. This step generates the following program modules in preparation for the link-edit step:

- ◆ `TOUPDPLS` (CICS server DPL module)
- ◆ `TOUPDTPS` (CICS server DTP module using CICS APIs)
- ◆ `MIRRDPLC` (CICS client DPL module)
- ◆ `MIRRDTPC` (CICS client DTP module using CPI-C verbs)

COBOL COMPILER EXAMPLE

Listing 6-8 shows a COBOL compiler example.

Listing 6-8 COBOL Compiler Example

```
//COB EXEC PGM=IGYCRCTL,REGION=GM,
//      PARM='NODYNAM,RENT,RES,APOST,MAP,XREF'
//STEPLIB DD DSN=SYS2.COB2.COB2COMP,DISP=SHR
//SYSLIB DD DSN=CICSXXX.SDFCOB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD DSN=&&SYSCIN,DISP=(OLD,DELETE)
//SYSLIN DD DSN=&&LOADSET,DISP=(MOD,PASS),
//      UNIT=&WORK,SPACE=(400,(20,20))
//SYSUT1 DD UNIT=WORK,SPACE=(460,(350,100))
//SYSUT2 DD UNIT=WORK,SPACE=(460,350,100))
//SYSUT3 DD UNIT=WORK,SPACE=(460,(350,100))
//SYSUT4 DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT5 DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT6 DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT7 DD UNIT=&WORK,SPACE=(460,(350,100))
```

C COMPILER EXAMPLE

Compile and pre-link your C language source. The BEA CONNECT SNA installation library contains C language compiler examples. Listing 6-9 shows a C/MVS compiler example.

In this example, the external storage variables must be re-entrant in the load module for the CICS/ESA environment. Additionally, the C language source contains long variable and function names. The pre-link step must be run to resolve the long names and create reentrant variables from the compiled object. Perform the pre-link step, use the RENT option in the compile program, and use the compile output in the link-edit step.

Listing 6-9 C/MVS Compiler Example

```
//*-----
//* COMPILE STEP:
//*-----
//COMPILE EXEC PGM=CBC310,REGION=2M,COND=(7,LT,TRN),
//      PARM=('OPT(1),LONGNAME,RENT,SOURCE')
//STEPLIB DD DSNAME=SYS1.SCEERUN,DISP=SHR
//      DD DSNAME=SYS1.SCBC3CMP,DISP=SHR
```

```
//SYMSGS      DD DSN=SYS1.SCBC3MSG(EDCMSGE),DISP=SHR
//SYSIN       DD DSN=SYS1.SYSCIN,DISP=SHR **FROM TRN STEP
//SYSLIB      DD DSN=SYS1.SCEEH.H,DISP=SHR
//            DD DSN=SYS1.SCEEH.SYS.H,DISP=SHR
//            DD DSN=CICSxxx.SDFHC370,DISP=SHR**CPIC
//            REQUIRED**
//SYSLIN      DD DSN=SYS1.PLNKSET,UNIT=VIO,
//            DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//            DCP=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSPRINT    DD SYSOUT=*
//SYSOUT      DD SYSOUT=*
//SYSPRINT    DD SYSOUT=*
//SYSUT1      DD UNIT=VIO,SPACE=(3200,(30,30)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT4      DD UNIT=VIO,SPACE=(3200,(30,30)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT5      DD UNIT=VIO,SPACE=(3200,(30,30)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=12800)
//SYSUT6      DD UNIT=VIO,SPACE=(3200,(30,30)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=12800)
//SYSUT7      DD UNIT=VIO,SPACE=(3200,(30,30)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=12800)
//SYSUT8      DD UNIT=VIO,SPACE=(3200,(30,30)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=12800)
//SYSUT9      DD UNIT=VIO,SPACE=(3200,(30,30)),
//            DCB=(RECFM=FB,LRECL=137,BLKSIZE=882)
//SYSUT10     DD SYSOUT=*
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT14     DD UNIT=VIO,SPACE=(3200,(30,30)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=128007)
//PLKED EXEC  PGM=EDCPRLK,COND=((7,LT,C),(7,LT,TRN)),
//            PARM="",REGION=2M
//STEPLIB     DD DSN=SYS1.SCEERUN,DISP=SHR
//SYMSGS      DD DSN=SYS1.SCEEMSGP(EDCPMSG),DISP=SHR
//SYSLIB      DD DSN=SYS1.LE370HLQ.SCEECPP,DISP=SHR
//            DD DUMMY
//SYSIN       DD DSN=SYS1.PLNKSET,DISP=(MOD,PASS)
//SYSMOD      DD DSN=SYS1.LOADSET,DISP=(,PASS),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSOUT      DD SYSOUT=*
//SYSPRINT    DD SYSOUT=*
```

Link-Editing the Compiled Source File

The next step is to take the compiled source and create the executable object. You should load the resulting object module into the application library that is concatenated with the CICS/ESA region datasets.

In the following COBOL and C program examples, SYSLIN is the name of the file containing the compiled source concatenated with other necessary executables, including interfaces for the CICS API verbs, interfaces for CPI-C verbs, and interfaces for SAA resource recovery verbs (sync-level 2).

In Listing 6-10, the module is linked as re-entrant and marked with 31-bit mode addressability. This is required for the module MIRRDTPC which performs CPI-C and SAA Resource/Recovery requests.

You must change the NAME to that of the executable being generated.

Listing 6-10 COBOL Link-Edit Sample for TOUPDPLS, TOUPDTPS, and MIRRDPLC

```
//LKED      EXEC PGM=IEWL,      * ** LINKAGE EDITOR **
//          PARM=AMODE=31,RENT,
//          REGION=512K//SYSPRINT DD SYSOUT=*
//SYSLIN     DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//          DD *
INCLUDE SYSLIB(DFHECI)
ORDER DFHECI
NAME xxxxxxxx(R)
/*
//SYSLIB     DD DSN=CICSxxx.SDFHLOAD,DISP=SHR
//          DD DSN=CICSxxx.SDFHCOB,DISP=SHR
//          DD DSN=SYS1.SCEELKED,DISP=SHR
//          DD DSN=SYS1.SIGYCOMP,DISP=SHR
//SYSLMOD    DD DSN=application.loadlib,DISP=(SHR,PASS)
//SYSUT1     DD UNIT=VIOD,SPACE=(1024,(50,20))
//
```

Listing 6-11 COBOL Link-Edit Sample for MIRRDTPC

```
//LKED      EXEC PGM=IEWL,      * ** LINKAGE EDITOR **
//          PARM=AMODE=31,RENT,
//          REGION=512K//SYSPRINT DD SYSOUT=*
//SYSLIN     DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//          DD *
INCLUDE SYSLIB(DFHECI)
INCLUDE SYSLIB(DFHCLPLC)
INCLUDE SYSLIB(DFHCLPLRR)
ORDER DFHECI
NAME xxxxxxxxx(R)
/*
//SYSLIB     DD DSN=CICSxxx.SDFHLOAD,DISP=SHR
//          DD DSN=CICSxxx.SDFHCOB,DISP=SHR
//          DD DSN=SYS1.SCEELKED,DISP=SHR
//          DD DSN=SYS1.SIGYCOMP,DISP=SHR
//SYSLMOD    DD DSN=application.loadlib(xxxxxxxxx),DISP=(SHR,PASS)
//SYSUT1     DD UNIT=VIOD,SPACE=(1024,(50,20))
//
```

Listing 6-12 C Link-Edit Sample

```
//LKED      EXEC PGM=HEWL,REGION=4M,
//          PARM='AMODE=31,RENT,
//          COND=((7,LT,C),(7,LT,PLKED),(7,LT,TRN))
//SYSLIB     DD DSN=CICSxxx.SDFHLOAD,DISP=SHR
//          DD DSN=SYS1.SCEELKED,DISP=SHR
//SYSLIN     DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//          DD *INCLUDE SYSLIB(DFHCLII)
//          INCLUDE SYSLIB(DFHCLPLC)
//          INCLUDE SYSLIB(DFHCLPLRR)NAME xxxxxxxxx(R)
/*
//SYSLMOD    DD DSN=application.loadlib(xxxxxxxxx),DISP=SHR
//SYSUT1     DD UNIT=VIOD,SPACE=(1024,(50,20))
//SYSPRINT   DD SYSOUT=*
```

Configuring the CICS/ESA Application

Your installed CONNECT SNA software contains two sample files that can be used to configure the CICS application:

- ◆ The BEACONN file contains the CICS/ESA configuration parameters to the host system. These include connection definitions and session definitions.
- ◆ The BEASNA file contains the application definitions that enable you to perform the installation verification in the CICS/ESA environment. These definitions are required to run the installation verification. They include the program definitions, transaction definitions, and for the CPI-C example, the partner definition.

Caution: The BEACONN file should only be added to the CICS/ESA System Definition (CSD) file if no definitions currently exist. Check with your system administrator.

One method of adding the files is to use the batch utility program DFHCSDUP. Listing 6-13 is an example of the *Job Control Language (JCL)* statements you can use to invoke DFHCSDUP as a batch program to add the BEASNA file:

Listing 6-13 JCL Example for Invoking DFHCSDUP

```
//YOURJOB JOB accounting info,name,MSGLEVEL=1
//STEP1 EXEC PGM=DFHCSDUP,REGION=512K,
//          PARM='CSD(READWRITE),PAGESIZE(60),NOCOMPAT'
//STEPLIB DD DSN=CICSxxx.SDFHLOAD,DISP=SHR
//DFHCSD DD UNIT=SYSDA,DISP=SHR,DSN=CICSxxx.DFHCSD
//SYSPRINT DD SYSOUT=A
//SYSIN DD DSN=YOUR.PDS(BEASNA),DISP=SHR
```

The definitions in the sample member use a CONNECT SNA *Resource Definition Online (RDO)* Group name. You may want to add these definitions to an existing RDO group, or you might consider adding them to your CICS/ESA start-up list if you plan to use them often. (This automatically installs the group on start-up of the CICS/ESA region.) To add the groups to the start-up list, un-comment the following statements in the sample RDO.

```
ADD GROUP (BEACONN) LIST (**YOURLIST**)
```

```
ADD GROUP (BEASNA) LIST (**YOURLIST**)
```

To manually install the groups after start-up of the CICS/ESA region, issue the following commands from a CICS/ESA terminal session.

```
CEDA I GROUP (BEACONN)
```

```
CEDA I GROUP (BEASNA)
```

BEACONN FILE: CONNECTION DEFINITION

The BEACONN file includes a sample connection definition.

Note: The sample connection definition achieves the minimum requirements for a connection over which the installation verification can be executed. Do not rely on it to provide optimal performance. Consult the *CICS/ESA Resource Definition Guide* for information about adding options not included in the sample.

The name of the sample connection definition is BEA, located under the installation group name BEACONN. It looks like this:

Listing 6-14 Sample Connection Definition in BEACONN File

```
DEFINE CONNECTION (BEA)          GROUP (BEACONN)
    DE (CONNECT SNA EXAMPLE RDO CONNECTION)
    ACCESSMETHOD (VTAM)          PROTOCOL (APPC)
    NETNAME (LUHP10A)
    ATTACHSEC (LOCAL)            AUTOCONNECT (NO)
```

The `NETNAME` option must be changed to the LU name of the remote system as known by VTAM. The `ATTACHSEC` option indicates the level of attach-time user security required for the connection. `LOCAL` is the simplest security. The authorization of the user is taken to be that of the link itself, relying on the authorization validation provided by the remote security utility. `AUTOCONNECT` indicates when the connection is acquired. `NO` is required. This means that CICS does not attempt to bind sessions when the connection is established by the stack.

Note: It is required that the CONNECT SNA software acquire the connection and negotiate the bind when the CONNECT SNA software starts up.

To install the sample connection definition, put it in on the host in a separate group which does not contain existing connection. Use the CEDA INSTALL command, for example:

```
CEDA INSTALL GROUP (BEACONN)
```

BEACONN FILE: SESSION DEFINITION

The BEACONN file also includes a sample session definition. When placed on the remote host, it defines the logical links by which the TUXEDO local domain communicates with the remote host.

The name of the sample session definition is BEATEST, located under the installation group name BEACONN. It looks like this:

Listing 6-15 Sample Session Definition in BEACONN File

```
DEFINE SESSION (BEATEST)      GROUP (BEACONN)
      CONNECTION (BEA)
      DE (CONNECT SNA EXAMPLE RDO SESSION)
      PROTOCOL (APPC)          AUTOCONNECT (YES)
      MODENAME ( **SMSNA100** ) MAXIMUM ( **6** , **3** )
```

AUTOCONNECT indicates how the activation of the session is to be negotiated. YES enables the CICS/ESA host to negotiate its own winner sessions when the connection is bound. (Remember that it is required that the CONNECT SNA software acquire the connection instead of the CICS/ESA host. However, when the stack acquires the connection, it can only bind the number of sessions identified as its winners. Setting the AUTOCONNECT parameter to YES causes the host to bind winner sessions immediately when the connection is acquired. Otherwise, the host's outbound clients must wait for winner sessions to bind.)

Replace ****MODE**** with either a CICS/ESA-supplied mode name, such as SMSNA100, or with your own defined mode name. If another set of session definitions exist for the BEA connection, this mode name must be unique among all sets defined to the connection. The mode name corresponds to the VTAM LOGMODE name.

The `MAXIMUM` option defines the total number of sessions in the set and the total number of contention winner sessions. To verify the installation, the total number of winner sessions must include those for the host and the remote stack. The installation verification process allows both sides to execute as the client. The total number of local contention winner sessions plus remote contention winner sessions must equal the number of sessions. The local number of sessions must equal the remote number of sessions.

BEASNA FILE: PROGRAM DEFINITION

The BEASNA file includes a sample program definition, shown in Listing 6-16. Replace the `LANGUAGE` variable `**LANG**` in the sample with either `C` or `COBOL` to identify the source type you have selected for the sample application.

Listing 6-16 Sample Program Definitions in BEASNA File

```
DEFINE PROGRAM(MIRRDPLC)      GROUP(BEASNA)
    DE(CONNECT SNA EXAMPLE CICS DPL CLIENT (MIRROR STRING))
    LANGUAGE(**LANG**)        DATALOCATION(ANY)

DEFINE PROGRAM(MIRRDTPC)      GROUP(BEASNA)
    DE(CONNECT SNA EXAMPLE CICS DTP CLIENT (MIRROR STRING))
    LANGUAGE(**LANG**)        DATALOCATION(ANY)

DEFINE PROGRAM(TOUPDTPS)      GROUP(BEASNA)
    DE(CONNECT SNA EXAMPLE CICS DTP SERVER (TOUPPER STRING))
    LANGUAGE(**LANG**)        DATALOCATION(ANY)

DEFINE PROGRAM(TOUPDPLS)      GROUP(BEASNA)
    DE(CONNECT SNA EXAMPLE CICS DPL SERVER (TOUPPER STRING))
    LANGUAGE(**LANG**)        DATALOCATION(ANY)
```

BEASNA FILE: REMOTE PROGRAM DEFINITION

The BEASNA file also contains a sample remote program definition, shown in Listing 6-17. The program definition is used by the CICS DPL client to identify the remote system and service for the DPL request. Replace the `REMOTESYSTEM` variable `**CONNECTION ID**` in the sample with the name of the connection for the remote LU.

Listing 6-17 Sample Remote Program Definition in BEASNA File

```
DEFINE PROGRAM(MIRRDPLS)      GROUP(BEASNA)
    DE(CONNECT SNA EXAMPLE DPL REMOTE PROGRAM DEFINITION)
    LANGUAGE(C)                DATALOCATION(ANY)
    REMOTESYSTEM(**CONNECTION ID**) REMOTENAME(MIRRDPLS)
```

BEASNA FILE: TRANSACTION DEFINITION

The BEASNA file also contains a sample transaction definition, shown in Listing 6-18.

Listing 6-18 Sample Transaction Definitions in BEASNA File

```
DEFINE TRANSACTION(DTPS)      GROUP(BEASNA)
    DE(CONNECT SNA EXAMPLE CICS DTP SERVER)
    TASKDATALOC(ANY)          PROGRAM(TOUPDTPS)

DEFINE TRANSACTION(H1PL)      GROUP(BEASNA)
    DE(CONNECT SNA EXAMPLE CICS DPL CLIENT - SYNCLEVEL 1)
    TASKDATALOC(ANY)          PROGRAM(MIRRDPLC)

DEFINE TRANSACTION(H2PL)      GROUP(BEASNA)
    DE(CONNECT SNA EXAMPLE CICS DTP CLIENT - SYNCLEVEL 2)
    TASKDATALOC(ANY)          PROGRAM(MIRRDPLC)

DEFINE TRANSACTION(H0TP)      GROUP(BEASNA)
    DE(CONNECT SNA EXAMPLE CICS DPL CLIENT - SYNCLEVEL 0)
    TASKDATALOC(ANY)          PROGRAM(MIRRDTPC)

DEFINE TRANSACTION(H2TP)      GROUP(BEASNA)
    DE(CONNECT SNA EXAMPLE CICS DTP CLIENT - SYNCLEVEL 2)
    TASKDATALOC(ANY)          PROGRAM(MIRRDTPC)
```

BEASNA FILE: PARTNER DEFINITION

The sample CICS/ESA client `MIRRDTPC` contains CPI-C verbs. The partner resource definition contains the CPI-C side information needed to allocate a conversation with the Tuxedo server. It contains the information about the remote LU and transaction program.

As shown in Listing 6-19, the `TPNAME` parameter identifies the transaction program that is invoked in the remote system. In this case, the name correlates to the `RNAME` value in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file. The `RNAME` there must match the `TPNAME` in the partner definition. (Notice in the sample `DMCONFIG` configuration file that a local service definition `MIRROR` exists. The `RNAME` in that definition matches the `TPNAME` in the sample partner definition.)

The profile resource definition can define conversation attributes, in particular `MODENAME`. In the sample, the `PROFILE` parameter can be replaced with a valid profile resource definition. The default profile name for the parameter is `DFHCICSA`. `DFHCICSA` is a CICS-delivered profile.

Use the `NETNAME` specified in the Connection definition of the remote LU (refer to “Create Connections at the Remote Host” on page 4-14).

Listing 6-19 Sample Partner Definition in BEASNA File

```
DEFINE PARTNER(MIRRDTPS)      GROUP(BEASNA)
DE(CONNECT SNA EXAMPLE CICS  DTP CLIENT USING CPIC VERBS
TPNAME(MIRRORSEV)           PROFILE(**DFHCICSA**)
NETNAME(**NETWORK NAME**))
```

Viewing Connection and Session Status

Once you have made the verification group definitions, you can view the status of connections and sessions using the following CICS/ESA system commands:

```
CEMT I CONN(BEA)              **view the status of the connection
CEMT I NET(**Netname**)       **View the status of the sessions
CEMT I MODENAME(**MODE**)     **View the status of the mode
```

Running the Sample Application

Now you are ready to validate the `CONNECT SNA` installation by running the sample application. At this stage, you should have completed the following prerequisites:

- ◆ Installed the BEA `CONNECT SNA` software
- ◆ Configured the `TUXEDO` and `CONNECT SNA` local domains
- ◆ Configured the `CICS/ESA` remote domain
- ◆ Initialized the stack processes
- ◆ Booted the servers

The sample application contains several scenarios. When the client runs as a `TUXEDO` client, the server runs as a `CICS/ESA` host. When the client runs as a `CICS/ESA` client, the server runs as a `TUXEDO` server.

Running the Application from a Tuxedo Client

In this scenario, the `toupclt` client performs a `tpcall` to the CICS/ESA host server. The server converts the text string you enter from lower-case to upper-case letters. The client may be run in transactional or non-transactional mode. The CICS/ESA server may be run as a DTP or DPL program

For example, enter the following command:

```
toupclt -s 0 -t DTP "hello world"
```

where:

`-s (0|2)`

Application Sync-Level.

0 = none (default)

2 = Transactional.

`-t (DPL|DTP)`

Remote Server Program.

DPL = Distributed Program Link (default)

DTP = Distributed Transaction Program

`-h`

Help

`" "`

Lowercase text string of up to 1915 characters.

If the TUXEDO client successfully executes the command, it displays the text string in upper-case letters, for example:

```
"HELLO WORLD"
```

Running the Application from a CICS/ESA Client

The following paragraphs depict two scenarios for running the application from a CICS/ESA client.

CICS/ESA Client with CPI-C

In this scenario, the CICS/ESA client sends a text string to the CONNECT SNA simple server. The string is re-displayed on the client's screen in reverse order.

Enter either of the following commands:

```
H0TP <string>
```

```
H2TP <string>
```

where:

```
H0TP = CPI-C Application Sync-Level 0
```

```
H2TP = CPI-C Application Sync-Level 2
```

<string> is a text string up to 1915 characters long.

If the CICS/ESA client successfully completes the transaction, it displays the text string in reverse order, for example:

You enter:

```
H0TP "HELLO WORLD"
```

The system returns:

```
"DLROW OLLEH"
```

CICS/ESA Client with DPL

Two transactions are available to execute the same program for the DPL sample. One is a simple request/response with the required sync-level 1, the other is a transactional request/response with sync-level 2.

Enter either of the following commands:

```
H1PL <string>
```

```
H2PL <string>
```

where:

H1PL = DPL Application Sync-Level 1

H2PL = DPL Application Sync-Level 2

<string> is a text string up to 955 characters long.

If the CICS/ESA client successfully completes the transaction, it displays a reverse (or mirror) image of the text string concatenated to the input text string, for example:

You enter:

```
H1PL "HELLO WORLD"
```

The system returns:

```
"HELLO WORLD : DLROW OLLEH"
```

7 Programming Considerations

Overview

This chapter is intended for application programmers who implement and integrate TUXEDO and host enterprise applications using *Application Program-to-Program Communication (APPC)*. Although primarily oriented toward TUXEDO application developers, it is also useful for non-TUXEDO application developers seeking to understand the relationship between the environments.

CONNECT SNA gives TUXEDO applications access to host APPC programs acting as servers. At the same time, APPC applications can act as clients and access TUXEDO system services. Because the client and server programs must be written for separate and very different environments, BEA CONNECT SNA allows the applications to be written using native programming facilities:

- ◆ The application programmer in a TUXEDO environment uses the familiar *Application-Transaction Monitor Interface (ATMI)* primitives in the application code. Service requests can be made to the CONNECT SNA Domain gateway just as they would be made to any other TUXEDO system server or gateway.
- ◆ The application programmer in the CICS/ESA environment can use CICS APPC API verbs. In addition, the programmer can use *Distributed Program Link (DPL)* techniques as either a client or server.
- ◆ The application programmer in the APPC environment can use *Common Programming Interface for Communications (CPI-C)* verbs for SNA-based applications.

- ◆ The application programmer in the *Information Management System (IMS)* environment can use implicit IMS verbs.

Programmers in each environment can continue to use the tools with which they are familiar to develop application software. It is not important that a programmer be well versed in the programming facilities of the other environments. What is important, however, is proper coordination between the applications running in each environment.

This chapter explains the various options offered by the CONNECT SNA domain and illustrates the most effective ways to implement these options.

APPC/IMS Programming

BEA CONNECT SNA supports non-transactional IMS servers using either the implicit APPC support for IMS or the explicit APPC interface using APPC/MVS calls from a user application.

Implicit APPC is similar and more convenient than the CICS/ESA DPL. Any IMS program that gets from and puts messages to the IMS message queue can be used without change as either a client or server.

To use the implicit APPC capabilities of IMS, you must modify the APPCM file in the SYS1.PARMLIB library provided with your CONNECT SNA product software. The configuration parameters in this file associate the LU with the IMS scheduler. You must identify the LU representing the application name used by CONNECT SNA to access the IMS region and the IMS system ID which provides scheduling for inbound requests. It is important to discuss with mainframe support personnel the changes you make to the APPCM file.

In Listing 7-1, the VTAM application major node is designated to be MVSLUO1 and the scheduling facility is designated to be the IMS control region IVP4.

Listing 7-1 APPCM File in SYS1.PARMLIB Library (Example Only)

```

SYS1.PARMLIB (APPCMxx)

LUADD ACBNAME(MVSLU01) BASE TPDATA(SYS1.APPCTP),
SCHED( IVP4),
SIDEINFO DATASET(SYS1.APPCSI)

SYS1.VTAMLST(MVSLU01)

MVSLU01 APPL  ACBNAME=MVSLU01,          ACBNAME FOR APPC          C
              APPC=YES,                  C
              AUTOSSES=3,                  C
              DDRAINL=NALLOW,              C
              DLOGMOD=APPCHOST,            C
              DMINWNL=3,                    C
              DMINWNR=3,                    C
              DRESPL=NALLOW,              C
              DSESLIM=6,                    C
              LMDENT=19,                    C
              MODETAB=APPCTAB,              C
              PARSESS=YES,                  C
              SECACPT=CONV,                  C
              SRBEXIT=YES,                  C
              VPACING=1                     C

```

The job which starts the IMS subsystem should have the APPC parameter set to Y. The example in Listing 7-2 illustrates such a job, but is not intended to be used under actual conditions. Use your own custom job for starting IMS.

Listing 7-2 IMS Subsystem Start Job (Example Only)

```

)IVP51F41 EXEC,PROC=IMSFP,TIME=(1440),,
              AGN=IVP,                    AGN NAME
              SOUT='*',                    SYSOUT CLASS
              MBR=DFSIVAG,                 PROGRAM NAME
              PSB=DFSIVPG,                 PSB NAME
              NBA=06,                      INITIAL FAST PATH DB BUFFERS
              OBA=05,                      SECONDARY FAST PATH DB BUFFERS
              TLIM=10,                     IFP TERMINATION LIMIT
              SOD=,                        SPIN-OFF DUMP CLASS
              IMSID=IVP4,                  IMSID OF IMS CONTROL REGION
              APPC=Y,                      ENABLE IMPLICIT APPC
              PREINIT=DC                   PROCLIB DFSINTXX MEMBER

```

CICS/ISC Programming

The CICS/ESA Intersystem Communications (ISC) facility permits multiple independent CICS/ESA systems, normally executing on different hosts and possibly different computing platforms, to cooperate while performing various computing tasks.

CICS/ESA defines a set of five distinct services with ISC as a common foundation:

- ◆ Asynchronous Processing (AP)
- ◆ Function Request Shipping (FRS)
- ◆ Transaction Routing (TR)
- ◆ Distributed Program Link (DPL)
- ◆ Distributed Transaction Processing (DTP)

The first four ISC services are categorized as *implicit*. These services are not used directly by user-written CICS/ESA applications, but are employed indirectly and automatically by the CICS/ESA system on behalf of them. The services are supported between peer CICS/ESA systems; for example, between two mainframe CICS/ESA systems connected by an ISC link. Currently, `CONNECT SNA` supports DPL for BEA TUXEDO.

DTP, on the other hand, is an *explicit* service that can be used by CICS/ESA applications to cooperatively distribute business functionality. `BEA CONNECT SNA` supports DTP services between peer CICS/ESA systems, as well as between CICS/ESA systems and non-CICS/ESA systems (for example, a DTP application conversing with an APPC/MVS application). `BEA CONNECT SNA` also supports DTP for BEA TUXEDO ATMI applications.

Multi-Region versus Multi-Processor Operations

CICS/ESA applications may also use another form of intersystem communication, called Multi-Region Operation (MRO). MRO is typically used to improve the performance of a CICS/ESA system by partitioning it into multiple, physically distinct regions.

MRO can be compared to a BEA TUXEDO Multi-Processor (MP) environment. In a two-platform MP application domain, the first platform might be dedicated to handle terminal and workstation clients, and the second platform might be dedicated to handle the application logic and file services. In a similar MRO configuration, all terminals might be placed in one CICS/ESA region (called a terminal-owning region), and applications and data stores might be placed in another CICS/ESA region (called an application-owning or file-owning region).

The biggest difference between this MP environment and the MRO configuration is that the MP environment spans multiple platforms, whereas the MRO configuration resides on a single host.

BEA CONNECT SNA does not relate directly to MRO due to MRO's single platform nature. However, it does support ISC operations.

ISC Operations

The ISC facility provides a mechanism for multiple CICS/ESA systems operating on different host computers to communicate with each other, but ISC is more than a simple communications mechanism. It implements a well-defined set of protocols by which connected CICS/ESA systems cooperate in the execution of recoverable, distributed units of work. This important concept is fundamental to understanding the relationship between connected CICS/ESA systems.

IBM's Systems Network Architecture (SNA) defines the protocols for communications between IBM systems in a multi-system environment. ISC is an implementation of a specific SNA protocol known as Advanced Program-to-Program Communications (APPC, also referred to as LU TYPE 6.2 or LU 6.2).

The ISC for APPC is not a strict implementation of the SNA architecture specification. In particular, CICS/ESA makes use of special Function Management Headers (FMH) which are not part of the SNA architecture specification. Other deviations are documented in the *IBM Distributed Transaction Programming Guide*.

Although ISC can be used to connect CICS/ESA systems residing on a single host computer, ISC is normally used to connect CICS/ESA systems which reside on physically-distinct host computers. Furthermore, systems which are geographically separated may be connected.

IBM's Advanced Communication Facility / Virtual Telecommunications Access Method (AFC/VTAM) is used to define and implement a logical connection between two CICS/ESA systems. Using the VTAM facilities, CICS/ESA systems establish one or more concurrent sessions for communications. Each session supports one logical conversation between the two systems. These logical conversations carry the inter-system traffic associated with the various ISC functions.

Similar to BEA TUXEDO conversational service requests, LU 6.2 is a half-duplex protocol; that is, session traffic can flow in only one direction at a time. The session partners are responsible for coordinating their use of the session such that when one is sending, the other is receiving. Both session partners cannot send (or receive) at the same time.

Asynchronous Processing

This ISC service allows an application to start a transaction on a remote system. Unlike DPL and Function Request Shipping, the calling application does not wait for the remote task to complete. The calling transaction initiates an asynchronous process with the CICS/ESA interval **START** command. Information can be passed to the back-end transaction in the **FROM** option of the **START COMMAND**. The back-end transaction can then issue a **RETRIEVE** command to get this data. The transaction can be defined as remote, indicating the system on which the transaction is to be run, or the remote system can be named explicitly in the **SYSID** option of the **START** command.

Function Request Shipping

Remote resources, such as VSAM files, DLI requests, Transient Data Queues, and Temporary Storage Queues can be made available to a local application. The application is not necessarily aware that the requested resource is on another CICS/ESA region. Files and Queues are defined as remote with system administration utilities, but could optionally be specified as remote using the **SYSID** option of the resource request command.

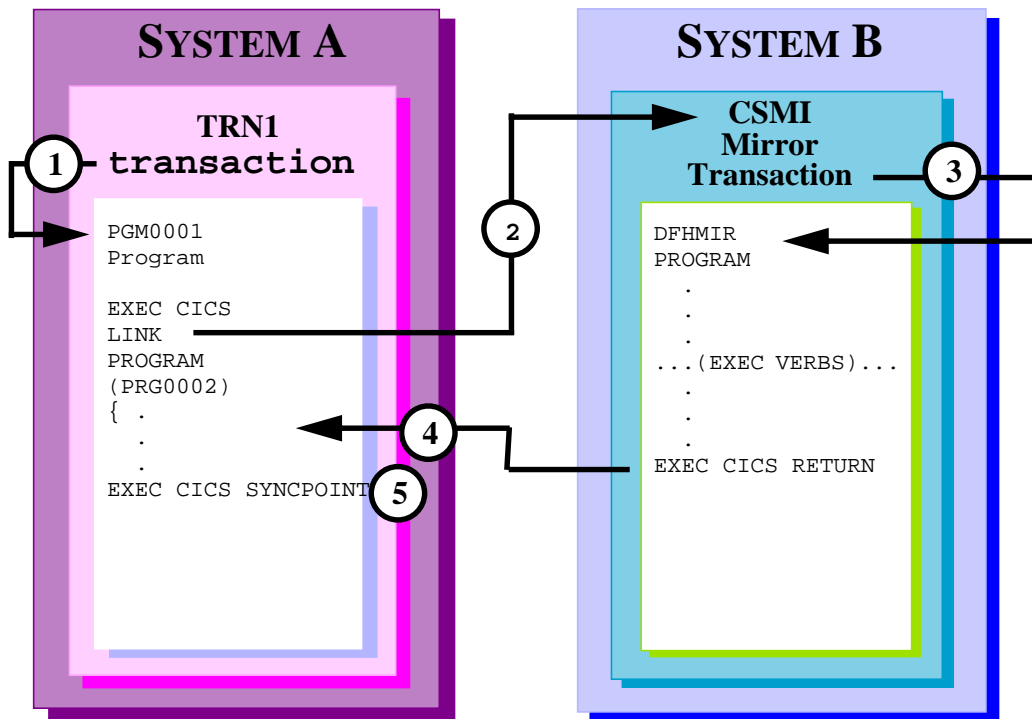
Transaction Routing

Transaction routing provides a terminal connected to a CICS/ESA region with the ability to run a transaction on another CICS/ESA region in a fashion transparent to the terminal user. A transaction can be defined as a remote transaction in the system initialization table. This definition indicates on which remote system the transaction is to run.

Distributed Program Link

Distributed Program Link (DPL) allows a CICS/ESA program to call (link to) another CICS/ESA program located on a remote system. This is a synchronous process; control does not return to the calling program until the called program has finished executing. See Figure 7-1 for a generic example of this process.

Figure 7-1 Generic DPL Transaction



Step-by-Step Description: Generic DPL Transaction

1. CICS/ESA transaction TRN1, executing on CICS/ESA System A, issues an EXEC CICS LINK to program PRG002, which resides on CICS/ESA System B. TRN1 “waits” while the LINK request is processed.
2. The LINK implementation internally initiates a special mirror transaction on system B, using an APPC conversation.
3. The mirror transaction runs the system program DFHMIR. It starts the user program PRG0002 and passes any input data to it. The mirror program uses APPC verbs to pass data to and from the user program.
4. Upon completion, PRG0002 returns control to the mirror program which sends any return data back to PGM0001. PGM001 continues execution and may invoke multiple DPL requests using the same conversation and mirror program task.
5. The conversation with the mirror program remains active and available to PGM0001 until a SYNCPOINT is issued. The conversation is used to coordinate the SYNCPOINT processing and is deallocated when the SYNCPOINT is complete.

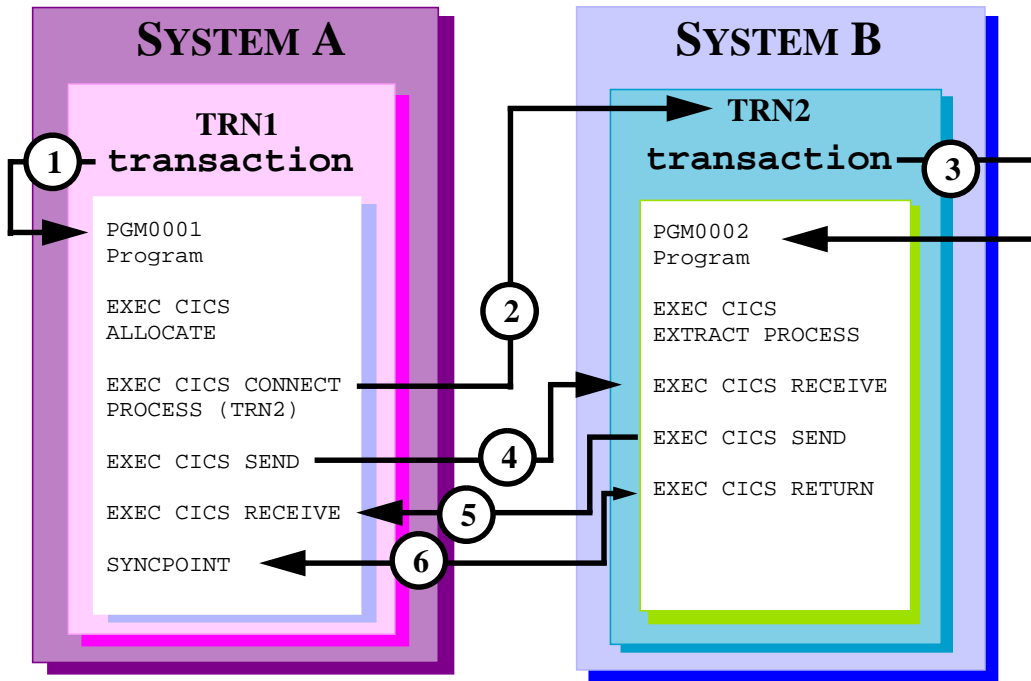
The functionality of the remote program is minimally constrained. It can access files and databases as well as linking to other programs, but does not do terminal I/O. In general, an application program is not aware that it is calling a program located on a remote system. In most cases, Distributed Program Link is completely transparent to both the calling and called programs. Implicit support for full recovery and restart of all CICS/ESA-controlled resources, such as files and databases, is automatically provided by the CICS/ESA system.

Note: If a client sends a data buffer to a remote CICS/ESA system as part of a DPL and no data is modified, the remote system returns no data and the length is zero.

Distributed Transaction Processing

Distributed Transaction Processing (DTP) is the only ISC services that provides for direct, application-to-application communications. DTP takes place between two conversation partners, both of which are normally user-written CICS/ESA applications. One conversation partner initiates a conversation with a counterpart on a remote system, and must explicitly identify the remote system. The conversation partner on the remote system must accept the requested conversation. Using the full capabilities of ISC, DTP applications send and receive application-defined data streams and engage in a series of application-defined interactions. (See Figure 7-2 for a generic example of this process.)

Figure 7-2 Generic DTP Transaction



Step-by-Step Description: Generic DTP Transaction

1. Transaction TRN1, invokes program PGM0001 executing on CICS/ESA System A.
2. Program PGM0001 initiates (requests) a conversation with transaction TRN2, located on CICS/ESA System B, which in turn, invokes program PGM0002.
3. Transaction TRN2, executing on CICS/ESA System B, invokes program PGM0002.
4. Program PGM0001 sends data to program PGM0002, then relinquishes control to PGM0002.
5. Program PGM0002 sends data to PGM0001, perhaps the response to the previous request. PGM0001 receives the data sent by PGM0002.
6. PGM0001 and PGM0002 synchronize the data, acknowledge the conversation, and end the conversation.

DTP offers the best of two worlds. On one hand, it provides support for completely customized application protocol semantics. On the other hand, it allows the applications to select the desired level of support for resource recovery and restart:

- ◆ At sync-level 0, the conversation partners elect to forego CICS/ESA system-assisted synchronization altogether.
- ◆ At sync-level 1, the conversation partners can exchange special synchronization verbs as an aid in maintaining synchronization.
- ◆ At sync-level 2, the CICS/ESA system provides full recovery and restart of all controlled resources, such as files and databases. More specifically, a task can initiate tasks on other systems and the collection of tasks can be synchronized into a common unit of work. If there is a failure in any of the tasks, the entire collection can be rolled back.

Note: It is important to understand that, by definition, ISC operates only at sync-level 2.

DTP conversations are LUTYPE 6.2 conversations. They are a half-duplex form of data exchange between two transactions. It is half-duplex because only one side of the conversation can send data at a time. A conversation is initiated by the application using the *Application Program-to-Program Communication (APPC)* protocol. Because of this, the application must be coded to respect the various conversation states. These states are documented in the *IBM CICS/ESA Intercommunication Guide* and identify which APPC verbs are legal at the various stages of a conversation. LUTYPE 6.2 conversations can be mapped or unmapped conversations.

Unmapped or base-level conversation support is the minimum support level required for LUTYPE 6.2 product implementation. Mapped conversations are at a higher level than base conversations and do not require the application to be aware of the SNA data stream format. The state transitions are less restrictive and the command set is more robust. In addition to this, the application has access to CICS/ESA data areas to determine the status of the conversation and the results of certain APPC commands.

CICS/ESA Sync-Levels

In general, the synchronization level determines the cooperation between communicating partners needed to coordinate changes to their respective resources (such as files and databases). It also defines specific protocol flows required to support a requested synchronization level. The APPC architecture defines three distinct levels of synchronization. These three levels are Level 0, Level 1, and Level 2 (corresponding to the SNA-defined levels of NONE, CONFIRM, and SYNCPOINT, respectively).

LEVEL 0 (NONE)

No synchronization at the protocol level is provided. This level is used when synchronization is not required or supported. This is very similar to non-transactional Tuxedo System service requests.

LEVEL 1 (CONFIRM)

This level supports the exchange of private, application-level, synchronization requests between conversing partners. These partners, or applications, are totally responsible for defining and executing the synchronization process to ensure the integrity of all resources. The CICS/ESA system is not involved in the synchronization process at this level. There is no confirm capability in the Tuxedo System. Any CICS/ESA application attempting to initiate a sync-level 1 operation with Tuxedo causes an error condition.

LEVEL 2(SYNCPOINT)

At this level, the CICS/ESA system is fully involved in the synchronization process and provides complete, automatic support of sync-pointing, including rollback. The CICS/ESA system defines and coordinates the synchronization process and is responsible for ensuring the integrity of all CICS/ESA-managed resources.

In the case of DTP, the communicating applications participate in the synchronization process and are responsible for the management and integrity of any non-CICS/ESA resources to which they gain access.

Because DTP takes place between two application-level conversation partners, it may operate at any of the three defined synchronization levels. In other words, the conversation partners are permitted to specify the desired level of synchronization and participate in the synchronization process only to the extent specified.

If the user DTP conversation is at sync-level 0 or 1, the CICS/ESA system does not propagate sync-points from one system to another via the conversation. Thus the user can explicitly suspend CICS/ESA *ACID Properties*. A DTP conversation at sync-level 2 causes a CICS/ESA system to propagate sync-points automatically via the conversation as well as via subordinate conversations.

DPL can also operate at sync-level 1, in which case the user is responsible for maintaining the ACID properties. Sync-level 1 DPL conversations usually occur because one or both of the systems are incapable of operating at sync-level 2. BEA CONNECT SNA can support DPL requests at both sync-levels 1 and 2, and can support DTP conversations at sync-levels 0 and 1.

Time-out and Error Handling

The /Domain's global timer is used to determine excessive time for a service request. If a time-out occurs, a FAILURE indication is raised and `tperrno` is set to `TPETIME`. The domain gateway cleans up any resources allocated to the failed communication.

CONNECT SNA sets `tperrno` values when an error occurs. Appendix B, “ATMI to CPI-C Function Mapping” describes the CPI-C return codes and their mapping to ATMI return codes.

Refer to Appendix D, “Error Messages” for error code listings.

Application-to-Application Programming

This section provides *Transaction* scenarios for the following programming environments supported by CONNECT SNA:

- ◆ Distributed Program Link (DPL)
- ◆ Distributed Transaction Processing (DTP)
- ◆ CPI-C

Caution: The scenarios in this section demonstrate how Tuxedo ATMI calls relate to CICS/ESA programming structures. They are intended neither for use in developing application code, nor for the replacement of existing application code. The use of any of these examples in actual situations may have unpredictable results.

Distributed Program Link (DPL)

Using CONNECT SNA, TUXEDO applications can make ATMI service calls to invoke programs running in a remote CICS/ESA DPL environment. CICS/ESA programs can in turn use the `EXEC CICS LINK` command to invoke user-defined TUXEDO services. This feature is available to either new or existing CICS/ESA programs and TUXEDO applications. DPL can only be used by BEA TUXEDO application domains connected to a CICS/ESA system.

Consider the program issuing the distributed link request or ATMI service request to be the *Client* program. Consider the executing remote program or service to be the *Server* program. The client does not have to be aware of the server's location.

TUXEDO programs can use both the request/response and conversational models to invoke programs on the host. (However, request/response models are preferred.)

Local and remote services are defined with the `FUNCTION=DPL` option in the `DM_LOCAL_SERVICES` and `DM_REMOTE_SERVICES` section in the `DMCONFIG` file, indicating that the DPL ISC function is invoked.

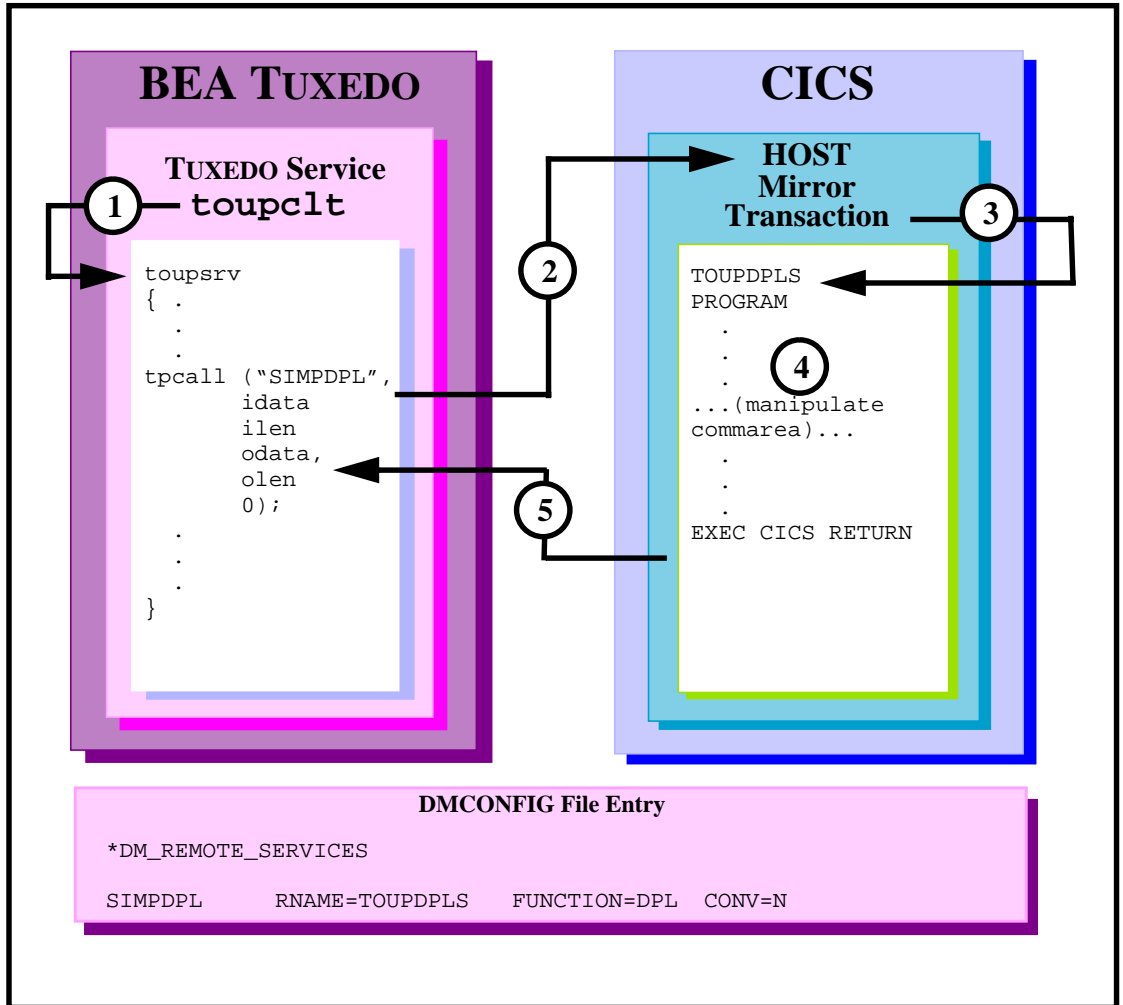
Host DPL requests, in turn, can also be compared to TUXEDO services using either call model. The request/response model is very similar to DPL. The conversational model, although supported in DTP, is less efficient and programmatically more complex than the request/response model. It is recommended that you use a DTP model when invoking conversational transactions.

It is also recommended you use a DPL model when invoking the same request/response or conversational request multiple times in a single transaction. With DPL, ISC starts a long-running mirror transaction to the CICS/ESA DPL server, enabling the server to process each of the requests over the same conversation. With DTP, each of the requests establishes a separate conversation with the server and the conversation remains outstanding until the transaction is committed.

The following examples represent a few of the many programming scenarios available for using DPL and TUXEDO service invocations. These examples employ the most natural and efficient approaches.

Note: To run *Transaction* client/server scenarios, the CONNECT SNA software must be licensed for sync-level 2 operations.

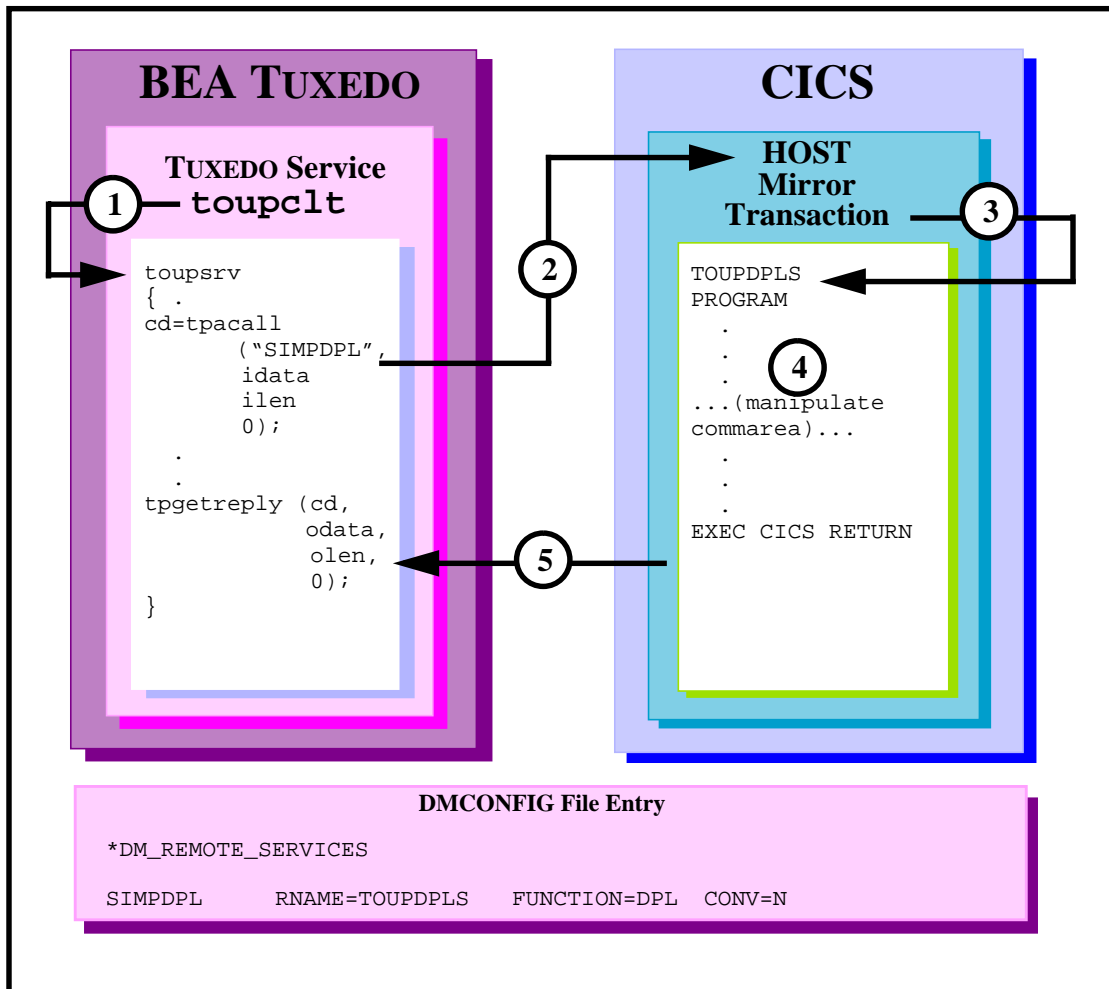
Figure 7-3 Tuxedo Client Request/Response to CICS/ESA DPL



Step-by-Step Description: Tuxedo Client Request/Response to CICS/ESA DPL

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpcall` for `SIMPDPL`, which is advertised in the `*DM_REMOTE_SERVICES` section of the `DMCONFIG` file.
3. Host mirror transaction starts `TOUPDPLS` program and passes `idata` buffer contents for processing.
4. The `TOUPDPLS` program processes data.
5. The CICS/ESA server returns the `commarea` into the client's `odata` buffer.

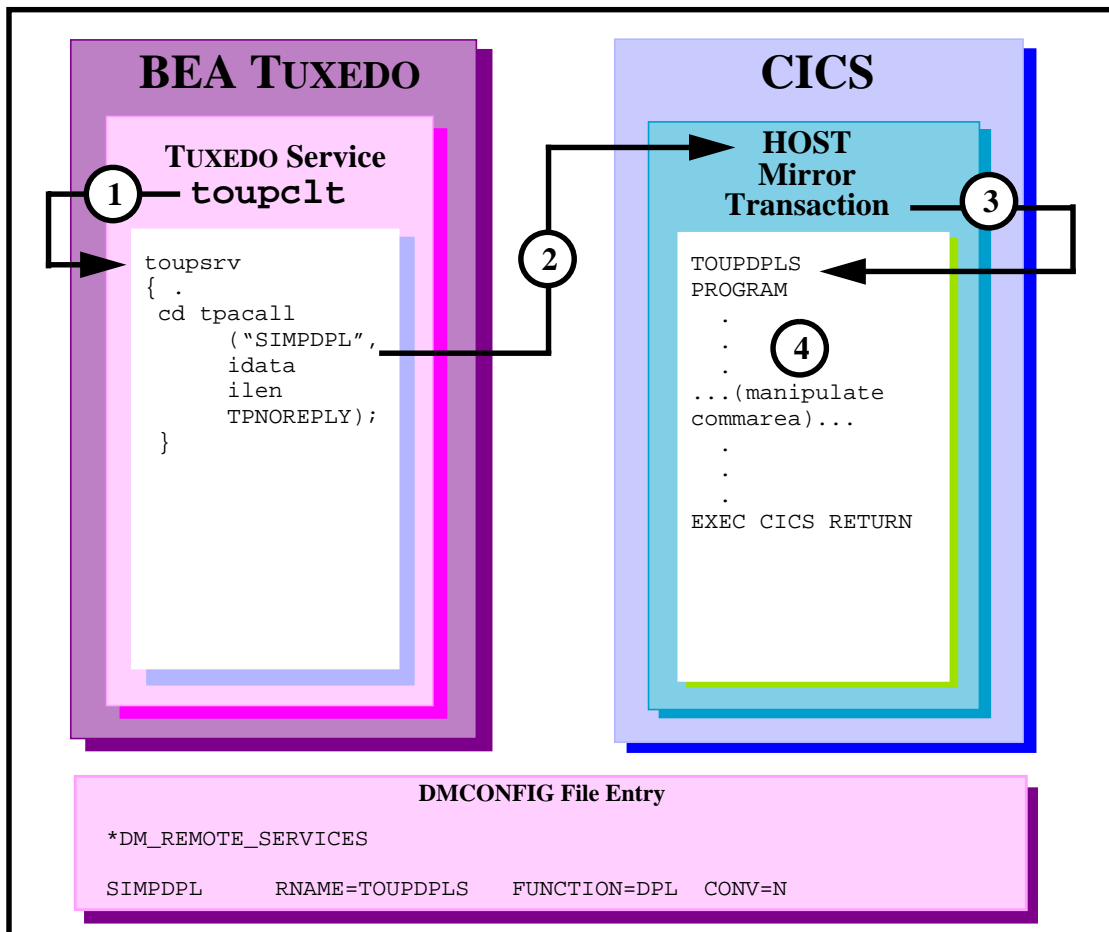
Figure 7-4 Tuxedo Client Asynchronous Request/Response to CICS/ESA DPL



Step-by-Step Description: Tuxedo Client Asynchronous Request/Response to CICS/ESA DPL

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpacall` for `SIMPDPL`, which is advertised in the `*DM_REMOTE_SERVICES` section of `DMCONFIG` file.
3. Host mirror transaction starts `TOUPDPLS` program and passes `idata` buffer contents for processing.
4. The `TOUPDPLS` program processes data.
5. The CICS/ESA system returns the `commarea` into the client's `tpgetreply` `odata` buffer.

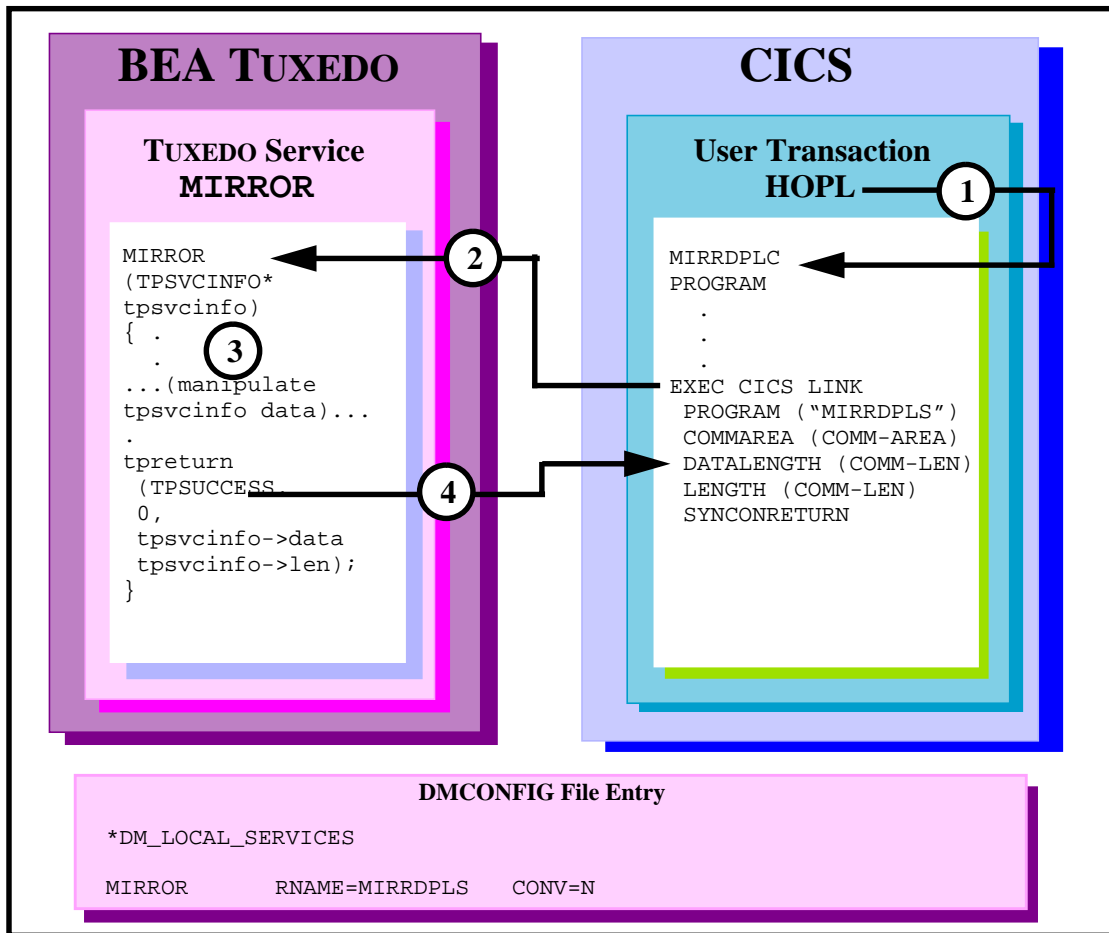
Figure 7-5 Tuxedo Client Asynchronous Request/Response with No Reply to CICS/ESA DPL



Step-by-Step Description: Tuxedo Client Asynchronous Request/Response with No Reply to CICS/ESA DPL

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpacall` for `SIMPDPL`, which is advertised in the `*DM_REMOTE_SERVICES` section of `DMCONFIG` file. The `toupsrv` service uses `TPNOREPLY` to specify that no reply is expected.
3. Host mirror transaction starts `TOUPDPLS` program and passes `idata` buffer contents for processing.
4. The `TOUPDPLS` program processes data.

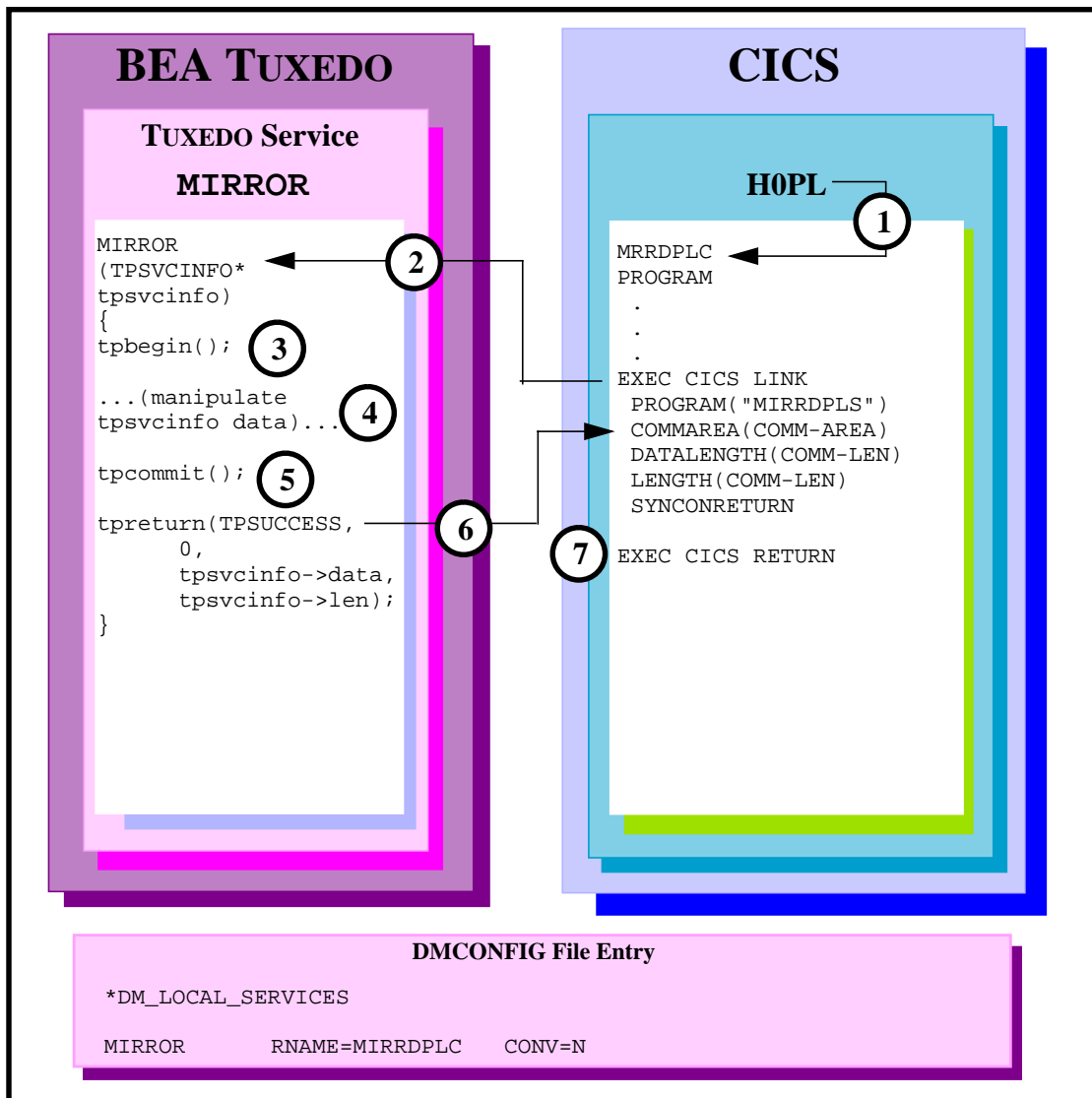
Figure 7-6 CICS/ESA DPL to Tuxedo Request/Response Server



Step-by-Step Description: CICS/ESA DPL to Tuxedo Request/Response Server

1. User-entered `HOPL` invokes `MIRRDPLC` program.
2. The `EXEC CICS LINK` command causes the advertised service mapped to `MIRRDPLS` (in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file) to execute.
3. The `MIRROR` service processes the data received in the service `TPSVCINFO` data buffer from the `EXEC CICS LINK`.
4. The `tpreturn` call returns the data into the `COMM-AREA` buffer.

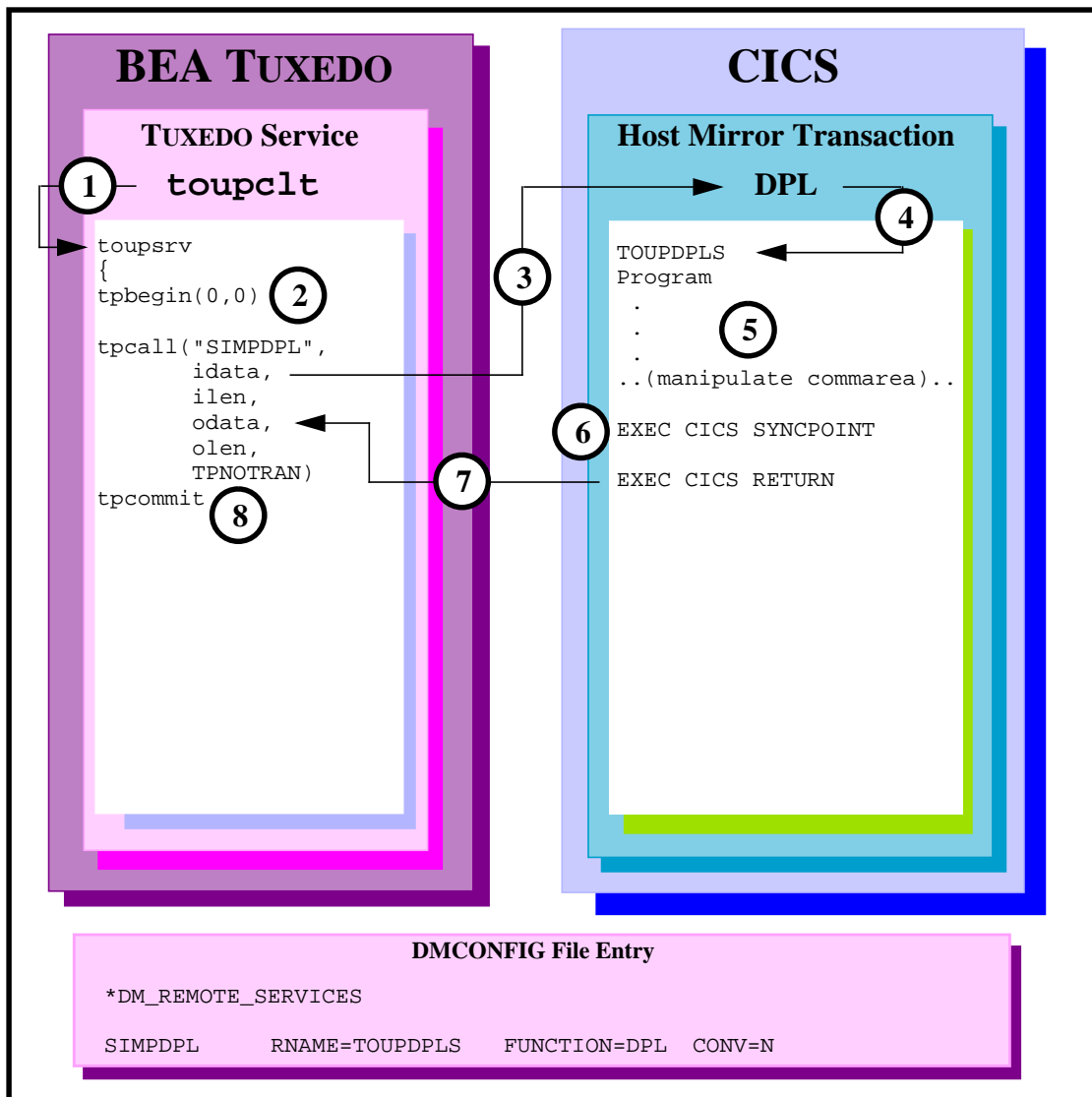
Figure 7-7 CICS/ESA DPL to Tuxedo Request/Response Server, Service in Autonomous Transaction



Step-by-Step Description: CICS/ESA DPL to Tuxedo Request/Response Server, Service in Autonomous Transaction

1. User-entered `H0PL` invokes `MIRRDPLC` program.
2. The `EXEC CICS LINK` command causes the advertised service mapped to `MIRRDPLS` (in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file) to execute. The `SYNCONRETURN` option indicates that the invoked service will not participate in the CICS/ESA transaction.
3. The `MIRROR` service request `tpbegin` incorporates all further operations in a transaction.
4. The `MIRROR` service processes the data.
5. The `tpcommit` indicates the end of the transaction; all updates performed within the service transaction are to be committed.
6. The `tpreturn` call returns the data into the `commarea` buffer.
7. The `EXEC CICS SYNCPOINT` is an explicit commit request. All updated resources in the CICS/ESA transaction are committed.

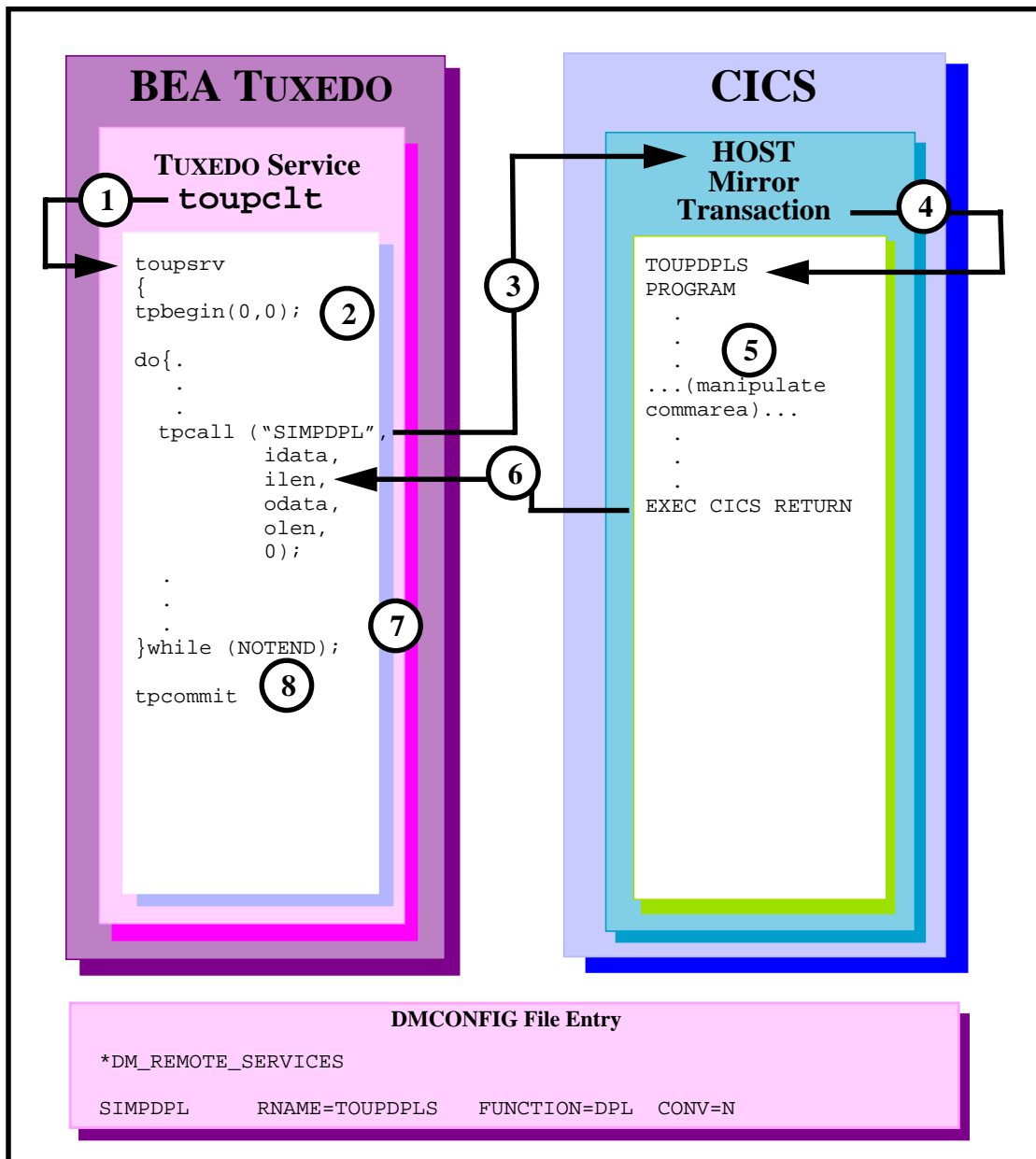
Figure 7-8 Tuxedo Client Request/Response to CICS/ESA DPL, in Autonomous Transaction



Step-by-Step Description: Tuxedo Client Request/Response to CICS/ESA DPL, in Autonomous Transaction

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpbegin` to start the transaction.
3. The `toupsrv` service issues `tpcall` for `SIMPDPL`, which is advertised in the `*DM_REMOTE_SERVICES` section of the `DMCONFIG` file. The `TPNOTRAN` parameter indicates the CICS/ESA application does not participate in the service transaction.
4. Host mirror transaction starts `TOUPDPLS` program and passes `idata` buffer contents for processing.
5. The `TOUPDPLS` program processes data.
6. The `EXEC CICS SYNCPOINT` is an explicit commit request. All updated resources in the CICS/ESA transaction are committed.
7. The CICS/ESA server returns the `commarea` into the client's `odata` buffer.
8. The `toupsrv` service `tpcommit` request signals the successful completion of the transaction, causing a commit of its own updated resources.

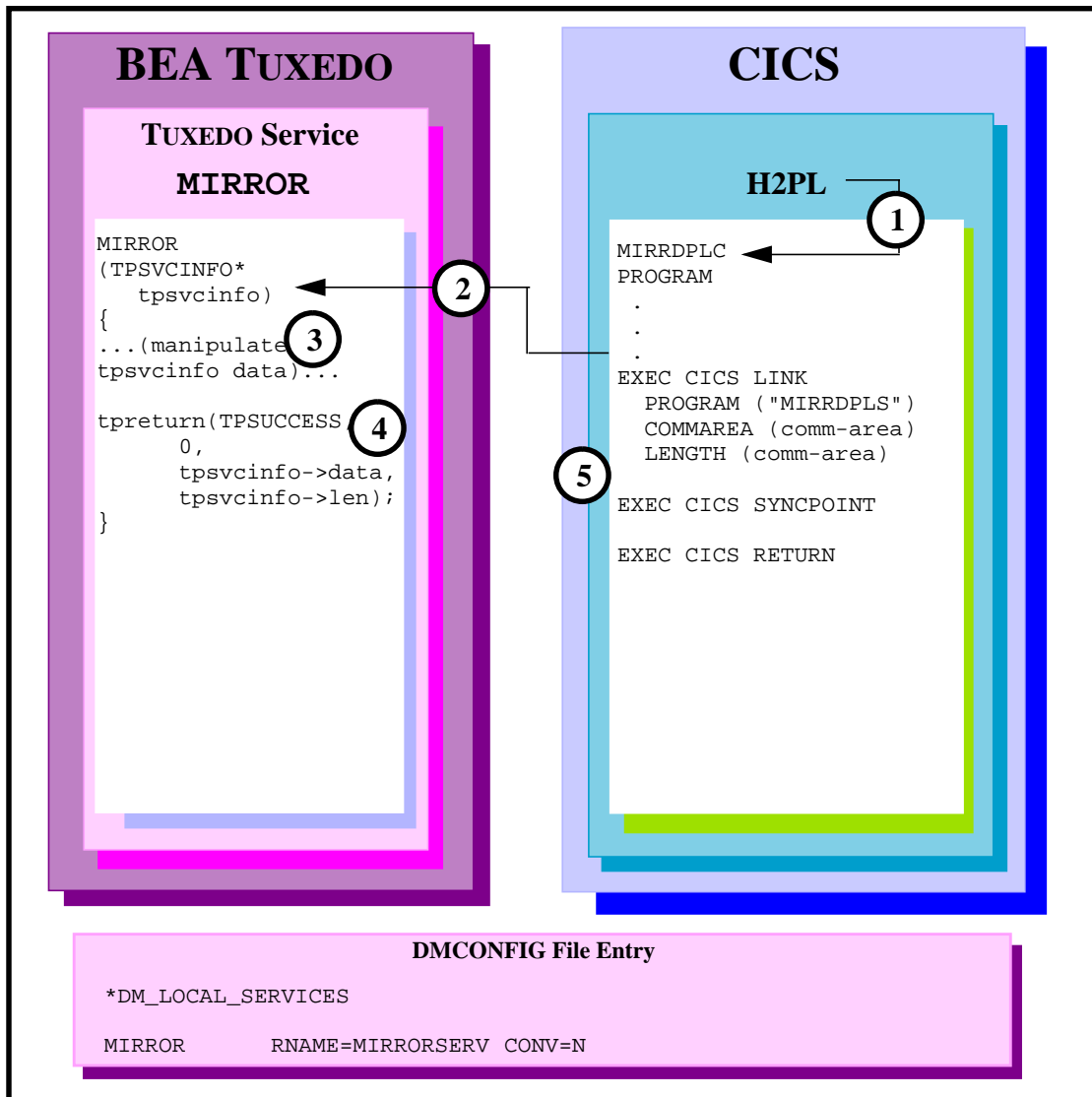
Figure 7-9 Transactional Tuxedo Client Multiple Requests/Responses to CICS/ESA DPL



Step-by-Step Description: Transactional Tuxedo Client Multiple Requests/Responses to CICS/ESA DPL

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpbegin` to start the transaction.
3. The `toupsrv` service issues `tpcall` for `SIMPDPL`, which is advertised in the `*DM_REMOTE_SERVICES` section of the `DMCONFIG` file. The `tpcall` is requested multiple times within the same transaction.
4. Host mirror transaction starts `TOUPDPLS` program and passes `idata` buffer contents for processing. The host mirror transaction remains as a long-running task to service all further requests on the transaction.
5. The `TOUPDPLS` program processes data.
6. The CICS/ESA system returns the `commarea` into the client's `odata` buffer.
7. Step 3 through Step 6 are repeated until the `toupsrv` service loop end conditions are met.
8. The `tpcommit` request indicates the successful completion of the transaction, causing a commit of its own resources and the resources held by the host mirror transaction.

Figure 7-10 Transactional CICS/ESA DPL to Tuxedo Request/Response Server



Step-by-Step Description: Transactional CICS/ESA DPL to Tuxedo Request/Response Server

1. User-entered `H2PL` invokes `MIRRDPLC` program.
2. The `EXEC CICS LINK` command causes the advertised service mapped to `MIRRDPLS` (in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file) to execute. The invoked service participates in the CICS/ESA transaction.
3. The `MIRROR` service processes the data.
4. The `tpreturn` call returns the data into the `commarea` buffer.
5. The `EXEC CICS SYNCPOINT` is an explicit commit request indicating a successful end of the conversation. All updated resources in the transaction are committed.

(This page intentionally blank)

Distributed Transaction Processing (DTP)

The following examples represent a few programming scenarios for using DTP and TUXEDO service invocations.

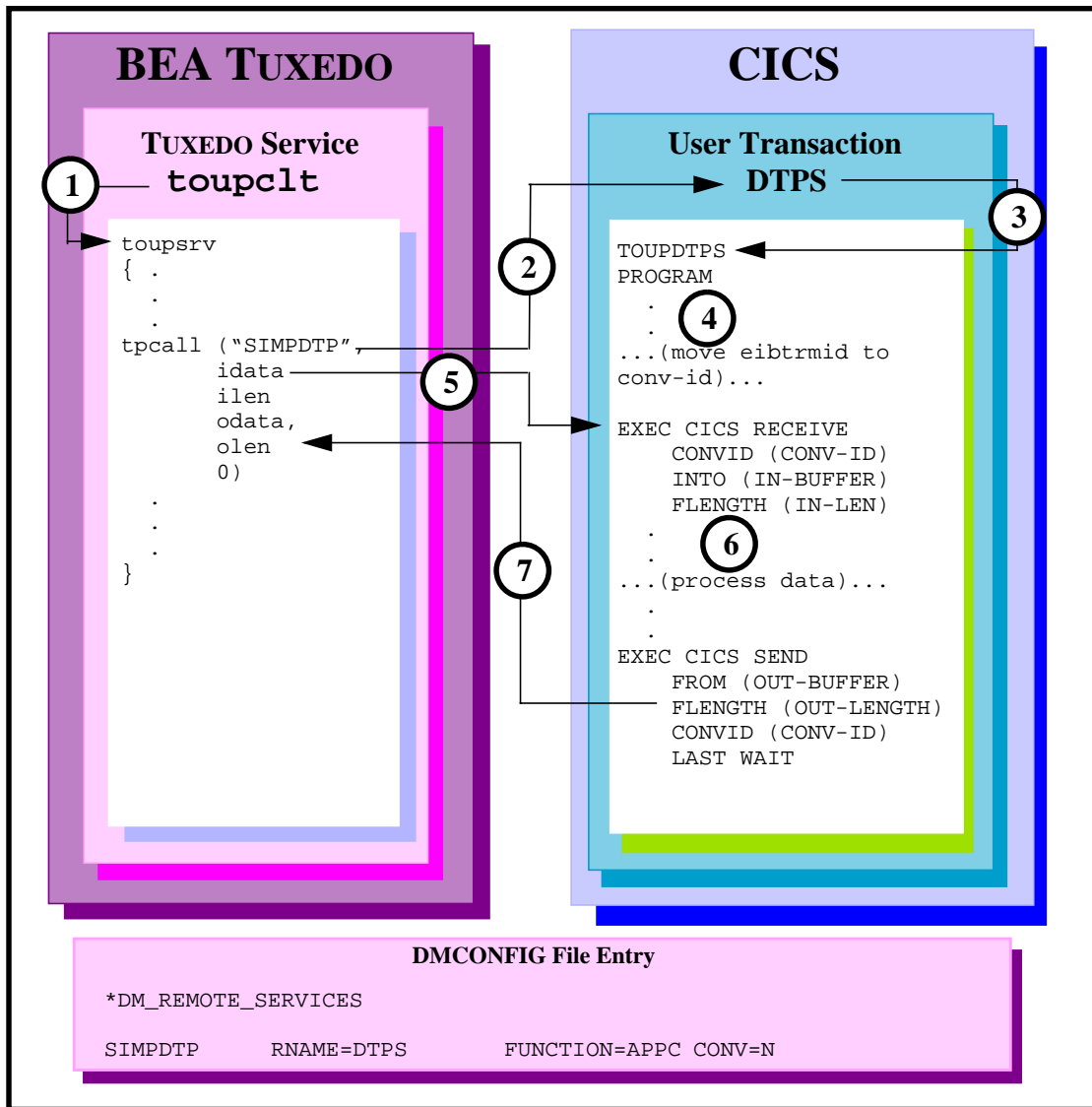
The recommended method of performing DTP services is to use TUXEDO conversational services instead of request/response services. The conversational service enables a client and a server to communicate as needed over the CONNECT SNA session conversation.

Although it is most suited for the DPL environment, the `tpcall` is used in the examples for a request/response to a DTP server.

It is recommended you use a DPL model when invoking multiple request/response calls in a single transaction. With DPL, a long-running mirror transaction to the CICS/ESA DPL server is started, enabling the server to process each of the requests over the same conversation. With DTP, each of the ATMI requests will establish a separate conversation; the conversation remains outstanding until the transaction is committed.

Note: To run *transactional* client/server scenarios, the CONNECT SNA software must be licensed for sync-level 2 operations.

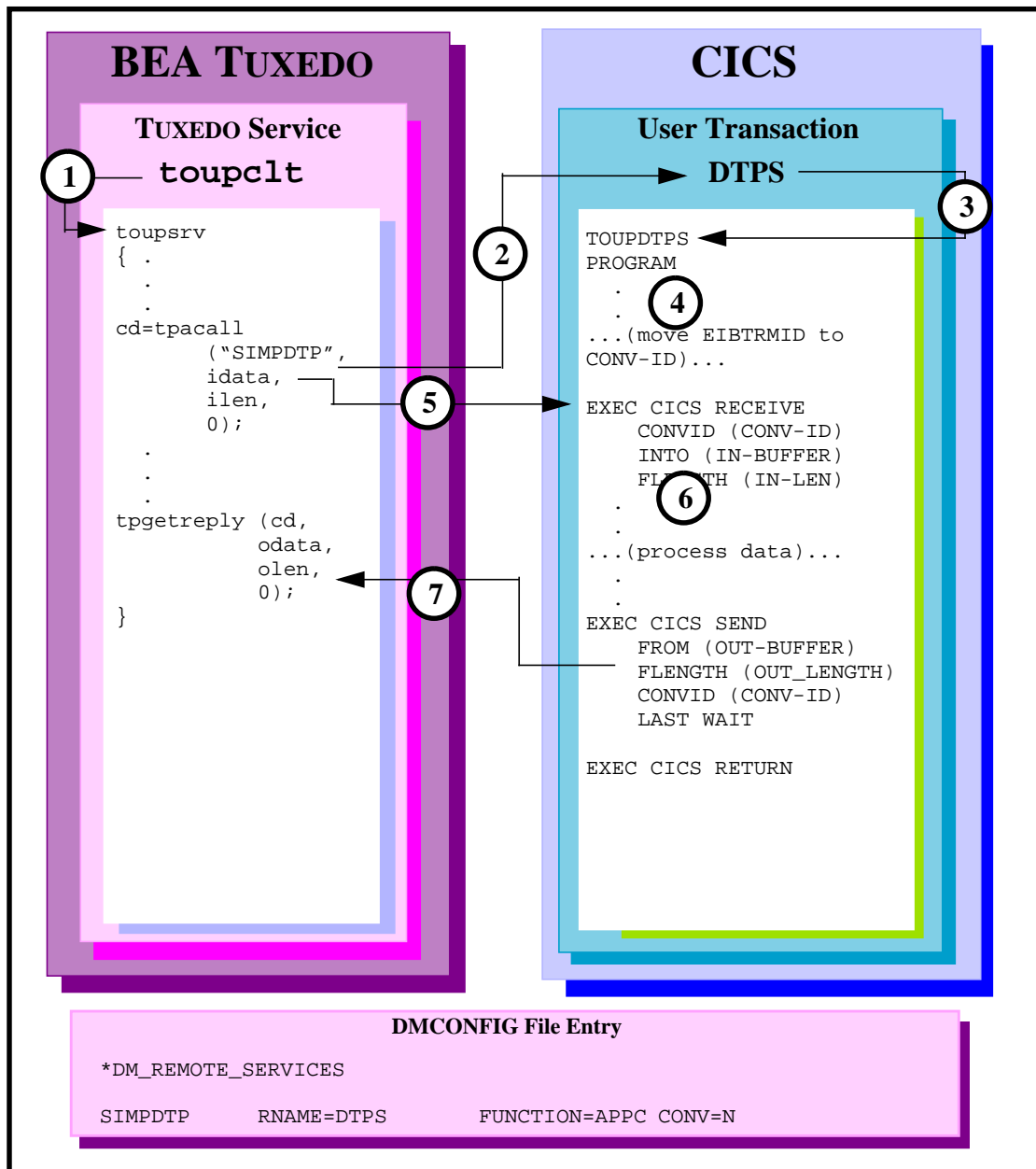
Figure 7-11 Tuxedo Client Request/Response to CICS/ESA DTP



Step-by-Step Description: Tuxedo Client Request/Response to CICS/ESA DTP

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpcall` for `SIMPDTP`, which is advertised in the `*DM_REMOTE_SERVICES` section of `DMCONFIG` file.
3. User transaction `DTPS` starts `TOUPDTPS` program.
4. It is recommended you save the `eibtrmid` to a program variable. This value may be used to identify the specific conversation in your CICS/ESA APPC verbs.
5. The `EXEC CICS RECEIVE` command receives the `idata` buffer contents for processing.
6. The `TOUPDTPS` program processes data.
7. The `EXEC CICS SEND` command returns the `OUT-BUFFER` contents into the clients `odata` buffer. `LAST` indicates the conversation is finished. `WAIT` suspends processing until the data has successfully been received.

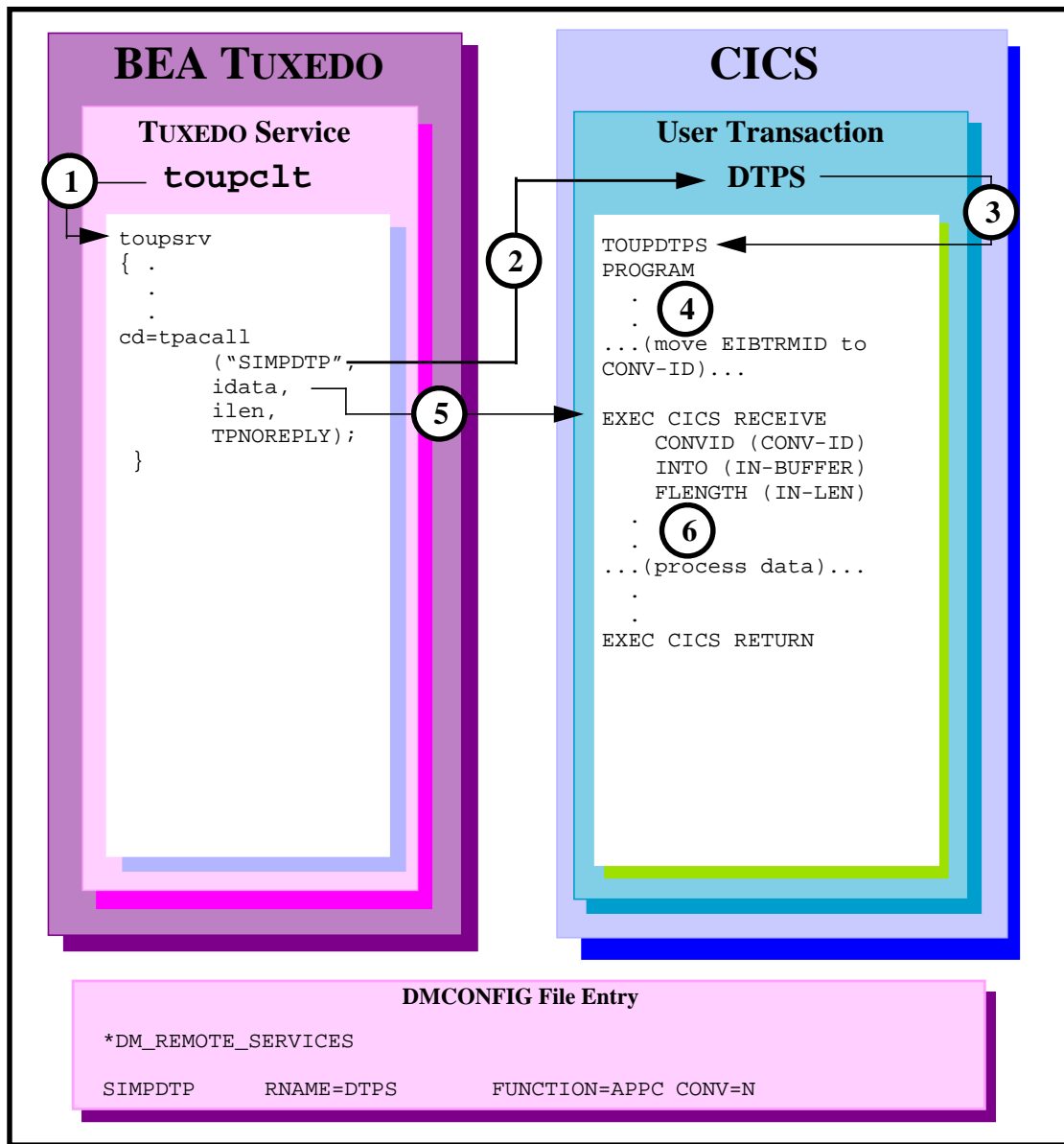
Figure 7-12 Tuxedo Client Asynchronous Request/Response to CICS/ESA DTP



Step-by-Step Description: Tuxedo Client Asynchronous Request/Response to CICS/ESA DTP

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpacall` for `SIMPDTP`, which is advertised in the `*DM_REMOTE_SERVICES` section of the `DMCONFIG` file.
3. User transaction `DTPS` starts `TOUPDTPS` program.
4. It is recommended you save the `EIBTRMID` to a program variable. This value may be used to identify the specific conversation in your CICS/ESA `APPC` verbs.
5. The `EXEC CICS RECEIVE` command receives the `idata` buffer contents for processing.
6. The `TOUPDTPS` program processes data.
7. The `EXEC CICS SEND` command returns the `OUT-BUFFER` contents into the clients `tpgetreply odata` buffer. `LAST` indicates the conversation is finished. `WAIT` suspends processing until the data has successfully been received.

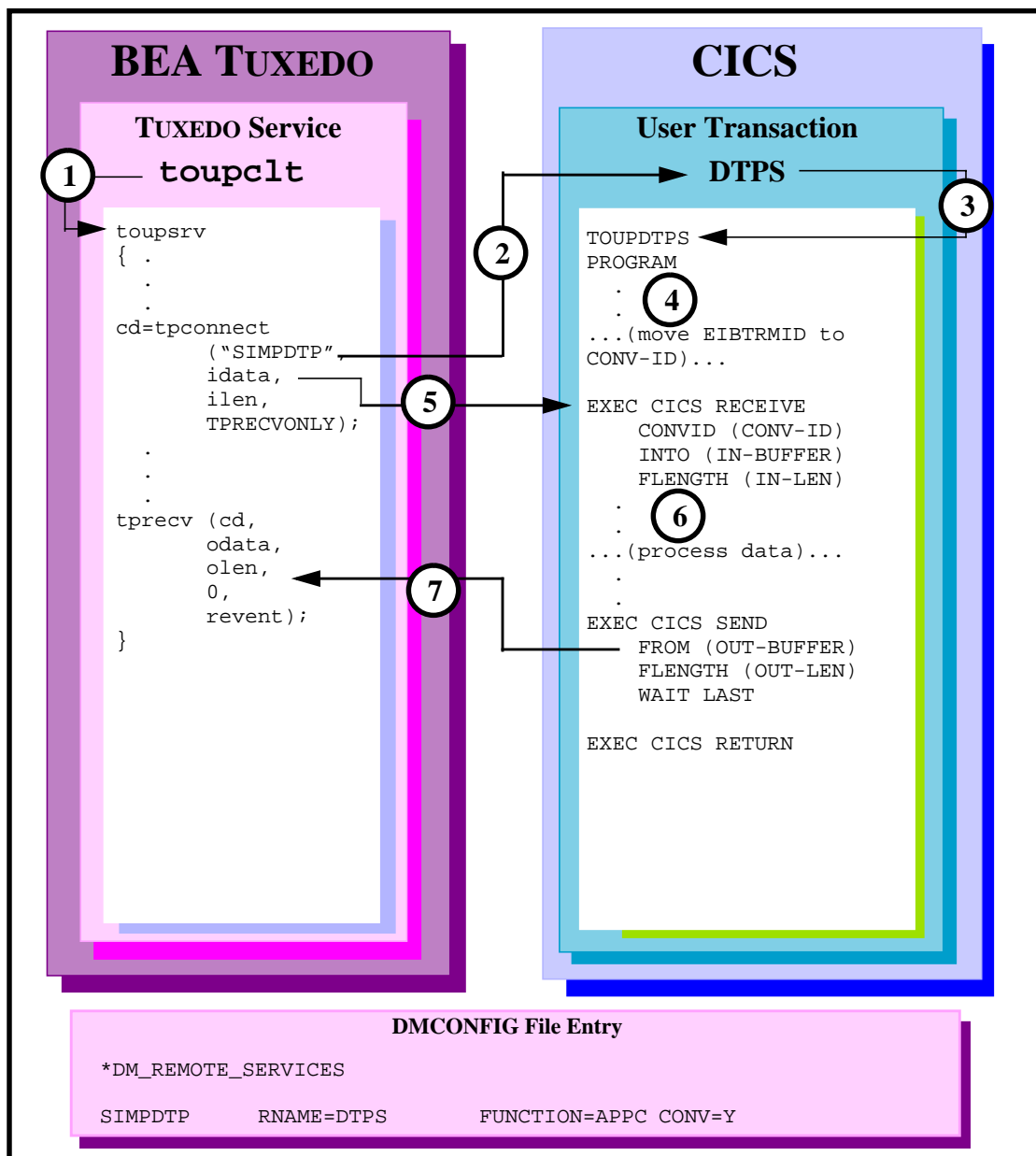
Figure 7-13 Tuxedo Client Asynchronous Request/Response with No Reply to CICS/ESA DTP



Step-by-Step Description: Tuxedo Client Asynchronous Request/Response with No Reply to CICS/ESA DTP

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpacall` with a `TPNOREPLY` request for `SIMPDTP`, which is advertised in the `*DM_REMOTE_SERVICES` section of `DMCONFIG` file.
3. User transaction `DTPS` starts `TOUPDTPS` program.
4. It is recommended you save the `EIBTRMID` to a program variable. This value may be used to identify the specific conversation on your CICS/ESA APPC verbs.
5. The `EXEC CICS RECEIVE` command receives the `idata` buffer contents for processing.
6. The `TOUPDTPS` program processes data.

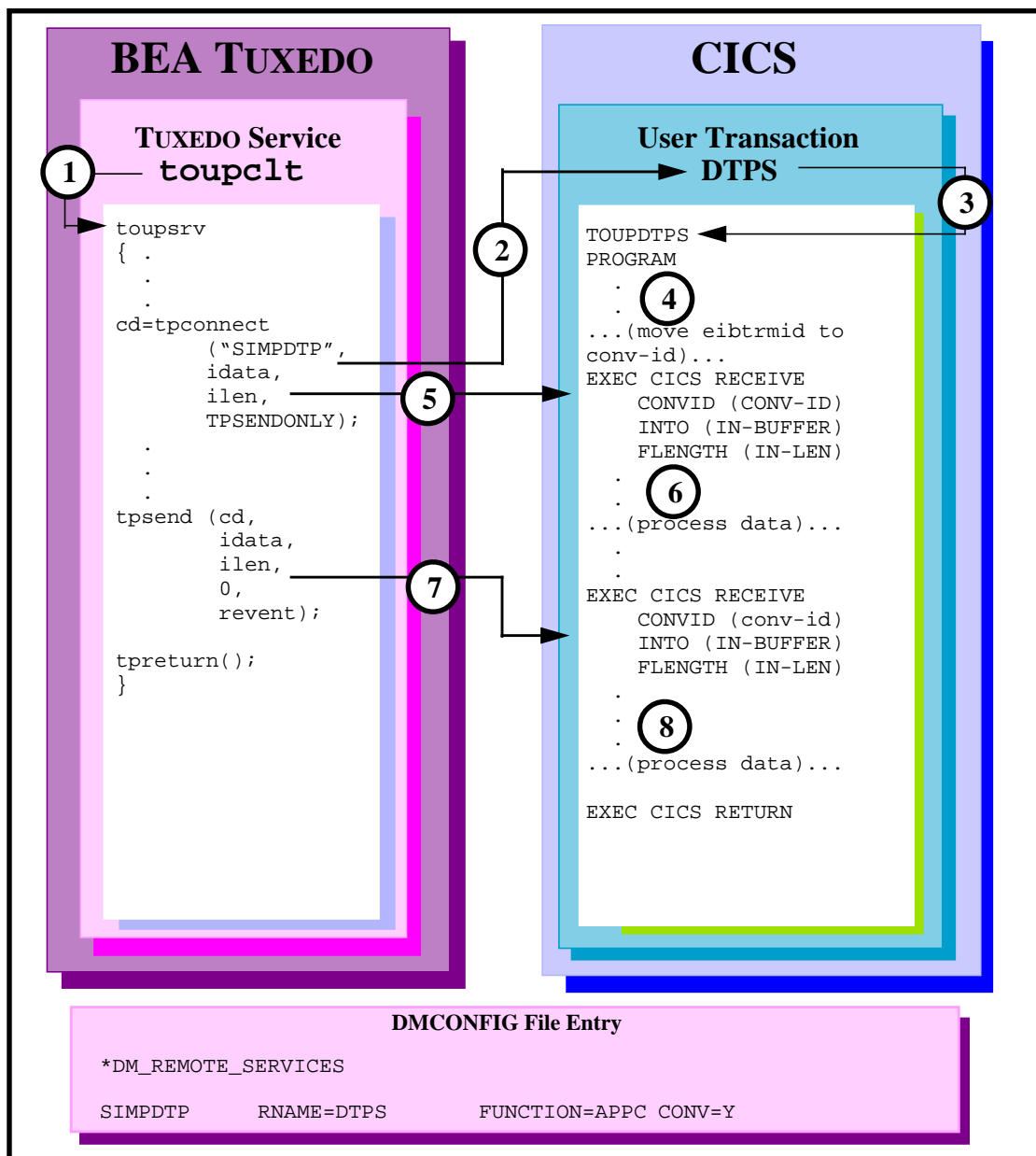
Figure 7-14 Tuxedo Conversational Client to CICS/ESA DTPS, Server Gets Control



Step-by-Step Description: Tuxedo Conversational Client to CICS/ESA DTP, Server Gets Control

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpconnect` for `SIMPDTP`, which is advertised in the `*DM_REMOTE_SERVICES` section of `DMCONFIG` file. The `TPRECVONLY` flag indicates the server gets control and the first conversation verb `toupsrv` can issue is `tprecv`. Data is sent on the `tpconnect` in the `idata` buffer.
3. User transaction `DTPS` starts `TOUPDTPS` program.
4. It is recommended you save the `EIBTRMID` to a program variable. This value may be used to identify the specific conversation on your CICS/ESA APPC verbs.
5. The `EXEC CICS RECEIVE` command receives the `idata` buffer contents for processing.
6. The `TOUPDTPS` program processes data.
7. The `EXEC CICS SEND` command returns the `OUT-BUFFER` contents into the clients `tprecv odata` buffer. `WAIT` suspends processing in `TOUPDTPS` until the data has successfully been received. `LAST` indicates the conversation is finished and is communicated to the `tprecv` as `TPEV_SVCSUCC`.

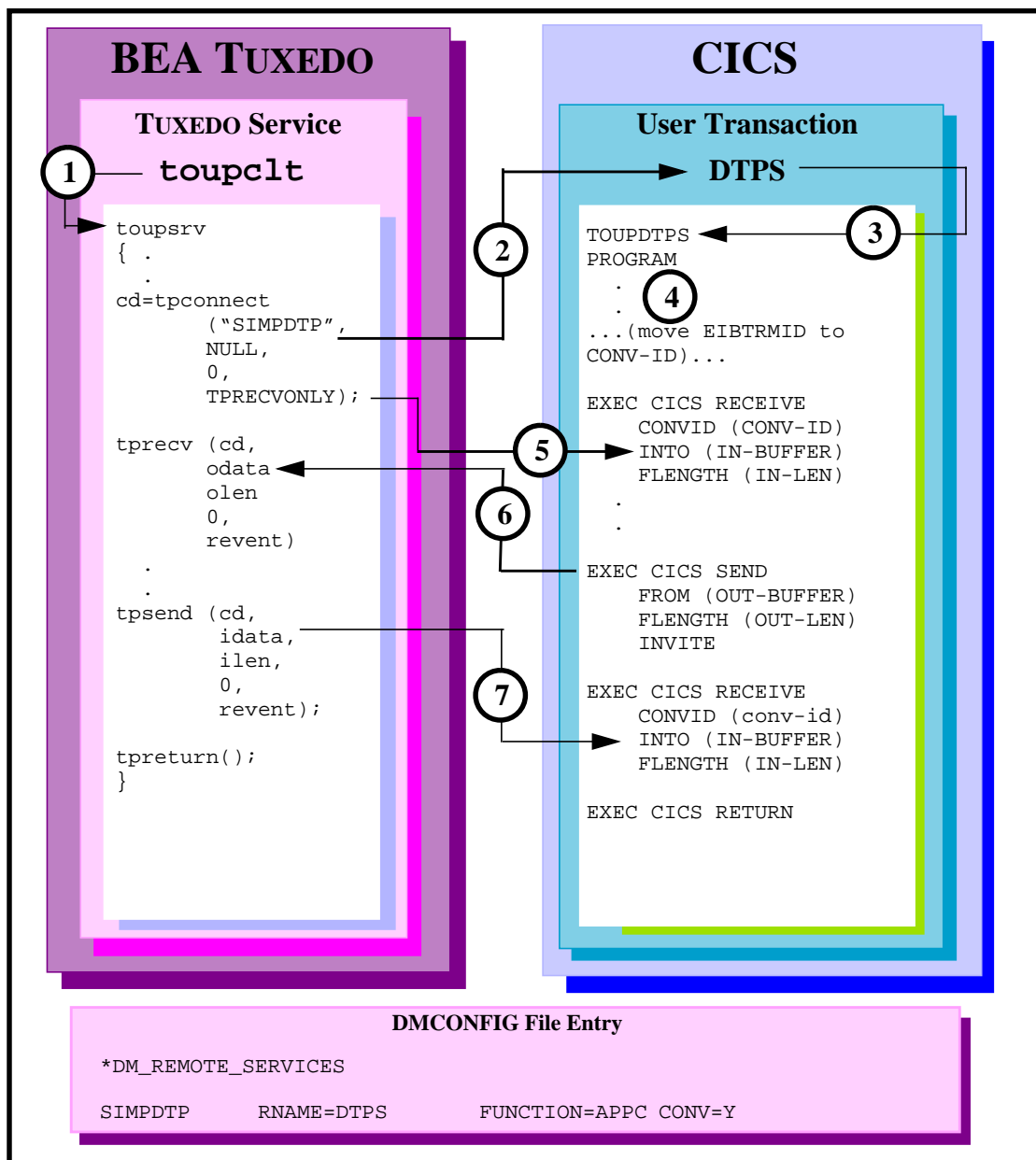
Figure 7-15 Tuxedo Conversational Client to CICS/ESA DTP, Client Retains Control



Step-by-Step Description: Tuxedo Conversational Client to CICS/ESA DTP, Client Retains Control

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpconnect` for `SIMPDTP`, which is advertised in the `*DM_REMOTE_SERVICES` section of `DMCONFIG` file. The `TPSENDONLY` indicates the client retains control and continues to send data. Data is sent on the `tpconnect` in the `idata` buffer.
3. User transaction `DTPS` starts `TOUPDTPS` program.
4. It is recommended you save the `EIBTRMID` to a program variable. This value may be used to identify the specific conversation on your CICS/ESA APPC verbs.
5. The `EXEC CICS RECEIVE` command receives the `tpconnect idata` buffer contents for processing.
6. The `TOUPDTPS` program processes data.
7. The `EXEC CICS RECEIVE` command receives the `tpsend idata` contents into the server's `IN-BUFFER`, along with notification that the conversation is over.
8. The server processes the data.

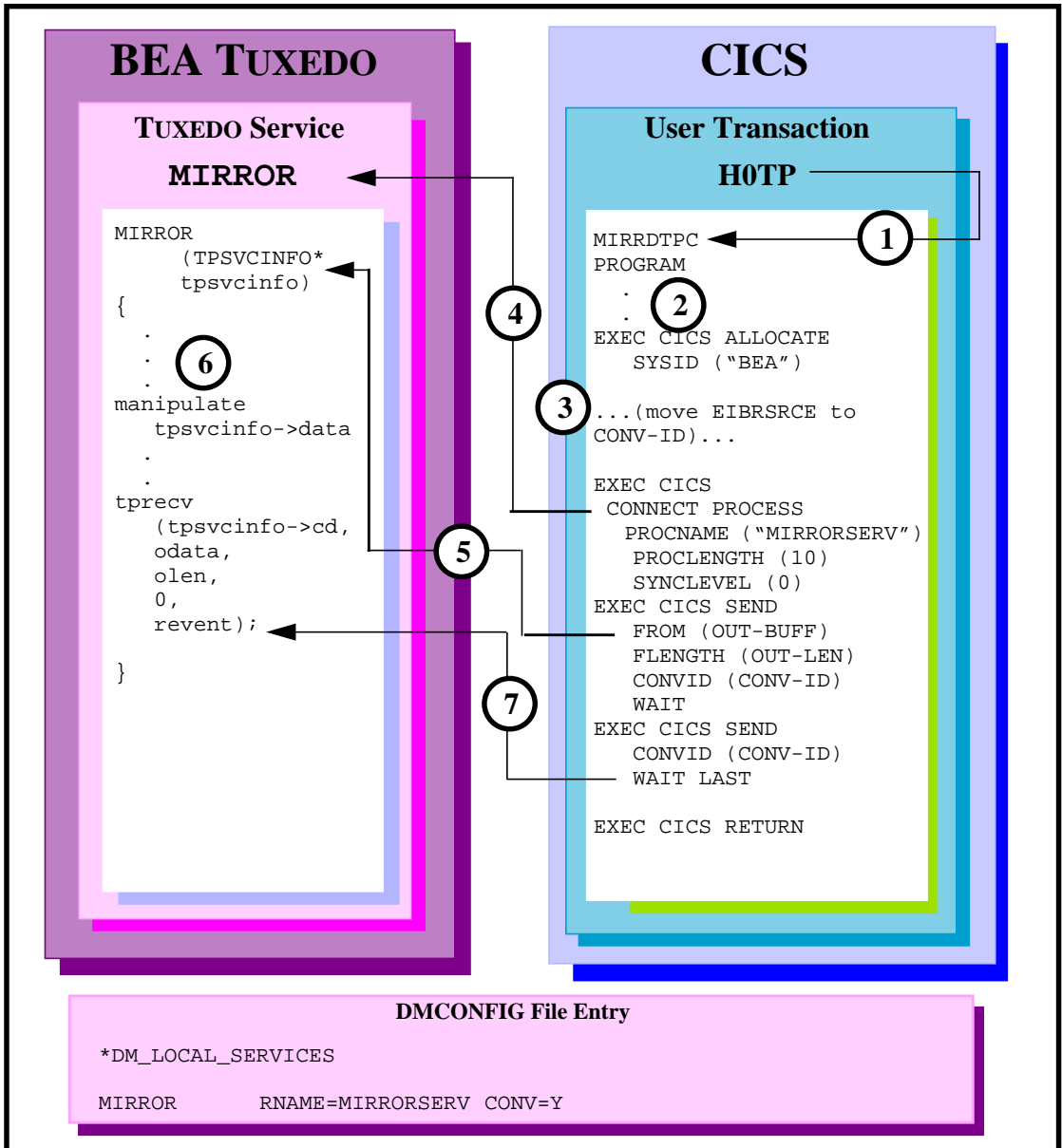
Figure 7-16 Tuxedo Conversational Client to CICS/ESA DTP, Client Grants/Gets Control



Step-by-Step Description: Tuxedo Conversational Client to CICS/ESA DTP, Client Grants/Gets Control

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpconnect` for `SIMPDTP`, which is advertised in the `*DM_REMOTE_SERVICES` section of `DMCONFIG` file. The `TPRECVONLY` indicates the server gets control and the first conversation verb `toupsrv` can issue is `tprecv`.
3. User transaction `DTPS` starts `TOUPDTPS` program.
4. It is recommended you save the `EIBTRMID` to a program variable. This value may be used to identify the specific conversation on your CICS/ESA APPC verbs.
5. The `EXEC CICS RECEIVE` command receives a `send state` indicator from the `tpconnect` `TPRECVONLY` flag. No data is received into the `INBUFFER`.
6. The `EXEC CICS SEND` command returns the `OUT-BUFFER` contents into the clients `tprecv odata` buffer. The `EXEC CICS SEND` command relinquishes control to the client by using the `INVITE` option. This is communicated to the `tprecv` as `TPEV_SENDOONLY`.
7. The `EXEC CICS RECEIVE` command receives the `tpsend idata` contents into the server's `IN-BUFFER`.

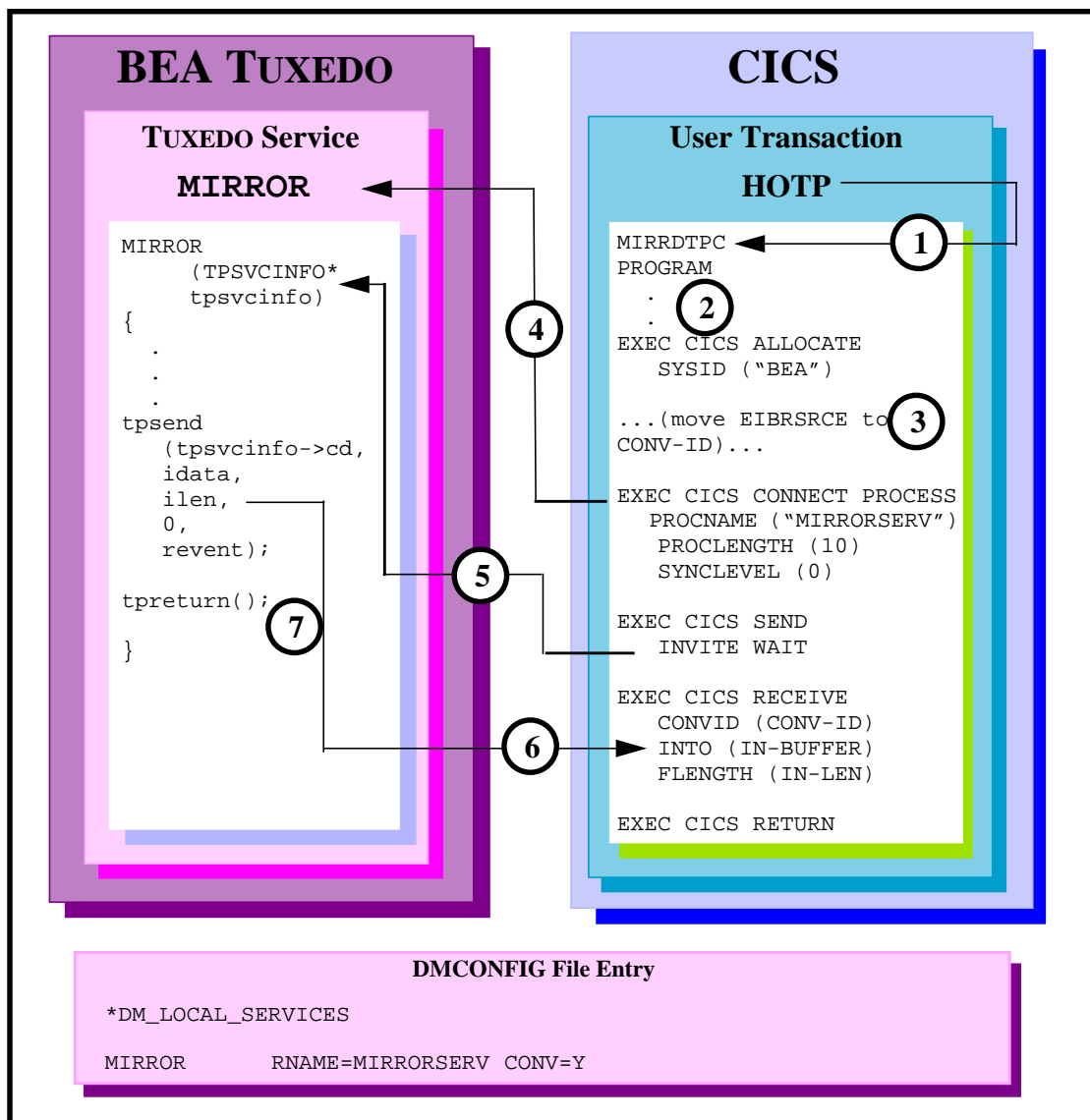
Figure 7-17 CICS/ESA DTP to Tuxedo Conversational Server, Client Retains Control



Step-by-Step Description: CICS/ESA DTP to Tuxedo Conversational Server, Client Retains Control

1. User-entered `H0TP` invokes `MIRRDTPC` program.
2. The `EXEC CICS ALLOCATE` acquires a session to the remote `TUXEDO` domain.
3. Save the conversation ID returned in `EIBRSRCE` to a program variable. This value is used to identify the specific conversation in your `CICS/ESA APPC` verbs.
4. The `EXEC CICS CONNECT PROCESS` command initiates the advertised service mapped to `MIRROR` in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file.
5. Execute the `EXEC CICS SEND WAIT` command to send immediately the contents of the `OUT-BUFFER` to the `TUXEDO` service in the `tpsvcinfol->data` buffer.
6. The `TUXEDO` service processes data.
7. The `revent` parameter on the `tprecv` receives a value of `TPEV_SVCSUCC` as a result of the `EXEC CICS SEND LAST` command issued by the host client. `WAIT` suspends processing until the data has successfully been received.

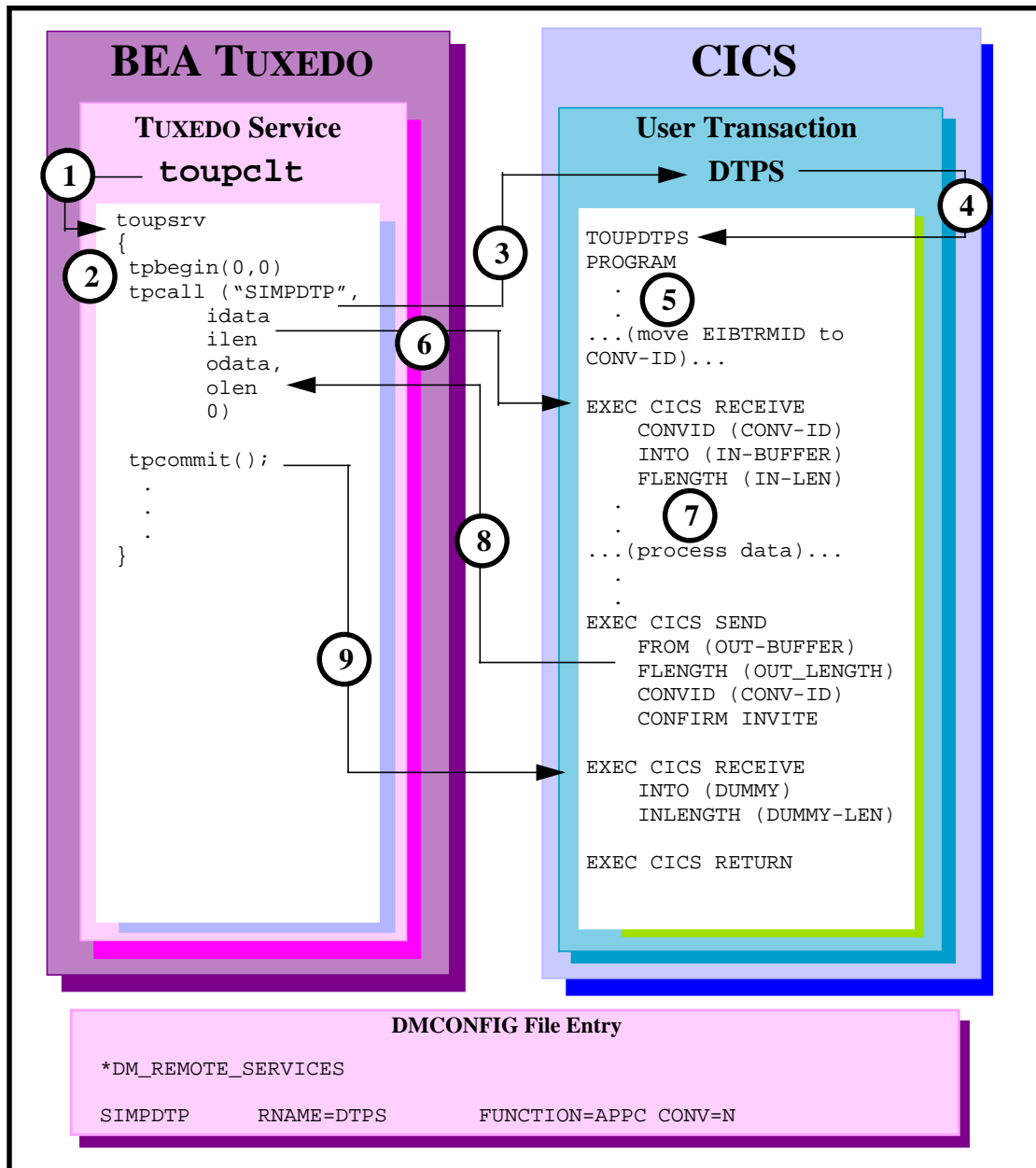
Figure 7-18 CICS/ESA DTP to Tuxedo Conversational Server, Client Relinquishes Control



Step-by-Step Description: CICS/ESA DTP to Tuxedo Conversational Server, Client Relinquishes Control

1. User-entered HOTP invokes MIRRDTPC program.
2. The EXEC CICS ALLOCATE acquires a session to the remote TUXEDO domain.
3. Save the conversation ID returned in EIBRSRCE to a program variable. This value is used to identify the specific conversation in your CICS/ESA APPC verbs.
4. The EXEC CICS CONNECT PROCESS command initiates the advertised service mapped to MIRROR in the DM_LOCAL_SERVICES section of the DMCONFIG file.
5. The EXEC CICS SEND command relinquishes control with the INVITE WAIT option.
6. The EXEC CICS RECEIVE command receives the tpsend idata buffer contents into the IN-BUFFER.
7. The tpreturn request tears down the conversation and indicates on the EXEC CICS RECEIVE that the conversation is over.

Figure 7-19 Transactional Tuxedo Client Request/Response to CICS/ESA DTP

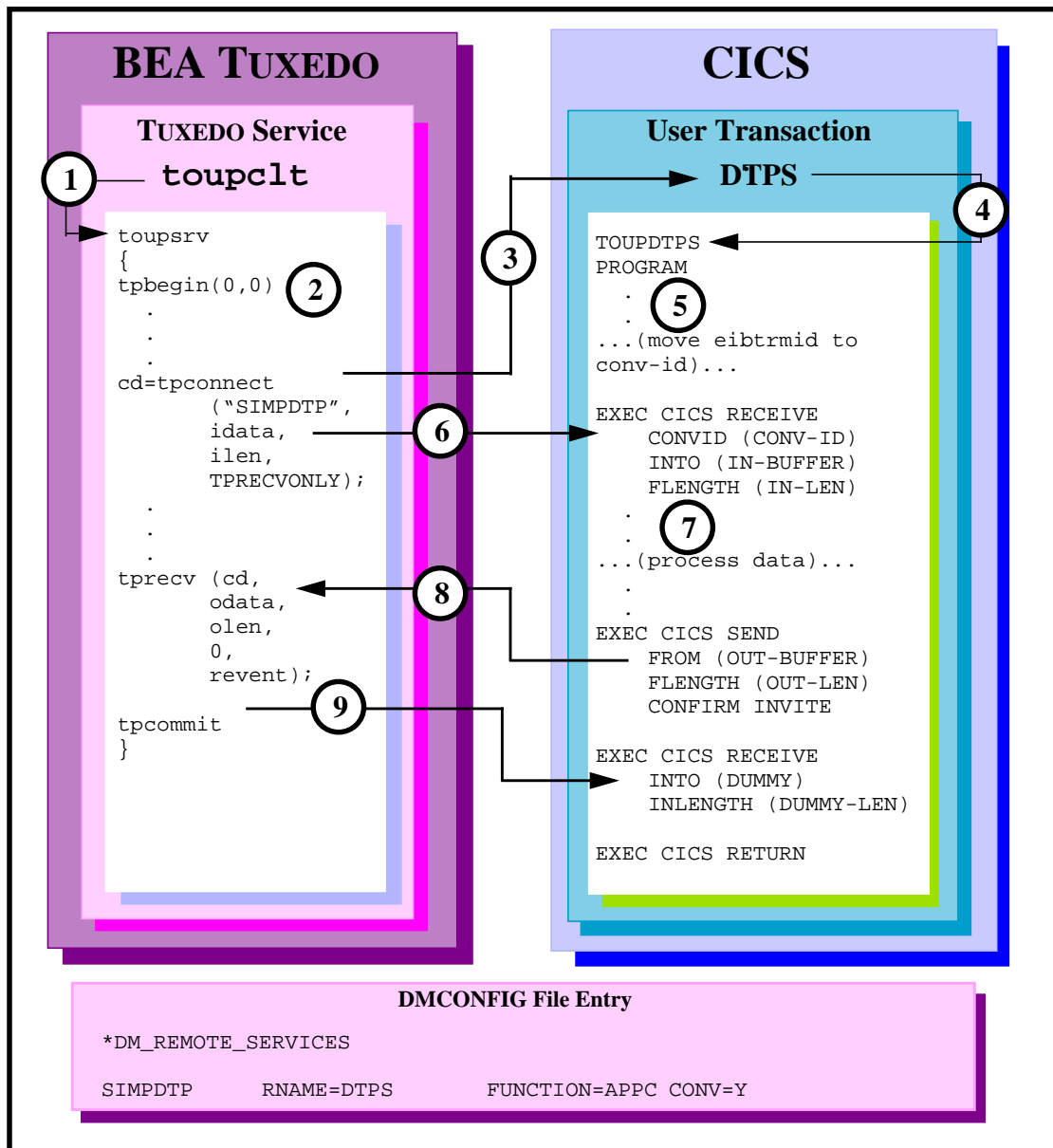


Step-by-Step Description: Transactional Tuxedo Client Request/Response to CICS/ESA DTP

Note: This is not the recommended method of performing a DTP transactional service. Please refer to the transactional DPL using request/response for the recommended method.

1. Tuxedo client `toupclt` invokes `toupsrv` service. (Note that each `tpcall` made in the program must be bookended with a `tpbegin` and a `tpcommit`.)
2. The service issues `tpbegin` to start a transaction.
3. The `toupsrv` service issues `tpcall` for `SIMPDTP`, which is advertised in the `*DM_REMOTE_SERVICES` section of the `DMCONFIG` file.
4. User transaction `DTPS` starts `TOUPDTPS` program.
5. Save the `EIBTRMID` to a program variable. This value is used to identify the specific conversation on your CICS/ESA APPC verbs.
6. The `EXEC CICS RECEIVE` command receives the `idata` buffer contents for processing.
7. The `TOUPDTPS` program processes data.
8. The `EXEC CICS SEND` command returns the `OUT-BUUFER` contents into the clients `odata` buffer. `CONFIRM` indicates the conversation is finished. `INVITE` allows the client to respond with a `COMMIT` request.
9. The `toupsrv` service issues `tpcommit` to end the transaction. The `COMMIT` is received on the `EXEC CICS RECEIVE` verb and the server issues `EXEC CICS RETURN` to commit the resources, terminate the transaction, and free the outstanding conversation.

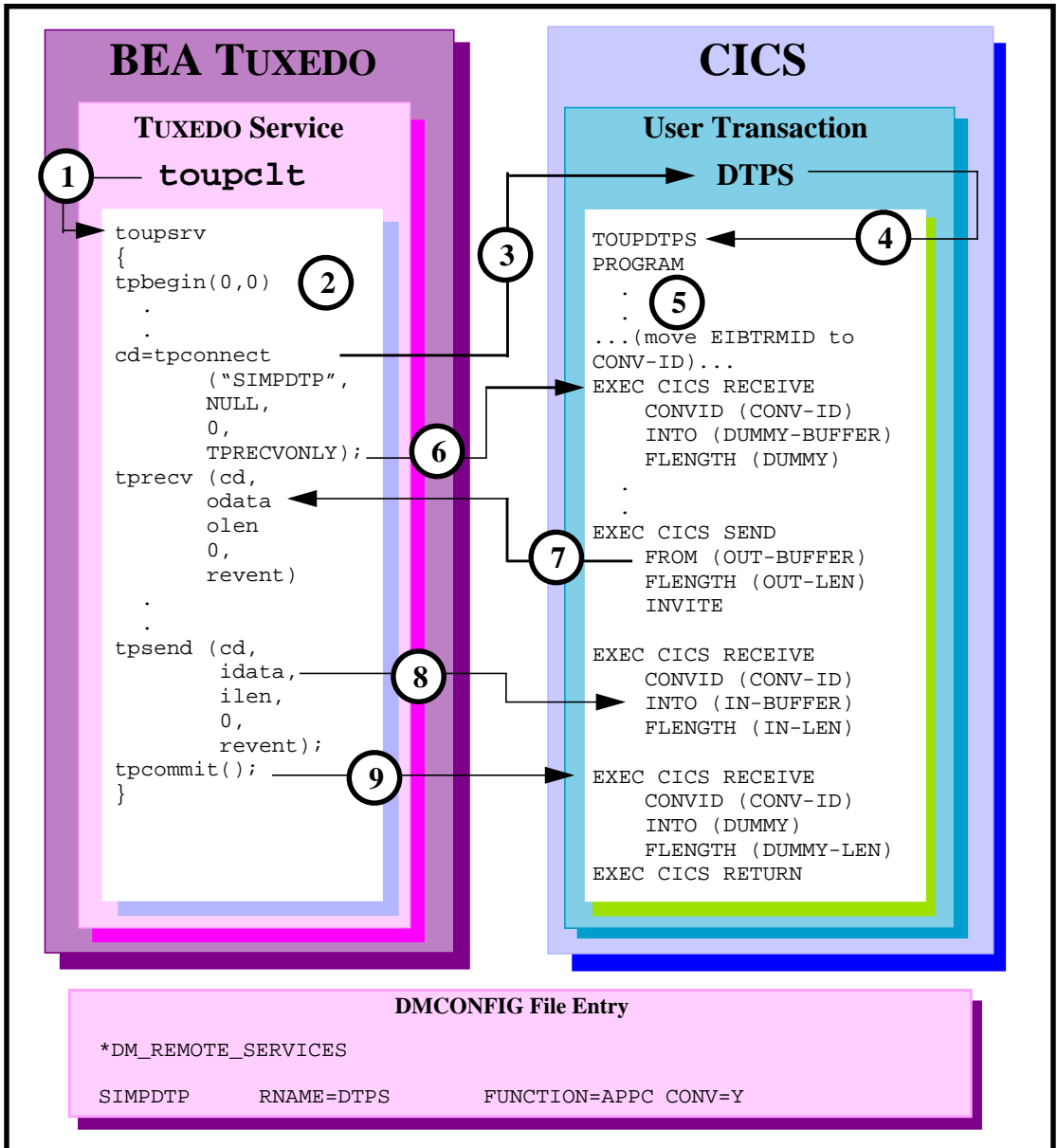
Figure 7-20 Transactional Tuxedo Conversational Client to CICS/ESA DTP, Server Gets Control



Step-by-Step Description: Transactional Tuxedo Conversational Client to CICS/ESA DTP, Server Gets Control

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpbegin` to start the transaction.
3. The `toupsrv` service issues `tpconnect` for `SIMPDTP`, which is advertised in the `*DM_REMOTE_SERVICES` section of `DMCONFIG` file. The `TPRECVONLY` indicates the server gains control and the first conversation verb `toupsrv` can issue is `tprecv`. Data is sent on the `tpconnect` in the `idata` buffer.
4. User transaction `DTPS` starts `TOUPDTPS` program.
5. It is recommended you save the `EIBTRMID` to a program variable. This value may be used to identify the specific conversation on your CICS/ESA APPC verbs.
6. The `EXEC CICS RECEIVE` command receives the `idata` buffer contents for processing.
7. The `TOUPDTPS` program processes data.
8. The `EXEC CICS SEND` command returns the `OUT-BUFFER` contents into the clients `tprecv odata` buffer. `CONFIRM` indicates that the conversation is finished and is communicated to the `tprecv` as `TPEV_SVCSUCC`. `INVITE` enables the client to respond with a `COMMIT` request.
9. The `toupsrv` service issues `tpcommit` to end the transaction. The `COMMIT` is received on the `EXEC CICS RECEIVE` verb and the server issues `EXEC CICS RETURN` to commit the resources, terminate the transaction, and free the outstanding conversation.

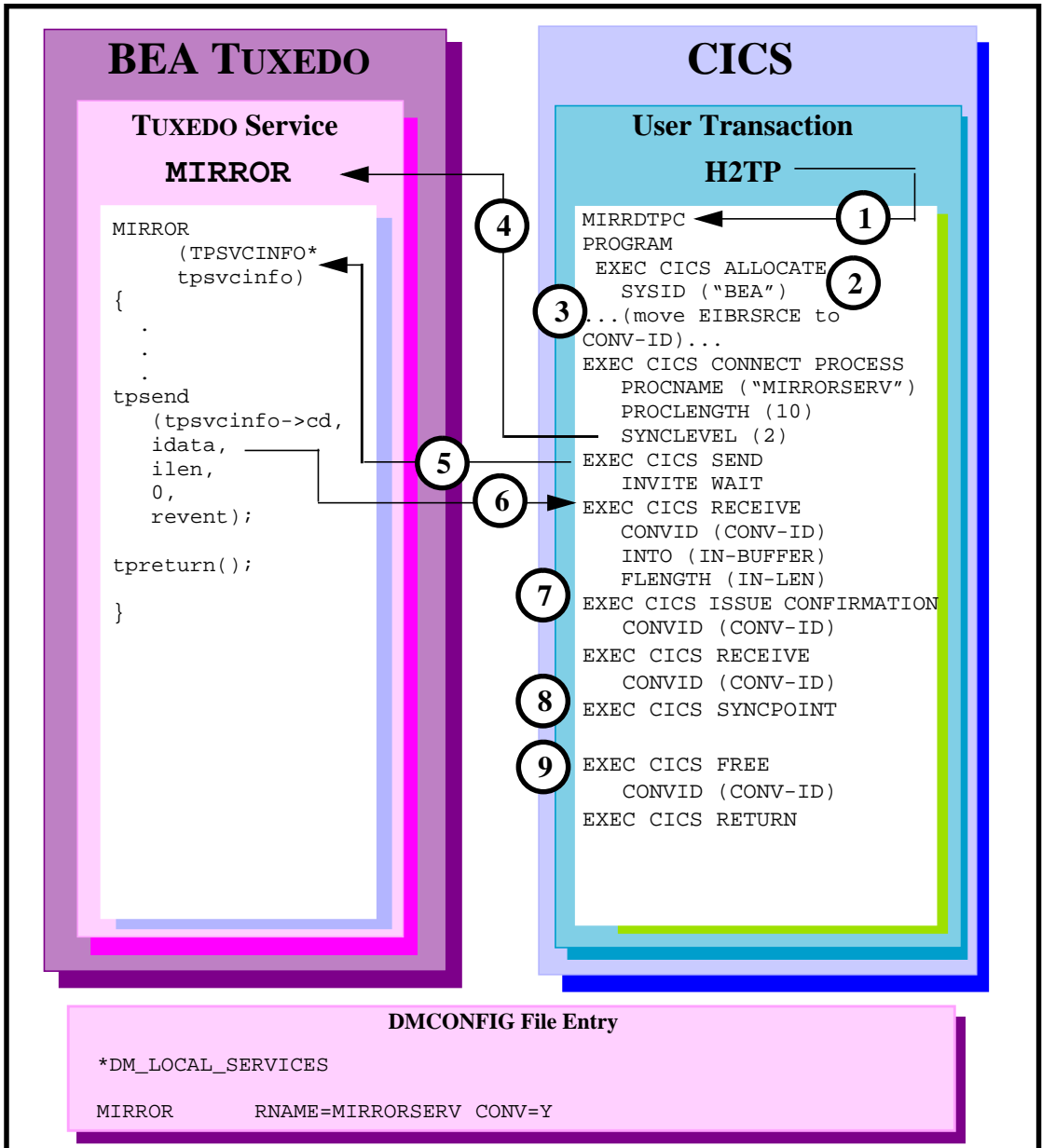
Figure 7-21 Transactional Tuxedo Conversational Client to CICS/ESA DTP, Client Grants/Gets Control



Step-by-Step Description: Transactional Tuxedo Conversational Client to CICS/ESA DTP, Client Grants/Gets Control

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpbegin` to start the transaction.
3. The `toupsrv` service issues `tpconnect` for `SIMPDTP`, which is advertised in the `*DM_REMOTE_SERVICES` section of `DMCONFIG` file. The `TPRECVONLY` indicates the server gains control and the first conversation verb `toupsrv` can issue is `tprecv`.
4. User transaction `DTPS` starts `TOUPDTPS` program.
5. It is recommended you save the `EIBTRMID` to a program variable. This value may be used to identify the specific conversation on your CICS/ESA APPC verbs.
6. The `EXEC CICS RECEIVE` command receives a `send state` indicator from the `tpconnect` `TPRECVONLY` flag. No data is received into the `INBUFFER`.
7. The `EXEC CICS SEND` command returns the `OUT-BUFFER` contents into the clients `tprecv odata` buffer. The `EXEC CICS SEND` command relinquishes control to the client by using the `INVITE` option. The data might not be sent immediately.
8. The `EXEC CICS RECEIVE` command flushes the request and receives the `tpsend idata` contents into the server's `IN-BUFFER`.
9. The `toupsrv` service issues `tpcommit` to end the transaction. The `COMMIT` is received on the `EXEC CICS RECEIVE` verb and the server issues `EXEC CICS RETURN` to commit the resources, terminate the transaction, and free the outstanding conversation.

Figure 7-22 Transactional CICS/ESA DTP to Tuxedo Conversational Server, Host Client Relinquishes Control



Step-by-Step Description: CICS/ESA DTP to Transactional Tuxedo Conversational Server, Host Client Relinquishes Control

1. User-entered H2TP invokes MIRRDTPC program.
2. The EXEC CICS ALLOCATE acquires a session to the remote TUXEDO domain.
3. Save the conversation ID returned in EIBRSRCE to a program variable. This value is used to identify the specific conversation on your CICS/ESA APPC verbs.
4. The EXEC CICS CONNECT PROCESS command initiates the advertised service mapped to MIRRDTPS. The SYNCLEVEL(2) parameter indicates the inclusion of the Tuxedo service in the CICS/ESA transaction.
5. The EXEC CICS SEND INVITE WAIT command causes the client to immediately relinquish control to the TUXEDO server. This is communicated to the service in TPSVCINFO as TPSENDONLY. No data is sent to the server on this request.
6. The EXEC CICS RECEIVE command receives the tpsend idata buffer contents into the IN-BUFFER. The EXEC CICS RECEIVE command receives a confirm request indicating the conversation should be terminated.
7. The EXEC CICS ISSUE CONFIRMATION verb responds positively to the confirm request.
8. The EXEC CICS SYNCPOINT is an explicit commit request to end the conversation and update all resources in the transaction.
9. The EXEC CICS FREE verb explicitly frees the outstanding conversation.

(This page intentionally blank)

CPI-C Programming

The Common Programming Interface for Communications (CPI-C) is an implementation choice for program-to-program communications. The reason that CPI-C may be used over other implementations is that CPI-C defines a single programming interface to the underlying network protocols across many different programming languages and environments.

Also, you can use the CPI Resource Recovery interface to provide a consistent interface to system services that allow your programs to coordinate data exchange and updates of databases and resources.

The examples in this section show the protocol exchanges between the local BEA Tuxedo and remote host application program. With BEA Tuxedo, the type of ATMI service request determines the nature of the client/server communication model. For requests initiated by the host application, the configuration information for the local service determines the protocol exchanges on the conversation.

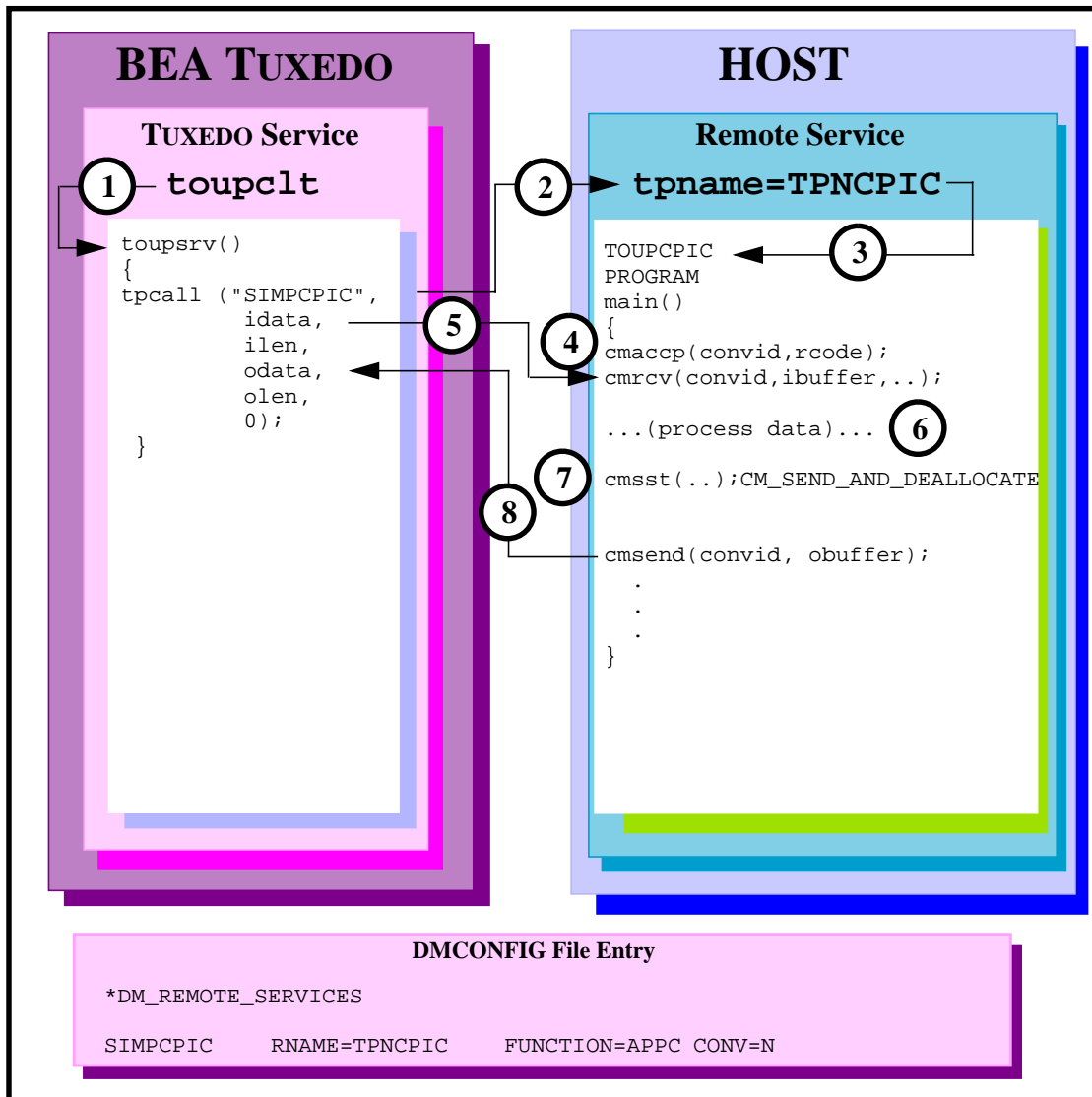
The recommended method of performing APPC services is to use TUXEDO conversational services instead of request/response services. The conversational service enables a client and a server to communicate as needed over the CONNECT SNA session conversation.

Although it is most suited for the DPL environment, the `tpcall` is used in the examples for a request/response to an APPC server.

It is recommended you use a DPL model when invoking multiple request/response calls in a single transaction. With DPL, a long-running mirror transaction to the CICS/ESA DPL server is started, enabling the server to process each of the requests over the same conversation. With APPC, each of the ATMI requests will establish a separate conversation; the conversation remains outstanding until the transaction is committed.

Note: To run *transactional* client/server scenarios or the CPI Resource Recovery interface, the CONNECT SNA software must be licensed for sync-level 2 operations.

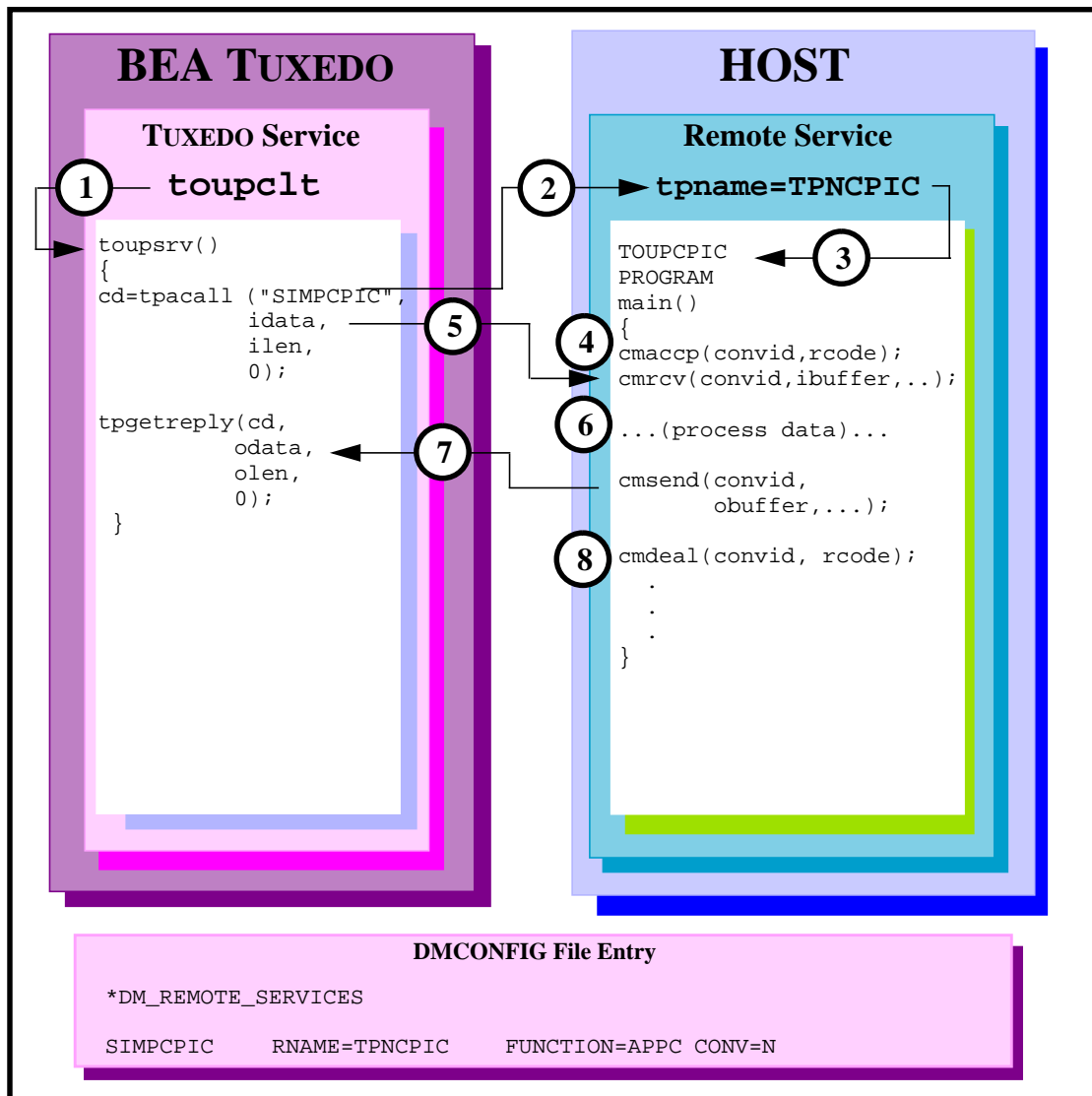
Figure 7-23 Tuxedo Client Request/Response to Host CPI-C



Step-by-Step Description: Tuxedo Client Request/Response to Host CPI-C

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpcall` for `SIMPCPIC`, which is advertised in the `*DM_REMOTE_SERVICES` section of the `DMCONFIG` file.
3. The remote service with the `tpname` `TPNCPIC` invokes `TOUPCPIC` program.
4. The server accepts the conversation with the `cmaccp` call. The conversation id returned on the request in `convid` is used for all other requests on this conversation.
5. The `cmrcv` request receives the `idata` buffer contents for processing
6. The `TOUPCPIC` program processes data.
7. The `cmsst` request prepares the next send request by setting the send type to `CM_SEND_AND_DEALLOCATE`.
8. The `cmsend` request returns the `obuffer` contents into the client `odata` buffer. The buffer is flushed, and the conversation ended.

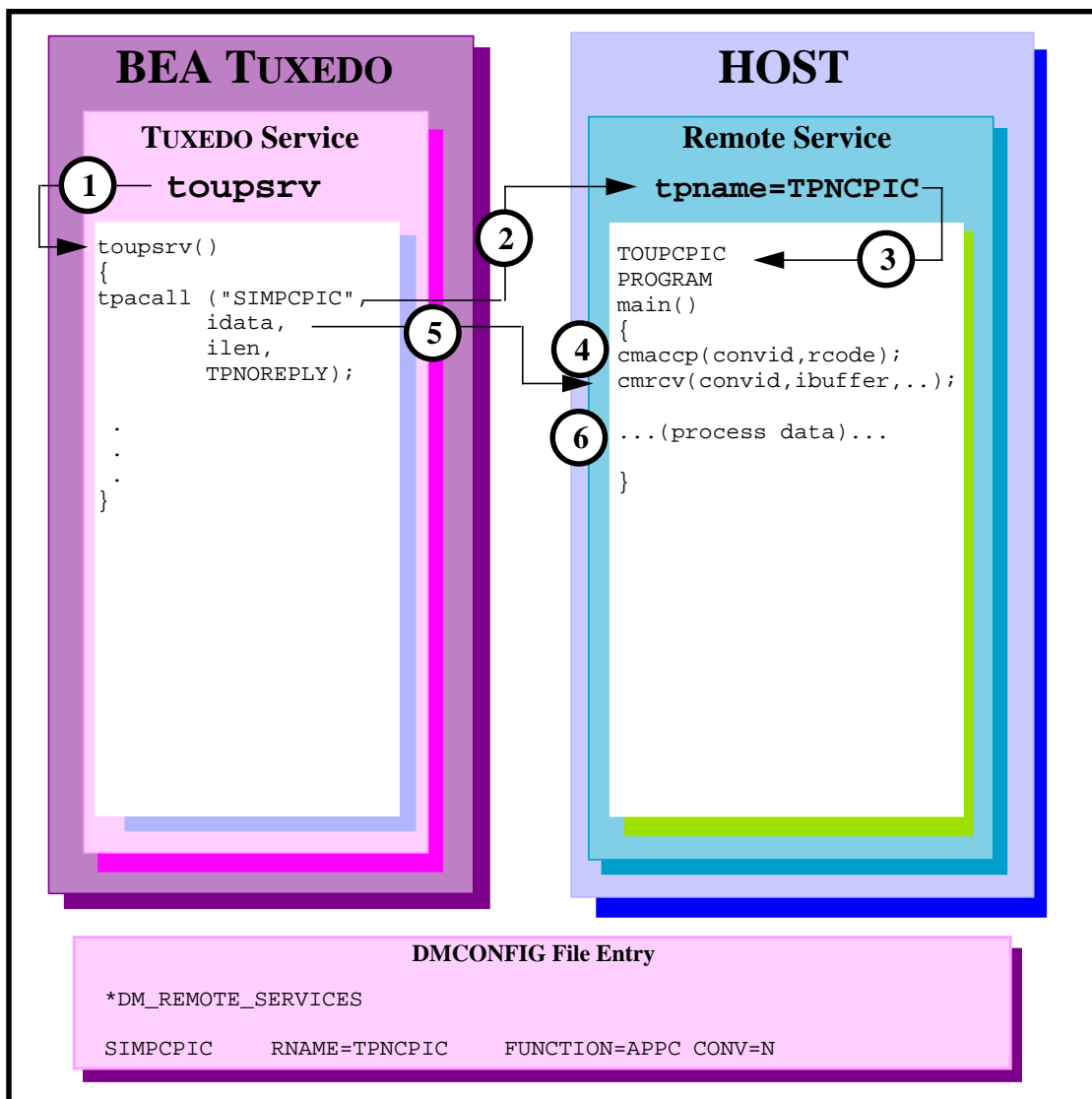
Figure 7-24 Tuxedo Client Asynchronous Request/Response to Host CPI-C



Step-by-Step Description: Tuxedo Client Asynchronous Request/Response to Host CPI-C

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpacall` for `SIMPCPIC`, which is advertised in the `*DM_REMOTE_SERVICES` section of the `DMCONFIG` file.
3. The remote service with `tpname` `TPNCPIC` invokes `TOUPCPIC` program.
4. The server accepts the conversation with the `cmaccp` call. The conversation id returned on the request in `convid` is used for all other requests on this conversation.
5. The `cmrcv` request receives the `idata` buffer contents for processing.
6. The `TOUPCPIC` program processes data.
7. The `cmsend` command returns the `obuffer` contents into the client `tpgetreply odata` buffer. The data may not be immediately sent to the `tpgetreply odata` buffer on this request.
8. The `cmdeal` flushes the data to the client, and indicates the conversation is finished.

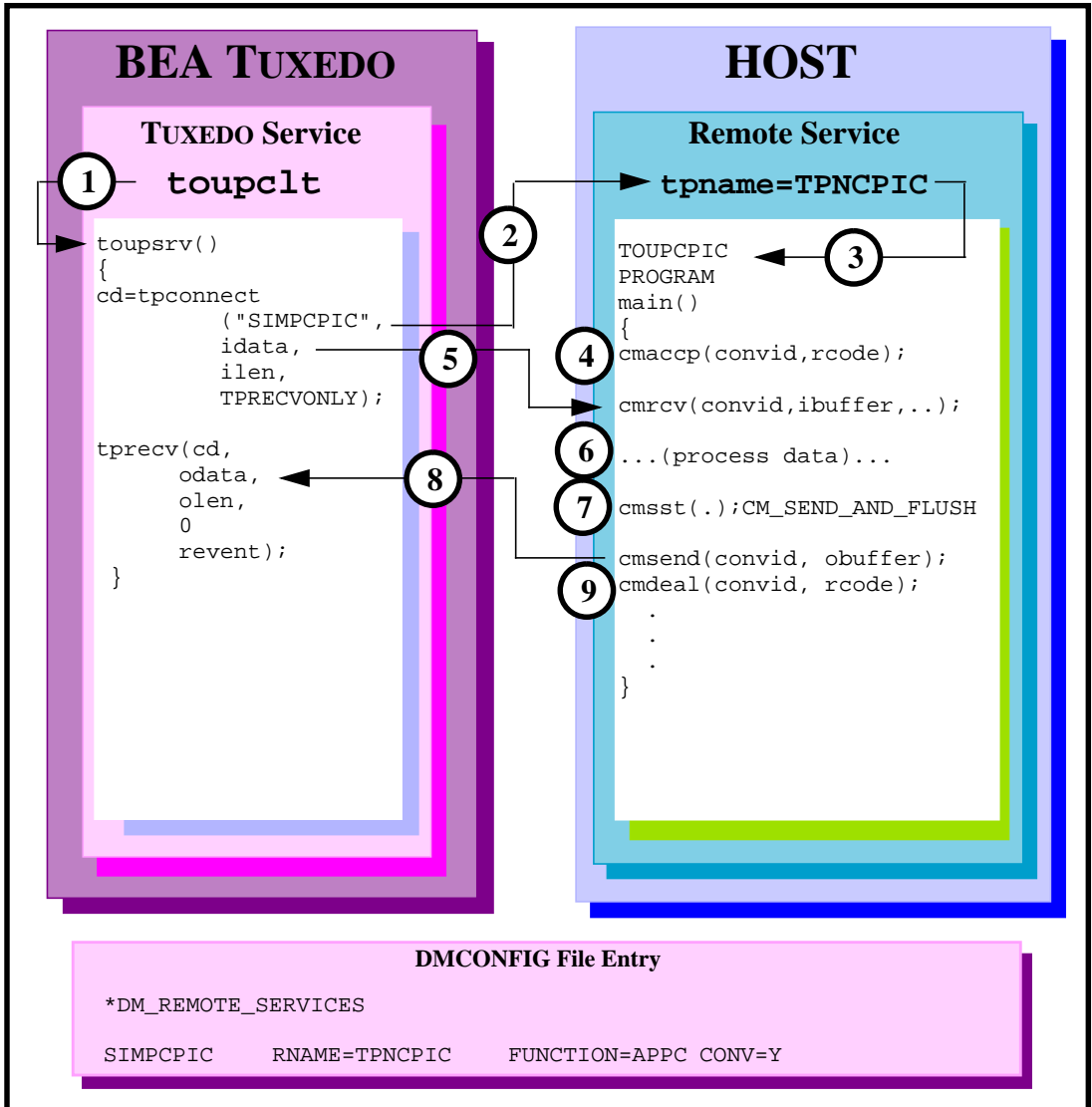
Figure 7-25 Tuxedo Client Asynchronous Request/Response to Host CPI-C with No Reply



Step-by-Step Description: Tuxedo Client Asynchronous Request/Response to Host with No Reply

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpacall` with a `TPNOREPLY` request for `SIMPCPIC`, which is advertised in the `*DM_REMOTE_SERVICES` section of the `DMCONFIG` file.
3. The remote service with `tpname` `TPNCPIC` invokes `TOUPCPIC` program.
4. The server accepts the conversation with the `cmaccp` call. The conversation id returned on the request in `convid` is used for all other requests on this conversation.
5. The `cmrcv` request receives the `idata` buffer contents for processing, and notification that the conversation has ended with the return code value of `CM_DEALLOCATED_NORMAL`.
6. The `TOUPCPIC` program processes data.

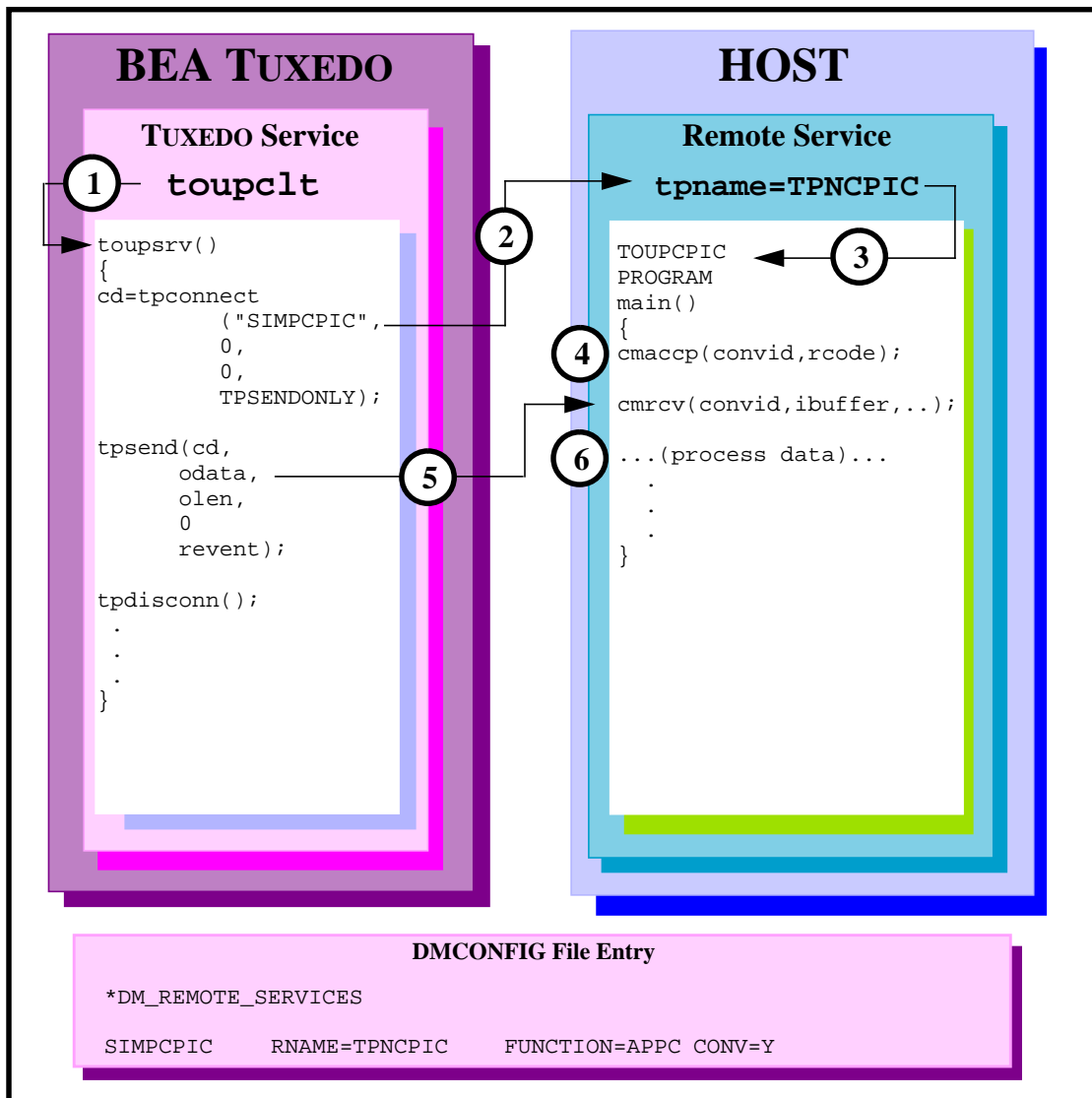
Figure 7-26 Tuxedo Conversational Client to Host CPI-C, Server Gets Control



Step-by-Step Description: Tuxedo Conversational Client to Host CPI-C, Server Gets Control

1. Tuxedo client invokes `toupsrv` service
2. The `toupsrv` service issues `tpconnect` for `SIMPCPIC`, which is advertised in the `*DM_REMOTE_SERVICES` section of the `DMCONFIG` file. The `TPRECVONLY` indicates the server gains control and the first conversation verb the `toupsrv` can issue is `tprecv`. Data is sent on the `tpconnect` in the `idata` buffer.
3. The remote service with `tpname` `TPNCPIC` invokes `TOUPCPIC` program.
4. The server accepts the conversation with the `cmaccp` call. The conversation ID returned on the request in `convid` is used for all other requests on this conversation.
5. The `cmrcv` request receives the `idata` buffer contents for processing.
6. The `TOUPCPIC` program processes data
7. The `cmsst` request prepares the next send request by setting the send type to `CM_SEND_AND_FLUSH`.
8. The `cmsend` command returns the `obuffer` contents into the client `tprecv` `odata` buffer. The data is immediately flushed on the send request.
9. The `cmdeal` request ends the conversation.

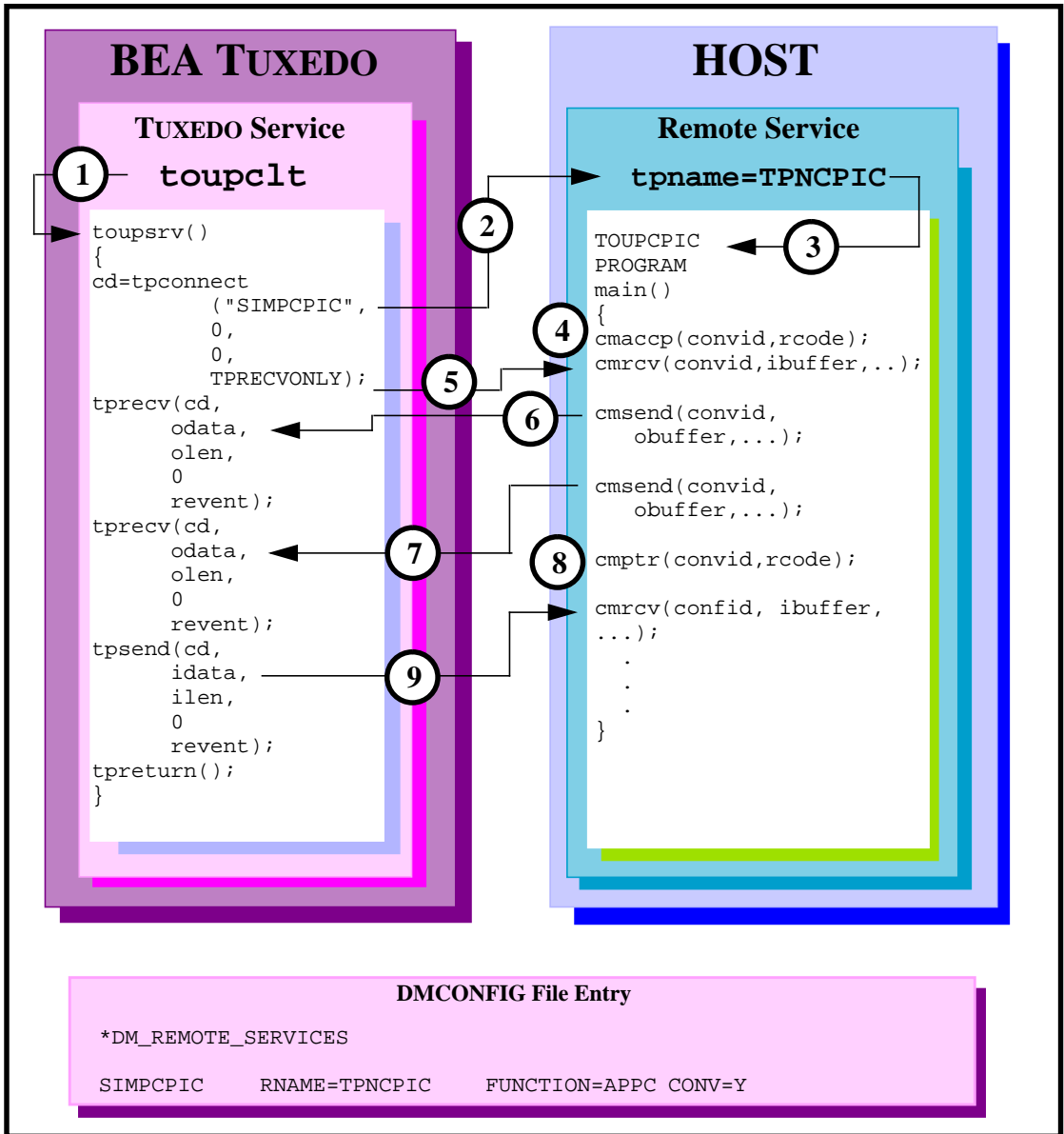
Figure 7-27 Tuxedo Conversational Client To Host CPI-C, Client Retains Control



Step-by-Step Description: Tuxedo Conversational Client to Host CPI-C, Client Retains Control

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpconnect` for `SIMPCPIC`, which is advertised in the `*DM_REMOTE_SERVICES` section of the `DMCONFIG` file. The `TPSENDONLY` indicates the client retains control and continues to send data. No data is sent with the `tpconnect`.
3. The remote service with `tpname` `TPNCPIC` invokes `TOUPCPIC` program.
4. The server accepts the conversation with the `cmaccp` call. The conversation id returned on the request in `convid` is used for all other requests on this conversation.
5. The `cmrcv` request receives the `tpsend idata` buffer contents for processing. The end of conversation is passed from the `tpreturn` service as `CM_DEALLOCATED_NORMAL`.
6. The `TOUPCPIC` program processes data.

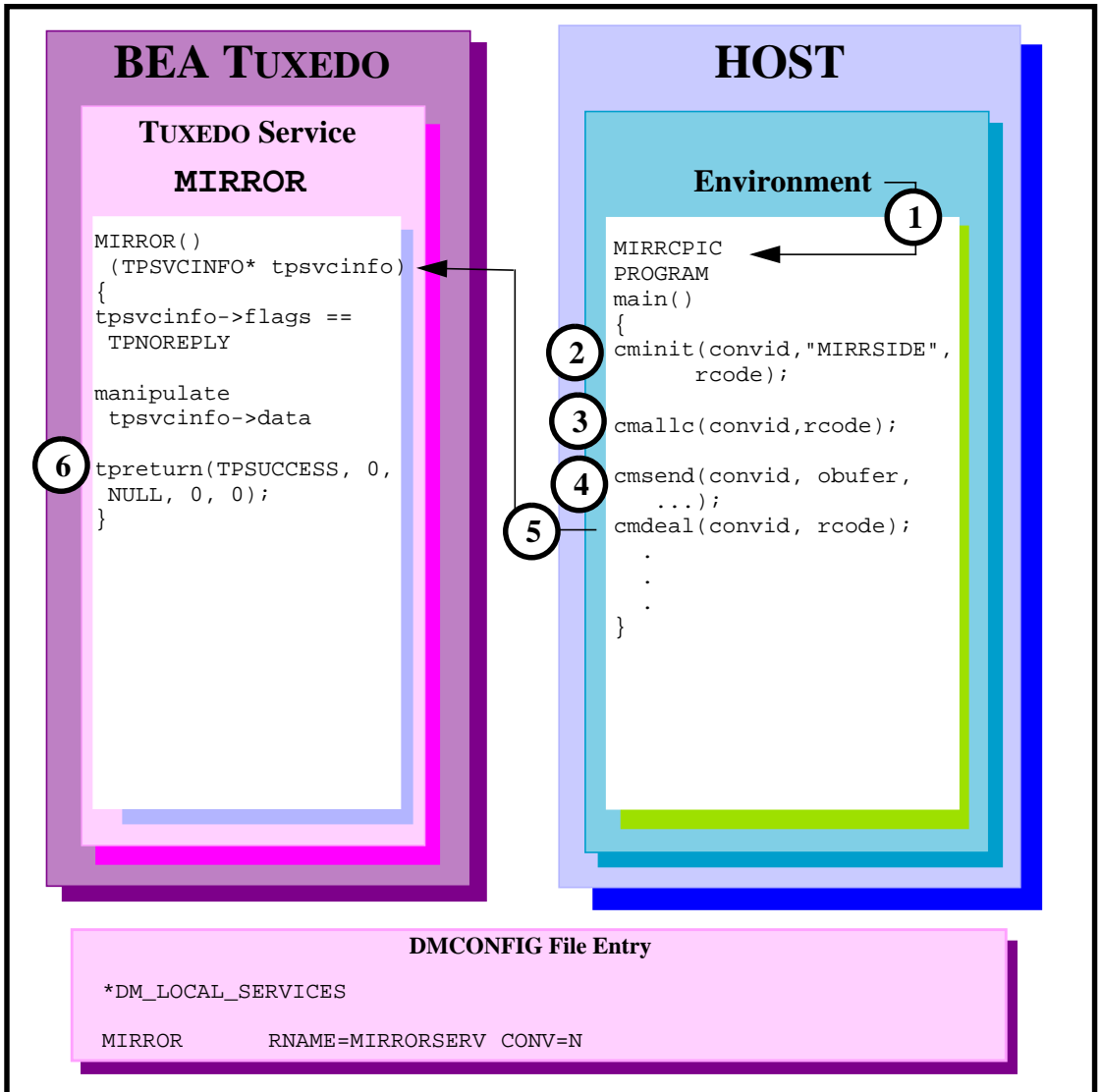
Figure 7-28 Tuxedo Conversational Client to Host CPI-C, Client Grants/gets Control



Step-by-Step Description: Tuxedo Conversational Client to Host CPI-C, Client Grants/Gets Control

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpconnect` for `SIMPCPIC`, which is advertised in the `*DM_REMOTE_SERVICES` section of the `DMCONFIG` file. The `TPRECVONLY` indicates the server gains control and the first conversation verb the `toupsrv` can issue is `tprecv`.
3. The remote service with `tpname` `TPNPCPIC` invokes `TOUPCPIC` program.
4. The server accepts the conversation with the `cmaccp` request. The conversation id returned on the request in `convid` is used for all other requests on this conversation.
5. The `cmrcv` request receives the indicator that control has been granted to the server.
6. The `cmsend` request returns its `obuffer` contents into the first client `tprecv` `odata` buffer. The data may not immediately be sent.
7. The `cmsend` request returns its `obuffer` contents into the second client `tprecv` `odata` buffer. The data may not immediately be sent.
8. The `cmptr` request flushes the data to the client, and grants control to the client.
9. The `cmrcv` request receives the `tpsend` `idata` buffer contents for processing. The end of conversation is passed from the `tpreturn` service as `CM_DEALLOCATED_NORMAL`.

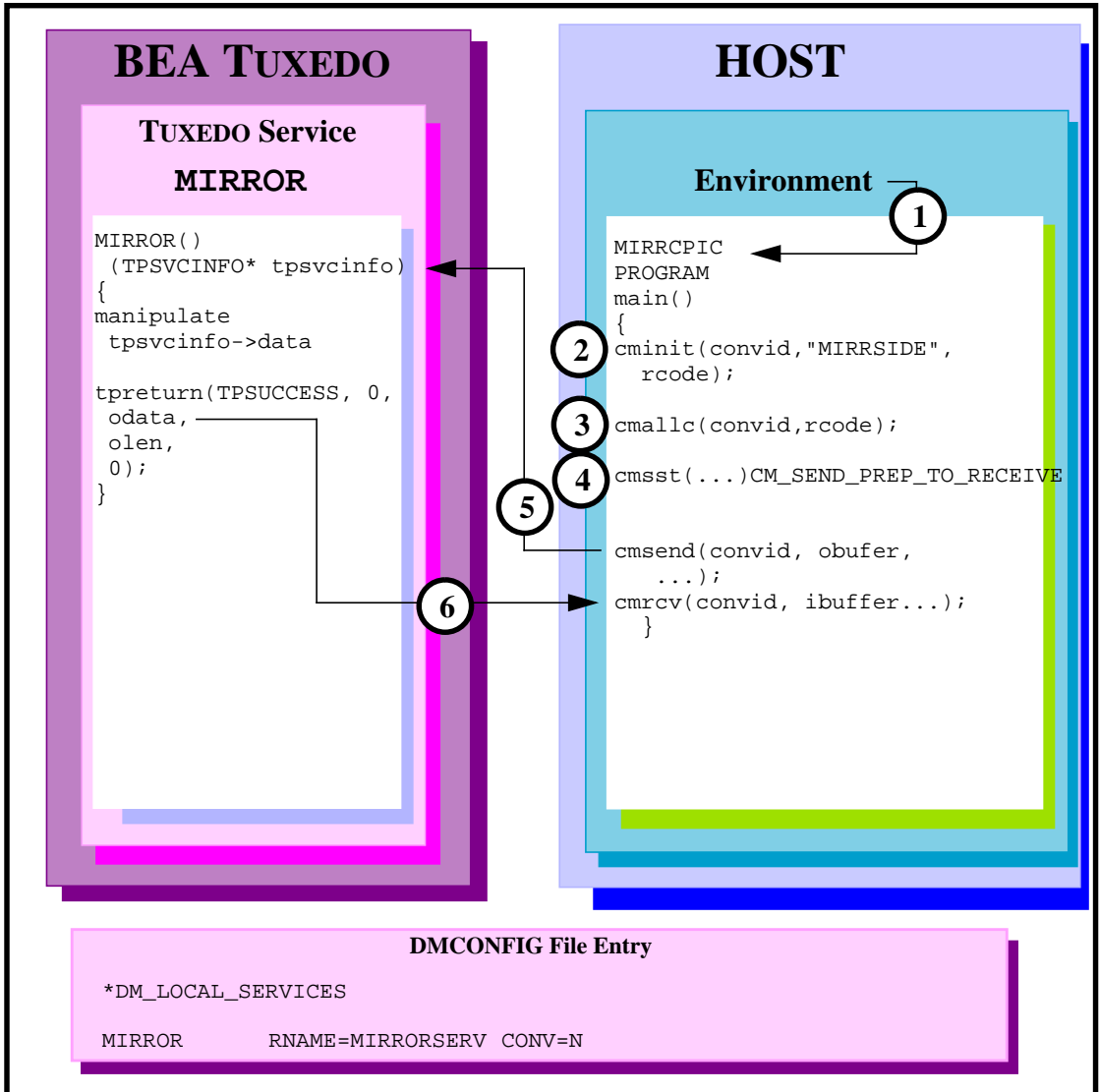
Figure 7-29 Host CPI-C to Tuxedo Asynchronous Request/Response Server with No Reply



Step-by-Step Description: Host CPI-C to Tuxedo Asynchronous Request/Response Server with No Reply

1. The CPI-C application program `MIRRCPI-C` is invoked using environment start-up specifications.
2. The `MIRRCPI-C` client requests `cminit` to establish conversation attributes and receive a conversation ID that will be used on all other requests on this conversation. The remote server and service are named in the CPI-C side information entry `MIRRSIDE`.
3. The `cmallc` request initiates the advertised service mapped to `MIRRORSERV` in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file.
4. The `cmsend` request sends the contents of `obuffer` to the Tuxedo service in the `tpsvcininfo->data` buffer.
5. The `cmdeal` request flushes the data, and indicates the conversation is finished with the `TPNOREPLY` in the `tpsvcininfo->flag` field.
6. The service completes with the `tpreturn`.

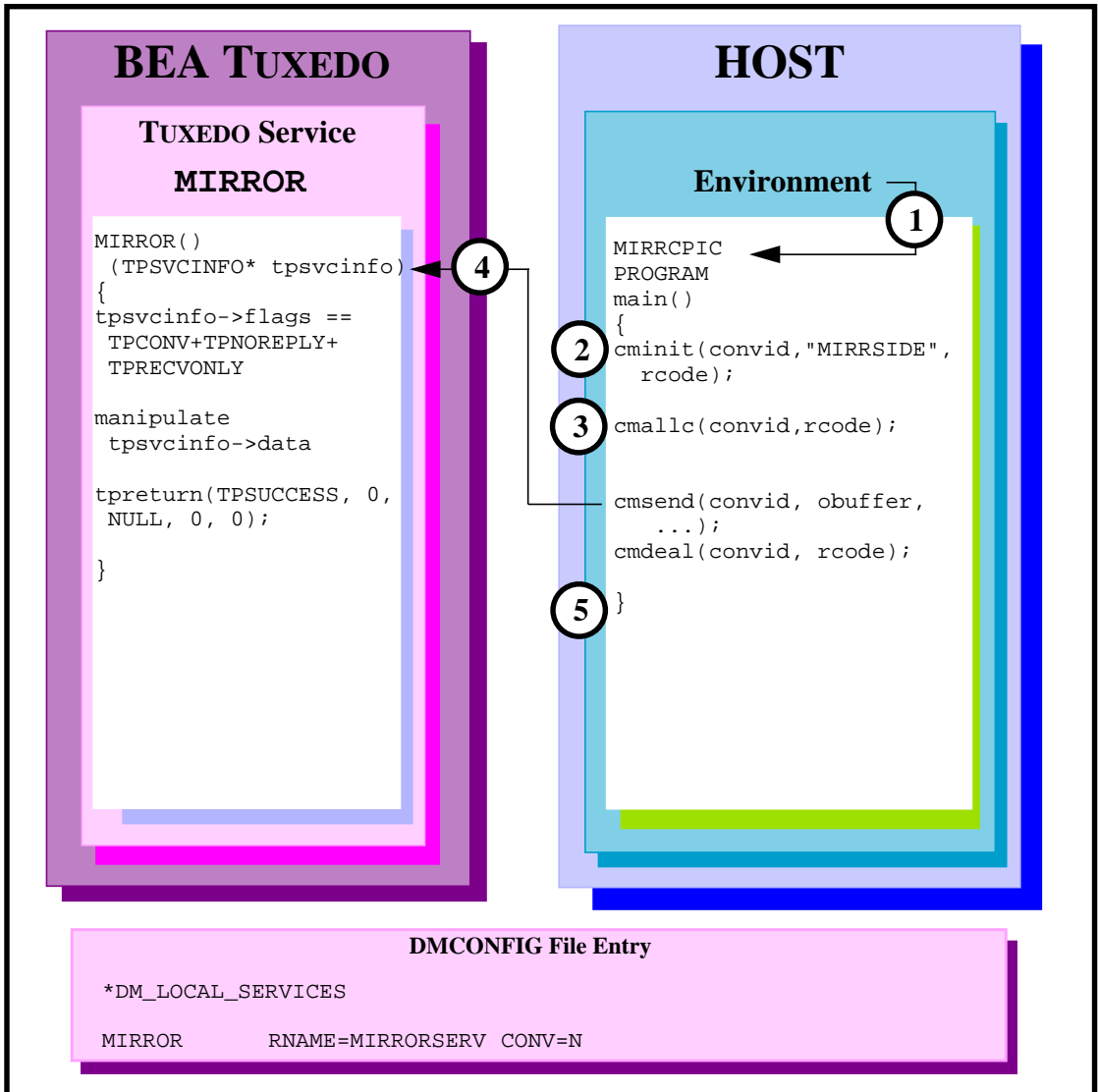
Figure 7-30 Host CPI-C to Tuxedo Server Request/Response



Step-by-Step Description: HOST CPI-C to Tuxedo Server Request/Response

1. The CPI-C application program MIRRCPIC is invoked using environment start-up specifications.
2. The MIRRCPIC client requests `cminit` to establish conversation attributes and receive a conversation id that will be used on all other requests on this conversation. The remote server and service are named in the CPI-C side information entry `MIRRSIDE`.
3. The `cmalloc` request initiates the advertised service mapped to `MIRRORSERV` in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file.
4. The `cmsst` request prepares the next send request by setting the send type to `CM_SEND_AND_PREP_TO_RECEIVE`.
5. The `cmsend` request immediately sends the contents of `obuffer` to the Tuxedo service in the `tpsvcinfo->data` buffer and relinquishes control to the `mirror serv` service.
6. The `cmrcv` request receives the contents of the `odata` returned on the Tuxedo `tpreturn` service, and notification that the conversation has ended with the return code value of `CM_DEALLOCATED_NORMAL`.

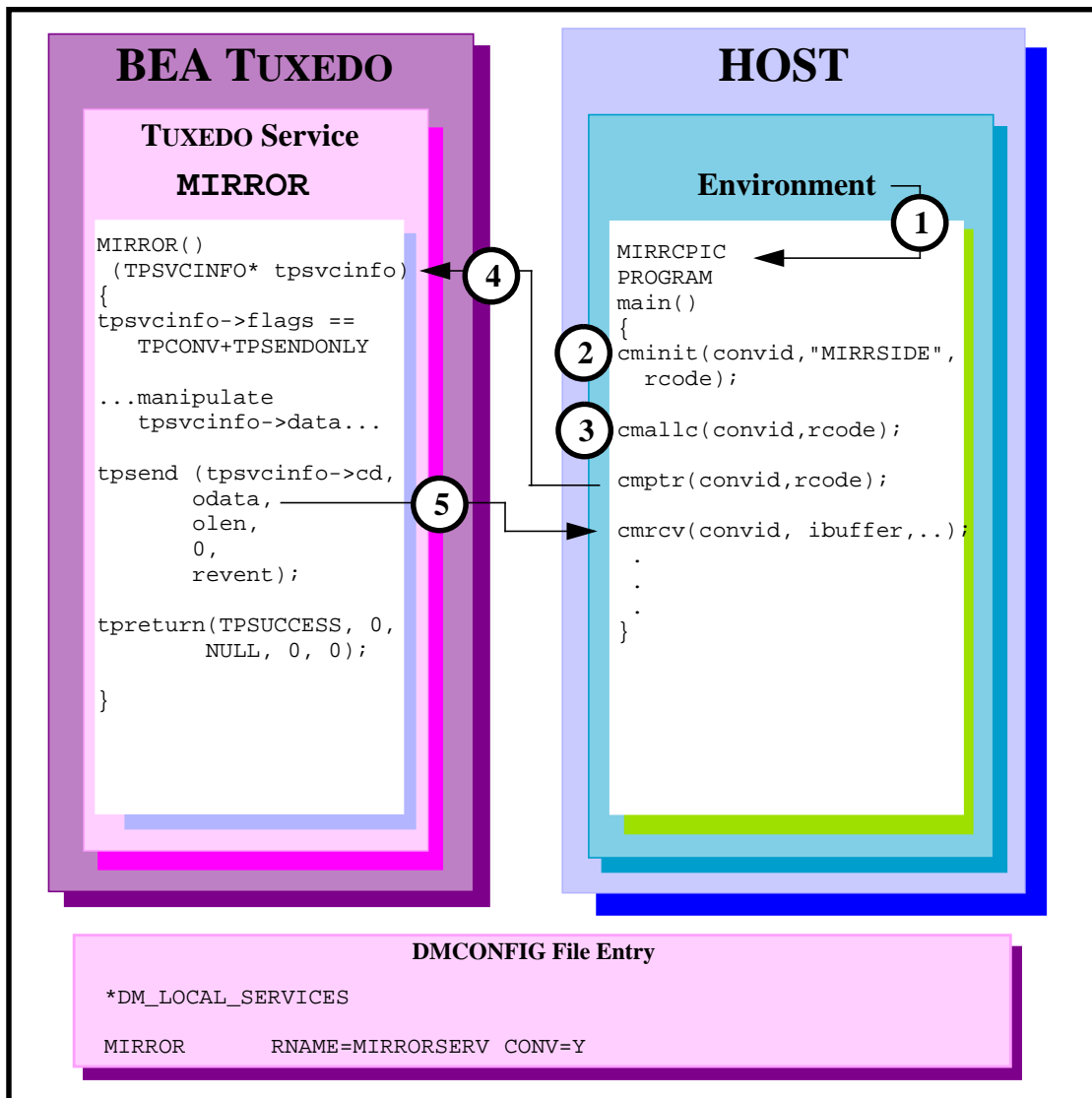
Figure 7-31 Host CPI-C to Tuxedo Conversational Service, Client Retains Control



Step-by-Step Description: Host CPI-C to Tuxedo Conversational Service, Client Retains Control

1. The CPI-C application program MIRRCPIC is invoked using environment start-up specifications.
2. The MIRRCPIC client requests `cminit` to establish conversation attributes and receive a conversation id that will be used on all other requests on this conversation. The remote server and service are named in the CPI-C side information entry `MIRRSIDE`.
3. The `cmallc` request initiates the advertised service mapped to `MIRRORSERV` in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file.
4. The `cmsend` request sends the contents of `obuffer` to the Tuxedo service in the `tpsvcininfo->data` buffer.
5. The `cmdeal` request flushes the data and ends the conversation, as indicated by `TPNOREPLY` in the `tpsvcininfo->flag` field.

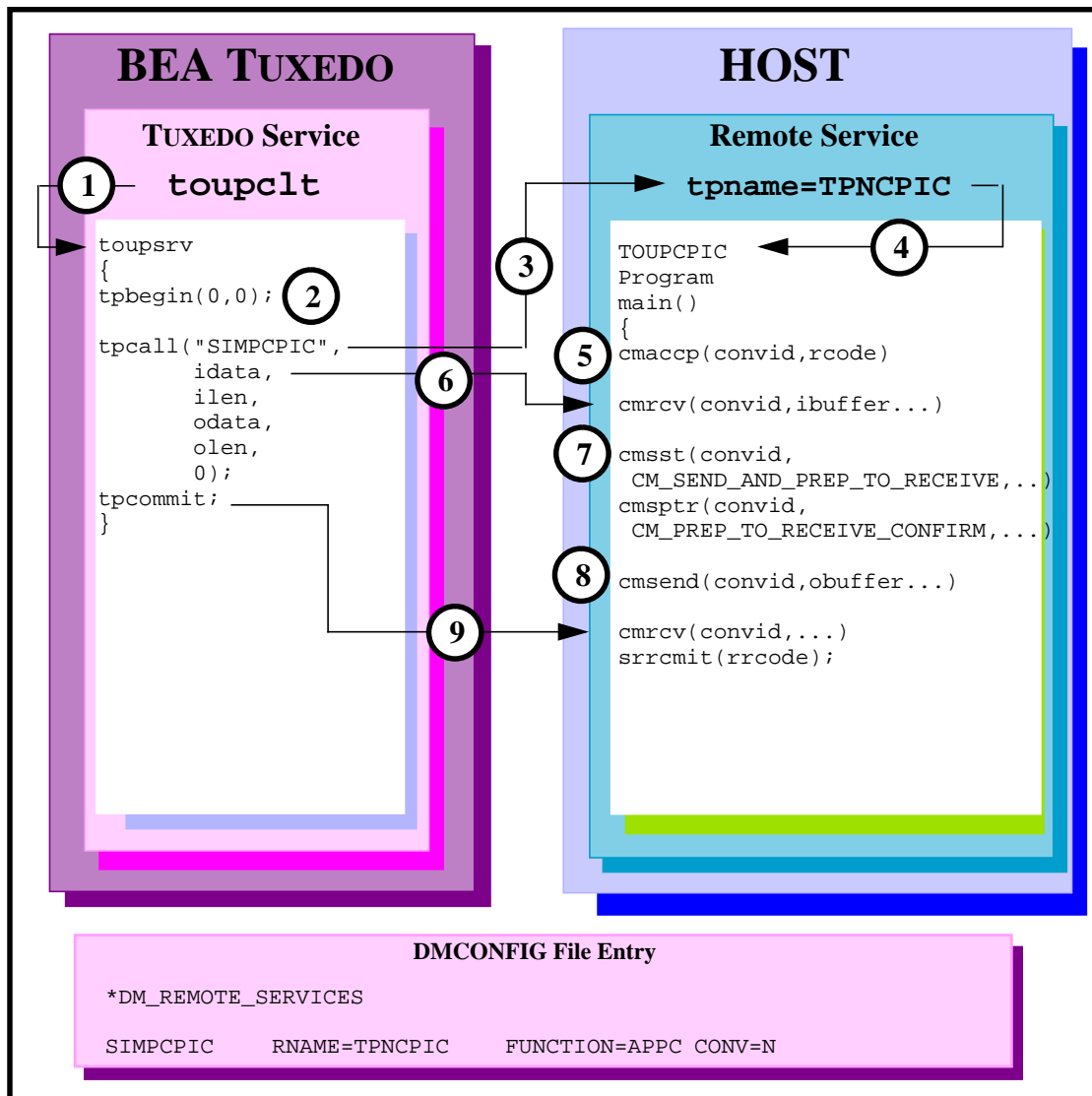
Figure 7-32 Host CPI-C Tuxedo to Conversational Service, Client Grants Control



Step-by-Step Description: Tuxedo Conversational Service, Client Grants Control

1. The CPI-C application program `MIRRPCIC` is invoked using environment start-up specifications.
2. The `MIRRPCIC` client requests `cminit` to establish conversation attributes and receive a conversation ID that will be used on all other requests on this conversation. The remote server and service are named in the CPI-C side information entry `MIRRSIDE`.
3. The `csmallc` request initiates the advertised service mapped to `MIRROR` in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file.
4. The `cmptr` relinquishes control of the conversation to the Tuxedo service indicated as `TPSENDONLY` in the `tpsvcininfo->flag` field. No data is passed in the `tpsvcininfo->data` field.
5. The `cmrcv` receives the contents of the `tpsend odata` into the `ibuffer`. The end of the conversation is passed from the `tpreturn` service as return code value `CM_DEALLOCATED_NORMAL`.

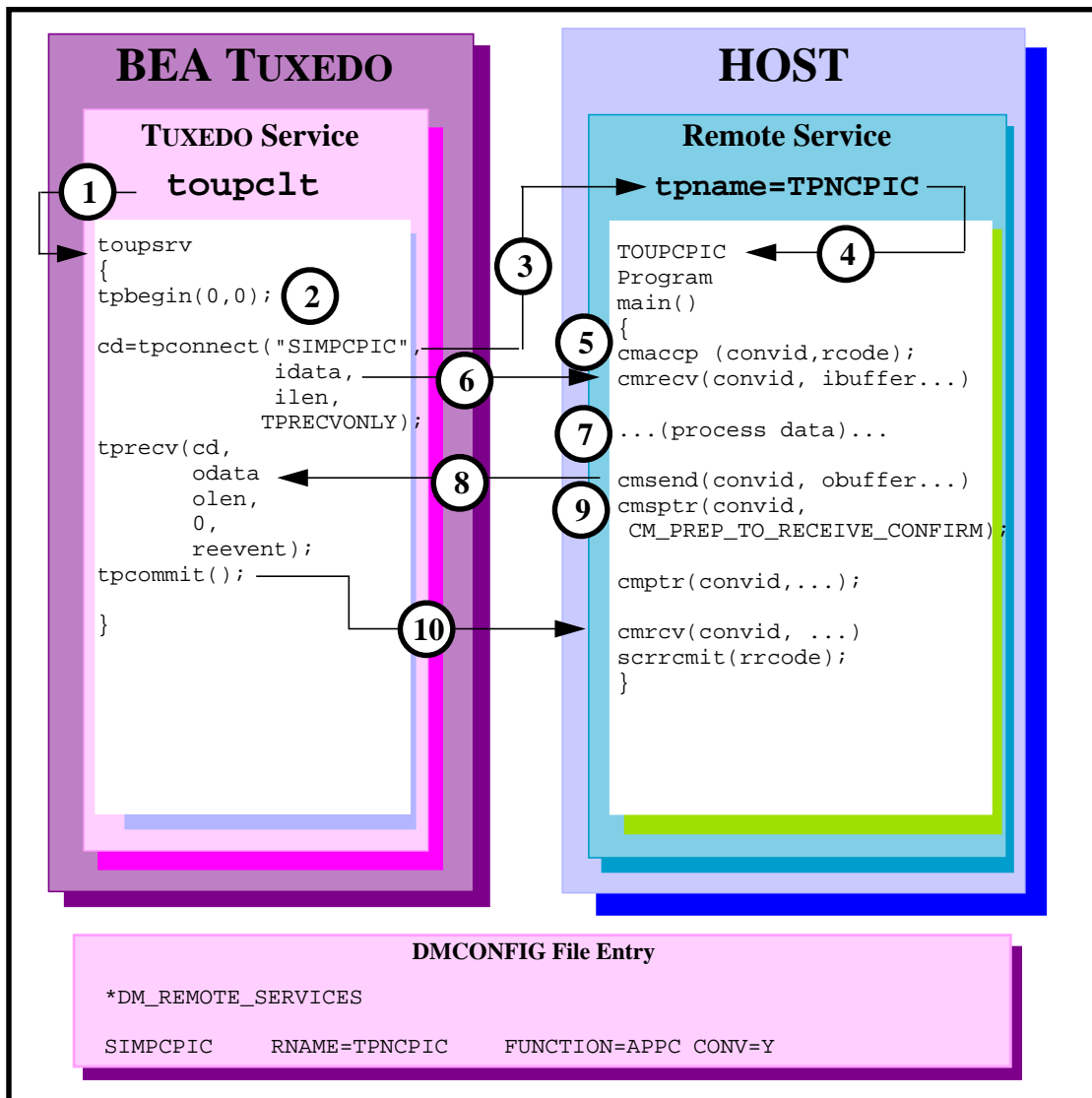
Figure 7-33 Transactional Tuxedo Client Request/Response to Host CPI-C



Step-by-Step Description: Transactional Tuxedo Client Request/Response to Host CPI-C

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpbegin` to start the transaction.
3. The `toupsrv` service issues `tpcall` for `SIMPCPIC`, which is advertised in the `*DM_REMOTE_SERVICES` section of the `DMCONFIG` file. Data is sent from the `idata` buffer on the `tpconnect`.
4. The remote service with `tpname` `TPNCPIC` invokes `TOUPCPIC` program.
5. The server accepts the conversation with the `cmaccp` call. The conversation ID returned on the request in `convid` is used for all other requests during this conversation.
6. The `cmrcv` request receives the `idata` buffer contents for processing.
7. The `cmsst` and `cmsptr` prepare the next send request by setting the send type to `CM_SEND_AND_PREP_TO_RECEIVE` and by setting the prepare-to-receive type to `CM_PREP_TO_RECEIVE_CONFIRM`.
8. The `cmsend` request immediately returns the `obuffer` contents into the client's `odata` buffer. The server relinquishes control to the server and indicates the end of the conversation with the `CONFIRM` request.
9. The `toupsrv` issues the `tpcommit` to successfully complete the transaction and commit all updated resources. The `cmrcv` request receives the commit request, and responds explicitly to the request with the SAA Resource/Recovery commit call `srrcmit`. The conversation is ended after the successful commit exchange.

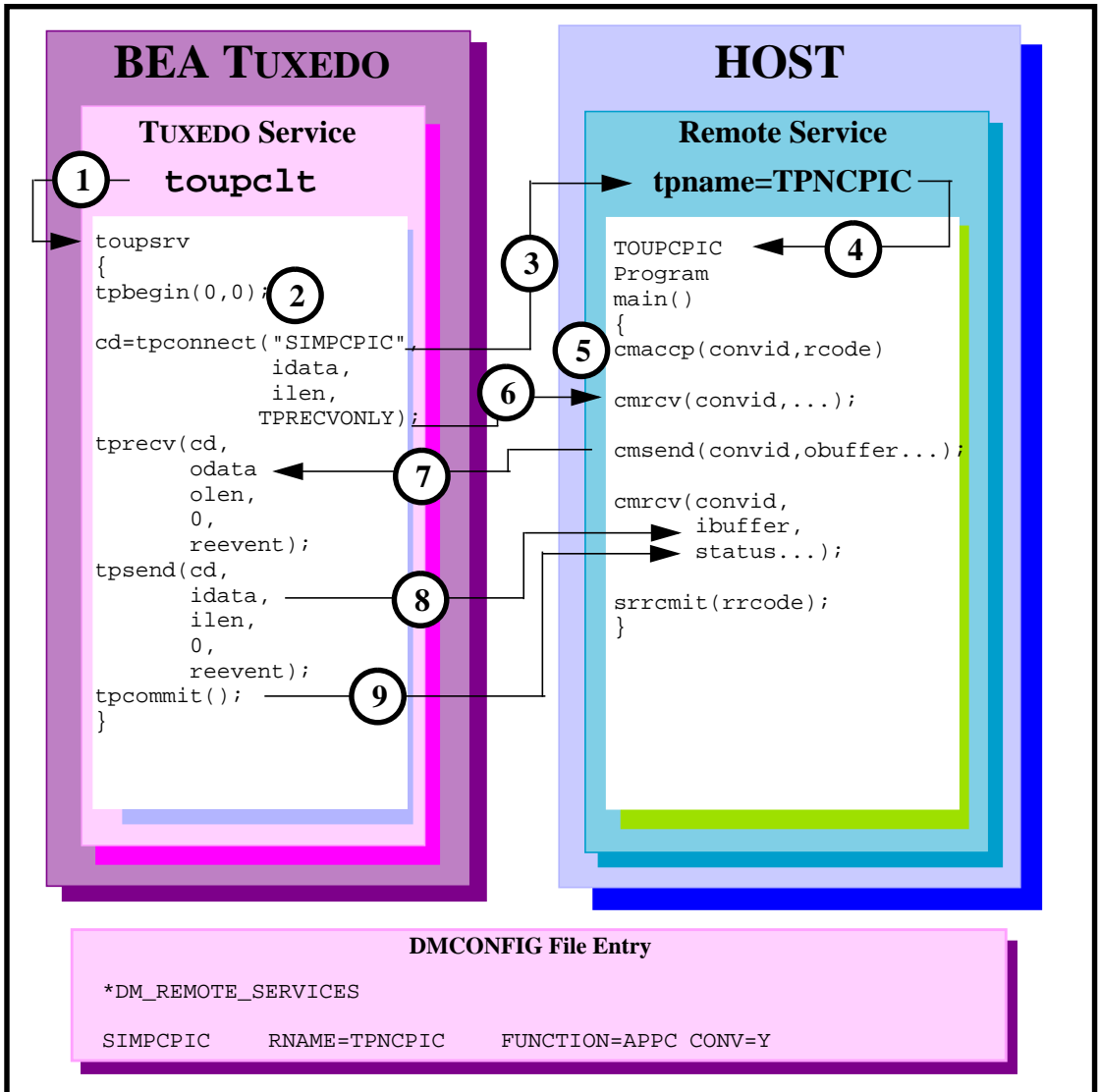
Figure 7-34 Transactional Tuxedo Conversational Client to Host CPI-C, Server Gets Control



Step-by-Step Description: Transactional Tuxedo Conversational Client to Host CPI-C, Server Gets Control

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpbegin` to start the transaction.
3. The `toupsrv` service issues a `tpconnect` service request for `SIMPCPIC`, which is advertised in the `*DM_REMOTE_SERVICES` section of the `DMCONFIG` file. Data is sent in the `idata` buffer on the `tpconnect`.
4. The remote service with `tpname` `TPNCPIC` invokes `TOUPCPIC` program.
5. The server accepts the conversation with the `cmaccp` call. The conversation ID returned on the request in `convid` is used for all other requests during this conversation.
6. The `cmrcv` request receives the `idata` buffer contents sent on the `tpconnect` for processing.
7. The `TOUPCPIC` program processes the data.
8. The `cmsend` returns the `obuffer` contents into the client's `tprecv` `odata` buffer. The buffer contents may not be sent immediately.
9. The `cmsptr` prepares the prepare-to-receive request with `CM_PREP_TO_RECEIVE_CONFIRM`. The `cmptr` request with `CONFIRM` indicates that the conversation is finished and is communicated to the `tprecv` as `TPEV_SVCSUCC`.
10. The `toupsrv` issues the `tpcommit` to successfully complete the transaction and commit all updated resources. The `cmrcv` request receives the commit request and responds explicitly to the request with the SAA Resource/Recovery commit call `srrcmit`. The conversation is ended after the successful commit exchange.

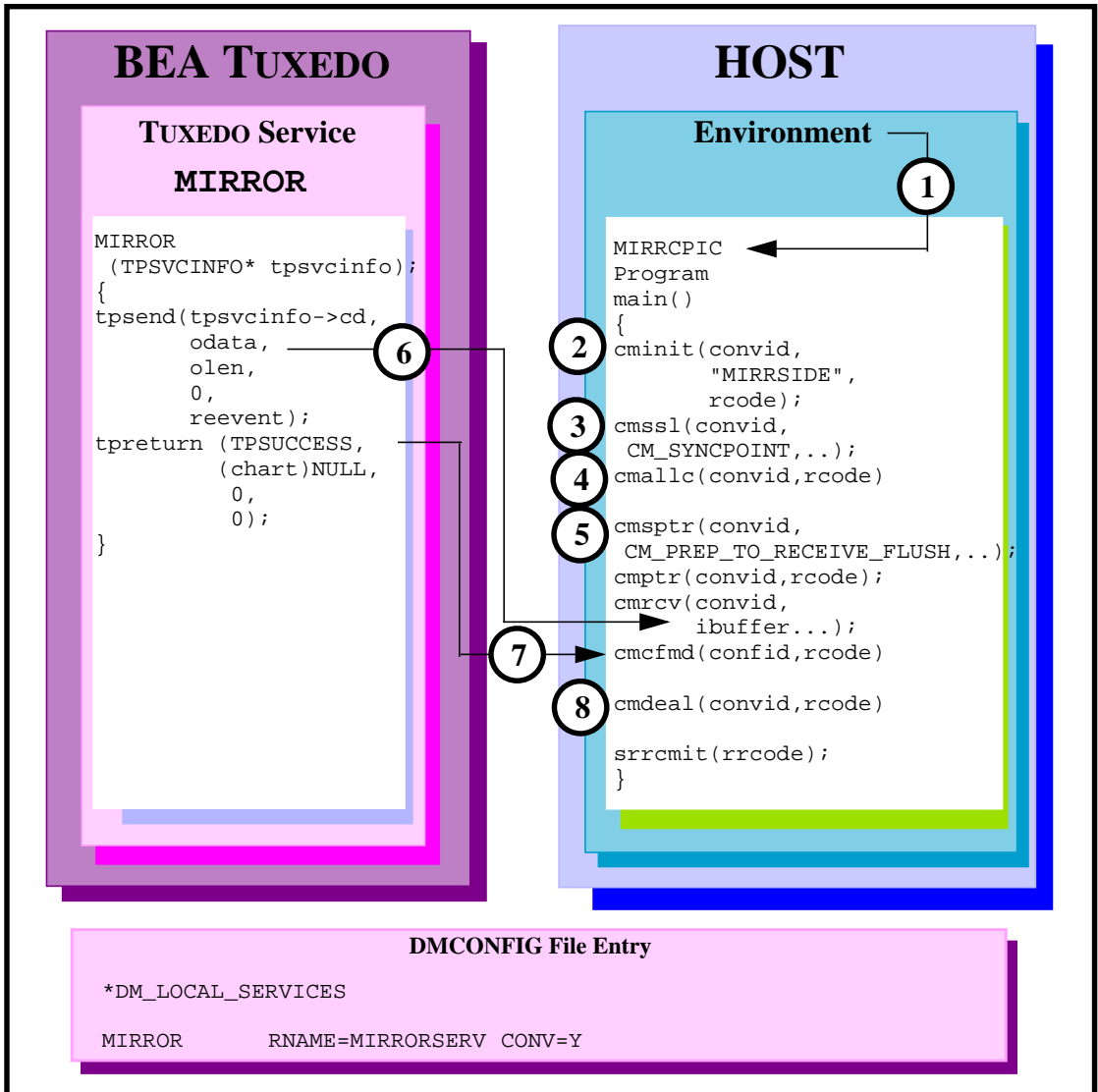
Figure 7-35 Transactional Tuxedo Conversational Client to Host CPI-C, Client Grants/Gets Control



Step-by-Step Description: Transactional Tuxedo Conversational Client to Host CPI-C, Client Grants/Gets Control

1. Tuxedo client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpbegin` to start the transaction.
3. The `toupsrv` service issues a `tpconnect` service request for `SIMPCPIC`, which is advertised in the `*DM_REMOTE_SERVICES` section of the `DMCONFIG` file. The `TPRECVONLY` indicates the server gains control and the first conversation verb `toupsrv` can issue is `tprecv`.
4. The remote service with `tpname` `TPNCPIC` invokes `TOUPCPIC` program.
5. The server accepts the conversation with the `cmaccp` call. The conversation ID returned on the request in `convid` is used for all other requests during this conversation.
6. The server receives control of the conversation on the `cmrcv` request.
7. The `cmsend` request sends the `obuffer` data contents into the `tprecv idata` buffer. The data might not be sent immediately.
8. The `cmrcv` request relinquishes control of the conversation and waits to receive the `tpsend ibuffer` data contents in the `odata` buffer.
9. The `toupsrv` issues the `tpcommit` to successfully complete the transaction and commit all updated resources. The `cmrcv` request receives the commit request and responds explicitly to the request with the SAA Resource/Recovery commit call `srrcmit`. The conversation is terminated after the successful commit exchange.

Figure 7-36 Transactional Host CPI-C to Tuxedo Conversational Server, Client Grants Control



Step-by-Step Description: Transactional Host CPI-C to Tuxedo Conversational Server, Client Grants Control

1. The CPI-C application program `MIRRCPIC` is invoked using environment start-up specifications.
2. The `MIRRCPIC` client requests `cminit` to establish conversation attributes and receive a conversation ID that will be used on all other requests on this conversation. The remote server and service are named in the CPI-C side information entry `MIRRSIDE`.
3. The `cmssl` sets the conversation attributes to sync-level 2 with `CM_SYNCPOINT`. This allows the Tuxedo service to participate in the transaction.
4. The `cmallc` request initiates the advertised service mapped to `MIRRORSERV` in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file.
5. The `MIRRCPIC` causes the client to relinquish control to the Tuxedo server with a prepare-to-receive request. The `cmsptr` sets the prepare-to-receive type to `CM_RECEIVE_AND_FLUSH`. The `cmptr` request immediately relinquishes control.
6. The `MIRROR` service sends the data contents of the `odata` buffer to the `cmrcv` `ibuffer`. The `cmrcv` receives a confirm request from the server indicating the conversation should be terminated.
7. The client replies positively to the confirm request with `cmcfmd`.
8. The `MIRRCPIC` client prepares to free the conversation with the `cmdeal` request. The conversation in `CM_DEALLOCATE_SYNC_LEVEL` commits all updated resources in the transaction and waits for the SAA resource recovery verb, `srrcmit`. After the `commit` sequence has completed, the conversation terminates.

Where to Find Additional Information

Additional information on the subject of CICS/ESA Intersystem Communications may be found in the following IBM publications:

- ◆ *CICS/ESA Intercommunication Guide*, IBM publication No. SC33-0657
- ◆ *CICS/ESA Distributed Transaction Programming Guide*, IBM publication No. SC33-00783
- ◆ *CICS/ESA Recovery and Restart Guide*, IBM publication No. SC33-0658

A Reference Pages

This appendix covers the following reference pages, formerly called man pages:

- ◆ addumap
- ◆ addusr
- ◆ delumap
- ◆ delusr
- ◆ DMADM
- ◆ dmadmin
- ◆ dmconfig
- ◆ dmloadcf
- ◆ dmunloadcf
- ◆ dmusradd
- ◆ dmusrmod
- ◆ GWADM
- ◆ modusr
- ◆ xsnacrm
- ◆ SNACRM
- ◆ CRMLOGS

addumap

Add a local-to-remote mapping for a local/remote domain pair

SYNOPSIS

DMCONFIG=dmconfig runs under dmadmin
addumap -d *local domain ID* -R *remote domain ID*
-p *local principal name* -u *remote username*

DESCRIPTION

addumap can only be executed as a subcommand of dmadmin(1). The purpose of this page is to describe options for the subcommand and to show examples.

The subcommand allows the administrator to add local-to-remote user mappings for a local/remote domain pair.

Mappings are defined to be inbound, outbound or both when the application is using /SNA Domain gateways and SECURITY is set to USER_AUTH, ACL, or MANDATORY ACL in the ubbconfig file and SECURITY is set to DM_PW or USER_PW in the DMCONFIG file.

The following options are available:

-d *local domain ID*

This is the name of the local domain gateway with which the ids and passwords are associated. This is the same ID as the one used when creating the domain definitions either in the DMCONFIG file or through the Graphical Administrative Interface.

-R *remote domain ID*

This is the name of the remote domain gateway with which the ids and passwords are associated. This is the same ID as the one used when creating the domain definitions either in the DMCONFIG file or through the Graphical Administrative Interface.

-p *local principal*

The user identification number. The *local principal* must be defined in the ACL user file and must be unique within the list of existing identifiers for the application.

-u *remote username*

The remote user name as defined in the ACL security application (for example, RACF) of the remote domain.

Before running this subcommand the application must be configured using either the Graphical Administrative Interface or `tmloadcf(1)` and `dmloadcf(1)`. `dmadmin addumap` may be run on any active node.

PORTABILITY

This subcommand is available only on non-/WS sites running TUXEDO System /T Release 6.4.

DIAGNOSTICS

The `dmadmin addumap` subcommand exits with a return code of 0 upon successful completion.

EXAMPLE

```
addumap -d ldom -R cics -p tuxusr -u cicsusr
/*maps principal tuxusr with
remote user cicsusr */
```

SEE ALSO

`dmadmin(1)`, `delumap(5)`

addusr

Add a user to the remote domain user and password file

SYNOPSIS

```
DMCONFIG=dmconfig  
addusr -d local domain ID -R remote domain ID -u remote username  
[-w ]
```

DESCRIPTION

addusr can only be executed as a subcommand of dmadmin(1). The purpose of this page is to describe options for the subcommand and to show an example.

The subcommand allows the administrator to add remote usernames and passwords to the remote domain remote user and password table. If -w is not specified, the user is prompted for a password.

The table entries created are used for passing remote user names and passwords to remote SNA domains when the application is using /SNA Domain gateways and SECURITY is set to USER_AUTH, ACL, or MANDATORY ACL in the ubbconfig file and SECURITY is set to DM_PW or USER_PW in the DMCONFIG file.

The following options are available:

-d *local domain ID*

This is the name of the local domain gateway with which the ids and passwords are associated. This is the same ID as the one used when creating the domain definitions either in the DMCONFIG file or through the Graphical Administrative Interface.

-R *remote domain ID*

This is the name of the remote domain gateway with which the ids and passwords are associated. This is the same ID as the one used when creating the domain definitions either in the DMCONFIG file or through the Graphical Administrative Interface.

-u *remote username*

The remote user name to be added.

-w

Do not prompt for password.

Before running this subcommand the application must be configured using either the Graphical Administrative Interface or `tmloadcf(1)` and `dmloadcf(1)`. `dmadmin addusr` may be run on any active node.

PORTABILITY

This subcommand is available only on non-/WS sites running TUXEDO System /T Release 6.4.

DIAGNOSTICS

The `dmadmin addusr` subcommand exits with a return code of 0 upon successful completion.

EXAMPLES

```
addusr -d tux -R cics -u cicsusr /*adds remote user cicsusr to
                                cics domain's user and
                                password file. The
                                administrator is prompted for
                                a password*/
```

SEE ALSO

`delusr(5)`, `modusr(5)`

delumap

Delete a local-to-remote mapping for a local/remote domain pair

SYNOPSIS

```
DMCONFIG=dmconfig  
delumap -d local domain ID -R remote domain ID  
-p local principal name -u remote username
```

DESCRIPTION

delumap can only be executed as a subcommand of dmadmin(1). The purpose of this page is to describe options for the subcommand and to show examples.

The subcommand allows the administrator to delete local-to-remote user mappings for a local/remote domain pair.

Mappings are defined to be inbound, outbound or both when the application is using /SNA Domain gateways and SECURITY is set to USER_AUTH, ACL, or MANDATORY ACL in the ubbconfig file and SECURITY is set to DM_PW or USER_PW in the DMCONFIG file.

The following options are available:

-d *local domain ID*

This is the name of the local domain gateway with which the ids and passwords are associated. This is the same ID as the one used when creating the domain definitions either in the DMCONFIG file or through the Graphical Administrative Interface.

-R *remote domain ID*

This is the name of the remote domain gateway with which the ids and passwords are associated. This is the same ID as the one used when creating the domain definitions either in the DMCONFIG file or through the Graphical Administrative Interface.

-p *local principal*

The user identification number. The *local principal* must be defined in the ACL user file and must be unique within the list of existing identifiers for the application.

-u remote username

The remote user name as defined in the ACL security application (for example, RACF) of the remote domain. Space is a valid remote username.

Before running this subcommand the application must be configured using either the Graphical Administrative Interface or `tmloadcf(1)` and `dmloadcf(1)`. `dmadmin delumap` may be run on any active node.

PORTABILITY

This subcommand is available only on non-/WS sites running TUXEDO System /T Release 6.4.

DIAGNOSTICS

The `dmadmin delumap` subcommand exits with a return code of 0 upon successful completion.

EXAMPLE

```
delumap -d ldom -R cics -p tuxusr -u cicsusr
/*deletes the mapping of principal
tuxusr with remote user cicsusr */
```

SEE ALSO

`dmadmin(1)`, `addumap(5)`

delusr

Delete a user from the remote domain user and password file

SYNOPSIS

DMCONFIG=dmconfig

delusr -d *local domain* -R *remote domain* -u *remote username*

DESCRIPTION

delusr can only be executed as a subcommand of dmadm(1). The purpose of this page is to describe options for the subcommand and to show an example.

The subcommand allows the administrator to remove remote usernames and passwords from the remote domain remote user and password table.

Once the entries are deleted they can no longer be used for mapping remote user names and passwords to local user names and passwords when the application is using /SNA Domain gateways and SECURITY is set to USER_AUTH, ACL, or MANDATORY ACL in the ubbconfig file and SECURITY is set to DM_USER_PW in the DMCONFIG file.

The following options are available:

-d *local domain ID*

This is the name of the local domain gateway with which the ids and passwords are associated. This is the same ID as the one used when creating the domain definitions either in the DMCONFIG file or through the Graphical Administrative Interface.

-R *remote domain ID*

This is the name of the remote domain gateway with which the ids and passwords are associated. This is the same ID as the one used when creating the domain definitions either in the DMCONFIG file or through the Graphical Administrative Interface.

-u *remote username*

The remote user name to be deleted.

Before running this subcommand the application must be configured using either the Graphical Administrative Interface or tmloadcf(1) and dmlloadcf(1). dmadm delusr may be run on any active node.

PORTABILITY

This subcommand is available only on non-/WS sites running TUXEDO System /T Release 6.4.

DIAGNOSTICS

The dmadmin delusr subcommand exits with a return code of 0 upon successful completion.

EXAMPLES

```
delusr -d tux -R cics -u cicsusr /*deletes remote user cicsusr to
                                cics domain users. The
                                administrator is prompted for a
                                password*/
```

SEE ALSO

addusr(5), modusr(5)

DMADM

/Domain administrative server

SYNOPSIS

DMADM SRVGRP = “*identifier*”

SRVID = “*number*”

REPLYQ = “*N*”

DESCRIPTION

The /DOMAIN administrative server (DMADM) is a TUXEDO-supplied server that provides run-time access to the BDMCONFIG file. When DMADM is booted the BDMCONFIG environment variable should be set to the pathname of the file containing the binary version of the DMCONFIG file.

DMADM is described in the *SERVERS section of the UBBCONFIG file as a server running within a group, e.g., DMADMGRP. There should be only one instance of the DMADM running in this group and it must not have a reply queue (REPLYQ must be set to “N”).

The following server parameters can also be specified for the DMADM server in the *SERVERS section: SEQUENCE, ENVFILE, MAXGEN, GRACE, RESTART, RQPERM and SYSTEM_ACCESS.

PORTABILITY

DMADM is supported as a TUXEDO-supplied server on UNIX System operating systems.

INTEROPERABILITY

The initial release of /SNA Domain can only be installed on a node running TUXEDO Release 6.4.

EXAMPLES

The following example illustrates the definition of the administrative server and a gateway group in the UBBCONFIG file.

```
#
*GROUPS
DMADMGRP LMID=mach1 GRPNO=1
gwgrp   LMID=mach1 GRPNO=2
#
*SERVERS
DMADM SRVGRP="DMADMGRP" SRVID=1001 REPLYQ=N RESTART=Y GRACE=0
GWADM SRVGRP="gwgrp" SRVID=1002 REPLYQ=N RESTART=Y GRACE=0
GWTDOMAIN SRVGRP="gwgrp" SRVID=1003 RQADDR="gwgrp" REPLYQ=Y
RESTART=Y MIN=1 MAX=1
```

SEE ALSO

dmadmin(1), tmboot(1), dmconfig(5), GWADM(5), servopts(5), ubbconfig(5)

TUXEDO /Domain User Guide

TUXEDO Administrator's Guide

dmadmin

TUXEDO System/T Domain Administration Command Interpreter

SYNOPSIS

dmadmin [-c]

DESCRIPTION

The dmadmin interactive command interpreter is used for the administration of domain gateway groups defined for a particular TUXEDO System/T application. The interpreter can operate in two modes: administration mode and configuration mode.

The dmadmin command interpreter enters *administration* mode when called with no parameters. This is the default. In this mode, dmadmin can be run on any active node (excluding workstations) within an active application. Application administrators can use this mode to obtain or change parameters on any active domain gateway group. Application administrators may also use this mode to create, destroy, or re-initialize the DMTLOG for a particular local domain. In this case, the domain gateway group associated with that local domain must not be active, and dmadmin must be run on the machine assigned to the corresponding gateway group.

The dmadmin command interpreter enters *configuration* mode when it is invoked with the -c option or when the config subcommand is invoked. Application administrators can use this mode to update or add new configuration information to the binary version of the domain configuration file (BDMCONFIG).

The dmadmin command interpreter requires the use of the DOMAIN administrative server (DMADM) for the administration of the BDMCONFIG file and the gateway administrative server (GWADM) for the re-configuration of active DOMAIN gateway groups (there is one GWADM per gateway group).

ADMINISTRATION MODE COMMANDS

Once dmadmin has been invoked, commands may be entered at the prompt (“>”) according to the following syntax:

command [arguments]

Several commonly occurring arguments can be given default values via the default command. Commands that accept parameters set via the default command check default to see if a value has been set. If no value is set, an error message is returned.

Once set, a default value remains in effect until the session is ended, unless changed by another default command. Defaults may be overridden by entering an explicit value on the command line, or reset by entering the value “*”. The effect of an override lasts for a single instance of the command.

Output from dmadmin commands is paginated according to the pagination command in use (see the paginate subcommand below).

Commands may be entered either by their full name or their abbreviation (shown in parentheses) followed by any appropriate arguments. Arguments appearing in square brackets, [], are optional; those in curly braces, {}, indicate a selection from mutually exclusive options. Note that for many commands *local_domain_name* is a required argument, but note also that it can be set with the default command.

The following commands are available in administration mode:

addumap [options]

Add local user mappings to remote user mappings for a local/remote domain pair. Mappings are defined to be inbound, outbound or both. See the addumap(5) reference page for an explanation of the available options and for examples.

addusr (addu) [options]

Add remote usernames and passwords to the remote user and password tables of a remote domain. See the addusr(5) reference page for an explanation of the available options and for examples.

advertise (adv) -d *local_domain_name* [{ -all | *service* }]

Advertise all remote services provided by the named local domain or the specified remote service.

audit (audit) -d *local_domain_name* [{off | on}]

Activate (on) or deactivate (off) the audit trace for the named local domain. If no option is given, then the current setting will be toggled between the values on and off, and the new setting will be printed. The initial setting is off.

chbktme (chbt) -d *local_domain_name* -t *bktme*

Change the blocking timeout for a particular local domain.

config (config)

Enter configuration mode. Commands issued in this mode follow the conventions defined in the section “Configuration Mode Commands” (see below).

crdmlog (crdlg) -d *local_domain_name*

Create the domain transaction log for the named local domain on the current machine (that is, the machine where dmadmin is running). The command uses the parameters specified in the DMCONFIG file. This command fails if the named local domain is active on the current machine or if the log already exists.

default (d) [-d *local_domain_name*]

Set the corresponding argument to be the default local domain. Defaults may be reset by specifying “*” as an argument.

If the default command is entered with no arguments, the current default values are printed.

delumap [options]

Delete local to remote user mappings for a local/remote domain pair. See the delumap(5) reference page for an explanation of the available options and for examples.

delusr (delu) [options]

Delete remote usernames and passwords from the remote user and password tables of a remote domain. See the delusr(5) reference page for an explanation of the available options and for examples.

dsdmlog (dsdlg) -d *local_domain_name* [-y]

Destroy the domain transaction log for the named local domain on the current machine (that is, the machine where dmadmin is running). An error is returned if a DMTLOG is not defined for this local domain, if the local domain is active, or if outstanding transaction records exist in the log. The term outstanding transactions means that a global transaction has been committed but an end-of-transaction has not yet been written. This command prompts for confirmation before proceeding unless the -y option is specified. dsdmlog is not supported for /SNA Domain.

echo (e) [{off | on}]

Echo input command lines when set to on. If no option is given, then the current setting is toggled, and the new setting is printed. The initial setting is off.

`forgettrans (ft) -d local_domain_name [-t tran_id]`

Forget one or all heuristic log records for the named local domain. If the transaction identifier *tran_id* is specified, then only the heuristic log record for that transaction will be forgotten. The transaction identifier *tran_id* can be obtained from the `printrans` command or from the ULOG file. `forgettrans` is not supported for /SNA Domain.

`help (h) [command]`

Print help messages. If *command* is specified, the abbreviation, arguments, and description for that command are printed. Omitting all arguments causes the syntax of all commands to be displayed.

`indmlog (indlg) -d local_domain_name [-y]`

Re-initialize the domain transaction log for the named local domain on the current machine (that is, the machine where `dmadmin` is running). An error is returned if a DMTLOG is not defined for this local domain, if the local domain is active, or if outstanding transaction records exist in the log. The term outstanding transactions means that a global transaction has been committed but an end-of-transaction has not yet been written. The command prompts for confirmation before proceeding unless the `-y` option is specified. `indmlog` is not supported for /SNA Domain.

`modusr (modu) [options]`

Change remote passwords in the password tables of a remote domain. See the `modusr(5)` reference page for an explanation of the available options and for examples.

`paginate (page) [{off | on}]`

Paginate output. If no option is given, then the current setting will be toggled, and the new setting is printed. The initial setting is on, unless either standard input or standard output is a non-tty device. Pagination may only be turned on when both standard input and standard output are tty devices. The shell environment variable `PAGER` may be used to override the default command used for paging output. The default paging command is the indigenous one to the native operating system environment, for example, the command `pg` is the default on UNIX System operating environments.

`passwd (passwd) [-r] local_domain_name remote_domain_name`

Prompts the administrator for new passwords for the specified local and remote domains. The `-r` option specifies that existing passwords and new passwords should be encrypted using a new key generated by the system. The password is truncated after at most eight characters.

`printdomain (pd) -d local_domain_name`

Print information about the named local domain. Information printed includes connected remote domains, global information shared by the gateway processes, and additional information that is dependent on the domain type instantiation.

`printstats (stats) -d local_domain_name`

Print statistical and performance information gathered by the named local domain. The information printed is dependent on the domain gateway type.

`printrans (pt) -d local_domain_name`

Print transaction information for the named local domain. `printrans` is not supported for /SNA Domain.

`quit (q)`

Terminate the session.

`resume (res) -d local_domain_name [{ -all | service}]`

Resume processing of the specified service or for all remote services handled by the named local domain.

`stats (stats) -d local_domain_name [{ off | on | reset }]`

Activate (*on*), deactivate (*off*), or reset (*reset*) statistics gathering for the named local domain. If no option is given, then the current setting will be toggled between the values *on* and *off*, and the new setting will be printed. The initial setting is *off*.

`suspend (susp) -d local_domain_name [{ -all | service}]`

Suspend one or all remote services for the named local domain.

`unadvertise (unadv) -d local_domain_name [{ -all | service}]`

Unadvertise one or all remote services for the named local domain.

`verbose (v) [{off | on}]`

Produce output in verbose mode. If no option is given, then the current setting will be toggled, and the new setting is printed. The initial setting is *off*.

`! shellcommand`

Escape to shell and execute *shellcommand*.

`!!`

Repeat previous shell command.

[text]

Lines beginning with "#" are comment lines and are ignored.

<CR>

Repeat the last command.

CONFIGURATION MODE COMMANDS

The `dmadmin` command enters configuration mode when executed with the `-c` option or when the `config` subcommand is used. In this mode, `dmadmin` allows run-time updates to the `BDMCONFIG` file. `dmadmin` manages a buffer that contains input field values to be added or retrieved, and displays output field values and status after each operation completes. The user can update the input buffer using any available text editor.

`dmadmin` first prompts for the desired section followed by a prompt for the desired operation.

The prompt for the section is as follows:

Sections:

- 1) LOCAL_DOMAINS 2) REMOTE_DOMAINS
- 3) LOCAL_SERVICES 4) REMOTE_SERVICES
- 5) ROUTING 6) ACCESS_CONTROL
- 7) PASSWORDS 8) TDOMAIN
- 9) OSITP 10) SNA
- 11) QUIT

Enter Section [1]:

The number of the default section appears in square brackets at the end of the prompt. You can accept the default by pressing RETURN or ENTER. To select another section enter its number, then press RETURN or ENTER.

`dmadmin` then prompts for the desired operation.

Operations:

- 1) FIRST 2) NEXT
- 3) RETRIEVE 4) ADD
- 5) UPDATE 6) DELETE
- 7) NEW_SECTION 8) QUIT

Enter Operation [1]:

The number of the default operation is printed in square brackets at the end of the prompt. Pressing RETURN or ENTER selects this option. To select another operation enter its number, then press RETURN or ENTER.

The currently supported operations are:

1. FIRST

Retrieve the first record from the specified section. No key fields are needed (they are ignored if in the input buffer).

2. NEXT

Retrieve the next record from the specified section, based on the key fields in the input buffer.

3. RETRIEVE

Retrieve the indicated record from the specified section by key field(s) (see fields description below).

4. ADD

Add the indicated record in the specified section. Any fields not specified (unless required) take their default values as specified in dmconfig(5). The current value for all fields is returned in the output buffer. This operation can only be done by the System/T administrator.

5. UPDATE

Update the record specified in the input buffer in the selected section. Any fields not specified in the input buffer remain unchanged. The current value for all fields is returned in the input buffer. This operation can only be done by the System/T administrator.

6. DELETE

Delete the record specified in the input buffer from the selected section. This operation can only be done by the System/T administrator.

7. NEW SECTION

Clear the input buffer (all fields are deleted). After this operation, dmadmin immediately prompts for the section again.

8. QUIT

Exit the program gracefully (dmadmin is terminated). A value of q for any prompt also exits the program.

For configuration operations, the effective user identifier must match the System/T administrator user identifier (UID) for the machine on which this program is executed. When a record is updated or added, all default values and validations used by `dmloadcf(1)` are enforced.

`dmadmin` then prompts whether or not to edit the input buffer.

Enter editor to add/modify fields [n]?

Entering a value of `y` will put the input buffer into a temporary file and execute the text editor. The environment variable `EDITOR` is used to determine which editor to be used; the default is `"ed"`. The input format is in field name/field value pairs and is described in the CONFIGURATION INPUT FORMAT section below. The field names associated with each `DMCONFIG` section are listed in tables in the subsections below. The semantics of the fields and associated ranges, default values, restrictions, etc., are described in `dmconfig(5)`. In most cases, the field name is the same as the `KEYWORD` in the `DMCONFIG` file, prefixed with `"TA_"`. When the user completes editing the input buffer, `dmadmin` reads it. If more than one line occurs for a particular field name, the first occurrence is used and other occurrences are ignored. If any errors occur, a syntax error will be printed and `dmadmin` prompts whether or not to correct the problem.

Enter editor to correct?

If the problem is not corrected (response `n`), then the input buffer will contain no fields. Otherwise, the editor is executed again.

Finally, `dmadmin` asks if the operation should be done.

Perform operation [y]?

When the operation completes, `dmadmin` prints the return value as in

Return value `TAOK`

followed by the output buffer fields. The process then begins again with a prompt for the section. All output buffer fields are available in the input buffer unless the buffer is cleared.

Entering `break` at any time restarts the interaction at the prompt for the section.

When `"QUIT"` is selected, `dmadmin` prompts for authorization to create a backup ASCII version of the configuration:

Unload `BDMCONFIG` file into ASCII backup [y]?

If a backup is selected, `dmadmin` prompts for the file name.

Backup filename [DMCONFIG]?

On success, `dmadmin` indicates that a backup was created, otherwise an error is printed.

CONFIGURATION INPUT FORMAT

Input packets consist of lines formatted as follows:

```
fldname<tabs>fldval
```

The field name is separated from the field value by one or more tabs (or spaces).

Lengthy field values can be continued on the next line by having the continuation line begin with one or more tabs (which are dropped when read back into `dmadmin`).

Empty lines consisting of a single newline character are ignored.

To enter an unprintable character in the field value or to start a field value with a tab, use a backslash followed by the two-character hexadecimal representation of the desired character (see ASCII(5) in a UNIX reference manual). A space, for example, can be entered in the input data as `\20`. A backslash can be entered using two backslash characters. `dmadmin` recognizes all input in this format, but its greatest usefulness is for non-printing characters.

CONFIGURATION LIMITATIONS

The following are general limitations of the dynamic domain re-configuration capability:

- ◆ Values for key fields (as indicated in the following sections) may not be modified. Key fields can be modified, when the system is down, by reloading the configuration file.
- ◆ Dynamic deletions cannot be applied when local domains are active (the corresponding gateway group is running).

RESTRICTIONS FOR CONFIGURATION FIELD IDENTIFIERS/UPDATES

The following sections describe, for each DMCONFIG section, what the field identifiers are for each DMCONFIG field, what the field type of the identifier is, and when the field can be updated. All applicable field values are returned with the retrieval operations. Fields that are allowed and/or required for adding a record are described in `dmconfig(5)`.

Fields indicated below as *key* are key fields that are used to uniquely identify a record within section. These key fields are required to be in the input buffer when updates are done and are not allowed to be updated dynamically. The Update column indicates when a field can be updated. The possible values are:

Yes

Can be updated at any time.

NoGW

Cannot be updated dynamically while the gateway group representing the local domain is running.

No

Cannot be updated dynamically while at least one gateway group is running.

CONFIGURING THE DM_LOCAL_DOMAINS SECTION

The following table lists the fields in the *DM_LOCAL_DOMAINS section.

Table A-1 *DM_LOCAL_DOMAINS SECTION

Field Identifier	Field Type	Update	Notes
TA_LDOM	string	NoGW	key
TA_AUDITLOG	string	Yes	
TA_BLOCKTIME	numeric	Yes	
TA_DOMAINID	string	NoGW	
TA_DMTLOGDEV	string	NoGW	
TA_DMTLOGNAME	string	NoGW	
TA_DMTLOGSIZE	numeric	NoGW	
TA_GWGRP	string	NoGW	
TA_MAXDATALEN	numeric	Yes	
TA_MAXRDOM	numeric	Yes	
TA_MAXRDTRAN	numeric	NoGW	
TA_MAXTRAN	numeric	NoGW	

Table A-1 *DM_LOCAL_DOMAINS SECTION

Field Identifier	Field Type	Update	Notes
TA_SECURITY	string	Yes	format: {NONE APP_PW DM_PW}
TA_TYPE	string	NoGW	format: {TDOMAIN OSITP SNA}

CONFIGURING THE DM_REMOTE_DOMAINS SECTION

The following table lists the fields in the *DM_REMOTE_DOMAINS section.

Table A-2 *DM_REMOTE_DOMAINS SECTION

Field Identifier	Field Type	Update	Notes
TA_RDOM	string	No	key
TA_DOMAINID	string	No	
TA_TYPE	string	No	format: {TDOMAIN OSITP SNA}

CONFIGURING THE DM_TDOMAIN SECTION

The *DM_TDOMAIN section contains the network addressing parameters required by TDOMAIN type domains. The following lists the fields in this section:

Table A-3 *DM_TDOMAIN SECTION

Field Identifier	Field Type	Update	Notes
TA_LDOM or TA_RDOM	string	No/NoGW	key
TA_NWADDR	string	No/NoGW	ASCII format (no embedded NULL characters)
TA_NWDEVICE	string	No/NoGW	

If the domain identifier (TA_LDOM) is a local domain identifier, then the TA_NWADDR and TA_NWDEVICE fields can be updated if the gateway group representing that local domain is not running.

CONFIGURING THE DM_OSITP SECTION

The *DM_OSITP section contains the network addressing parameters required by OSITP type domains. The following lists the fields in this section:

Table A-4 *DM_OSITP SECTION

Field Identifier	Field Type	Update	Notes
TA_LDOM or TA_RDOM	string	No/NoGW	key
TA_APT	string	No/NoGW	
TA_AEQ	string	No/NoGW	
TA_AET	string	No/NoGW	
TA_ACN	string	No/NoGW	
TA_APID	string	No/NoGW	
TA_AEID	string	No/NoGW	
TA_PROFILE	string	No/NoGW	

If the domain identifier (TA_LDOM) is a local domain identifier, then the other fields in this table can be updated if the gateway group representing that local domain is not running.

CONFIGURING THE DM_LOCAL_SERVICES SECTION

The following table lists the fields in the *DM_LOCAL_SERVICES section.

Table A-5 *DM_LOCAL_SERVICES SECTION

Field Identifier	Field Type	Update	Notes
TA_SERVICENAME	string	No	key
TA_LDOM	string	Yes	
TA_RNAME	string	Yes	
TA_ACLNAME	string	Yes	

Table A-5 *DM_LOCAL_SERVICES SECTION

Field Identifier	Field Type	Update	Notes
TA_BUFTYPE	string	Yes	
TA_BUFSTYPE	string	Yes	
TA_OBUFTYPE	string	Yes	
TA_OBUFSTYPE	string	Yes	

CONFIGURING THE DM_REMOTE_SERVICES SECTION

The following table lists the fields in the *DM_REMOTE_SERVICES section.

Table A-6 *DM_REMOTE_SERVICES SECTION

Field Identifier	Field Type	Update	Notes
TA_SERVICENAME	string	No	key
TA_RDOM	string	No	key
TA_LDOM	string	No	key
TA_RNAME	string	Yes	
TA_CONV	string	NoGW	format: { Y N }
TA_BUFTYPE	string	Yes	
TA_BUFSTYPE	string	Yes	
TA_OBUFTYPE	string	Yes	
TA_OBUFSTYPE	string	Yes	
TA_ROUTINGNAME	string	Yes	
TA_TRANTIME	numeric	Yes	
TA_FUNCTION	string	No	

CONFIGURING THE DM_ROUTING SECTION

The following table lists the fields in the *DM_ROUTING section.

Table A-7 *DM_ROUTING SECTION

Field Identifier	Field Type	Update	Notes
TA_ROUTINGNAME	string	No	key
TA_FIELD	string	Yes	
TA_RANGE	string	Yes	
TA_BUFTYPE	string	Yes	

CONFIGURING THE DM_ACCESS_CONTROL SECTION

The following table lists the fields in the *DM_ACCESS_CONTROL section.

Table A-8 *DM_ACCESS_CONTROL SECTION

Field Identifier	Field Type	Update	Notes
TA_ACLNAME	string	No	key
TA_RDOM	string	Yes	

CONFIGURING THE DM_PASSWORDS SECTION

The following table lists the fields in the *DM_PASSWORDS section.

Table A-9 *DM_PASSWORDS SECTION

Field Identifier	Field Type	Update	Notes
TA_LDOM	string	No	key
TA_RDOM	string	No	key
TA_LPWD	string	Yes	format: { Y N U }
TA_RPWD	string	Yes	format: { Y N U }

The TA_LPWD and TA_RPWD show the existence of a defined password for the local and/or the remote domain. Passwords are not displayed. If an UPDATE operation is selected, the value of the corresponding field must be set to U. The program will then prompt with echo turned off for the corresponding passwords.

DIAGNOSTICS IN CONFIGURATION MODE

dmadmin fails if it cannot allocate an FML typed buffer, if it cannot determine the /etc/passwd entry for the user, or if it cannot reset the environment variables FIELDTBLS or FLDTBLDIR.

The return value printed by dmadmin after each operation completes indicates the status of the requested operation. There are three classes of return values.

The following return values indicate a problem with permissions or a TUXEDO System/T communications error. They indicate that the operation did not complete successfully.

[TAEPERM]

The calling process specified an ADD, UPDATE, or DELETE operation but it is not running as the System/T administrator. Update operations must be run by the administrator (that is, the user specified in the UID attribute of the RESOURCES section of the TUXCONFIG file).

[TAESYSTEM]

A TUXEDO System/T error has occurred. The exact nature of the error is written to userlog(3).

[TAEOS]

An operating system error has occurred.

[TAETIME]

A blocking timeout occurred. The input buffer is not updated so no information is returned for retrieval operations. The status of update operations can be checked by doing a retrieval on the record that was being updated.

The following return values indicate a problem in doing the operation itself and generally are semantic problems with the application data in the input buffer. The string field TA_STATUS will be set in the output buffer and will contain short text describing the problem. The string field TA_BADFLDNAME will be set to the field name for the field containing the value that caused the problem (assuming the error can be attributed to a single field).

[TAECONFIG]

An error occurred while reading the BDMCONFIG file.

[TAEDUPLICATE]

The operation attempted to add a duplicate record.

[TAEINCONSIS]

A field value or set of field values are inconsistently specified.

[TAENOTFOUND]

The record specified for the operation was not found.

[TAENOSPACE]

The operation attempted to do an update but there was not enough space in the BDMCONFIG file.

[TAERANGE]

A field value is out of range or is invalid.

[TAEREQUIRED]

A field value is required but not present.

[TAESIZE]

A field value for a string field is too long.

[TAEUPDATE]

The operation attempted to do an update that is not allowed.

The following return values indicate that the operation was successful.

[TAOK]

The operation succeeded. No updates were done to the BDMCONFIG file.

[TAUPDATED]

The operation succeeded. Updates were made to the BDMCONFIG file.

When using `dmunloadcf` to print entries in the configuration, optional field values are not printed if they are not set (for strings) or 0 (for integers). These fields will always appear in the output buffer when using `dmadmin`. In this way, it makes it easier for the administrator to retrieve an entry and update a field that previously was not set. The entry will have the field name followed by a tab but no field value.

CONFIGURATION EXAMPLE

In the following example, `dmadmin` is used to add a new remote domain. For illustration purposes, `ed` is used for the editor.

```
$ EDITOR=ed dmadmin
> config
Sections:
    1) LOCAL_DOMAINS    2) REMOTE_DOMAINS
    3) LOCAL_SERVICES    4) REMOTE_SERVICES
    5) ROUTING           6) ACCESS_CONTROL
    7) PASSWORDS         8) TDOMAIN
    9) OSITP             10) SNA
    11) QUIT
Enter Section [1]: 2
Operations:
    1) FIRST             2) NEXT
    3) RETRIEVE          4) ADD
    5) UPDATE            6) DELETE
    7) NEW_SECTION       8) QUIT
Enter Operation [1]: 4
Enter editor to add/modify fields [n]? y
a
TA_RDOM                B05
TA_DOMAINID            BA.BANK05
TA_TYPE                TDOMAIN
w
53
q
Perform operation [y]? <return>
Return value TAUPDATED
Buffer contents:
TA_OPERATION           4
TA_SECTION             2
TA_DOMAINID            BA.BANK05
TA_RDOM                B05
TA_TYPE                TDOMAIN
TA_STATUS              Update completed successfully
Operations:
    1) FIRST             2) NEXT
    3) RETRIEVE          4) ADD
```

```

5) UPDATE          6) DELETE
7) NEW_SECTION    8) QUIT
Enter Operation [4]: 7
Sections:
1) LOCAL_DOMAINS  2) REMOTE_DOMAINS
3) LOCAL_SERVICES 4) REMOTE_SERVICES
5) ROUTING        6) ACCESS_CONTROL
7) PASSWORDS      8) TDOMAIN
9) OSITP          10) QUIT
Enter Section [1]: 8
Operations:
1) FIRST          2) NEXT
3) RETRIEVE       4) ADD
5) UPDATE         6) DELETE
7) NEW_SECTION    8) QUIT
Enter Operation [6]: 4
Enter editor to add/modify fields [n]? y
a
TA_RDOM           B05
TA_NWADDR         0x00020401c0066d05
TA_NWDEVICE       /dev/tcp
w
55
q
Perform operation [y]? <return>
Return value TAUPDATED
Buffer contents:
TA_OPERATION      4
TA_SECTION        8
TA_RDOM           B05
TA_NWADDR         0x00020401c0066d05
TA_NWDEVICE       /dev/tcp
TA_STATUS         Update completed successfully
Operations:
1) FIRST          2) NEXT
3) RETRIEVE       4) ADD
5) UPDATE         6) DELETE
7) NEW_SECTION    8) QUIT
Enter Operation [4]: 8

```

```
> quit
```

The dmadm program ends.

SECURITY

If dmadm is run with the application administrator's UID, it assumes a trusted user and Security is bypassed. If dmadm is run with another user ID, and if the security option is enabled in the TUXCONFIG file, then the corresponding application password is required to start the dmadm program. If standard input is a terminal, then dmadm will prompt the user for the password with echo turned off. If standard input is not a terminal, the password is retrieved from the environment variable, APP_PW. If this environment variable is not specified and an application password is required, then dmadm will fail to start.

When running with another user ID (other than the UID of the administrator) only a limited set of commands is available.

ENVIRONMENT VARIABLES

dmadm resets the FIELDTBLS and FLDTBLDIR environment variables to pick up the \${TUXDIR}/udataobj/dmadm field table. Hence, the TUXDIR environment variable should be set correctly.

If the application requires security and the standard input to dmadm is not from a terminal, then the APP_PW environment variable must be set to the corresponding application password.

The TUXCONFIG environment variable should be set to the pathname of the TUXEDO System/T configuration file.

GENERAL DIAGNOSTICS

If the dmadm command is entered before the system has been booted, the following message is displayed:

No bulletin board exists. Only logging commands are available.

dmadm then prompts for the corresponding commands.

If an incorrect application password is entered or is not available to a shell script through the environment, then a log message is generated, the following message is displayed, and the command terminates:

Invalid password entered.

INTEROPERABILITY

dmadmin for /SNA must be installed on TUXEDO System/T R6.4. Other nodes in the same domain with an R6.4 gateway may be TUXEDO System/T R4.2.2 or later.

PORTABILITY

dmadmin for /SNA is supported as a TUXEDO System/T-supplied administrative tool on UNIX operating systems only.

SEE ALSO

dmloadcf(1), tadmin(1), dmconfig(5), DMADM(5), addusr(5), delusr(5)

TUXEDO /Domain User Guide

dmconfig

TUXEDO System/T ASCII domain configuration file

DESCRIPTION

dmconfig is the ASCII version of a TUXEDO System/Domain domain configuration file; it is also referred to by its environmental variable name: DMCONFIG. The dmconfig file is parsed and loaded into a binary version by the dmloadcf(1) utility. The binary configuration file, called the BDMCONFIG file, contains information used by domain gateways to initialize the context required for communications with other domains. dmadmin(1) uses the binary file (or a copy of it) in its monitoring activity. There will be one BDMCONFIG file for each TUXEDO System/Domain application that uses the /Domain feature.

A DMCONFIG file, and its binary BDMCONFIG counterpart, are analogous to the UBBCONFIG and TUXCONFIG files of a non-/Domain System/T application. The DMCONFIG file extends the definition of a non-/Domain System/T application so that the application becomes a domain.

Definitions

A TUXEDO System/Domain Application is defined as the environment described in a single TUXCONFIG file. A System/T Application can communicate with another System/T Application or with another TP Application via a domain gateway group. In “TUXEDO System/Domain” terms, an Application is the same as a TP Domain.

A Gateway Group is a collection of domain gateway processes that provide communication services with a specific type of TP Domain.

A Domain Gateway is a TUXEDO System/Domain process that relays requests and replies to another TP Domain.

A Local Domain characterizes a part of the application (set or subset of services) that is made available to other domains. A Local Domain is always represented by a Domain Gateway Group, and both terms are used as synonyms.

A Remote Domain is a remote application that is accessed through a Gateway Group. The remote application may be another TUXEDO System/Domain application or an application running under another TP system.

A Remote Service is a service provided by a remote domain that is made available to the local application through a Gateway Group.

A Local Service is a service of a local domain that is made available to remote domains through a Gateway Group.

Configuration File Format

The format of a domain configuration file is as follows:

- ◆ The file is made up of eight possible specification sections. Lines beginning with an asterisk (*) indicate the beginning of a specification section. Each such line contains the name of the section immediately following the *. Allowable section names are: DM_LOCAL_DOMAINS, DM_REMOTE_DOMAINS, DM_SNACRM, DM_SNASTACKS, DM_SNALINKS, DM_LOCAL_SERVICES, DM_REMOTE_SERVICES, DM_ROUTING, DM_ACCESS_CONTROL, DM_OSITP, and DM_TDOMAIN. The DM_LOCAL_DOMAINS section must precede the DM_REMOTE_DOMAINS /.
- ◆ Parameters are generally specified by: *KEYWORD* = *value*. This sets *KEYWORD* to *value*. Valid keywords are described below within each section. *KEYWORDS* are reserved; they can not be used as *values* unless they are quoted.

Lines beginning with the reserved word, *DEFAULT:*, contain parameter specifications that apply to any lines that follow them in the section in which they appear. Default specifications can be used in all sections. They can appear more than once in the same section. The format for these lines is:

```
DEFAULT: [ KEYWORD1 = value1 [ KEYWORD2 = value2 [...]]]
```

The values set on this line remain in effect until reset by another *DEFAULT:* line, or until the end of the section is reached. These values can also be overridden on non-*DEFAULT:* lines by placing the optional parameter setting on the line. If on a non-*DEFAULT:* line, the parameter setting is valid for that line only; lines that follow revert to the default setting. If *DEFAULT:* appears on a line by itself, all previously set defaults are cleared and their values revert to the system defaults.

If a value is *numeric*, standard C notation is used to denote the base (that is, 0x prefix for base 16 (hexadecimal), 0 prefix for base 8 (octal), and no prefix for base 10 (decimal)). The range of values acceptable for a numeric parameter are given under the description of that parameter.

If a value is an *identifier*, standard C rules are used. An *identifier* must start with an alphabetic character or underscore and contain only alphanumeric characters or underscores. The maximum allowable length of an identifier is 30 (not including the terminating null). An identifier cannot be the same as any *KEYWORD*.

A value that is neither an integer number or an identifier must be enclosed in double quotes. Certain special characters can be escaped inside a string using a backslash. “\\” translates to a single backslash. “\” translates to a double quote. “\n” translates to a newline. “\t” translates to a tab. “\f” translates to a form feed. “\x” (where ‘x’ is any character other than one of the previously mentioned special characters) translates to ‘x’.

- ◆ Input fields are separated by at least one space (or tab) character.
- ◆ “#” introduces a comment. A newline ends a comment.
- ◆ Blank lines and comments are ignored.
- ◆ Comments can be freely attached to the end of any line.
- ◆ Lines are continued by placing at least one tab after the newline. Comments can not be continued.

VERSION=*string_value*

where *string_value* can be any value. The field is not checked by the software; it is provided simply as a place where the customer can enter a string that may have some documentation value to the application.

The DM_LOCAL_DOMAINS Section

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

Entries have the form:

LDOM required parameters [optional parameters]

where *LDOM* is an *identifier* value used to name each local domain. *LDOM* must be unique within a particular configuration. As you will see in the description of the *DM_LOCAL_SERVICES section, *LDOM* is the identifier that connects local services with a particular gateway group.

The following are the required parameters:

GWGRP = *identifier*

specifies the name of the gateway server group (the name provided in the TUXCONFIG file) representing this local domain. There is a one-to-one relationship between a *DOMAINID* (see below) and the name of the gateway server group, that is, each GWGRP must have its own, unique *DOMAINID*.

TYPE = *identifier*

is used for grouping local domain into classes. TYPE can be set to one of the following values: TDOMAIN, OSITP or SNAX. The TDOMAIN value indicates that this local domain can only communicate with another TUXEDO System/Domain. The OSITP value indicates that this local domain communicates with another TP Domain via the OSI-TP protocol. The SNA value indicates that this local domain communicates with an MVS/CICS region via the LU6.2 protocol. Domain types must be defined in the \$TUXDIR/udataobj/DMTYPE file.

DOMAINID = *string*

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

DMTLOGDEV = *string*

specifies the TUXEDO filesystem that contains the Domain transaction log (DMTLOG) for this machine. The DMTLOG is stored as a TUXEDO System VTOC table on the device. If this parameter is not specified (and it should not be specified if TYPE=SNADOM), the domain gateway group is not allowed to process requests in transaction mode. Local domains running on the same machine can share the same DMTLOGDEV filesystem, but each local domain must have its own log (a table in the DMTLOGDEV) named as specified by the DMTLOGNAME keyword (see below).

Optional parameters describe resources and limits used in the operation of domain gateways:

AUDITLOG = *string*

specifies the name of the audit log file for this local domain. The audit log feature is activated from the dmadm(1) command and records all the operations within this local domain. If the audit log feature is active and this parameter is not specified, the file DMmmddy.LOG (where mm=month, dd=day, and yy=year) is created in the directory specified by the \$APPDIR environment variable or the APPDIR keyword of the *MACHINES section of the TUXCONFIG file.

BLOCKTIME = *numeric*

specifies the maximum wait time allowed for a blocking call. The value sets a multiplier of the SCANUNIT parameters specified in the TUXCONFIG file. The value SCANUNIT * BLOCKTIME must be greater than or equal to SCANUNIT and less than 32,768 seconds. If this parameter is not specified, the default value is set to the value of the BLOCKTIME parameter specified in the TUXCONFIG file. A time-out always implies a failure of the affected request. Notice that the time-out specified for transactions in the TUXCONFIG will always be used when the request is issued within a transaction.

DMTLOGNAME = *identifier*

specifies the name of the domain transaction log for this domain. This name must be unique when the same DMTLOGDEV is used for several local domains. If not specified, the default is the string “DMTLOG”. The name must be 30 characters or less. Since transactions are not support for /SNA Domain, this parameter has no meaning when TYPE=SNADOM.

DMTLOGSIZE = *numeric*

specifies the numeric size, in pages, of the Domain transaction log for this machine. It must be greater than 0 and less than the amount of available space on the TUXEDO filesystem. If not specified, the default is 100 pages. Since transactions are not support for /SNA Domain, this parameter has no meaning when TYPE=SNADOM.

MAXDATALEN = *numeric*

specifies a maximum amount of data (in bytes) that can be sent to or from any services advertised by this local domain. There is no limit if this parameter is not specified.

MAXRDOM = *numeric*

specifies the maximum number of connections (or dialogues if the domain is of type *OSITP*) allowed per gateway. There is no limit if this parameter is not specified.

MAXRDTRAN = *numeric*

specifies the maximum number of domains that can be involved in a transaction. It must be greater than 0 and less than 32,768. If not specified, the default is 16. Since transactions are not support for /SNA Domain, this parameter has no meaning when TYPE=SNADOM.

MAXTRAN = *numeric*

specifies the maximum number of simultaneous global transactions allowed on this local domain. It must be greater than or equal to 0 and less than or equal to the MAXGTT parameter specified in the TUXCONFIG file. If not specified, the default is the value of MAXGTT.

MAXSENDLEN = *numeric*

specifies the maximum length (in bytes) of messages sent or received by this local domain. If this parameter is set all messages sent or received will be broken up into packets of no more than MAXSENDLEN bytes. There is no limit if this parameter is not specified.

SECURITY = *value*

specifies the type of application security to be enforced. The following description applies to security in an /SNA Domain.

The combined settings of the SECURITY parameters in the UBBCONFIG and the DMCNFIG files have the following effects:

- ◆ When the DM_LOCAL_DOMAINS Security parameter is set to NONE or APP_PW, no action is taken by the CONNECT SNA gateway with regard to security.
- ◆ However, when the UBBCONFIG file Security parameter is set to APP_PW, the application password is validated by an AUTHSVC when clients join the application. The AUTHSVC is provided by the user application.

If security is to be enforced by both the local domain and the host system for each request outbound from the local domain, the UBBCONFIG file Security parameter must be set to one of the following: USER_AUTH, ACL, or MANDATORY_ACL; the DMCNFIG file *DM_LOCAL_DOMAINS section Security parameter must be set to DM_USER_PW; and the *DM_SNALINKS Security parameter must be set to IDENTIFY or VERIFY.

If security is to be enforced by both the local domain and the host system for each request inbound from the host system to the local domain, the UBBCONFIG file Security parameter must be set to one of the following: USER_AUTH, ACL, or MANDATORY_ACL; the DMCNFIG file *DM_LOCAL_DOMAINS section Security parameter must be set to DM_USER_PW; and the *DM_SNALINKS Security parameter must be set to IDENTIFY or VERIFY.

For a request sent to the host system, the local principal userid is located in the domain security table and the associated remote userid, or userid and password, are put into the conversation start-up request before being sent over the LU6.2 conversation. (This occurs if SECURITY is set to IDENTIFY or VERIFY in the DM_SNALINKS section of the DMCONFIG file.)

For requests sent from the host system, the local domain extracts the remote userid, or userid and password, from the conversation start-up request and checks the domain security table. That table contains pairs of local principal userids and remote userids, maintained on a service-by-service basis. The remote userid is mapped to the local principal userid. The local principal userid and password are used for further Access Control List (ACL) checking, as specified in the UBBCONFIG file.

When a request is received from the host system, the local domain checks the DMCONFIG file ACL for the local service to see if requests from the remote domain are permitted. If the DMCONFIG file does not contain an ACL for the local service, the service is accessible to all requests.

Therefore, if the ATTACHSEC level for the connection definition in the host system is Identify or Verify, the DMCONFIG SECURITY parameter must be set to DM_USER_PW so that a userid and a password are sent on the conversation start-up requests.

The DM_REMOTE_DOMAINS Section

This section identifies the known set of remote domains and their characteristics.

Entries have the form:

RDOM required parameters

where *RDOM* is an *identifier* value used to identify each remote domain known to this configuration. *RDOM* must be unique within the configuration.

The following parameters are required:

TYPE = *identifier*

is used for grouping remote domain into classes. TYPE can be set to one of the following values: TDOMAIN, OSITP or SNAX. The TDOMAIN value indicates that this remote domain can only communicate with another TUXEDO System/Domain Domain. The OSITP value indicates that this remote domain communicates with another TP domain via the OSI-TP protocol. The SNAX value indicates that this remote domain communicates with an MVS/CICS region via the LU6.2 protocol.

DOMAINID = *string*

is used to identify a remote domain. DOMAINID must be 32 octets or fewer in length. If the value is a string, it must be 32 characters or fewer (counting the trailing null). DOMAINID must be unique across remote domains. The value of *string* can be a sequence of characters or a sequence of hexadecimal digits preceded by “0x”.

There are no optional parameters for this section.

The DM_TDOMAIN Section

This section defines the addressing information required by domains of type TDOMAIN. This section should have an entry per local domain if requests from remote domains to local services are accepted on that local domain (gateway group), and an entry per remote domain accessible by the defined local domains.

Entries have the form:

DOM required parameters [optional parameters]

where *DOM* is an *identifier* value used to identify either a local domain (LDOM) or a remote domain (RDOM) in the *DM_LOCAL_DOMAINS section or in the *DM_REMOTE_DOMAINS section. The *DOM* identifier must match a previously defined *LDOM* in the *DM_LOCAL_DOMAINS sections or *RDOM* in the *DM_REMOTE_DOMAINS section.

The following parameter is required:

NWADDR = *string*

This parameter specifies the network address used by a local or a remote domain to accept connections from other TUXEDO System/Domain Domains. If *string* has the form “0xhex-digits”, it must contain an even number of valid hexadecimal digits.

The following parameter is optional:

NWDEVICE = *string*

Specifies the device file name to be used when binding to the listening address of a local or a remote domain. If the networking functionality is TLI-based, the device name must be an absolute pathname. If the networking functionality is Sockets-based, this parameter does not need to be specified.

NWIDLETIME = numeric

This parameter specifies the maximum time allowed for a connection to be idle (that is, unused). When this time is reached, the idle connection is be terminated. The numeric value represents a time in minutes. If this keyword is not specified, then idle connections will be maintained until the gateway handling the connection is shutdown.

Notice that multiple entries for a particular domain may be defined in this table. Multiple addresses specified for a remote domain mean that the first address (the first entry in the table for the remote domain) should be used to establish the connection and the other addresses should be used as back-up addresses in case of failure of the connection setup to the first address. Multiple addresses specified for a local domain mean that multiple listening ports are available on the same or different types of networks.

The DM_OSITP Section

This section defines the addressing information required by domains of type OSITP. This section should have one entry per gateway group (local domain), and one entry per remote domain of type OSITP.

Entries have the form:

DOM required parameters [optional parameters]

where *DOM* is an *identifier* value used to identify a local domain (LDM) or a remote domain (RDM) in the *DM_LOCAL_DOMAINS section or in the *DM_REMOTE_DOMAINS section. The *DOM* identifier must match a previously defined *LDM* in the *DM_LOCAL_DOMAINS sections or *RDM* in the *DM_REMOTE_DOMAINS section.

The following are required parameters:

APT = *string*

This parameter specifies an OSI Application Process Title (APT). An APT may be a name (i.e., the Directory Name of an Application Process Title) or an object identifier (i.e., a sequence of integer values separated by periods).

AEQ = *string*

This parameter specifies an OSI Application Entity Qualifier (AEQ). An AEQ may be a name (i.e., the relative distinguished name of a particular Application Entity) or an integer (i.e., if the APT is an object identifier).

NWDEVICE = *string*

Specifies the device name to be used when binding to an XAP dialogue instance. It may be an absolute pathname of a device filename or just an identifier of a device. The value of the string is specific to the XAP-TP provider.

The following are optional parameters:

AET = *string*

This parameter specifies an OSI Application Entity Title (AET). An AET is formed from an Application Process Title (APT) and an Application Entity Qualifier (AEQ), i.e. in ASN.1 AET is defined as a SEQUENCE { APT, AEQ } where APT and AET are of type ANY. Three main formats are accepted for the value of *string*:

encoded string

This is a single value as a hexadecimal octet string which a represents a valid BER encoding of the AET, e.g. AET = "0x06062B80CE0F0107".

{object identifier}, {integer}

The first element represents the APT defined as an object identifier (i.e., a sequence of integer values separated by periods) and the second element represents an AEQ defined as an integer constant, e.g., AET = "{1.3.15.0.3},{1}".

{string}, {string}

This format allows the APT and the AEQ to be defined as string constants, e.g., AET = "{BA.CENTRAL01},{TUXEDO}".

ACN = {XATMI / UDT}

This parameter specifies the object identifier of the Application Context Name (ACN) used by this domain. Current allowed application contexts are: the XATMI-ASE (XATMI) and the UDT-ASE (UDT). If this parameter is not specified, the ACN is set to the object identifier of the XATMI-ASE Application Context.

APID = *integer*

This parameter specifies an OSI Application Process Invocation Identifier (APID).

AEID = *integer*

This parameter specifies an OSI Application Entity Invocation Identifier (AEID).

PROFILE = *identifier*

This parameter specifies the OSI TP profile used by this domain and is used to determine the required OSI TP functional units. PROFILE can be set to one of the following values: ATP11, ATP21, ATP31, ATP12, ATP22, and ATP32. The UDT ASE application context allows the use of any of these profiles. The XATMI-ASE application context only allows profiles ATP11, ATP21 and ATP31. Profiles ATP11, ATP21 and ATP31 use the Dialogue, Polarized Control and Handshake functional units. Profiles ATP12, ATP22 and ATP32 use the Dialogue, Shared Control, and Handshake functional units. Profiles ATP11 and ATP12 do not use OSI TP transactions (the Commit functional unit is not used). Profiles ATP21 and ATP22 require the Commit, Unchained Transactions, and Recovery functional units. Profiles ATP31 and ATP32 require the Commit, Chained Transactions, and Recovery functional units. By default, the ATP21 profile is always selected.

URCH = *string*

This parameter specifies the user portion of the OSITP Recovery Context Handle. It may be required by the XAP-TP provider in order to perform recovery of distributed transactions after a communications line or system failure.

The DM_SNACRM Section

The DM_SNACRM section provides three (3) keywords used to identify the SNA Communications Resource Manager that will provide ATMI transaction semantics between a given domain and it's partners. Entries have the general form:

<SNA CommunicationsResourceManagerName> parameters

Where <SNA CommunicationsResourceManagerName> is the locally known name of this SNACRM definition to be used when referencing this SNACRM in subsequent sections. This name is an ASCII string 1 to 30 characters in length. The parameters are the keyword/value pairs that makeup the definition. All keywords are required for a valid SNACRM definition. Keywords can be in any order.

LDOM <LocalDomainName>

LDOM associates this SNACRM with a defined local domain.

<LocalDomainName> is the reference to an entry in the DM_LOCAL_DOMAINS section. This name is an ASCII string 1 to 30 characters in length. This parameter is required. This parameter has no default.

NWDEVICE <DeviceName>

<DeviceName> is the logical name used to access the network, for example /dev/tcp.

SNACRMADDR <HexSocketAddress>

SNACRMADDR provides the socket address the domain gateway will use to communicate with the SNACRM. If the SNACRM is started independent of the domain gateway this address must be used on the SNACRM command line.

<HexSocketAddress> is a TCP/IP address using the sockaddr_in format of family,port,address:

<0xFFFFPPPPAAAAAAAA>

FFFF is the hex value of the protocol family, always 0x0002 for the INET family.

PPPP is the hex value of an unused TCP/IP port

AAAAAAAA is the hex value of the IP address for the machine running the SNACRM

Therefore if the SNACRM was running on a machine with an IP address of 206.189.43.13, and we wanted to use port 6000 for the SNACRM then SNACRMADDR would be:

0x00021770CEBD2B0D

This parameter is required. This parameter must contain an even number of hex characters. This parameter has no default.

The DM_SNASTACKS Section

The DM_SNASTACKS section provides five (5) keywords which identify the third party SNA stack that should be used for connections established between a given domain and its partners. Entries have the general form:

<StackReference> parameters

Where <StackReference> is the locally known name of this stack definition to be used when referencing this stack in subsequent sections. This name is an ASCII string 1 to 30 characters in length. The parameters are the keyword/value pairs that makeup the definition. All keywords are required for a valid stack definition. Keywords can be in any order.

LOCALLU <LocalLUAlias>

LOCALLU provides a reference to an LU alias defined in the third party SNA stack. <LocalLUAlias> is the name used to identify the local LU definition as specified by the third party SNA stack configuration. This is a name that represents the end node for an LU6.2 connection. The value for this parameter is an ASCII string, 1 to 64 characters in length. This parameter is required. This parameter has no default. The third party SNA stack will require a corresponding definition for a local LU.

LTPNAME <LocalTransactionProgramName>

LTPNAME identifies the inbound transaction programs which will be serviced by any SNACRM using this stack definition. <LocalTransactionProgramName> is the name used to identify inbound transaction programs for which an attach will be accepted. The only useful value is an asterisk. This indicates all inbound attaches will be accepted. This parameter is required. This parameter has no default. Partial TP names are not supported. The third party SNA stack will require a corresponding definition for inbound TP names.

SNACRM <SNACommunicationsResourceMangerName>

SNACRM provides a name by which to reference the associated SNACRM definition. <SNACommunicationsResourceMangerName> is the name used to associate the DM_SNACRM definition with this DM_SNASTACKS entry. The value for this parameter is an ASCII string, 1 to 32 characters in length. This parameter is required. This parameter has no default.

STACKPARMS <parameters required for third party sna stack>

STACKPARMS provides a method for the domain gateway to pass any required parameters to the third party SNA stack. <parameters required for third party sna stack> is an ASCII string, 1 to 128 characters in length. Currently, the only value used is the TCP/IP hostname for the machine running the third party SNA stack. This parameter is required. This parameter has no default.

STACKTYPE={sun91 | hp51 | ibm51}

This option is used to indicate which vendor SNA stack is being used. It is also used to determine the name of specific BEA Connect system libraries. It is essential that the value of this option be coded correctly. These values are mapped to the equivalent BEA Connect system library.

The DM_SNALINKS Section

This section defines the SNA Link information required by domains of type SNA. Entries have the form:

LINK parameters

Where *LINK* is an *identifier* value used to identify a connection between a local domain (LDOM) and a remote domain (RDOM). The *RDOM* identifier must match a previously defined *RDOM* in the *DM_REMOTE_DOMAINS section.

The following parameters are available:

STACKREF = *string*

This required parameter defines the stack that will be used for establishment of this link. The STACKREF string is the tag that was used in a previous definition established in the *DM_SNASTACKS section.

RDOM = *string*

The RDOM string should match a previous RDOM definition in the *DM_REMOTE_DOMAINS section.

LSYSID = *string*

Reserved for future use. LSYID is the 4 character identifier that is to be used for this link. This should match the connection ID used by a partner CICS to communicate to the SNACRM across this link.

RSYSID = *string*

Reserved for future use. RSYID is the 4 character remote sysid of the partner. Typically it is the sysid of a CICS region, but could also be the subsystem id of an IMS control region. This parameter should match the actual sysid of the remote partner.

RLUNAME = *string*

The RLUNAME value represents an alias known to the third party SNA stack that resolves to a VTAM netname for the remote application. This would most likely be the VTAM applid for a CICS region, however it could also be an APPC/MVS LU defined for use with IMS. The value must be unique within the SNA network. *string* should be from 1 to 8 characters. This parameter is required. This parameter has no default. The third party stack configuration requires a matching definition.

MODENAME = *string*

MODENAME is VTAM mode entry, defined to the third party SNA stack, to be used for this link. For a CICS link this must be compatible with the RDO session definition for the corresponding connection. For an IMS connection this must be compatible with the DLOGMOD entry on the LU definition used to access the IMS scheduler. *string* should be from 1 to 8 ASCII characters. This parameter is required. This parameter must match the third party SNA stack configuration and must be compatible with the corresponding entries defined to VTAM and/or CICS.

SECURITY = *string*

SECURITY_TYPE specifies the security setting in CICS/RACF or partner. Legal values are LOCAL, IDENTIFY, VERIFY, PERSISTENT or MIXIDPE. *string* should be from 1 to 10 characters. The default setting is LOCAL.

MAXSESS = number

MAXSESS is the maximum number of parallel sessions that can be started on this link. MAXSESS must be greater than or equal to four.

MINWIN = number

The minimum number of contention winners. This value is typically half the MAXSESS value.

MAXSYNCLVL = number

This value represents the maximum transaction synchronization level that can be supported over this link.

A value of zero (0) means this link is non-transactional. No synchronization will be maintained. This can be used for sending and receiving messages from IMS via the APPC/MVS transparency interface. The default sync-level is sync-level 0.

A value of one (1) means this link will support everything supported with zero (0), in addition to:

Outbound ATMI tpcall() as a CICS distributed program link request with the semantics of SYNCONRETURN.

Inbound EXEC CICS LINK requests with the semantics of SYNCONRETURN. The program name must match the RNAME on the local service definition and the SYSID must match the LSYSID for the link.

A value of two (2) means this link will support everything supported with zero (0) and one (1) for partners able to exchange logs and compare states, in addition to:

The exchange logs and compare states function with a partner CICS.
Outbound ATMI tpcall() as a CICS distributed program link request with full two phase commit transaction semantics using tpcommit().
Outbound ATMI tpconnect() as APPC or CPIC distributed transaction processing with full two phase commit transaction semantics using tpcommit().

Inbound EXEC CICS LINK requests with full two phase commit transaction semantics using Prepare Rollback and Syncpoint verbs.

Inbound APPC or CPIC conversations with full two phase commit transaction semantics using Prepare Rollback and Syncpoint verbs.

The partner must be able to negotiate a CICS style exchange logs and compare states for successful initialization of a sync-level 2 link.

If the installation is not licensed for sync-level 2, this parameter must be set to 0 or 1 for the link to be established. Transaction support is only available at sync-level 2. Distributed Program Link can be accessed as SYNCONRETURN, that is, not transactional if the link sync-level is 1.

Caution: If you set MAXSYNCLVL=2 or make no entry for this parameter (that is, accept the default) without having installed the Connect SNA software licensed for that level, the system configuration automatically reverts to Sync-level 1 and an error message is sent to the error log. To clear that error message, you must either reset the MAXSYNCLVL parameter to an appropriate value or purchase and install the correct software.

STARTTYPE = {warm | cold}

This option sets the recovery mode for transactional links. When set to WARM, the system restarts using configuration and link data recovered from the in-flight transaction log. When set to COLD, the system uses configuration data taken from the current dmconfig file and loses any in-flight link data. Changing dmconfig file parameters and performing a WARM start results in a message warning that changed parameters are ignored until the next cold start. To force a cold start and disregard the STARTTYPE setting, delete the SNA*LOG files in \$APPDIR.

The DM_ACCESS_CONTROL Section

This section specifies the access control lists used by local domain. Lines in this section are of the form:

ACL_NAME required parameters

where *ACL_NAME* is a (*identifier*) name used to identify a particular access control list; it must be 15 characters or less in length.

Required parameters are:

ACLIST = *identifier* [,*identifier*]

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

The DM_LOCAL_SERVICES Section

This section provides information on the services exported by each local domain. This section is optional and if it is not specified then all local domains defined in the *DM_LOCAL_DOMAINS section accept requests to all of the services advertised by the TUXEDO System/Domain application. If this section is defined then it should be used to restrict the set of local services that can be requested from a remote domain.

Lines within this section have the form:

service [optional parameters]

where *service* is the (*identifier*) local name of the exported service, and it must be 1-15 characters in length. This name corresponds to a name advertised by one or more servers running with the local TUXEDO System/Domain application. Notice that exported services inherit the default or special properties specified for the service in an entry in the SERVICES section of the TUXCONFIG file. Some of these parameters are: LOAD, PRIO, AUTOTRAN, ROUTING, BUFTYPE, and TRANTIME.

Optional parameters are:

ACL = *identifier*

specifies the name of the access control list (ACL) to be used by the local domain to restrict requests made to this service by remote domains. The name of the ACL is defined in the *DM_ACCESS_CONTROL section. If this parameter is not specified then access control will not be performed for requests to this service.

CONV = { Y | N }

specifies whether (Y) or not (N) the remote service is a conversational service. The default value is N.

LDOM = *identifier*

specifies the name identifying the local domain exporting this service. If this keyword is not specified, then the first local domain entry in the *DM_LOCAL_DOMAINS section accepts requests for this local service.

INBUFTYPE = *type[:subtype]*

restricts the buffer type naming space of data types accepted by this service to a single buffer type. This parameter should be defined when the service is going to be used from an OSITP type gateway that uses the UDT ASE Application Context. For /SNA Domain buffer types, see the discussion in the DM_REMOTE_SERVICES section below.

OUTBUFTYPE = *type[:subtype]*

restricts the buffer type naming space of data types returned by this service to a single buffer type. This parameter should be defined when the service is going to be used from an OSITP type gateway that uses the UDT ASE Application Context. The FML buffer type cannot be used for OSITP type gateways. For /SNA Domain buffer types, see the discussion in the DM_REMOTE_SERVICES section below.

RNAME = *string*

specifies the name exported to remote domains. This name will be used by the remote domains for request to this service. If this parameter is not specified, the local service name is supposed to be the name used by any remote domain.

The DM_REMOTE_SERVICES Section

This section provides information on services “imported” and available on remote domains. Lines within this *DM_REMOTE_SERVICES section have the form:

service [optional parameters]

where *service* is the (*identifier*) name used by the local TUXEDO System/Domain application for a particular remote service. Remote services are associated with a particular remote domain.

Optional Parameters are:

AUTOTRAN = { Y | N }

specifies whether or not a transaction should automatically be started if a request message is received that is not already in transaction mode. The default is N.

BLOCKTIME = numeric

specifies the maximum wait time allowed for a reply to this remote service. The value sets a multiplier of the SCANUNIT parameters specified in the TUXCONFIG file. The value SCANUNIT * BLOCKTIME must be greater than or equal to SCANUNIT and less than 32,768 seconds. A time-out always implies a failure of the affected transaction or request.

CONV = { Y | N }

specifies whether (Y) or not (N) the remote service is a conversational service. The default value is N.

FUNCTION = { APPC|DPL }

enables outbound Tuxedo service requests to map to APPC transaction programs or CICS programs. The default value APPC indicates the remote service is a transaction program that may or may not be running under CICS. The DPL value indicates the remote service maps to a program running under CICS.

LDOM = *identifier*

specifies the name of a local domain in charge of routing requests to this remote service. The gateway group associated with the local domain advertises *service* in the TUXEDO System/Domain Bulletin Board. If this parameter is not specified then all the local domains will be able to accept requests to this remote service. The service request will be then redirected to a remote domain of the same type (see RDOM keyword below).

LOAD = integer

specifies that the remote service imposes a load of integer units. The value of LOAD can be between 1 and 32767 inclusive. If not specified, the default is 50. A higher number indicates a greater load.

INBUFTYPE = *type[:subtype]*

restricts the buffer type naming space of data types accepted by this service to a single buffer type. This parameter should be defined when the service is going to be used from an OSITP type gateway that uses the UDT ASE Application Context. The FML buffer type cannot be used for OSITP type gateways.

OUTBUFTYPE = *type[:subtype]*

restricts the buffer type naming space of data types returned by this service to a single buffer type. This parameter should be defined when the service is going to be used from an OSITP type gateway that uses the UDT ASE Application Context. The FML buffer type cannot be used for OSITP type gateways.

PRIO = integer

specifies the dequeing priority of service requests to this remote service. The value of PRIO must be greater than 0 and less than or equal to 100, with 100 being the highest priority. The default is 50.

RDOM = *identifier*

specifies the name of the remote domain responsible for the actual execution of this service. If this parameter is not specified and a routing criteria (see below ROUTING keyword) is not specified, then the local domain assumes that any remote domain of the same type accepts this service and it selects a known domain (a domain to which a connection already exists) or remote domain from the `**DM_REMOTE_DOMAINS` section.

RNAME = *string*

specifies the actual service name expected by the remote domain. If this parameter is not specified, the remote service name is the same as the name specified in *service*.

The RNAME option is the name of the host TP_NAME. For non-CICS systems, this name can be up to 64 characters in length. For CICS systems, this name is the trans-id name for APPC-defined requests and the program name for DPL requests. CICS trans-id names cannot exceed four characters and CICS program names cannot exceed eight characters. The RNAME option must observe these requirements.

ROUTING = *identifier*

when more than one remote domain offers the same service, a local domain can perform data dependent routing if this optional parameter is specified. The *identifier* specifies the name of the routing criteria used for this data dependent routing. If not specified, data dependent routing is not done for this service. *identifier* must be 15 characters or less in length. If multiple entries exist for the same service name but with different RDOM parameters, the ROUTING parameter should be the same for all of these entries.

TRANTIME = *integer*

specifies the default time-out value in seconds for a transaction automatically started for the associated service. The value must be greater than or equal to 0 and less than 2147483648. The default is 30 seconds. A value of 0 implies the maximum time-out value for the machine.

The DM_ROUTING Section

This section provides information for data dependent routing of service requests using FML, VIEW, X_C_TYPE, and X_COMMON typed buffers. Lines within the DM_ROUTING section have the form:

CRITERION_NAME required parameters

where *CRITERION_NAME* is the (*identifier*) name of the routing entry that was specified on the services entry. *CRITERION_NAME* must be 15 characters or less in length.

Required parameters are:

FIELD = *identifier*

specifies the name of the routing field. It must be 30 characters or less. This field is assumed to be a field name that is identified in an FML field table (for FML buffers) or an FML view table (for VIEW, X_C_TYPE, or X_COMMON buffers). The FLDTBLDIR and FIELDTBLS environment variables are used to locate FML field tables, and the VIEWDIR and VIEWFILES environment variables are used to locate FML view tables.

RANGES = *string*

specifies the ranges and associated remote domain names (RDOM) for the routing field. *string* must be enclosed in double quotes. The format of *string* is a comma-separated ordered list of range/RDOM pairs (see EXAMPLES below).

A range is either a single value (signed numeric value or character string in single quotes), or a range of the form “lower - upper” (where lower and upper are both signed numeric values or character strings in single quotes). Note that “lower” must be less than or equal to “upper”. To embed a single quote in a character string value (as in O’Brien, for example), it must be preceded by two backslashes (‘O’Brien’). The value MIN can be used to indicate the minimum value for the data type of the associated FIELD; for strings and arrays, it is the null string; for character fields, it is 0; for numeric values, it is the minimum numeric value that can be stored in the field. The value MAX can be used to indicate the maximum value for the data type of the associated FIELD; for strings and arrays, it is effectively an unlimited string of octal-255

characters; for a character field, it is a single octal-255 character; for numeric values, it is the maximum numeric value that can be stored in the field. Thus, “MIN - -5” is all numbers less than or equal to -5 and “6 - MAX” is all numbers greater than or equal to 6. The meta-character “*” (wild-card) in the position of a range indicates any values not covered by the other ranges previously seen in the entry; only one wild-card range is allowed per entry and it should be last (ranges following it will be ignored).

The routing field can be of any data type supported in FML. A numeric routing field must have numeric range values and a string routing field must have string range values.

String range values for string, array, and character field types must be placed inside a pair of single quotes and can not be preceded by a sign. Short and long integer values are a string of digits, optionally preceded by a plus or minus sign. Floating point numbers are of the form accepted by the C compiler or atof(): an optional sign, then a string of digits optionally containing a decimal point, then an optional e or E followed by an optional sign or space, followed by an integer.

When a field value matches a range, the associated RDOM value specifies the remote domain to which the request should be routed. A RDOM value of “*” indicates that the request can go to any remote domain known by the gateway group.

Within a range/RDOM pair, the range is separated from the RDOM by a “:”.

`BUFTYPE = ~type1[:subtype1[,subtype2 . . .]][:type2[:subtype3[, . . .]]] . . . ~`

is a list of types and subtypes of data buffers for which this routing entry is valid. The types are restricted to be either FML, VIEW, X_C_TYPE, or X_COMMON. No subtype can be specified for type FML and subtypes are required for the other types (“*” is not allowed). Duplicate type/subtype pairs can not be specified for the same routing criterion name; more than one routing entry can have the same criterion name as long as the type/subtype pairs are unique. This parameter is required. If multiple buffer types are specified for a single routing entry, the data types of the routing field for each buffer type must be the same.

If the field value is not set (for FML buffers), or does not match any specific range and a wild-card range has not been specified, an error is returned to the application process that requested the execution of the remote service.

FILES

The BDMCONFIG environment variable is used to find the BDMCONFIG configuration file.

EXAMPLE 1

The following configuration file defines a 5-site domain configuration. The example shows 4 Bank Branch domains communicating with a Central Bank Branch. Three of the Bank Branches run within other TUXEDO System/Domain domains. The fourth Branch runs under the control of another TP Domain and OSI-TP is used in the communication with that domain.

```
# TUXEDO DOMAIN CONFIGURATION FILE FOR THE CENTRAL BANK
#
#
*DM_LOCAL_DOMAINS
#<local domain name> <Gateway Group name> <domain type> <domain id> <log device>
#           [<audit log>] [<blocktime>]
#           [<log name>] [<log offset>] [<log size>]
#           [<maxrdom>] [<maxrdtran>] [<maxtran>]
#           [<maxdatalen>] [<security>]
#           [<tuxconfig>] [<tuxoffset>]
#
#
DEFAULT: SECURITY = NONE

c01      GWGRP = bankg1
          TYPE = TDOMAIN
          DOMAINID = "BA.CENTRAL01"
          DMTLOGDEV = "/usr/apps/bank/DMTLOG"
          DMTLOGNAME = "DMTLG_C01"

c02      GWGRP = bankg2
          TYPE = OSITP
          DOMAINID = "BA.CENTRAL01"
          DMTLOGDEV = "/usr/apps/bank/DMTLOG"
          DMTLOGNAME = "DMTLG_C02"
          NWDEVICE = "OSITP"
          URCH = "ABCD"

#
*DM_REMOTE_DOMAINS
#<remote domain name> <domain type> <domain id>
#
```

```

b01    TYPE = TDOMAIN
        DOMAINID = "BA.BANK01"

b02    TYPE = TDOMAIN
        DOMAINID = "BA.BANK02"

b03    TYPE = TDOMAIN
        DOMAINID = "BA.BANK03"

b04    TYPE = OSITP
        DOMAINID = "BA.BANK04"
        NWDEVICE = "/dev/osi"
        URCH = "ABCD"

*DM_TDOMAIN
#
# <local or remote domain name> <network address> [<nwdevice>]
#
# Local network addresses
c01    NWADDR = "0x0002ff98c00b9d6d" NWDEVICE = "/dev/tcp"
c01    NWADDR = "newyork01.65432"    NWDEVICE = "/dev/starlan"
# Remote network addresses
b01    NWADDR = "0x00020401c00b6d05" NWDEVICE = "/dev/tcp"
b02    NWADDR = "dallas.65432" NWDEVICE = "/dev/starlan"
b03    NWADDR = "0x00021094c00b6d9c" NWDEVICE = "/dev/tcp"

*DM_OSITP
#
#<local or remote domain name> <apt> <aeq>
#
#           [<aet>] [<acn>] [<apid>] [<aeid>]
#
#           [<profile>]
#
c02    APT = "BA.CENTRAL01"
        AEQ = "TUXEDO.R.4.2.1"
        AET = "{1.3.15.0.3},{1}"
        ACN = "XATMI"

```

```
b04      APT = "BA.BANK04"
          AEQ = "TUXEDO.R.4.2.1"
          AET = "{1.3.15.0.4},{1}"
          ACN = "XATMI"
```

***DM_LOCAL_SERVICES**

```
#<service_name> [<Local Domain name>] [<access control>] [<exported svcname>]
#           [<inbuftype>] [<outbuftype>]
#
open_actACL = branch
close_actACL = branch
credit
debit
balance
loan          LDOM = c02ACL = loans
```

***DM_REMOTE_SERVICES**

```
#<service_name> [<Remote domain name>] [<local domain name>]
#           [<remote svcname>] [<routing>] [<conv>] [<trantime>]
#           [<inbuftype>] [<outbuftype>]
#
tlr_add  LDOM = c01  ROUTING = ACCOUNT
tlr_bal  LDOM = c01  ROUTING = ACCOUNT
tlr_add  RDOM = b04  LDOM = c02 RNAME ="TPSU002"
tlr_bal  RDOM = b04  LDOM = c02 RNAME ="TPSU003"
```

***DM_ROUTING**

```
# <routing criteria><field> <typed buffer> <ranges>
#
ACCOUNTFIELD = branchid  BUFTYPE ="VIEW:account"
          RANGES ="MIN - 1000:b01, 1001-3000:b02, *:b03"
```

***DM_ACCESS_CONTROL**

```
#<acl name>  <Remote domain list>
#
```

```
branch  ACLIST = b01, b02, b03
loans   ACLIST = b04
```

EXAMPLE 2

This example shows the TUXEDO System/Domain Configuration file required at one of the Bank Branches (BANK01).

```
#
#TUXEDO DOMAIN CONFIGURATION FILE FOR A BANK BRANCH
#
#
*DM_LOCAL_DOMAINS
#
b01      GWGRP = auth
        TYPE = TDOMAIN
        DOMAINID = "BA.BANK01"
        DMTLOGDEV = "/usr/apps/bank/DMTLOG"

*DM_REMOTE_DOMAINS
#
c01      TYPE = TDOMAIN
        DOMAINID = "BA.CENTRAL01"

*DM_TDOMAIN
#
b01      NWADDR = "0x00021094c00b689c" NWDEVICE = "/dev/tcp"
c01      NWADDR = "0x0002ff98c00b9d6d" NWDEVICE = "/dev/tcp"
*DM_LOCAL_SERVICES
#
tlr_add      ACL = central
tlr_bal      ACL = central

*DM_REMOTE_SERVICES
#
OPA001      RNAME = "open_act"
CLA001      RNAME = "close_act"
CRD001      RNAME = "credit"
DBT001      RNAME = "debit"
BAL001      RNAME = "balance"
```

```
*DM_ACCESS_CONTROL
#
central          ACLIST = c01
```

EXAMPLE 3

This example shows the configuration file entries for a BEA Connect SNA application:

```
#=====
# DMCONFIG
#      Application Domain Gateway Test Configuration
#
# See also
#      See $(TOP)/Makefile for more information.
#
# @(#)SNA Devel apps/simpsna DMCONFIG 1.6 98/03/03 15:35:29
# Copyright 1997, BEA Systems, Inc., all rights reserved.
#-----

*DM_LOCAL_DOMAINS
simpsnad
    GWGRP=GROUP2
    TYPE=SNAX
    DOMAINID="simpsnad"
    BLOB_SHM_SIZE=1000000
    DMTLOGDEV=<your TUXEDO filesystem device and name for
    DMTLOG>

#example DMTLOGDEV="/home/me/bin/DMTLOG"

*DM_REMOTE_DOMAINS

SIMPSNAG TYPE=SNAX DOMAINID="SIMPSNAG"

*DM_SNACRM

simpercm      SNACRMADDR="<your Host Socket Listen Address>"
              NWDEVICE="/dev/tcp"
              LDOM="simpsnad"

#example SNACRMADDR="0x00021770cfbd2b0d" INET family 0x0002 port 6000 host
```

207.189.43.13

*DM_SNASTACKS

simpstk

```

SNACRM="simperm"
STACKTYPE=<SNACRM Stack Library Named Token>
LOCALLU=<Local LU definition specified in
stack product>
LTPNAME="*"
STACKPARMS=<Parameters passed to Stack
Product>

```

```
#example STACKTYPE="HP51"
```

```
# LOCALLU="HPTEST"
```

```
# STACKPARMS="testhp" Name of the host machine
```

*DM_SNALINKS

simplk1

```

STACKREF="simpstk"
RDOM="SIMPSNAG"
LSYSID=<Connection ID of remote (CICS)
region>
RSYSID=<SYSID of remote (CICS) region>
RLUNAME=<Alias of Applid for remote region>
MODENAME=<Mode name VTAM mode entry>
SECURITY="LOCAL"
STARTTYPE="COLD"
MAXSESS=<Total Session number>
MINWIN=<Session Local Winners>
MAXSYNCLVL=<0|1|2 Maximum Syncpoint Level>

```

```
#example LSYSID="BEA"
```

```
# RSYSID="TEST"
```

```
# RLUNAME="CICSTEST"
```

```
# MODENAME="SMSNA100"
```

```
# MAXSESS=10
```

```
# MINWIN=5
```

```
# MAXSYNCLVL=2
```

*DM_LOCAL_SERVICES

```
MIRROR LDOM="simpsnad"
      CONV=N
      RNAME="MIRRORSESV"
      INBUFTYPE="STRING"
      OUTBUFTYPE="STRING"
      API="ATMI"
```

```
*DM_REMOTE_SERVICES
```

```
SIMPDPLAUTOTRAN=N
      LDOM="simpsnad"
      RDOM=SIMPSNAG
      CONV=N
      RNAME="TOUPDPLS"
      INBUFTYPE="STRING"
      OUTBUFTYPE="STRING"
      API="ATMI"
      FUNCTION="DPL"
```

```
SIMPDTPAUTOTRAN=N
      LDOM="simpsnad"
      RDOM=SIMPSNAG
      CONV=N
      RNAME="DTPS"
      INBUFTYPE="STRING"
      OUTBUFTYPE="STRING"
      API="ATMI"
      FUNCTION="APPC"
```

```
*DM_ROUTING
```

```
#example SNACRMADDR="0x00021770cfbd2b0d" INET family 0x0002 port 6000 host
207.189.43.13
```

```
*DM_SNASTACKS
```

```
simpstk
      SNACRM="simperm"
      STACKTYPE=<SNACRM Stack Library Named Token>
```

```

LOCALLU=<Local LU definition specified in
stack product>
LTPNAME="*"
STACKPARMS=<Parameters passed to Stack
Product>

```

```

#example STACKTYPE="HP51"
#   LOCALLU="HPTEST"
#   STACKPARMS="testhp" Name of the host machine

```

```
*DM_SNALINKS
```

```

simplk1  STACKREF="simpstk"
         RDOM="SIMPSNAG"
         LSYSID=<Connection ID of remote (CICS)
         region>
         RSYSID=<SYSID of remote (CICS) region>
         RLUNAME=<Alias of Applid for remote region>
         MODENAME=<Mode name VTAM mode entry>
         SECURITY="LOCAL"
         STARTTYPE="COLD"
         MAXSESS=<Total Session number>
         MINWIN=<Session Local Winners>
         MAXSYNCLVL=<0|1|2 Maximum Syncpoint Level>

```

```

#example LSYSID="BEA"
#   RSYSID="TEST"
#   RLUNAME="CICSTEST"
#   MODENAME="SMSNA100"
#   MAXSESS=10
#   MINWIN=5
#   MAXSYNCLVL=2

```

```
*DM_LOCAL_SERVICES
```

```

MIRROR LDOM="simpsnad"
        CONV=N
        RNAME="MIRRORSESV"
        INBUFTYPE="STRING"
        OUTBUFTYPE="STRING"
        API="ATMI"

```

*DM_REMOTE_SERVICES

SIMPDPL AUTOTRAN=N
 LDOM="simpsnad"
 RDOM=SIMPSNAG
 CONV=N
 RNAME="TOUPDPLS"
 INBUFTYPE="STRING"
 OUTBUFTYPE="STRING"
 API="ATMI"
 FUNCTION="DPL"

SIMPDTP AUTOTRAN=N
 LDOM="simpsnad"
 RDOM=SIMPSNAG
 CONV=N
 RNAME="DTPS"
 INBUFTYPE="STRING"
 OUTBUFTYPE="STRING"
 API="ATMI"
 FUNCTION="APPC"

*DM_ROUTING

SEE ALSO

build_dgw(1), dmadmin(1), tmboot(1), tmshutdown(1), dmloadcf(1), dmunloadcf(1)

dmgwopts(5), GWADM(5), DMADM(5)

TUXEDO /Domain User Guide

TUXEDO Administrator's Guide

TUXEDO Programmer's Guide

dmloadcf

Parse a DMCONFIG file and load binary BDMCONFIG configuration file

SYNOPSIS

```
dmloadcf [-c] [-n] [-y] [-b blocks] {dmconfig_file | - }
```

DESCRIPTION

dmloadcf reads a file or the standard input that is in DMCONFIG syntax, checks the syntax, and optionally loads a binary BDMCONFIG configuration file. The BDMCONFIG environment variable points to the path name of the BDMCONFIG file where the information should be stored.

dmloadcf prints an error message if it finds any required section of the DMCONFIG file missing. If a syntax error is found while parsing the input file, dmloadcf exits without performing any updates to the BDMCONFIG file.

dmloadcf requires the existence of the \$TUXDIR/udataobj/DMTYPE file. This file defines the valid domain types. If this file does not exist, *dmloadcf* exits without performing any updates to the BDMCONFIG file.

The effective user identifier of the person running dmloadcf must match the UID in the RESOURCES section of the TUXCONFIG file.

The -c option to dmloadcf causes the program to print minimum IPC resources needed for each local domain (gateway group) in this configuration. The BDMCONFIG file is not updated.

The -n option to dmloadcf causes the program to do only syntax checking of the ASCII DMCONFIG file without actually updating the BDMCONFIG file.

After syntax checking, dmloadcf checks to see if the file pointed to by BDMCONFIG exists, is a valid TUXEDO System file system, and contains BDMCONFIG tables. If these conditions are not true, the user is prompted to create and initialize the file with

Initialize BDMCONFIG file: *path* [y, q]?

where *path* is the complete file name of the BDMCONFIG file. Prompting is suppressed if the standard input or output are not terminals, or if the -y option is specified on the command line. Any response other than “y” or “Y” will cause dmloadcf to exit without creating the configuration file.

If the BDMCONFIG file is not properly initialized, and the user has given the go-ahead, `dmloadcf` creates the TUXEDO file system and then creates the BDMCONFIG tables. If the `-b` option is specified on the command line, its argument is used as the number of blocks for the device when creating the TUXEDO file system. If the value of the `-b` option is large enough to hold the new BDMCONFIG tables, `dmloadcf` will use the specified value to create the new file system; otherwise, `dmloadcf` will print an error message and exit. If the `-b` option is not specified, `dmloadcf` will create a new file system large enough to hold the BDMCONFIG tables. The `-b` option is ignored if the file system already exists. The `-b` option is highly recommended if BDMCONFIG is a raw device (that has not been initialized) and should be set to the number of blocks on the raw device. The `-b` option is not recommended if BDMCONFIG is a regular UNIX file.

If the BDMCONFIG file is determined to already have been initialized, `dmloadcf` ensures that the local domain described by that BDMCONFIG file is not running. If a local domain is running, `dmloadcf` prints an error message and exits. Otherwise, `dmloadcf`, to confirm that the file should be overwritten, prompts the user with:

“Really overwrite BDMCONFIG file [y, q]?”

Prompting is suppressed if the standard input or output are not a terminal or if the `-y` option is specified on the command line. Any response other than “y” or “Y” will cause `dmloadcf` to exit without overwriting the file.

If the SECURITY parameter is specified in the RESOURCES section of the TUXCONFIG file, then `dmloadcf` will flush the standard input, turn off terminal echo and prompt the user for an application password as follows:

Enter Application Password?

The password is truncated to 8 characters. The option to load the ASCII DMCONFIG file via the standard input (rather than a file) cannot be used when this SECURITY parameter is turned on. If the standard input is not a terminal, that is, if the user cannot be prompted for a password (as with a here file, for example), then the environment variable `APP_PW` is accessed to set the application password. If the environment variable `APP_PW` is not set with the standard input not a terminal, then `dmloadcf` will print an error message, generate a log message and fail to load the BDMCONFIG file.

Assuming no errors, and if all checks have passed, `dmloadcf` loads the DMCONFIG file into the BDMCONFIG file. It will overwrite all existing information found in the BDMCONFIG tables.

PORTABILITY

dmloadcf is supported as a TUXEDO-supplied administrative tool on UNIX operating systems only.

ENVIRONMENT VARIABLES

The environment variable APP_PW must be set for applications that require security (the SECURITY parameter in the TUXCONFIG file is set to APP_PW) and dmloadcf is run with something other than a terminal as the standard input.

The BDMCONFIG environment variable should point to the BDMCONFIG file.

EXAMPLES

The following example shows how a binary configuration file is loaded from the bank.dmconfig ASCII file. The BDMCONFIG device is created (or re-initialized) with 2000 blocks:

```
dmloadcf -b 2000 -y bank.dmconfig
```

DIAGNOSTICS

If an error is detected in the input, the offending line is printed to standard error along with a message indicating the problem. If a syntax error is found in the DMCONFIG file or the system is currently running, no information is updated in the BDMCONFIG file and dmloadcf exits with exit code 1.

If dmloadcf is run on an active node, the following error message is displayed:

```
*** dmloadcf cannot run on an active node ***
```

If dmloadcf is run by a person whose effective user identifier doesn't match the UID specified in the TUXCONFIG file, the following error message is displayed:

```
*** UID is not effective user ID ***
```

Upon successful completion, dmloadcf exits with exit code 0. If the BDMCONFIG file is updated, a userlog message is generated to record this event.

SEE ALSO

dmunloadcf(1), dmconfig(5), ubbconfig(5)

TUXEDO /Domain User Guide

TUXEDO Administrator's Guide

dmunloadcf

Unload binary BDMCONFIG domain configuration file

SYNOPSIS

dmunloadcf

DESCRIPTION

dmunloadcf translates the BDMCONFIG configuration file from the binary representation into ASCII. This translation is useful for transporting the file in a compact way between machines with different byte ordering and backing up a copy of the file in a compact form for reliability. The ASCII format is the same as is described in dmconfig(5).

dmunloadcf reads values from the BDMCONFIG file pointed to by the BDMCONFIG environment variable and writes them to its standard output.

PORTABILITY

dmunloadcf is supported as a TUXEDO-supplied administrative tool on UNIX operating systems only.

EXAMPLES

To unload the configuration in /usr/tuxedo/BDMCONFIG into the file bdmconfig.backup:

```
BDMCONFIG=/usr/tuxedo/BDMCONFIG dmunloadcf > bdmconfig.backup
```

DIAGNOSTICS

dmunloadcf checks that the file pointed to by the BDMCONFIG environment variable exists, is a valid TUXEDO file system, and contains BDMCONFIG tables. If any of these conditions is not met, dmunloadcf prints an error message and exits with error code 1. Upon successful completion, dmunloadcf exits with exit code 0.

SEE ALSO

dmloadcf(1), dmconfig(5)

TUXEDO /Domain User Guide

dmusradd

Add a user to the domain password file

SYNOPSIS

```
DMCONFIG=dmconfig
dmusradd [ -ld local domain ID ][ -rd remote domain ID ]
[ -lu local uid ] [ -lp local password ]
[ -ru remote username ] [-rp remote password]
```

DESCRIPTION

This command allows the administrator to add local userids and passwords and corresponding remote userids and passwords to the domain password table.

The entries created are used for mapping remote user names and passwords to local user names and passwords when the application is using /SNA Domain gateways and SECURITY is set to USER_AUTH, ACL, or MANDATORY ACL in the ubbconfig file and SECURITY is set to DM_PW or USER_PW in the DMCONFIG file.

The system file entries created with this command have a limit of 512 characters per line. Specifying long arguments to several options may exceed this limit.

The following options are available:

-ld *local domain ID*

This is the domainID of the local domain gateway with which the ids and passwords are associated. This is the same ID as the one used when creating the domain definitions either in the DMCONFIG file or through the Graphical Administrative Interface.

-rd *remote domain ID*

This is the domainID of the remote domain gateway with which the ids and passwords are associated. This is the same ID as the one used when creating the domain definitions either in the DMCONFIG file or through the Graphical Administrative Interface.

-lu *local uid*

The user identification number. The local *uid* must be a positive decimal number below 128K. *uid* must be unique within the list of existing identifiers for the application. *uid* defaults to the next available (unique) identifier greater than 0. The wildcard character ('*') is a valid uid.

-lp local password

The local password that is associated with the local uid to be added.

-ru remote username

The name of the remote user as defined in the remote domain's security application, for example, RACF. The wildcard character ('*') and space are valid remote usernames.

-rp remote password

The remote password that is associated with the local uid to be added.

The administrator is prompted for the local password and for the remote password if not specified on the command line.

Before running this command, the application must be configured using either the Graphical Administrative Interface or `tmloadcf(1)` and `dmloadcf(1)`. `dmusradd` may be run on any active node. If the application is not active, `dmusradd` must be run on the MASTER node.

PORTABILITY

This command is available only on non-/WS sites running TUXEDO System /T Release 6.4 or later.

DIAGNOSTICS

The `dmusradd` command exits with a return code of 0 upon successful completion.

EXAMPLES

1. Associate uid 408 with theirname in CICS security
`dmusradd -ld ldom -rd cics -lu 408 -ru theirname`
2. Associate uid 409 with any remote user
`dmusradd -ld ldom -rd cics -lu 409 -ru '*'`
3. Associate any local user with remote theirname
`dmusradd -ld ldom -rd cics -lu '*' -ru theirname`

SEE ALSO

`dmusrdel(1)`, `dmusrmod(1)`,

dmusrmod

Change a user password in the remote domain password file

SYNOPSIS

```
DMCONFIG=dmconfig
dmusrmod [ -ld local domain ID ][ -rd remote domain ID ]
[ -lu local uid ] [ -ru remote username ] [ -rp remote password ]
```

DESCRIPTION

This command allows the administrator to modify the username and password of remote domain users in the corresponding remote domain password file.

Once the entry is modified, the old information can no longer be used for mapping remote user names and passwords to local user names and passwords when the application is using /SNA Domain gateways and SECURITY is set to USER_AUTH, ACL, or MANDATORY ACL in the ubbconfig file and SECURITY is set to DM_PW or USER_PW in the DMCONFIG file.

The following options are available:

-ld local domain ID

This is the domainID of the local domain gateway with which the ids and passwords are associated. This is the same ID as the one used when creating the domain definitions either in the DMCONFIG file or through the Graphical Administrative Interface.

-rd remote domain ID

This is the domainID of the remote domain gateway with which the ids and passwords are associated. This is the same ID as the one used when creating the domain definitions either in the DMCONFIG file or through the Graphical Administrative Interface.

-lu local uid

The user identification number. The local *uid* must be a positive decimal number below 128K. *uid* must be unique within the list of existing identifiers for the application. *uid* defaults to the next available (unique) identifier greater than 0. The wildcard character ('*') is a valid uid.

-ru remote username

The name of the remote user that is associated with the local uid to be modified.

-rp remote password

The remote password that is associated with the local uid to be modified.

Before running this command, the application must be configured using either the Graphical Administrative Interface or `tmloadcf(1)` and `dmloadcf(1)`. `dmusrmod` may be run on any active node. If the application is not active, `dmusrmod` must be run on the MASTER node.

PORTABILITY

This command is available only on non-/WS sites running TUXEDO System /T Release 6.4.

DIAGNOSTICS

The `dmusrmod` command exits with a return code of 0 upon successful completion.

EXAMPLES

Change uid 408

```
dmusrmod -ld ldom -rd cics -lu 408 -ru newname -rp*****
```

SEE ALSO

`dmusrdel(1)`, `dmusradd(1)`,

GWADM

/Domain gateway administrative server

SYNOPSIS

```
GWADM SRVGRP = "identifier" SRVID = "number" REPLYQ = "N"  
CLOPT = "-A -- [-a { on | off } ] [-s services ]  
[-t { on | off } ]"
```

DESCRIPTION

The gateway administrative server (GWADM) is a TUXEDO-supplied server that provides administrative functions for a /Domain gateway group.

GWADM should be defined in the *SERVERS section of the UBBCONFIG file as a server running within a particular gateway group, that is, SRVGRP must be set to the corresponding GRPNAME tag specified in the *GROUPS section. The SVRID parameter is also required and its value must consider the maximum number of gateways allowed within the gateway group.

There should be only one instance of a GWADM per /Domain gateway group, and it should NOT be part of the MSSQ defined for the gateways associated with the group. Also, GWADM should have the REPLYQ attribute set to N.

The CLOPT option is a string of command line options that is passed to the GWADM when it is booted. This string has the following format:

```
CLOPT="-A -- <gateway group runtime parameters>"
```

The following runtime parameters are recognized for a gateway group:

```
-a { on | off }
```

This option turns off or on the audit log feature for this local domain. The default is off. The dmadm program can be used to change this setting while the gateway group is running (see dmadm(1)).

-s services

Specifies the remote *services* that should be initially offered by the domain gateway. The specifications for these services are found in the DMCONFIG file. For example, the specification

-s x,y,z

implies that the gateway should initially advertise remote services x, y, and z. Spaces are not allowed between commas and the *-s* option may appear several times.

-t { on | off }

This option turns off or on the statistics gathering feature for the local domain. The default is off. The dmadmin program can be used to change this setting while the gateway group is running (see dmadmin(1)).

The GWADM server must be booted before the corresponding gateways.

PORTABILITY

GWADM is supported on TUXEDO-supplied servers, using UNIX System operating systems.

INTEROPERABILITY

The initial release of /SNA Domain can only be installed on a node running TUXEDO Release 6.4.

EXAMPLES

The following example illustrates the definition of the administrative server in the UBBCONFIG file.

```
#
*GROUPS
DMADMGRP GRPNO=1
gwgrp GRPNO=2
#
*SERVERS
DMADM SRVGRP="DMADMGRP" SRVID=1001 REPLYQ=N RESTART=Y GRACE=0
GWADM SRVGRP="gwgrp" SRVID=1002 REPLYQ=N RESTART=Y GRACE=0
CLOPT="-A -- -a on -t on"
GWTDOMAIN SRVGRP="gwgrp" SRVID=1003 RQADDR="gwgrp" REPLYQ=Y
RESTART=Y MIN=1 MAX=1
```

SEE ALSO

dmadmin(1), tnboot(1)

dmconfig(5), DMADM(5), servopts(5), ubbconfig(5)

TUXEDO /Domain User Guide

TUXEDO Administrator's Guide

modusr

Modify a remote user password

SYNOPSIS

DMCONFIG=dmconfig

modusr -d *local domain ID* -R *remote domain ID* -u *remote username*

DESCRIPTION

modusr can only be executed as a subcommand of dmadmin(1). The purpose of this page is to describe options for the subcommand and to show an example.

The subcommand allows the administrator to modify passwords in the remote password table. The administrator is prompted for the remote password.

The table entries modified are used for passing remote user names and passwords to remote SNA domains when the application is using /SNA Domain gateways and SECURITY is set to USER_AUTH, ACL, or MANDATORY ACL in the ubbconfig file and SECURITY is set to DM_USER_PW in the DMCONFIG file.

The following options are available:

-d *local domain ID*

This is the name of the local domain gateway with which the ids and passwords are associated. This is the same ID as the one used when creating the domain definitions either in the DMCONFIG file or through the Graphical Administrative Interface.

-R *remote domain ID*

This is the name of the remote domain gateway with which the ids and passwords are associated. This is the same ID as the one used when creating the domain definitions either in the DMCONFIG file or through the Graphical Administrative Interface.

-u *remote username*

The remote user whose password is being modified.

Before running this subcommand the application must be configured using either the Graphical Administrative Interface or tmloadcf(1) and dmloadcf(1). dmadmin modusr may be run on any active node.

PORTABILITY

This subcommand is available only on non-/WS sites running TUXEDO System /T Release 6.4.

DIAGNOSTICS

The dmadm modusr subcommand exits with a return code of 0 upon successful completion.

EXAMPLES

```
modusr -d tux -R cics -u cicsusr /*modifies remote user's password  
sent to CICS. The administrator  
is prompted for the password*/
```

SEE ALSO

delusr(5), addusr(5)

xsnacrm

xsnacrm: X/Motif real-time monitor for running SNACRMs

SYNOPSIS

```
xsnacrm [ X overrides ] address [ address . . . ]
```

(See syntax examples.)

DESCRIPTION

The `xsnacrm` program provides real-time monitoring of running SNACRMs and displays information describing the activity occurring in each SNACRM. The `xsnacrm` utility is intended to be used by administrators and system operators only. Therefore, usage may be restricted by the installation (by setting the execute permissions). `xsnacrm` requires Motif libraries.

COMMAND LINE OPTIONS

`xsnacrm` supports the standard X Toolkit command line arguments (see `X(1)`). The following additional arguments are supported as well.

TRACE OPTIONS

The `xsnacrm` trace level determines the level of detail of SNACRM tracing as follows:

- ◆ `-t 0`—No tracing. Setting this level effectively disables SNACRM tracing and closes the trace file if there is one. If tracing is subsequently restarted, a new file will be created with an incremented numerical suffix.
- ◆ `-t 1`—Minimum tracing. At this level, the SNACRM traces only major events and is sufficient only to determine the sequence of application conversations.
- ◆ `-t 2`—Medium tracing. At this level, SNACRM also traces all I/O buffers.
- ◆ `-t 3`—Maximum tracing. At this level, SNACRM also traces all APPC verbs.

The APPC Protocol Stack API trace is either **enabled** or **disabled**. If enabled, it generally shows the parameters and results of all API calls. Depending on the Stack being used, other options (such as vendor-specified environment variables) may have to be activated for SNACRM to enable the trace.

GENERAL OPTIONS

address

Specifies the INET socket address of a SNACRM to monitor. This value must match the corresponding parameter on the command line used to start the SNACRM you wish to monitor.

There must be at least one address specified. Any number of SNACRMs may be monitored by specifying all their associated addresses.

The format of an address can be either hexadecimal or symbolic. The hexadecimal form consists of the characters “0x” followed by 16 hexadecimal characters representing the INET address (no spaces).

The symbolic format of an address consists of the characters “//” followed by a host name or address followed by a “:” followed by a service name or decimal port number (no spaces).

If a host name is used, it should be an entry in the file /etc/hosts. If a host address is used, it should be specified in the format nnn.nnn.nnn.nnn where each group of nnn represents a decimal number between 1 and 255. This host should identify the computer where the SNACRM you wish to monitor is running, ***not*** the host where `xsnacrm` is to run.

If a service name is used, it should be an entry in the file /etc/services. If a decimal port number is used, it should be a decimal number in the range 4000 - 32767. This number must match the corresponding port number on the command line used to start the SNACRM you wish to monitor. (If the SNACRM was started automatically, the address is specified in the DMCONFIG file).

xsnacrm Window

`xsnacrm` displays a single window consisting of the following sections from top to bottom:

TITLE FRAME

Displays the application title “*BEA Connect SNA CRM Status*”

MENU BAR

Displays the menu items “File” and “Trace.” The File menu consists of a single “Exit” button that terminates xsnacrm. The xsnacrm window may also be terminated by selecting “close” on the X/Motif system menu for the window.

The Trace menu contains two sections that send commands to the currently selected SNACRM to change it’s own tracing function, and the tracing function of the APPC Protocol Stack the SNACRM is using, respectively. To change either current tracing option, select the corresponding menu button (For more information on tracing, please refer to the “Trace Options” section above).

BEA LOGO

Displays the BEA Logo.

SNACRM SELECT PANE

Displays the list of SNACRM’s specified on the command line. The list consists of a set of radio buttons. The selected button determines which SNACRM’s data is displayed in the other panes below.

The phrase “**not active or invalid address**” means that xsnacrm is unable to connect to the INET address specified, because the:

- ◆ Address is incorrect
- ◆ SNACRM is not monitoring the address (probably because it is not running)
- ◆ Path to SNACRM is not available (perhaps due to a network problem)

TRACE STATUS PANE

Displays the current trace options for the selected SNACRM.

LINK STATUS PANE

Displays the current status of all remote links for the selected SNACRM. The text may be scrolled if it is not entirely visible.

LINK STATISTICS PANE

Displays the current statistics for all remote links for the selected SNACRM. The text may be scrolled if it is not entirely visible.

MESSAGE LINE

Displays messages showing the results of either automatic attempts by xsnacrm to connect to the specified SNACRMs or commands issued to change the trace options.

The space in the window allocated to each of the four panes can be adjusted by dragging the sashes (little rectangles) located on the dividers between them.

EXAMPLES

The default geometry for xsnacrm is 630x480+150+150. This places an appropriately sized window for the default font in approximately the center of a 1024x768 Xterm. The following command places this window in the lower-right corner at start-up:

```
xsnacrm -geometry 630x480-0-0 //somehost:4999 //otherhost:6666
```

The following command starts xsnacrm as an icon:

```
xsnacrm -iconic //252.148.37.16:5555
```

The following command shows the use of a hexadecimal address:

```
xsnacrm -bg cyan -fg blue 0x0002123fceb3b1e
```

The following command changes the name of the trace menu to *Commands* and uses the service name *snacrm* for the port number:

```
xsnacrm -xrm "*"tracemenu.labelString: Commands" //me:snacrm
```

CUSTOMIZING X RESOURCES

The default X resources for xsnacrm correspond to the distributed contents of the associated file xsnacrm. To customize the application, copy the xsnacrm file to your home directory and edit it.

WIDGETS

The widget structure of the xsnacrm window is given in the text of the xsnacrm file as follows:

```

! English US resource file for xsnacrm program
!
! "@(#)ISC Devel SNACRM Xsnacrm 1.1 97/08/12 17:49:57";
!
! The values shown below are the fallback resource values
!
! The widget hierarchy is:
!
! XsnacrmApp Shell
!   mainWindowMain Window
!     logoFrame
!       logobitmapLabel
!       menubarRow/Column
!         filemenuPull-down Menu
!         quitPush Button
!         tracemenuPull-down Menu
!           tracebutton0Push Button
!           tracebutton1Push Button
!           tracebutton2Push Button
!           tracebutton3Push Button
!           traceSepSeparator
!           tracebuttonYPush Button
!           tracebuttonNPush Button
!         mainpanePaned Window
!           selectFrameFrame
!             selectFrameLabelLabel
!             selectRadioBoxRow/Column
!               selectButton<n>Toggle Button
!             traceFrameFrame
!               traceFrameLabelLabel
!               traceDataLabel
!             statusFrameFrame
!               statusFrameLabelLabel
!               stusScrollScrolled Window
!                 stusScrollDataLabel
!               statisticsFrameFrame
!                 statFrameLabelLabel
!                 statScrollScrolled Window
!                   statScrollDataLabel
!               mainmessageLabel
!             quitDialog      Message Dialog
!
!
! *title:BEA Connect SNA CRM Status
! *geometry:630x480+150+150
! *foreground:white
! *background:purple

```

```
*fontList:*courier-medium-r-normal--12*
*filemenu.labelString:File
*quitDialog.okLabelString:Exit
*quitDialog.messageString:Exit SNA CRM Status Display now?
*quit.labelString:Exit
*tracemenu.labelString:Trace
*traceButton0.labelString:Stop CRM Trace
*traceButton1.labelString:Set Minimum CRM Trace
*traceButton2.labelString:Set Medium CRM Trace
*traceButton3.labelString:Set Maximum CRM Trace
*traceButtonY.labelString:Start APPC Stack Trace
*traceButtonN.labelString:Stop APPC Stack Trace
```

SEE ALSO

CRMLOGS and SNACRM

SNACRM

Launches the SNA Communications Resource Manager.

SYNOPSIS

```
SNARCM [ -t 0|1|2|3 ] [-s] [-o]] <addr> <group>
```

DESCRIPTION

SNACRM provides all of the sync-level two logic for an SNA domain gateway and directly communicates with the PU2.1 server.

You can manually start SNACRM from the command line or during tmboot with the DMINIT server. You can either start all SNACRM processes with a single DMINIT process or individually start each one with a DMINIT server that is defined to each gateway server group.

When you start SNACRM from the UNIX command line, the SNACRM Command Line Console puts its prompt in this window, and if exited, shuts down all of the active links. When started from DMINIT, the console is redirected to the null device. In this case, you can use the `xsnacrm` command to monitor the console and enable/disable tracing of SNACRM and the SNA stack.

When using TMADMIN to start and stop servers by group id, include the DMINIT server in the same group so that SNARCM can be restarted with its corresponding SNA Domain Gateway.

You must configure one SNACRM for each SNA Domain Gateway, as well as configuring one stack for each SNACRM definition. Each stack can manage one or more SNA links, which is equivalent to a TUXEDO remote domain.

SNACRM has two types of log files stored in \$APPPDIR, RSTRTLOG and BLOBLOG. RSTRTLOG is the transaction state log used during the recovery process, while the BLOBLOG log stores session and link information. Deleting the log files require a cold start for each link involved. You can use the CRMLOGS command to display the contents and state of the SNARCM log files.

TRACE OPTIONS

When initiating the SNACRM command, you can specify any of the following trace levels that determine the level of detail of SNACRM tracing:

- ◆ -t 0—No tracing. Setting this level effectively disables SNACRM tracing and closes the trace file, if there is one. If tracing is subsequently restarted, a new file is created with an incremented numerical suffix.
- ◆ -t 1—Minimum tracing. At this level, SNACRM traces only major events and is sufficient only to determine the sequence of application conversations.
- ◆ -t 2—Medium tracing. At this level, SNACRM also traces all I/O buffers.
- ◆ -t 3—Maximum tracing. At this level, SNACRM also traces all APPC verbs.

The APPC Protocol Stack API trace is either enabled or disabled. If enabled, it generally shows the parameters and results of all API calls. Depending on the stack being used, other options (such as vendor-specified environment variables) may have to be activated for SNACRM to enable the trace.

GENERAL OPTIONS

The following parameters apply to this command:

- s
APPC Stack API trace (default none)
- o
Output gateway load module
- addr*
Socket listening address (required)
0x<16 hex chars> or //<hst><port> (Note: Only the port is significant.)
- group*
SNA Domain Group Name (required)

ENVIRONMENT VARIABLES

You must set the following environment variables before starting SNARCM from the command line or by using a DMINIT server:

- ◆ FIEDLTBLS32 must contain fmb.def

- ◆ FLDTBLDIR32 must contain \$TUXDIR/lib
- ◆ APPDIR must be set to the application directory

CONFIGURING DM_SNACRM

This section describes the DM_SNACRM section in DMCONFIG that pertains to the SNA Communications Resource Manager, and contains the network addressing parameters required by SNA domains. It specifies the field identifiers for each DMCONFIG field, what the field type of the identifier is, and when the it can be updated. All applicable field values are returned with the retrieval operations. Fields that are allowed and/or required for adding a record are described in dmconfig(5). Fields indicated below as *key* are key fields used to uniquely identify a record within a section. These fields are required to be in the input buffer when updates are done and are not allowed to be dynamically updated. The Update column indicates when a field can be updated. The possible values are:

Yes

Can be updated at any time.

NoGW

Cannot be dynamically updated while the gateway group representing the local domain is running.

No

Cannot be dynamically updated while at least one gateway group is running.

The following table lists the fields in this section:

Table A-10 *DM_SNACRM SECTION

Field Identifier	Field Type	Update	Notes
LDOM	string	No/NoGW	Associates this SNACRM with a defined local domain definition.
NWDEVICE	string	No/NoGW	Defines the network device used to communicate between the SNA Domain Gateway and SNACRM. It must be set to /dev/tcp.

Table A-10 *DM_SNACRM SECTION

Field Identifier	Field Type	Update	Notes
SNACRMADDR (required)	string	No/NoGW	Provides the socket of the domain gateway. The hex format for this option is <0xFFFFPPPPAAAAAAA> where the hex values are: FFFF—Protocol family (always 0x0002 for INET) PPPP—TCP/IP port AAAAAAA—IP address of the machine running the SNACRM process

PORTABILITY

The following initial stacks and operating systems support SNACRM:

- ◆ SNA PLUS2, running HP-UX 10.20
- ◆ SUN Link 9.1, running Solaris 2.5.x
- ◆ IBM Comm Server 5.1, running AIX 4.2.1

INTEROPERABILITY

SNARCM is interactive with the following:

- ◆ CICS 3.3 or higher
- ◆ IMS 4.1 or higher
- ◆ MVS 5.22 9510 or higher
- ◆ OS/390 1.2 or higher
- ◆ VTAM for MVS/ESA, version 4.3 or higher

DIAGNOSTICS

SNACRM exits with a return code of 0 upon successful completion.

EXAMPLES

Following is an example of this command:

```
SNACRM -s 0 000215b300000000 GROUP2 /dev/null>std.out 2>std.err &
```

When you start SNACRM from the UNIX command line, the following SNARCM command Line Console appears:

```
$ SNACRM -t 0 000215d300000000 GROUP2
BEA Connect SNA Resource Manager started Thu Dec 11 18:40:49.098 1997
[SNACRM]
```

Console active. Enter commands

?>

da => Display active tasks

dl => Display remote links

ds => Display link statistics

dt => Display trace status

st => Start all links

sh => Stop all links and terminate

si => Terminate immediately (no quiesce)

To launch SNARCM with the console running in the background:

```
$ SNACRM -t0 000215d300000000 GROUP2 <dev/null>std.out 2>std.err &
```

To launch SNARCM with detailed tracing and APPC Stack API tracing turned on from the command line using the host/port address, type:

```
SNARCM -t2 -s //me:SNACRM
```

The following command shows the use of a hexadecimal address:

```
SNARCM -t2 -s 0x0002123fceb3b1e
```

When using the DMINIT server to launch SNARCM, you must specify the CLOPT option in UBCONFIG.CFG as follows:

```
CLOPT="-- -f filename"
```

Where the filename is the name of a shell script containing the start-up command line for one or more SNARCM processes.

SEE ALSO CRMLOGS, dmadmin, dmconfig, and xsnacrm

CRMLOGS

Displays the content and state of the SNA Communications Resource Manager (SNACRM) log files.

SYNOPSIS

CRMLOGS <group> [<crm name>]

DESCRIPTION

You can use the CRMLOGS command to display the contents and state of the two SNARCM log files. RSTRTLOG is the transaction state log used during the recovery process and the BLOBLOG log stores session and link information. Deleting the log files require a cold start for each link involved.

CRMLOGS requires the following parameters:

group

SNA domain group name (required)

crm name

SNACRM name (default SNARCM)

PORTABILITY

The following initial stacks and operating systems support CRMLOGS:

- ◆ SNA PLUS2, running HP-UX 10.20
- ◆ SUN Link 9.1, running Solaris 2.5.x
- ◆ IBM Comm Server 5.1, running AIX 4.2.1

DIAGNOSTICS

CRMLOGS exits with a return code of 0 upon successful completion.

EXAMPLES

To display the RSTRTLOG log file for group2, type:

```
CRMLOGS GROUP2 SNARCM.GROUP2.RSTRTLOG
```

To display the BLOBLOG log file for group1, type:

```
CRMLOGS GROUP1 SNARCM.GROUP1.BLOBLOG
```

SEE ALSO SNACRM and xsnacrm

B ATMI to CPI-C Function Mapping

ATMI Calls Mapped to CPI-C Verbs

This appendix lists the most common ATMI function calls and shows how their parameters map to CPIC verbs. The mappings are shown in the following order:

- ◆ tpcal
- ◆ tpacall with or without reply
- ◆ tpgetrply
- ◆ tpservice
- ◆ tpreturn
- ◆ tpcancel
- ◆ tpconnect
- ◆ tpsend
- ◆ tprecv
- ◆ tpdicon
- ◆ tpforward

B ATMI TO CPI-C FUNCTION MAPPING

The tables on the following pages show the parameters of the ATMI call, what the content or meaning of the parameters is, and notes that indicate the usage with the CPIC verbs.

Table B-1 tpcal

tpcall()	Parameters	Contents	CPIC Notes
svc		Service Name	Used in CMALLC to identify the CICS transaction to be invoked.
idata		User data	This data is sent in CMSENDS until completely transmitted
len		Length of User data	0 < data <= 32K
odata		Reply data	CMRCV receives the data until it has been completely transmitted (data_received is set to CM_COMPLETE_DATA_RECEIVED) and return code is set to CM_OK or CM_DEALLOCATE_NORMAL
olen		Reply data length	0 < data < 32K
flags	TPNOTRAN	Not part of a transaction	
	TPNOCHANGE	N/A	Local
	TPNOBLOCK	N/A	Local
	TPNOTIME	N/A	Local
	TPSIGRSTRT	N/A	Local

Table B-2 tpacall

tpacall()	Parameter	Contents	CPIC Notes
svc		Service Name	Used in CMALLC to identify the CICS transaction to be invoked.

Table B-2 tpacall

tpacall()	Parameter	Contents	CPIC Notes
data		User data	This data is sent in CMSENDS until completely transmitted
len		Length of user data	0 < data <= 32K
flags	TPNOREPLY	false	The last data is sent with a CMSEND with send_type set to CMSEND_AND_PREP_TO_RECEIVE. This changes the state of the conversation to receive and a CMRCV is issued to await the reply
		true	Since no reply is expected, a CMDEAL deallocates the conversation after all data has been received
	TPNOTRAN	Not part of a transaction	Sync level 2 not supported in this release, therefore the service call should not be in transaction mode
	TPNOBLOCK	N/A	Local
	TPNOTIME	N/A	Local
	TPSIGRSTRT	N/A	Local

Table B-3 tpgetrply

tpgetrply()	Parameters	Contents	CPIC Notes
cd		call descriptor	The call descriptor is mapped to the CONVID returned by the CMINIT when the LU6.2 was initiated
data		User data	Data received from CMRCV if WHAT_RECEIVED set to DATA_COMPLETE
len		Length of user data	0 < data <= 32K

B ATMI TO CPI-C FUNCTION MAPPING

Table B-3 tpgetrply

tpgetrply()	Parameters	Contents	CPIC Notes
flags	TPGETANY	If true, data is returned from any conversation. If false, data is returned from conversation associated with the cd	Data available on any conversation is returned to the requestor
	TPNOCHANGE	Local to the requestor	Limited buffer types supported.
	TPNOBLOCK	N/A	Local
	TPNOTIME	N/A	Local
	TPSIGRSTRT	N/A	Local

Table B-4 tpservice

tpservice()	Parameters	Contents	CPIC Notes
svcinfo		Service information and data	User Data captured from a CMRCV populates the TPSVCINFO structure user data area. Service characteristics are obtained from the service attributes in the DMCONFIG and UBBCONFIG files.
name		Service name	The service name associated with the 8 character RNAME sent from CICS.
data		User data	Data captured from CMRCV
len		Length of user data	0 < data <= 32K Length of data received
cd		call descriptor	The call descriptor associated with the CONVID returned by the CMINIT when the LU6.2 was initiated
appkey		32-bit key (if used)	For security
cldid		set by BEA TUXEDO	For security

Table B-4 tpservice

tpservice()	Parameters	Contents	CPIC Notes
flags	TPCONV	If true, service is conversational	
	TPTRAN	N/A	Sync level 2 not supported in this release, therefore the service cannot be called in transaction mode
	TPNOREPLY	If true, requestor not expecting a reply	The conversation is terminated with a CMDEAL normal
	TPSENDONLY	N/A	If set, the CPIC conversation in CICS should be in receive state. If not set, the CICS CPIC conversation state will be in send state.
	TPRECVONLY	N/A	If set, the CPIC conversation in CICS remains in send state.

Table B-5 tpreturn

tpreturn()	Parameters	Contents	CPIC Notes
rval		TPSUCCESS	Set to TPSUCCESS when conversation terminates with a normal deallocation.
		TPSVCERR	Set to TPESVCERR when the conversation has terminated with a non-normal deallocation type or other error.
rcode		Set by the application	N/A
data		User data	Data is returned to the CICS transaction from a successful CMRCV with data received set to CM_DATA_COMPLETE and return code of CM_DEALLOCATE_NORMAL. If the service fails, no data is returned to the caller and the conversation is deallocated abnormally.
len		Length of data returned	0 < data <= 32K
flags		N/A	N/A

Table B-6 tpcancel

tpcancel()	Parameters	Contents	CPIC Notes
cd		The connection descriptor on which a tpgetreply() is waiting.	CMDEAL abnormal is issued on the conversation with CONVID mapped from call descriptor

Table B-7 tpconnect

tpconnect()	Parameters	Contents	CPIC Notes
svc		The local service name representing the service to be invoked. in CICS	The name is used to find the RNAME. The RNAME should match the TPName in CICS and will be used by CMINIT and CMALLC to initiate and allocate the conversation
data		User data	This data is sent in CMSENDs until completely transmitted
len		Length of User data	0 < data <= 32K
flags	TPNOTRAN	True	Sync level 2 not supported in this release, therefore the service call should not be in transaction mode
	TPSENDONLY	If true, the conversation stays in or changes to send state	The conversation remains in send state. This is the default.
	TPRECVONLY	If true, the conversation stays in or changes to receive state	Immediately after the allocate BEA TUXEDO sends a CMSEND with no data and send_type set to CM_SEND_AND_PREP_TO_RECEIVE
	TPNOBLOCK	N/A	Local
	TPNOTIME	N/A	Local
	TPSIGRSTRT	N/A	Local

Table B-8 tpsend

tpsend()	Parameters	Contents	CPIC Notes
cd		The connection descriptor	This locally assigned connection descriptor has been mapped to the CONVID returned in the CMINIT and CMALLC on behalf of the tpconnect()
data		User data	ASCII/EBCDIC conversion may be required before sending to CICS. See Chapter 5 “Buffers”
len		Length of User data	0 < data <= 32K
flags	TPRECVONLY	If true, the conversation changes to receive state	The state of the conversation changes from send to receive. A CMSEND is sent with send_type set to CM_SEND_AND_PREP_TO_RECEIVE
	TPNOBLOCK	N/A	Local
	TPNOTIME	N/A	Local
	TPSIGRSTRT	N/A	Local
revent	TPEV_DISCONIMM	If set, the LU6.2 conversation has been terminated abnormally	If the return code from a CMRCV is deallocate_abnormal, the conversation is terminated. A disconnect event is sent to the sending process.
	TPEV_SVCERR	If set, the LU6.2 conversation has been terminated abnormally	Any return code other than CM_OK or CM_DEALLOCATE_NORMAL is treated as a TPEV_SVCERR
	TPEV_SVCFAIL	If set, the LU6.2 conversation has been terminated abnormally	If the return code from CMRCV is CM_TP_NOT_AVAIL_NO_RETRY or CM_TP_RESOURCE_FAILURE_NO_RETRY, revent is set to TPEV_SVCFAIL

Table B-9 tprecv

tprecv()	Parameters	Contents	CPIC Notes
cd		The connection descriptor	This locally assigned connection descriptor has been mapped to the CONVID returned in the CMINIT and CMALLC issued by the initiator of this conversation
data		User data	Date to be received using a CMRCV_immediate and returned to the BEA TUXEDO service
len		Length of User data	0 < data <= 32K
flags	TPNOCHANGE	Local	Must be a supported buffer type
	TPNOBLOCK	N/A	Local
	TPNOTIME	N/A	Local
	TPSIGRSTRT	N/A	Local
revent	TPEV_DISCONIMM	If set, the LU6.2 conversation has been terminated abnormally	If the return code from a CMSEND is deallocate_abnormal, the conversation is terminated. A disconnect event is sent to the sending process.
	TPEV_SENDOONLY	If set, the LU6.2 conversation changes to send if partner allows it	The sending partner has sent a CMSEND with send_type set to CM_SEND_AND_PREP_TO_RECEIVE
	TPEV_SVCERR	If set, the LU6.2 conversation has been terminated abnormally	Any return code other than CM_OK or CM_DEALLOCATE_NORMAL is treated as a TPEV_SVCERR
	TPEV_SVCFAIL		If the return code from CMRCV is CM_TP_NOT_AVAIL_NO_RETRY or CM_TP_RESOURCE_FAILURE_NO_RETRY, revent is set to TPEV_SVCFAIL

Table B-9 tprecv

tprecv()	Parameters	Contents	CPIC Notes
	TPEV_SVCSUCC	If set, the conversation has completed normally	The return code from CMRCV was set to CM_DEALLOCATE_NORMAL. This indicates that the sending TP has completed and deallocated the conversation normally.

Table B-10 tpdiscn

tpdiscn()	Parameters	Contents	CPIC Notes
cd		The connection descriptor	This connection descriptor is mapped to the CONVID returned from CMINIT or CMACCP to the originator of the conversation

Table B-11 tpforward

tpforward ()	Parameters	Contents	CPIC Notes
svc		Service name	tpforward() is treated as if it were a tpacall(). A CMINIT and subsequent CMALLC are issued to initialize and allocate a session for a conversation. ClientID must be propagated to the CICS transaction in a TPSVCINFO record
data		User data	Data is sent using CMSEND. The last CMSEND is sent with send_type of deallocate_normal
len		Length of data returned	0 < data <= 32K
flags		Refer to tpacall()	

C CPI-C Parameters and Values

CPI-C Verbs

The following table shows the parameters required for each CPI-C verb and the permissible values when used with BEA Connect SNA.

Table C-1 CMINIT

Parameter	In/Out	Value	Usage
conversation_ID	Output	Returned by the SNA Stack	Used on every other call
sym_dest_name	Input	Symbolic name that represents the partner LU, the modename, and the TP (TUXEDO service)	If blank, SET calls must be made to initialize conversation parameters
return code	Output	See list of CPI-C return codes	If not CM_OK, conversation fails.

Table C-2 CMALLC

Parameter	In/Out	Value	Usage
conversation_ID	Input	Value returned from CMINIT	
return_code	Output	See list of CPI-C return codes	If not CM_OK, or CM_ALLOCATION_FAILURE_RETRY, conversation fails

Table C-3 CMRCV

Parameter	In/Out	Value	Usage
conversation_ID	Input	Value returned from CMINIT	
buffer	Input	Name of variable to be populated by data received	
requested_length	Input	Name of variable where amount of data to be received is posted	0 <= requested_length < 32K
data_received	Output	CM_COMPLETE	Indicates enough data in LU buffer to satisfy requested length
		CM_INCOMPLETE	Another CMRCV must be issued to get enough data to satisfy the requested length
received-length	Output	Name of variable where amount of data returned is posted	Returned data may be shorter than requested amount including zero
status_received	Output	CM_NO_STATUS	No state change.
		CM_SEND_RECEIVED	State of the conversation is changed to send
request_to_send_received	Output	CM_REQ_TO_SEND_RECEIVED	Receiver enters send state.
		CM_REQ_TO_SEND_NOT_RECEIVED	Partner has not requested a state change
return_code	Output	See list of CPI-C return codes	If not CM_OK or CM_DEALLOCATE_NORMAL, the conversation is deallocated.

Table C-4 CMSEND

Parameter	In/Out	Value	Usage
conversation_ID	Input	Value returned from CMINIT	
buffer	Input	Name of buffer area containing data to be sent	
send_length	Input	Length of data in buffer to be sent	0 <= data < 32K
req_to_send_received	Output	The name of the variable that contains the indication that indicates CM_REQ_TO_SEND_RECEIVED	Partner request to change state to send
		CM_REQ_TO_SEND_NOT_RECEIVED	Partner has not requested a state change
return_code	Output	See list of CPI-C return codes	If not CM_OK, the conversation is deallocated

Table C-5 CMACCP

Parameter	In/Out	Value	Usage
conversation_ID	Output	Returned from the LU	
return_code	Output	See list of CPI-C return codes	If not CM_OK, the accept fails and no conversation is started

D Error Messages

SNACRM Messages

The following messages are logged to the USERLOG file with a 90xx prefix and are also written to the stderr file. For any SNACRM messages not listed, contact BEA Connect Technical Support. The SNACRM error messages are listed in alphabetical order.

ERROR	Allocation Failure for <trancode> on <remotesysid>: <error>	
	Description	An allocation error occurred.
	Action	The reason for the failure is described by <error>. Correct the problem with configuration or application.
	See Also	<i>None</i>
ERROR	Attempt to connect as second master refused!	
	Description	A second GWSNAX is attempting to connect to the SNACRM as a master gateway. Only one master gateway is allowed.
	Action	Ensure that multiple TUXEDO configurations do not use the same SNACRM address.
	See Also	<i>None</i>

ERROR	Attempt to connect with incorrect group name refused!	
	Description	The group name in the DMCONFIG file does not match the group name specified in the SNACRM command line.
	Action	Correct the group name that is in error and restart.
	See Also	<i>None</i>
ERROR	Attempt to send <nnnnn> bytes (> 32767)	
	Description	The length of a send request exceeded 32767 (including overhead).
	Action	Check application program and correct.
	See Also	<i>None</i>
ERROR	CNOS Notification Received for unknown partner!	
	Description	Multiple instances of the SNACRM may be using the same local LU.
	Action	Ensure that multiple TUXEDO configurations do not use the same local LU.
	See Also	<i>None</i>
ERROR	Configuration change on <linkref> requires cold start	
	Description	Attempting to do a warm start after changing the domain configuration.
	Action	Change start type to "COLD" and restart.
	See Also	<i>None</i>
ERROR	Conversation terminated without confirm for context <correlator>	
	Description	Sync level 2 conversation was terminated without confirm.
	Action	Check application program and correct.
	See Also	<i>None</i>

ERROR	Exchange logs failed with <linkref>	
	Description	An error occurred during the exchange logs process.
	Action	Run CRMLOGS to examine the SNACRM log file. Cold start the TUXEDO application.
	See Also	<i>None</i>
ERROR	Exchange Logs Rejected for <remotesysid>. Restart Type or Log Name Mismatch	
	Description	The log files for the SNACRM have been incorrectly modified.
	Action	Run CRMLOGS to examine the SNACRM log file. Cold start the TUXEDO application.
	See Also	<i>None</i>
WARNING	Inbound Exchange Logs Rejected for <remotesysid>	
	Description	Link not configured for sync level 2.
	Action	None. This message is for information only.
	See Also	<i>None</i>
ERROR	Invalid cold start received from <remotesysid> Unrecovered Local transactions are pending.	
	Description	Attempting to cold start host while warm starting TUXEDO.
	Action	Run CRMLOGS to examine the SNACRM log file. Cold start the TUXEDO application.
	See Also	<i>None</i>

ERROR	Invalid warm start received from <remotesysid>. Unknown log name.	
	Description	The log files for the SNACRM have been incorrectly modified.
	Action	Run CRMLOGS to examine the SNACRM log file. Cold start the TUXEDO application.
	See Also	<i>None</i>
WARNING	Link not configured for sync-level 2.	
	Description	This warning results when the inbound exchange log is rejected.
	Action	<i>None</i>
	See Also	<i>None</i>
ERROR	Link refers to undefined APPC stack!	
	Description	The stackref in the link configuration is incorrect.
	Action	Correct the stackref that is in error, run dmloadcf, and restart.
	See Also	<i>None</i>
ERROR	Mixed Heuristic on link <linkref>	
	Description	One side has reported committed while the other side has reported aborted.
	Action	Check the ULOG for any additional messages.
	See Also	<i>None</i>
ERROR	No Available Session on link <linkref> for context <correlator>	
	Description	Maximum number of sessions has been exceeded.
	Action	Check session limits in DMCONFIG, stack configuration, CICS or VTAM. Increase session limits if necessary.
	See Also	<i>None</i>

ERROR	Missing send last from host <ATMI r/r> for context <correlator>	
	Description	Host application did not issue send last during an out-bound request/ response service. The host application may have abended.
	Action	Check application program and correct.
	See Also	<i>None</i>
ERROR	No blob with recovery request. Tranid = <tctx> dropped	
	Description	Attempting to warm start after the SNACRM BLOBLOG has been modified.
	Action	Change start type to "COLD" and restart.
	See Also	<i>None</i>
ERROR	Requested Synclevel not supported by link <linkref> for context <correlator>	
	Description	Attempted to issues a request at sync level 2 on a sync level 0 link.
	Action	Correct application.
	See Also	<i>None</i>
ERROR	Server <stackref> Creation Failed	
	Description	SNACRM was unable to instantiate the stack object due to an error.
	Action	Check for additional messages in stderr. Could be that the shared library for the stack or the stack interface could not be loaded due to an incorrect library path.
	See Also	<i>None</i>

ERROR Service Request at SL=2 Rejected on pending link <linkref>	
Description	An attempt to start a new sync level 2 request has been received and the Link is currently processing recovery information.
Action	Wait until recovery is complete to request sync level 2 services.
See Also	None
ERROR Server <stackref> Failed, Code = <returncode>	
Description	SNACRM received a bad return code from the stack start-up.
Action	<p>The <returncode> is the value returned by the SNA Stack software. Check the status of the stack and the configuration of the stack and the DMCONFIG.</p> <p>Note: You must manually kill the SNACRM process.</p>
See Also	None
ERROR Unable to start recovery task for link <linkref>	
Description	An error occurred during the warm start of TUXEDO.
Action	Cold start the TUXEDO application.
See Also	None
ERROR Unknown tranid = <tctxt> dropped	
Description	Recovery was requested by TUXEDO on a transaction that was already forgotten by the SNACRM.
Action	None. This message for information only.
See Also	None

ERROR	Undefined Remote LU on link <linkref>	
	Description	The remote LU does not exist as defined.
	Action	Check the DMCONFIG file and the stack configuration and correct the mismatch.
	See Also	<i>None</i>
ERROR	Unable to start session on link <linkref>. <reason>	
	Description	Link activation failure due to SNA error.
	Action	<reason> is the description of the stack return code. Determine the cause and correct it.
	See Also	<i>None</i>
ERROR	Unable to initialize link <linkref>. <reason>	
	Description	Link initialization failure due to SNA error.
	Action	<reason> is the description of the stack return code. Determine the cause and correct it.
	See Also	<i>None</i>
ERROR	Unrecovered tranid=<tctx> blob dropped. Presumed forgotten.	
	Description	SNACRM found recovery information but GWSNAX did not request recovery during warm start.
	Action	<i>None</i>
	See Also	<i>None</i>

GWSNAX Messages

For any GWSNAX messages not listed, contact your BEA Connect Technical Support.

3051 ERROR	Exceed local transaction limit<max_tran>!	
	Description	Number of concurrent transactions using this domain gateway exceeded the configured limit.
	Action	Check the dmconfig file for MAXTRAN parameter, this number is the limit. If MAXTRAN is not specified, then check the ubbconfig file for MAXGTT parameter. In the later case MAXGTT is the limit. Modify the limit if necessary.
	See Also	<i>None</i>
3053 ERROR	Create transaction tree node failed!	
	Description	Failed to create transaction tree node object. The reasons can be either because memory allocation failed, or the configured transaction limit reached.
	Action	Check the prior messages in the USERLOG. If the cause is the memory allocation failure, then update the system and/or system configuration. If the cause is the limit reached, then may need to modify the dmconfig or ubbconfig.
	See Also	<i>None</i>

3067 ERROR	Cannot set remote domain for transaction!	
	Description	Several possibilities for this failure. The first one is unable to get shared memory lock, the other one is it reached the configured limit for number of remote domains that can get involved in the same transaction. This limit is specified in the MAXRDTRAN in the dmconfig.
	Action	Check the configuration, and compare with number of remote domains which may get involved in the transaction. Update the configuration file if necessary. If the cause is not in the configuration file, then contact your BEA Connect Technical Support.
	See Also	<i>None</i>
3071 ERROR	Defined transaction limit max_tran reached!	
	Description	The number of concurrent transactions using this domain gateway exceeded the limit.
	Action	If MAXTRAN is not specified, then check the ubbconfig file for MAXGTT parameter. In the later case MAXGTT is the limit. Modify the limit if necessary.
	See Also	<i>None</i>
3085 ERROR	Transaction node is de-allocated because of ABEND!	
	Description	Informational. Detected ABEND condition or received ABEND event from SNACRM causing the node object to be destroyed.
	Action	No action need to be taken.
	See Also	<i>None</i>

3124 ERROR	Send verb <verb_id> to snacrm failed!	
	Description	Failed to send SNACRM verb through network connection.
	Action	Check the network connection and determine if the SNACRM is still alive.
	See Also	<i>None</i>
3127 ERROR	Send verb <verb_id> to SNACRM failed!	
	Description	Failed to send SNACRM verb through network connection.
	Action	Check the network connection and determine if the SNACRM is still alive.
	See Also	<i>None</i>
3130 ERROR	Send verb <verb_id> to SNACRM failed!	
	Description	Failed to send SNACRM verb through network connection.
	Action	Check the network connection and determine if the SNACRM is still alive.
	See Also	<i>None</i>
3146 ERROR	Unable to find SNACRM for superior remote domain<rdom_name>	
	Description	Failed to find remote domain definition in the configuration in order to get SNACRM identification.
	Action	Check whether there is a configuration change between the boot that would modify or remove the remote definition from dmconfig. If there is no change, then contact your BEA Connect Technical Support.
	See Also	<i>None</i>

3149 ERROR	Could not create transaction tree structure!	
	Description	Unable to allocate a transaction tree node structure to represent the transaction recovery object.
	Action	Check the configuration to determine if there is a change of MAXTRAN in dmconfig and/or a change of MAXGTT in ubbconfig. This condition may occur if there is a decrease in the number of transactions allowed between boot-ups while many transactions must be recovered. If no change is made, then contact your BEA Connect Technical Support.
	See Also	<i>None</i>
3502 ERROR	Buffer type <type,subtype> not defined in the buffer type switch!	
	Description	The buffer type associated with the inbound message is not specified in the type switch.
	Action	Check the application, and configuration about which buffer type it is using. GWSNAX only supports standard TUXEDO buffer types over the domain gateway. If the buffer type is VIEW, VIEW32, FML, or FML32, then also check the environmental variable VIEWFILES, VIEWFILES32, VIEWDIR, VIEWDIR32.
	See Also	<i>None</i>
3505 ERROR	Input buffer too big<buflen>!	
	Description	Input buffer exceeding 32K limit allowed for SNA Gateway.
	Action	Check the application, if necessary contact your BEA Connect Technical Support.
	See Also	<i>None</i>

3506 ERROR	Missing subtype specification for buftype	
	Description	The subtype specification is required for this buffer type.
	Action	Check the application.
	See Also	<i>None</i>
3511 ERROR	Missing subtype for FML type buffer!	
	Description	Missing subtype specification needed for decoding for FML buffer type.
	Action	Check the application, and check environmental variable setting of VIEWDIR, VIEWDIR32, VIEWFILE, VIEWFILE32. If necessary contact your BEA Connect Technical Support.
	See Also	<i>None</i>
3515 ERROR	Convert buffer type<buftype,bufstype> failed, convert failure!	
	Description	Conversion from C structure to fielded buffer failed.
	Action	Check memory situation, if the error did not due to memory allocation failure then check the whether the actual data is larger than the definition.
	See Also	<i>None</i>
3517 ERROR	Missing subtype definition for FML32 buffer type!	
	Description	FML32 buffer subtype is missing from domain configuration in either local or remote service definition.
	Action	Correct the domain configuration.
	See Also	<i>None</i>

3519 ERROR	Convert FML32 buffer type<buftype,bufstype> failed, convert failure!	
	Description	Conversion from FML32 buffer type to C structure failed.
	Action	Check whether the buffer type definition is too large, and whether the view name can be found in the file specified by VIEWDIR32, VIEWFILE3. Also, check the view syntax.
	See Also	<i>None</i>
3521 ERROR	Convert FML32 buffer with subtype<bufstype> failed!	
	Description	Conversion from C structure to FML32 failed.
	Action	Check whether the data is too large.
	See Also	<i>None</i>
3523 ERROR	Failed to convert for FML buffer <bufstype>, exceeding limit!	
	Description	Failed to convert the FML buffer; the required buffer size to process the conversion exceeds the limit.
	Action	Check the view file definition for this FML buffer subtype.
	See Also	<i>None</i>
3524 ERROR	Failed to convert for FML32 buffer <bufstype>, exceeding limit!	
	Description	Failed to convert the FML32 buffer, the required buffer size to process the conversion exceeds the limit.
	Action	Check the view file definition for this FML32 buffer subtype.
	See Also	<i>None</i>

3525 ERROR	Fail to retrieve remote service definition!	
	Description	Could not find remote service definition in the shared memory.
	Action	Check the /Domain configuration using dmunloadcf command to check the remote service definition. If error found, then correct the dmconfig and reload it. If no error found contact your BEA Connect Technical Support.
	See Also	<i>None</i>
3527 ERROR	Must specify subtype for <buftype>	
	Description	Missing specification for the buffer subtype.
	Action	Check the /Domain configuration for SNA gateway. If no error found in the configuration, contact your BEA Connect Technical Support.
	See Also	<i>None</i>
3529 ERROR	Convert buffer type <buftype> with subtype <bufstype> failed<rc=Error>!	
	Description	Conversion of VIEW or VIEW32 data buffer to target record type failed.
	Action	Check the error code in the message, and make any correction according to the error code suggested. If nothing is wrong, then contact your BEA Connect Technical Support.
	See Also	<i>None</i>
3530 ERROR	Convert buffer type buftype with subtype bufstype failed<rc=Error>!	
	Description	Convert VIEW or VIEW32 data buffer to target record type failed because it's length exceeds the limit.
	Action	Check the view definition.
	See Also	<i>None</i>

3534 ERROR	Convert FML buffer with subtype <bufstype> failed<Error=rc>!	
	Description	Conversion of FML buffer to C structure failed with reason code.
	Action	Check the reason code, and make any correction if necessary. If no error can be found, contact your BEA Connect Technical Support.
	See Also	<i>None</i>
3536 ERROR	Convert FML buffer with subtype <bufstype> failed<Error=rc>!	
	Description	Conversion FML buffer to C structure failed with reason code.
	Action	Check the reason code, and make any correction if necessary. If no error can be found, contact your BEA Connect Technical Support.
	See Also	<i>None</i>
3538 ERROR	FML32 buffer type requires subtype to be specified!	
	Description	The required buffer type is not specified.
	Action	Do a <code>dmunloadcf</code> to find whether the configuration for the service has missing subtype for FML32 buffer type, and correct it. After the correction, reload the SNA gateway.
	See Also	<i>None</i>
3540 ERROR	Convert FML32 buffer with subtype <bufstype> failed<Error=rc>!	
	Description	Conversion of FML32 buffer to C structure failed with reason code.
	Action	Check the reason code, and make any correction if necessary. If no error can be found, contact your BEA Connect Technical Support.
	See Also	<i>None</i>

3542 ERROR	Convert FML32 buffer with subtype <bufstype> failed<Error=rc>!	
	Description	Conversion of FML32 buffer to C structure failed with reason code.
	Action	Check the reason code, and make any correction if necessary. If no error can be found, contact your BEA Connect Technical Support.
	See Also	<i>None</i>
5021 ERROR	Encode reply failed!	
	Description	Failed to encode reply message data to SNACRM.
	Action	Look into the USERLOG for prior failure message which gives more detailed description of the reason why it failed. The most common reason is the buffer subtype not specified or not found. Check the /Domain configuration for OUTBUFTYPE of the failed service response.
	See Also	<i>None</i>
5023 ERROR	Could not get transaction	
	Description	Failed to associate a transactional ACALL service request to a transaction object.
	Action	The most likely reasons for this failure are, exceeding local configured limit, and memory allocation failed to create the transaction object. First check the prior failure message to identify the real cause of the failure. If the failure is for local configuration limit exceeded, then check whether the application is licensed to do transactions, check the DMCONFIG file for MAXTRAN, and check that SYNC LEVEL is set to 2 for remote domain. If the failure is caused by memory allocation failure, then check the system memory configuration.
	See Also	<i>None</i>

5024 ERROR	Could not get API	
	Description	The API information for the remote service is incorrect.
	Action	Check the DMCONFIG for any API not configured correctly. If everything looks fine, contact your BEA Connect Technical Support.
	See Also	<i>None</i>
5025 ERROR	Could not get transaction	
	Description	Failed to associate a transactional CONNECT service request to a transaction object.
	Action	Most likely reason for this failure are, exceeding local configured limit, and memory allocation failed to create the transaction object. First check the prior failure message to identify the real cause of the failure. If the failure is for local configuration limit exceeded, then check if the application is licensed to do transaction, check the DMCONFIG file for MAXTRAN, and check that SYNC LEVEL is set to 2 for remote domain. If the failure is caused by memory allocation failure, then check the system memory configuration.
	See Also	<i>None</i>
5026 ERROR	Could not get API	
	Description	The API information for the remote service is incorrect.
	Action	Check the DMCONFIG for any API not configured correctly. If everything looks fine, contact your BEA Connect Technical Support.
	See Also	<i>None</i>

5027 ERROR	Could not get API	
	Description	The API information for the remote service is incorrect.
	Action	Check the DMCONFIG for any API not configured correctly. If everything looks fine, contact your BEA Connect Technical Support.
	See Also	<i>None</i>
5029 ERROR	Link status down for remote domain <rdom>	
	Description	The link status for the remote domain is down.
	Action	Check whether the stack is up, and then check that the session between the stack provider and CICS is up.
	See Also	<i>None</i>
5030 ERROR	Link status pending for remote domain rdom	
	Description	The link status for the remote domain is pending.
	Action	The transaction recovery is still in progress, wait a while before sending request to remote domain.
	See Also	<i>None</i>
5031 ERROR	Transaction not allowed, request rejected!	
	Description	Failed to send ACALL request to remote domain because transaction is not allowed in the current configuration.
	Action	Check the following 3 things, first whether the GWSNAX is licensed to do transaction; second check whether the DMCONFIG file is configured to do transaction; third check whether the remote domain is configured to do SYNC level 2 transaction.
	See Also	<i>None</i>

5032 ERROR	Transaction not allowed, request rejected!	
	Description	Failed to send CONNECT request to remote domain because the transaction is not allowed in the current configuration.
	Action	Check the following 3 things, first whether the GWSNAX is licensed to do transaction; second check whether the DMCONFIG file is configured to do transaction; third check whether the remote domain is configured to do SYNC level 2 transaction.
	See Also	<i>None</i>
5034 ERROR	Allocate context failed!	
	Description	Failed to allocate ACALL request context.
	Action	Check any prior message to tell the real reason of failure. The most common reason for failure is memory allocation failed.
	See Also	<i>None</i>
5043 ERROR	Trans Event Error	
	Description	Failed to find transaction object using TCTXT which is specified in the SNACRM_QUERY_SVC_TYPE. This will cause GWSNAX to shut down.
	Action	This could be a stale inbound request not cleaned up by SNACRM, contact your BEA Connect Technical Support.
	See Also	<i>None</i>

5050 ERROR	SHUTDOWN message received, shutdown gateway!	
	Description	Gateway GWSNAX received SNACRM_SHUTDOWN message from SNACRM, and will shut down immediately.
	Action	Check any problem encountered in SNACRM and contact your BEA Connect Technical Support.
	See Also	<i>None</i>
5062 ERROR	Failed to find local service<resource> definition!	
	Description	The gateway GWSNAX failed to find local service in the configuration specified in the SNACRM_QUERY_SVC_TYPE message received from SNACRM.
	Action	Check the DMCONFIG for any missing definition of the local service. If not, then contact your BEA Connect Technical Support.
	See Also	<i>None</i>
5065 ERROR	Enter shutdown	
	Description	Failed to send SNACRM_QUERY_SVC_TYPE response to SNACRM.
	Action	Check any communication problem between GWSNAX and SNACRM.
	See Also	<i>None</i>
5067 ERROR	Decode inbound ACALL request failed	
	Description	Decode data in the SNACRM_ACALL request from SNACRM failed.
	Action	Check for prior error message, and handle it accordingly.
	See Also	<i>None</i>

5068 ERROR	Could not get FUNC	
	Description	Failed to get FUNCTION specification from configuration, or the FUNCTION specification in the configuration is not supported.
	Action	Check the DMCONFIG configuration for the service.
	See Also	<i>None</i>
5069 ERROR	Could not get FUNC	
	Description	Failed to get FUNCTION specification from configuration, or the FUNCTION specification in the configuration is not supported.
	Action	Check the DMCONFIG configuration for the service.
	See Also	<i>None</i>
5070 ERROR	Failed in security checking for RRCORR<svcReqCorr>!	
	Description	Security checking failed for the inbound ACALL request.
	Action	Check for prior failure message to identify real cause of security failure.
	See Also	<i>None</i>
5073 ERROR	Could not get API	
	Description	Failed to retrieve API information from the configuration, or retrieved API information is not supported for inbound request.
	Action	Check the DMCONFIG for the API specified for the local service.
	See Also	<i>None</i>

5077 ERROR	Enter shutdown	
	Description	Failed to send SNACRM_SIGNAL_EVENT message to SNACRM.
	Action	Check the connection between GWSNAX and SNACRM.
	See Also	<i>None</i>
5078 ERROR	Enter shutdown	
	Description	Failed to send SNACRM_CONNECT response message to SNACRM.
	Action	Check the connection between GWSNAX and SNACRM.
	See Also	<i>None</i>
5081 ERROR	Inbound connect request security check failed, RRCORR<svcReqCorr>!	
	Description	Failed to pass the security checking for the inbound SNACRM_CONNECT request
	Action	Check the security setting in the DMCONFIG file.
	See Also	<i>None</i>
5083 ERROR	Invalid buffer type, encode failed!	
	Description	The buffer type used by the TUXEDO client or server is not recognized in GWSNAX gateway. The GWSNAX only recognizes standard buffer types, such as STRING, CARRAY, X_OCTET, X_COMMON, X_C_TYPE, VIEW, VIEW32, FML, and FML32.
	Action	Check the application programs which will communicate through GWSNAX gateway.
	See Also	<i>None</i>

5084 ERROR	Outbound ACALL encode failed!	
	Description	Failed to encode data for outbound SNACRM_ACALL request message.
	Action	Check for prior error message for the real cause of failure, and handle the error accordingly.
	See Also	<i>None</i>
5087 ERROR	Invalid buffer type, decode failed!	
	Description	Unrecognized buffer type found for inbound message from SNACRM.
	Action	Check the applications, and the DMCONFIG file for the INBUFTYE. Only STRING, CARRAY, X_OCTET, X_COMMON, X_C_TYPE, VIEW, VIEW32, FML, and FML32 are allowed.
	See Also	<i>None</i>
5090 ERROR	Outbound CONNECT encode failed	
	Description	Failed to encode data for the outbound SNACRM_CONNECT request message.
	Action	Check for prior error message for actual failure reason. The most common reasons are either memory allocation failure, OUTBUFTYPE specification error in the DMCONFIG, or could not find the definition file such as VIEW, VIEW32, FML, and FML32.
	See Also	<i>None</i>

5091 ERROR	Outbound CONVSEND encode failed	
	Description	Failed to encode data for the outbound SNACRM_SEND_DATA request message.
	Action	Check for prior error message for actual failure reason. The most common reasons are either memory allocation failure, OUTBUFTYPE specification error in the DMCONFIG, or could not find the definition file such as VIEW, VIEW32, FML, and FML32.
	See Also	<i>None</i>
5094 ERROR	Unable to send data on sna_conv_idx <ctx_idx>, gpnd <gpnd>	
	Description	Failed to send SNACRM_CONNECT response to SNACRM, GWSNAX shutdown itself.
	Action	Usually a serious network problem has occurred. Check the network connection to determine if the SNACRM is still up.
	See Also	<i>None</i>
5101 ERROR	Unable to find view file for buftype <bufstype>	
	Description	Failed to find view file for the FML buffer type specified in the INBUFTYPE for the inbound request.
	Action	Check the VIEWFILES and VIEWDIR environmental variable, and whether the view file is in the directory specified by VIEWDIR.
	See Also	<i>None</i>

5107 ERROR	Unable to get appkey for this outbound request	
	Description	Failed to find the remote domain configuration information in the shared memory.
	Action	Check the DMCONFIG for the remote domain configuration and its security setting. Contact your BEA Connect Technical Support.
	See Also	<i>None</i>
5109 ERROR	Unable to find view file for buftype <bufstype>	
	Description	Failed to find view file definition for encoding outbound FML data as specified in the OUTBUFTYPE.
	Action	First check the VIEWDIR, and VIEWFILES environment to find out whether the specified VIEW file for FML is there. Next, check the DMCONFIG for the OUTBUFTYPE to find out if it is correct.
	See Also	<i>None</i>
5110 ERROR	Unable to find view file for buftype <bufstype>	
	Description	Failed to find view file definition for encoding outbound FML32 data as specified in the OUTBUFTYPE.
	Action	First check the VIEWDIR32, and VIEWFILES32 environment to find out whether the specified VIEW file for FML is there. Next, check the DMCONFIG for the OUTBUFTYPE to find out if it is correct.
	See Also	<i>None</i>

5112 ERROR	System full, can't create action, Enter shutdown	
	Description	Failed to create GWSNAX gateway internal scheduled event to handle failure or disconnect.
	Action	Either memory allocation failed or system limit reached. Check and update system memory configuration. If the reason is the later one, then contact your BEA Connect Technical Support.
	See Also	<i>None</i>
6000 ERROR	YOUR CONNECT SNA LICENSE IS EITHER INVALID OR EXPIRED	
	Description	The BEA CONNECT SNA v2.0 license is missing or expired.
	Action	Contact your BEA sale representative.
	See Also	<i>None</i>
6001 ERROR	YOUR CONNECT SNA LICENSE IS NOT ENABLED FOR SYNCLEVEL 2	
	Description	The BEA CONNECT SNA v2.0 license does not include the license to run request at sync level 2.
	Action	Contact your BEA sale representative.
	See Also	<i>None</i>
90xx: INFO	SNACRM Message: <string1>	
	Description	ERROR or informational message sent from SNACRM to GWSNAX, and logged in the USERLOG file. Refer to SNACRM Messages.
	Action	Refer to SNACRM message file.
	See Also	<i>None</i>

E Sample VTAM Configurations

Introduction

This appendix provides sample environments showing how BEA Connect SNA can be configured for use with an Ethernet LAN and an APPN system 390. Considerations for token ring and subarea-style configurations are included. It is assumed that hardware and operating system installation have been completed.

An environment properly configured for BEA Connect SNA involves two general components, a local environment and remote environment.

Local Environment

A local environment is a UNIX-based machine running BEA Connect SNA 2.0. Connect SNA 2.0 is a fully bi-directional program, supporting the local system as either a client or server. This environment consists of the following components:

- ◆ Hardware
 - ◆ Any workstation and network interface supported by the required software.
- ◆ Software
 - ◆ Operating Systems supporting:
Solaris 2.5.1

HPUX 10.20
AIX 4.2.1

- ◆ SNA Protocol Stacks (aka PU servers) supporting:
 - Sunlink 9.1
 - Brixton 4.0
 - HPSnaPlus2
 - IBM Communications Server for AIX
- ◆ Tuxedo 6.4

Remote Environment

A remote environment is an IBM mainframe that may or may not be on the same local network. As in the local environment, BEA Connect SNA 2.0 is a fully bi-directional program, supporting the remote system as either a client or server. This environment consists of the following components:

- ◆ Hardware
 - ◆ Any workstation and network interface supported by the required software.
- ◆ Software
 - ◆ MVS 5.22 9510 or higher, including OS/390 through 1.3
 - ◆ VTAM 4.3 or higher
 - ◆ CICS 3.3 or higher
 - ◆ IMS/ESA DC 5.01 (optional)
 - ◆ Support software for a P390 or R390 (optional)

Sample Environments

Samples of each environment are provided to illustrate a starting point when first configuring your system. These samples are not intended to be used without modifications. Any similarity between them and any actual system is coincidental.

Machine Attributes (LAN Descriptions)

The attributes of the sample environment machines are listed below for reference. The sample configurations refer to these attributes as required.

10BaseT carrying SNA/DLC (IEEE 802.3) and TCP/IP (DIX) traffic.

Table E-1 SPARCstation 5

Name	Attribute
OS	Solaris 2.5.1 (SunOS 5.5.1)
SNA	Brixton/CNT 4.0
APP	Tuxedo 6.4
MAC	08:00:20:7C:47:50
IP	206.189.43.14
NAME	beasun2

Table E-2 SPARCstation 5

Name	Attribute
OS	Solaris 2.5.1 (SunOS 5.5.1)
SNA	SunLink 9.0
APP	Tuxedo 6.4
MAC	08:00:20:87:47:2d
IP	206.189.43.54
NAME	dalsun4

Table E-3 HP 9000/847

Name	Attribute
OS	HP-UX B.10.20 (patches:PHNE_9663,976 1,9889)
SNA	HPSNAPlus2 5.1
APP	Tuxedo 6.4
MAC	08:00:09:30:24:77
IP	206.189.43.13
NAME	dalhp10

Table E-4 P390 Server 500

Name	Attribute
OS	MVS 5.22 9510
SNA	VTAM 4.3
APP	CICS 4.1 / IMS DC 5.1
MAC	10:00:5a:d4:3e:8e
IP	206.189.43.98
NAME	beavs

Table E-5 P390 Server 500

Name	Attribute
OS	OS/390 1.2
SNA	VTAM 4.3
APP	CICS 4.1
MAC	10:00:5a:d4:c1:e0
IP	206.189.43.96

Table E-5 P390 Server 500

Name	Attribute
NAME	dalvs2

Local Environment Configuration

(For future reference.) HPSNAPLus2 configurations are usually setup using the HP xSnapAdmin utility, resulting in the configuration text file `/etc/opt/sna/sna_node.cfg`. This file can be manually created and/or maintained using a text editor, however, using the HP xSnapAdmin utility is recommended. The example below is the `sna_node.cfg` file for the sample environment.

HPSNAPLus2 Configuration

[define_node_config_file]

```
major_version = 5
minor_version = 1
update_release = 1
revision_level = 116
```

[define_node]

```
node_name = dalhp10
description = snacrm development
node_type = END_NODE
fqcp_name = BEALAN.DALHP10
cp_alias = dalhp10
mode_to_cos_map_supp = NO
mds_supported = YES
node_id = <05ffffff>
max_locates = 100
dir_cache_size = 255
max_dir_entries = 0
locate_timeout = 60
reg_with_nn = YES
reg_with_cds = YES
mds_send_alert_q_size = 100
cos_cache_size = 24
tree_cache_size = 40
tree_cache_use_limit = 40
```

```
max_tdm_nodes = 0
max_tdm_tgs = 0
max_isr_sessions = 1000
isr_sessions_upper_threshold = 900
isr_sessions_lower_threshold = 800
isr_max_ru_size = 16384
isr_rcv_pac_window = 8
store_endpt_rscvs = NO
store_isr_rscvs = NO
store_dlur_rscvs = NO
dlur_support = YES
pu_conc_support = NO
nn_rar = 128
ptf_flags = NONE
```

[define_ethernet_dlc]

```
dlc_name = ETHER0
description = ""
neg_ls_supp = YES
initially_active = NO
adapter_number = 0
```

[define_ethernet_port]

```
port_name = ethl0
description = 1st ethernet adapter
dlc_name = ETHER0
port_type = PORT_SATF
port_number = 1
max_rcv_btu_size = 1033
tot_link_act_lim = 64
inb_link_act_lim = 0
out_link_act_lim = 0
ls_role = LS_NEG
act_xid_exchange_limit = 9
nonact_xid_exchange_limit = 5
ls_xmit_rcv_cap = LS_TWS
max_ifrm_rcvd = 7
target_pacing_count = 7
max_send_btu_size = 1033
mac_address = <000000000000>
lsap_address = 0x08
implicit_cp_cp_sess_support = NO
implicit_limited_resource = NO
implicit_deact_timer = 0
```

```

effect_cap = 3993600
connect_cost = 0
byte_cost = 0
security = SEC_NONSECURE
prop_delay = PROP_DELAY_LAN
user_def_parm_1 = 0
user_def_parm_2 = 0
user_def_parm_3 = 0
initially_active = YES
test_timeout = 5
test_retry_limit = 2
xid_timeout = 5
xid_retry_limit = 2
t1_timeout = 5
t1_retry_limit = 5

```

[define_ethernet_ls]

```

ls_name = P390HP10
description = P390 – beavs
port_name = eth10
adj_cp_name = P390.USS3270
adj_cp_type = LEARN_NODE
mac_address = <0020af543176>
lsap_address = 0x08
auto_act_supp = NO
tg_number = 0
limited_resource = NO
solicit_sscp_sessions = NO
pu_name = <0000000000000000>
disable_remote_act = NO
default_nn_server = NO
dspu_services = NONE
dspu_name = <0000000000000000>
dlus_name = <00000000000000000000000000000000>
bkup_dlus_name = <00000000000000000000000000000000>
link_deact_timer = 0
use_default_tg_chars = YES
ls_attributes = SNA
adj_node_id = <00000000>
local_node_id = <00000000>
cp_cp_sess_support = YES
effect_cap = 3993600
connect_cost = 0
byte_cost = 0

```

```
security = SEC_NONSECURE
prop_delay = PROP_DELAY_LAN
user_def_parm_1 = 0
user_def_parm_2 = 0
user_def_parm_3 = 0
target_pacing_count = 7
max_send_btu_size = 1033
ls_role = USE_PORT_DEFAULTS
initially_active = NO
react_timer = 30
react_timer_retry = 65535
test_timeout = 5
test_retry_limit = 2
xid_timeout = 5
xid_retry_limit = 2
t1_timeout = 5
t1_retry_limit = 5
```

[define_local_lu]

```
lu_name = LUHP10A
description = Test LU #1
lu_alias = LUHP10A
nau_address = 0
syncpt_support = YES
lu_session_limit = 0
default_pool = NO
pu_name = <000000000000000000>
sys_name = ""
timeout = -1
back_level = NO
```

[define_local_lu]

```
lu_name = LUHP10B
description = Test LU #2
lu_alias = LUHP10B
nau_address = 0
syncpt_support = YES
lu_session_limit = 0
default_pool = NO
pu_name = <000000000000000000>
sys_name = ""
timeout = -1
back_level = NO
```

[define_local_lu]

```
lu_name = LUHP10C
description = Test LU #3
lu_alias = LUHP10C
nau_address = 0
syncpt_support = YES
lu_session_limit = 0
default_pool = NO
pu_name = <0000000000000000>
sys_name = ""
timeout = -1
back_level = NO
```

[define_partner_lu]

```
description = APPC MVS LU for IMS
fqplu_name = P390.MVSLU01
plu_alias = MVSLU01
plu_un_name = MVSLU01
max_mc_ll_send_size = 32767
conv_security_ver = NO
parallel_sess_supp = YES
```

[define_partner_lu]

```
description = backend cics #1
fqplu_name = P390.C410XB01
plu_alias = C410XB01
plu_un_name = C410XB01
max_mc_ll_send_size = 32767
conv_security_ver = NO
parallel_sess_supp = YES
```

[define_partner_lu]

```
description = Second backend cics
fqplu_name = P390.C410XB02
plu_alias = CICS2
plu_un_name = C410XB02
max_mc_ll_send_size = 32767
conv_security_ver = NO
parallel_sess_supp = YES
```

[define_mode]

```
mode_name = SMSNA100
description = Sessions: 10 -- 5,5
max_ru_size_upp = 1024
```

```
receive_pacing_win = 4
default_ru_size = YES
max_neg_sess_lim = 256
plu_mode_session_limit = 10
min_conwin_src = 5
cos_name = #CONNECT
cryptography = NONE
auto_act = 0
```

Remote Environment Configurations

You must involve your mainframe system support personnel early in the process. In a large shop there will most likely be separate individuals responsible for MVS, VTAM, and CICS. Make sure everyone is involved. Hopefully most of the configuration for your mainframe has already been done.

These samples are provided for illustration. Mainframe technical support is not trivial, and this is not intended to train you in all of the possible configurations. These samples represent one way a P390 can be configured to work in an APPN LAN environment.

ATCSTRxx VTAM Start List

The example below is the VTAM start list for the BEAVS P390 machine. It supports both the subarea and APPN environments.

```
*
-----
* VTAM START LIST FOR SYS1
-----
BN=YES ,                                X
BNDYN=FULL ,                            X
XNETALS=YES ,                           X
SSCPID=06 ,NOPROMPT ,                   X
CONFIG=00 ,MAXSUBA=31 ,SUPP=NOSUP ,      X
SSCPNAME=USS3270 ,                       X
NETID=P390 ,                             X
NODETYPE=NN ,                            X
HOSTSA=6 ,                               X
CRPLBUF=( 208 , ,15 , ,1,16 ) ,          X
IOBUF=( 100,512,19 , ,1,20 ) ,            X
LFBUF=( 104 , ,0 , ,1,1 ) ,              X
LPBUF=( 64 , ,0 , ,1,1 ) ,                X
SFBUF=( 163 , ,0 , ,1,1 )
```

XCA Major Node Defines the LAN Adapter for SYS1

This definition is set up for use with a 3172 (emulated) for connecting an APPN network node to another APPN node. Note that it is for an Ethernet LAN and the SAPADDR specified must be the same as the LSap specified for the local link station.

```
*****
*   Emulated 3172 XCA MAJOR NODE FOR HOST beavs
*****
XETH2LP1 VBUILD TYPE=XCA ** EXTERNAL COMMUNICATION ADAPT**
PORTE2   PORT  ADAPNO=1,      ** 3172 RELATIVE ADAPTER NUMBER**
          CUADDR=E22,        ** CHANNEL UNIT ADDRESS          **
          MEDIUM=CSMACD,    ** LAN TYPE=ETHERNET              **
          SAPADDR=8,         ** SERVICE ACCESS POINT ADDRESS**
          TIMER=120          ** CHANNEL ACTIVATE RESP TIME    **
*
G1ETH2   GROUP DIAL=YES,      ** YES required for putype 2    **
          DYNPU=YES,
          CALL=INOUT,
          ANSWER=ON,
          ISTATUS=ACTIVE
LETH20   LINE
PETH20   PU
LETHE3   LINE
PETHE3   PU
LETHF3   LINE
PETHF3   PU
```

Switched Network (SWNET) Definitions

The three switched network definition examples in this section specify the VTAM PU, representing the local link stations that expect to connect with the host machine. The 10BLK and IDNUM definitions are provided to support 3270 traffic and must be unique, as well as match the values specified in the local link definition.

SWNET MAJOR NODE (DALHP10)

```

SWNETHHP      VBUILD  TYPE=SWNET ,MAXNO=3 ,MAXGRP=3
P390HP10      PU      ADDR=02 ,
                  IDBLK=05F ,
                  IDNUM=FFFFFF ,
                  PUTYPE=2 ,
                  NETID=BEALAN ,
                  CPNAME=DALHP10 ,
                  MAXPATH=3 ,
                  DWACT=YES ,
                  CONNTYPE=APPN ,
                  CPCP=YES ,
                  DYNLU=YES

* -----
* SNA SAP & HP10 MAC ADDRESS BIT REVERSED FOR TRFMT
* -----

PATHHP        PATH  DIALNO=00041000900C24EE ,
                  GRPNM=G1ETH2

LUHP10A      LU      LOCADDR=0
LUHP10B      LU      LOCADDR=0
LUHP10C      LU      LOCADDR=0

```

SWNET MAJOR NODE (SUN2)

```
SWNETH2      VBUILD  TYPE=SWNET,MAXNO=3,MAXGRP=3
P390ETH2     PU   ADDR=04,
              IDBLK=019,
              IDNUM=10092,
              PUTYPE=2,
              NETID=BEALAN,
              CPNAME=SUN2,
              MAXPATH=3,
              DWACT=YES,
              CONNTYPE=APPN,
              CPCP=YES,
              DYNLU=YES

* -----
* SNA SAP & SUN2 MAC ADDRESS BIT REVERSED FOR TRFMT
* -----

PATH01       PATH DIALNO=00081000043EE20A,
              GRPNM=G1ETH2

LUSUN2A     LU   LOCADDR=0
LUSUN2B     LU   LOCADDR=0
LUSUN2C     LU   LOCADDR=0
```

SWNET MAJOR NODE ((SUN4)

```

SWNESUN4    VBUILD  TYPE=SWNET ,MAXNO=3 ,MAXGRP=3
P390ETH4    PU    ADDR=03 ,
              IDBLK=018 ,
              IDNUM=10092 ,
              PUTYPE=2 ,
              NETID=BEALAN ,
              CPNAME=SUN4 ,
              MAXPATH=3 ,
              DWACT=YES ,
              CONNTYPE=APPN ,
              CPCP=YES ,
              DYNLU=YES

* -----
*  SNA SAP & SUN4 MAC ADDRESS BIT REVERSED FOR TRFMT
* -----

PATHSL4      PATH  DIALNO=0008100002E1E2B4 ,
              GRPNM=G1ETH2

LUSUN4A    LU    LOCADDR=0
LUSUN4B    LU    LOCADDR=0
LUSUN4C    LU    LOCADDR=0

```

VTAM Application Major Nodes for CICS Regions

These examples represent the partner LU definitions to be accessed from the local environment. The APPL names must match those specified in the partner LU definitions on the local machine.

```
BEACICS VBUILD TYPE=APPL                      APPLICATION MAJOR NODE
* APPL DEFINITION STATEMENTS FOR CICS
* CICS 4.10 BACKEND REGION #1 SYSID=B41A
C410XB01 APPL EAS=64,                          ESTIMATED CONCURRENT SESSIONS
      MODETAB=ISTINCLM,                        MAKE SURE DEFAULT MODETAB
      PARSESS=YES,
      AUTH=(ACQ,BLOCK,PASS) CICS CAN ACQUIRE & PASS TMLS
                                CICS CAN REQUEST BLOCKED INPUT

C410XB02 APPL EAS=64,                          ESTIMATED CONCURRENT SESSIONS
      MODETAB=ISTINCLM,                        MAKE SURE DEFAULT MODETAB
      PARSESS=YES,
      AUTH=(ACQ,BLOCK,PASS) CICS CAN ACQUIRE & PASS TMLS
                                CICS CAN REQUEST BLOCKED INPUT

C410XB03 APPL EAS=64,                          ESTIMATED CONCURRENT SESSIONS
      MODETAB=ISTINCLM,                        MAKE SURE DEFAULT MODETAB
      PARSESS=YES,
      AUTH=(ACQ,BLOCK,PASS) CICS CAN ACQUIRE & PASS TMLS
                                CICS CAN REQUEST BLOCKED INPUT

* #####
* END OF BEACICS APPL DEF
* #####
```

CICS Resource Definition Entries (RDO)

CICS connection and session definitions map the VTAM path definitions for the CICS application. Each connection represents one local LU definition in the local SNA configuration, therefore, the names must match.

CICS session definitions associate a VTAM mode with the LU specified in the connection. The mode names and session count characteristics must match those specified in the mode definitions for the local SNA configuration. Note that these definitions set AUTOCONNECT to “YES,” allowing automatic session acquisition for a CICS client application.

```
LIST GROUP(BEAHP10) OBJECTS
GROUP NAME: BEAHP10
-----
CONNECTIONS:          FHPA    FHPB    FHPC
SESSIONS:             FHPA    FHPB    FHPC

CONNECTION(FHPA)      GROUP(BEAHP10)
DESCRIPTION(1ST HP SNAP2+ CONNECTION)

CONNECTION-IDENTIFIERS
  NETNAME(LUHP10A)          INDSYS()
REMOTE-ATTRIBUTES
  REMOTESYSTEM() REMOTENAME() REMOTESYSNET()
CONNECTION-PROPERTIES
  ACCESSMETHOD(VTAM)        PROTOCOL(APPC)    CONNTYPE()
  SINGLESESS(NO)            DATASTREAM(USER)  RECORDFORMAT(U)
  QUEUELIMIT(NO)            MAXQTIME(NO)
OPERATIONAL-PROPERTIES
  AUTOCONNECT(NO)           INSERVICE(YES)
SECURITY
  SECURITYNAME()             ATTACHSEC(LOCAL)  BINDSECURITY(NO)
  USEDFLTUSER(NO)
RECOVERY
  PSRECOVERY(SYSDEFAULT)
```

CONNECTION(FHPB)	GROUP(BEAHP10)	
	DESCRIPTION(2ND HP SNA+ 2 CONNECTION)	
CONNECTION-IDENTIFIERS		
NETNAME(LUHP10B)	INDSYS()	
REMOTE-ATTRIBUTES		
REMOTESYSTEM()	REMOTENAME()	REMOTESYSNET()
CONNECTION-PROPERTIES		
ACCESSMETHOD(VTAM)	PROTOCOL(APPC)	CONNTYPE()
SINGLESESS(NO)	DATASTREAM(USER)	RECORDFORMAT(U)
QUEUELIMIT(NO)	MAXQTIME(NO)	
OPERATIONAL-PROPERTIES		
AUTOCONNECT(YES)	INSERVICE(YES)	
SECURITY		
SECURITYNAME()	ATTACHSEC(LOCAL)	BINDSECURITY(NO)
USEDFTUSER(NO)		
RECOVERY		
PSRECOVERY(SYSDEFAULT)		

CONNECTION(FHPC)	GROUP(BEAHP10)	
	DESCRIPTION(3RD HP SNA+ 2 CONNECTION)	
CONNECTION-IDENTIFIERS		
NETNAME(LUHP10C)	INDSYS()	
REMOTE-ATTRIBUTES		
REMOTESYSTEM()	REMOTENAME()	REMOTESYSNET()
CONNECTION-PROPERTIES		
ACCESSMETHOD(VTAM)	PROTOCOL(APPC)	CONNTYPE()
SINGLESESS(NO)	DATASTREAM(USER)	RECORDFORMAT(U)
QUEUELIMIT(NO)	MAXQTIME(NO)	
OPERATIONAL-PROPERTIES		
AUTOCONNECT(NO)	INSERVICE(YES)	
SECURITY		
SECURITYNAME()	ATTACHSEC(LOCAL)	BINDSECURITY(NO)
USEDFTUSER(NO)		
RECOVERY		
PSRECOVERY(SYSDEFAULT)		

SESSIONS(FHPA)		GROUP(BEAHP10)
DESCRIPTION(1ST HP SNAP2+ SESSION)		
SESSION-IDENTIFIERS		
CONNECTION(FHPA)	SESSNAME()	NETNAMEQ()
MODENAME(SMSNA100)		
SESSION-PROPERTIES		
PROTOCOL(APPC)	MAXIMUM(32,16)	RECEIVEPFX()
RECEIVECOUNT()	SENDPFX()	SENDCOUNT()
SENDSIZE(4096)	RECEIVESIZE(4096)	SESSPRIORITY(0)
PRESET-SECURITY		
USERID()		
OPERATIONAL-PROPERTIES		
AUTOCONNECT(NO)	BUILDCHAIN(YES)	USERAREALEN(0)
IOAREALEN(0,0)	RELREQ(NO)	DISCREQ(NO)
NEPCLASS(0)		
RECOVERY		
RECOVOPTION(SYSDEFAULT)		

SESSIONS(FHPB)		GROUP(BEAHP10)
DESCRIPTION(2ND HP SNAP2+ SESSION)		
SESSION-IDENTIFIERS		
CONNECTION(FHPB)	SESSNAME()	NETNAMEQ()
MODENAME(SMSNA100)		
SESSION-PROPERTIES		
PROTOCOL(APPC)	MAXIMUM(32,16)	RECEIVEPFX()
RECEIVECOUNT()	SENDPFX()	SENDCOUNT()
SENDSIZE(4096)	RECEIVESIZE(4096)	SESSPRIORITY(0)
PRESET-SECURITY		
USERID()		
OPERATIONAL-PROPERTIES		
AUTOCONNECT(YES)	BUILDCHAIN(YES)	USERAREALEN(0)
IOAREALEN(0,0)	RELREQ(NO)	DISCREQ(NO)
NEPCLASS(0)		
RECOVERY		
RECOVOPTION(SYSDEFAULT)		

SESSIONS(FHPC)		GROUP(BEAHP10)
DESCRIPTION(3RD HPSNAP2+ SESSION)		
SESSION-IDENTIFIERS		
CONNECTION(FHPC)	SESSNAME()	NETNAMEQ()
MODENAME(SMSNA100)		
SESSION-PROPERTIES		
PROTOCOL(APPC)	MAXIMUM(10,5)	RECEIVEPFX()
RECEIVECOUNT()	SENDPFX()	SENDCOUNT()
SENDSIZE(4096)	RECEIVESIZE(4096)	SESSPRIORITY(0)
PRESET-SECURITY		
USERID()		
OPERATIONAL-PROPERTIES		
AUTOCONNECT(YES)	BUILDCHAIN(YES)	USERAREALEN(0)
IOAREALEN(0,0)	RELREQ(NO)	DISCREQ(NO)
NEPCLASS(0)		
RECOVERY		
RECOVOPTION(SYSDEFAULT)		

Glossary

A

Access Control Lists (ACL)

A Tuxedo security feature that controls client access to services by means of lists that are automatically checked each time a service is requested.

ACID Properties

The essential characteristic of transaction processing systems:

Atomicity: All changes that a transaction makes to a database are made permanent, or else are nullified.

Consistency: A successful transaction transforms a database from a previous valid state to a new valid state.

Isolation: Changes that a transaction makes to a database are not visible to other operations until the transaction completes its work.

Durability: Changes that a transaction makes to a database survive future system or media failures.

Application

A BEA TUXEDO System/T *application* is bounded by the environment described in a single TUXCONFIG file. In /Domain, a BEA TUXEDO System/T application can communicate with another application via a domain gateway group.

Application Domain

When used alone, the term *Domain* can mean a number of things. In order to avoid confusion, the term *application domain* is used to refer to a BEA TUXEDO application bounded by the configuration of a `tmconfig` file. This application domain can be restricted to a single platform, or shared memory (SHM) environment, or could be scaled across multiple machines in a multiple processor (MP) environment.

Application Programming Interface (API)

1) The verbs and environment that exist at the application level to support a particular system software product. 2) A set of code that enables a developer to initiate and complete client/server requests within an application. 3) A set of calling conventions that define how to invoke a service. A set of well-defined programming interfaces (entry points, calling parameters, and return values) by which one software program utilizes the services of another

Application Program-to-Program Communication (APPC)

An interface to LU6.2 services; provides a set of primitives to conduct conversations in LU6.2 sessions.

Application-Transaction Monitor Interface (ATMI)

The Application Programming Interface (API) to Tuxedo that includes transaction routines, message handling routines, service interface routines, and buffer management routines.

C

Client

A program designed to request information from a server.

CNOS

CNOS are service programs implemented as part of an LU6.2. The *CNOS* programs negotiate session limits between the two communication LU's.

Conversation

In this guide *conversation* has two meanings; the context determines which meaning is intended. In BEA TUXEDO System/T, conversation identifies a mode of communication between processes in which a connection is opened and stays open until brought down. Communication is achieved through sends and receives. This is distinguished from the request/response model in which communication is achieved through calls and replies. In SNA terms, a conversation uses a session as long as the conversation continues. In an SNA conversation, communication can be either the BEA TUXEDO System/T conversation or request/response model. Each SNA conversation is assigned a CONVID (Conversation ID) at the time it is initialized by the LU. SNA conversations can be *mapped* or *basic*:

Conversation, Mapped

Conversations that allow programmers to send and receive buffers without having to worry about the sizes of underlying request units (RUs) used for communication. The LU takes the buffer and divides it, if necessary, into appropriate Logical Records with associated length fields and data type fields. This is the style supported in BEA Connect SNA for applications.

Conversation, Basic

Conversations in which logical records with appropriate type and length fields must be formatted for transmission and parsed on receipt. The service transaction programs in an LU use basic conversations to communicate.

Common Programming Interface for Communications (CPI-C)

An interface to LU6.2 services. It is a simpler set of primitives than the APPC interface and is intended for use in program-to-program communications.

Customer Information Control System/Extended System Architecture (CICS/ESA)

An operating environment devised by IBM that provides a foundation upon which to write customer applications programs. Several facilities useful for programming are supplied by the CICS environment, including basic mapping services (BMS), transient data queues (TD), temporary storage files (TS), memory services, etc. Customer applications are built as separate transaction programs, and are invoked as transactional tasks. CICS/ESA is a trademark of International Business Machines (IBM), Inc.

D

Distributed Program Link (DPL)

Function of CICS ISC that supports LINK requests between CICS regions, and is similar to a BEA TUXEDO request/response.

Distributed Transaction Processing (DTP)

A CICS intercommunication in which processing is distributed among transactions that communicate synchronously over intersystem or inter-region links. It is roughly equivalent to BEA TUXEDO conversations.

Domain

A *domain* can be another BEA TUXEDO System/T application that is independently administered, an application that is under the control of another transaction processing system, or an application in a remote CICS/ESA region. Domains can be local or remote.

Domain Gateway

A BEA TUXEDO System/T process that provides connectivity to remote BEA TUXEDO application environments, such as OSI, MVS/APPC, CICS/MVS, and IMS operating environments. BEA CONNECT OSI, BEA CONNECT SNA, and BEA CONNECT TCP/IP are domain gateways.

Domain Gateway Group

A *Domain Gateway Group* is a collection of domain gateway processes that provide communication services with other domains.

E

ESA

(ESA) Enterprise Systems Architecture is the conceptual structure and functional behavior of IBM's latest range of mainframe computers. ESA/370 is the fourth step in an evolution of which the first three steps were System/360, System/370, and System/370 extended architecture (370-XA).

F

Field Manipulation Language (FML)

A set of C language functions for defining and manipulating storage structures called field buffers. Cooperating processes can send and receive data in fielded buffers.

FML Buffer

A buffer of self-describing data items accessed through the Field Manipulation Language API.

G

Graphical Administrative Interface

A Tuxedo System component that enables an authorized user to configure and control an application through a Motif-based set of screens and icons.

I

Inbound

A generic term referring to request message direction relative to the server, or response message direction relative to the client.

InterSystem Communications (ISC)

Communication between separate systems by means of SNA networking facilities or by means of the application-to-application facilities. ISC links CICS systems to other systems, and may be used for communication between user applications, or to transparently execute CICS functions on a remote CICS system.

Information Management System (IMS)

A database manager used by CICS/ESA to allow access to data. IMS provides for the arrangement of data in an hierarchical structure and a common access approach in application programs that manipulate IMS databases.

J

Job Control Language (JCL)

Control language used to describe a job and its requirements to an operating system.

L

Local Domain

A *Local Domain* is a part of an application (set or subset of 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 Service

A *Local Service* is a service of a local domain that is made available to remote domains through a Domain Gateway Group.

Logical Unit (LU)

In SNA, a port through which a user gains access to the services of a network. Also, see System Network Architecture (SNA).

LU6.2

LU6.2 is a particular SNA logical unit that identifies a specific set of services for program to program communication. Services include syncpoint, mapping of buffers into records, message confirmation, and security.

M

MODENAME

MODENAME is a configuration parameter that names a set of characteristics for a group of BEA Connect SNA sessions. In the CICS region, the mode is defined in VTAM and referenced in CIC and the DMCONFIG file.

Multiple Virtual Storage (MVS)

An operating system for processing systems consisting of one or more mainframe processors.

O

Outbound

A generic term referring to request message direction relative to the client, or response message direction relative to the server.

P

Partitioned Data Set (PDS).

A CICS/ESA data set in direct access storage that is divided into partitions called members. A member can contain a program or data. Program libraries are held in partitioned data sets.

R

re-entrant

The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks.

Remote Domain

A *Remote Domain* is a part of an application accessed through a Domain Gateway Group. The remote domain may be another BEA TUXEDO System/T application, an application running under another TP system, or a BEA Connect SNA application.

Remote Service

A *Remote Service* is a service of a remote domain that is made available to the local application through a Domain Gateway Group.

Resource Definition Online (RDO)

The recommended method of defining resources to CICS/ESA. Resource definitions are created interactively by a CEDA transaction, or by the DFHCSDUP utility. Both methods store definition in the CICS/ESA system definition data set (CSD). At CICS initialization, CSD definitions are selectively installed as CICS system tables controlled by a user-supplied list of definitions. CEDA-defined resource definitions can be installed while CICS is active and used immediately.

S

Server

A computer or program that is dedicated to providing information in response to external requests.

Session

When two LU's bind with each other, that is, when they have successfully negotiated how they will communicate, they are said to be in *session*. SNA has fixed limits on the number of sessions configurable for an LU type.

Stack

Platform vendor-supplied software that provides connectivity to an SNA network.

Synchronization Level (sync level)

The level of synchronization (0, 1, or 2) established for an APPC session between intercommunicating CICS/ESA transactions. Level 0 gives no synchronization support, level 1 allows the exchange of private synchronization requests, and level 2 gives full CICS/ESA synchronization support, with backout of all updates to recoverable resources if failure occurs.

System Definition Data Set (CSD)

A VSAM KSDS cluster that contains a resource definition record for every resource defined to CICS using resource definition online (RDO).

System Network Architecture (SNA)

A seven-layer networking protocol. Each layer of the protocol has a set of associated data communication services. The services of the uppermost layer are embodied in a Logical Unit (LU). Each LU type defined in SNA has its own specific set of services available to an end user for communicating. The end user may be a terminal device, or an application program. The SNA structure enables the end user to operate independently, unaffected by the specific facilities used for information exchange.

SNA Communication Resource Manager (SNACRM)

A process that provides all of the sync-level two logic for an SNA domain gateway and directly communicates with the PU2.1 server.

SYM_DEST_NAME

A symbolic name for a combination of Partner LUNAME, MODENAME and TPNAME that uniquely identifies the destination for a conversation start-up request.

T

Transaction

- 1) A complete unit of work that transforms a database from one consistent state to another. In DTP, a transaction can include multiple units of work performed on one or more systems.
- 2) A logical construct through which applications perform work on shared resources (e.g., databases). The work done on behalf of the transaction conforms to the four ACID Properties: atomicity, consistency, isolation, and durability.

Transaction Processing (TP)

A form of immediate data processing in which user requests are entered directly to the terminal and on-line programs satisfy the requests; for example, by updating database files and displaying output messages.

Typed Buffer

A buffer for message communication involving data of a specific data type.

V

Virtual Telecommunications Access Method (VTAM)

A set of programs that control communication across a network between terminals and application programs.

