BEA eLink Adapter into WebLogic Enterprise CORBA

User Guide

BEA eLink Adapter into WebLogic Enterprise CORBA 1.0
Document Edition 1.0
July 2000

Copyright

Restricted Rights Legend

Trademarks or Service Marks

BEA eLink Adapter into WebLogic Enterprise CORBA User Guide

Document Edition	Part Number	Date	Software Version
1.0		July 2000	BEA eLink Adapter into WebLogic Enterprise CORBA 1.0

Contents

4. eLink Adapter for CORBA Configuration

5. Running BEA eLink Adapter for CORBA

A. Using the Sample Application

B. Error and Information Messages

C. Troubleshooting

D. BEA eLink Platform Reference

Glossary

Index

About This Document

This document describes the BEA eLink Adapter for CORBA component and provides instructions on how to provide reliable communication between eLink components and CORBA-based applications using the eLink infrastructure and eLink Adapter into WebLogic Enterprise CORBA.

The *BEA eLink Adapter into WebLogic Enterprise CORBA User Guide* is organized as follows:

- *Understanding the BEA eLink Solution* introduces the eLink Adapter for CORBA component and explains how eLink Adapter for CORBA fits into the BEA eLink Platform environment.
- *Installing BEA eLink Adapter for CORBA* provides instructions for installing the eLink Adapter for CORBA.
- *Understanding CORBA Objects* provides information on what a CORBA object is and the object terminology used throughout the eLink Adapter for CORBA information set.
- *eLink Adapter for CORBA Configuration* provides information for configuring the servers required to run eLink Adapter for CORBA.
- *Running BEA eLink Adapter for CORBA* provides information for booting and shutting down the application.
- *Using the Sample Application* provides the information you need to build a client application to access the WLE CORBA Simpapp application to test your installation.
- *Error and Information Messages* describes error and informational messages as well as actions to resolve the errors.

- *Troubleshooting* provides various solutions to problems you might encounter while using the eLink Adapter for CORBA.
- *BEA eLink Platform Reference* provides you with basic information about BEA eLink Platform and FML buffers
- *Glossary* provides a list of terms used within this document.

What You Need to Know

This document is intended for system administrators who will install the eLink Adapter for CORBA on various platforms. It is also intended for programmers who will configure the eLink Adapter for CORBA and set up eLink Platform services to execute information transfers between CORBA based applications and third-party applications. This guide assumes knowledge of BEA eLink Platform, WebLogic Enterprise, and CORBA-based products.

e-docs Web Site

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

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser. A PDF version of this document is available on the eLink documentation Home page on the e-docs Web site (and also on the documentation CD).

You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the eLink documentation Home page, click the PDF files button and select the document you want to print. If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe Web site at http://www.adobe.com/.

Related Information

The following BEA publications are also available for more information:

- *BEA eLink Platform Online Documentation*
- *BEA WebLogic Enterprise Online Documentation*

Contact Us

Your feedback on the BEA eLink documentation is important to us. Send us e-mail at docsupport@beasys.com if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the eLink documentation. In your e-mail message, please indicate that you are using the documentation for the BEA eLink Adapter for CORBA 1.0 release.

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

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

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes

- The name and version of the eLink Adapter for CORBA you are using
- A description of the problem and the content of pertinent error messages

Document Conventions

The following documentation conventions are used throughout this document:

Item	Examples
Variable names	Variable names represent information you must supply or output information that can change; they are intended to be replaced by actual names. Variable names are displayed in italics and can include hyphens or underscores. The following are examples of variable names in text: *error_file_name* The *when-return* value...

User input and screen output	For screen displays and other examples of input and output, user input appears as in the first of the following lines; system output appears as in the second through fourth lines:
	`dir c:\accounting\data`
	`Volume in drive C is WIN_NT_1`
	`Volume Serial Number is 1234-5678`
	`Directory of C:\BEADIR\DATA`

Syntax	Code samples can include the following elements: ■ Variable names can include hyphens or underscores (e.g., *error_file_name)* ■ Optional items are enclosed in square brackets: [ ]. If you include an optional item, do not code the square brackets. ■ A required element for which alternatives exist is enclosed in braces { }. The alternatives are separated by the pipe (vertical bar) character: |. You must include only one of the alternatives for that element. Do not code the braces or pipe character. ■ An ellipsis (...) indicates that the preceding element can be repeated as necessary.

Item	Examples
Omitted code	An ellipsis (...) is used in examples to indicate that code that is not pertinent to the discussion is omitted. The ellipsis can be horizontal or vertical.
Environment variables	Environment variables are formatted in an uppercase font. ENVFILE=${APPDIR}
Key names	Key names are presented in boldface type. Press Enter to continue.
Literals	Literals are formatted in a monospace font. `class extendSample`
Window items	Window items are presented in boldface type. Window items can be window titles, button labels, text edit box names or other parts of the window. Type your password in the Logon window. Select Export to make the service available to the client.

1 Understanding the BEA eLink Solution

This section contains the following topics:

- BEA eLink Solution Overview
- BEA eLink Adapter into WebLogic Enterprise CORBA Overview

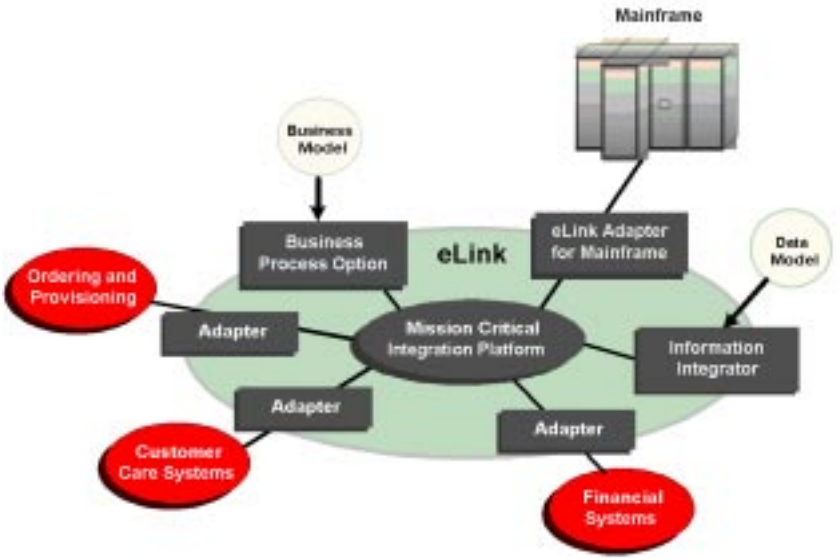
BEA eLink Solution Overview

BEA eLink™ provides an open Enterprise Application Integration (EAI) solution that allows applications throughout organizations to communicate seamlessly. Using EAI, you gain the long-term flexibility and investment protection you need to keep up with today's ever-changing business environment.

Typically, companies use packaged applications to automate internal operations, such as financial, manufacturing, or human resources. While they successfully address the needs of these specific areas, these proprietary platforms often do not work together. To compete today, you need a much greater exchange of information. Systems need to communicate at a process level within your own organization, as well as with customer's and supplier's systems. BEA eLink Platform is the underlying basis of BEA eLink, a family of off-the-shelf enterprise application integration (EAI) products. BEA eLink leverages the BEA transaction platform to integrate existing legacy applications with customer-focused and business-to-business e-commerce initiatives.

BEA eLink Platform provides a proven infrastructure for integrating applications within the enterprise and across the Web. BEA eLink Platform ensures high-performance, secure transactions and transparent access to mission-critical applications and information throughout the enterprise and across the Web. Figure 1-1 illustrates the eLink logical architecture and shows where the eLink Adapters fit into the process.

Figure 1-1 BEA eLink Solution Illustration



The entire BEA eLink family (including all options and adapters) is highly scalable. Multiple instances of BEA eLink components can collaborate so that work is divided between eLink domains. BEA eLink includes Simple Network Management Protocol (SNMP) integration for enterprise management.

The current BEA eLink Platform leverages the BEA Tuxedo infrastructure because it is based on a service-oriented architecture. Both BEA Tuxedo and BEA eLink communicate directly with each other and with other applications through the use of services. Multiple services are grouped into “application servers” or “servers”. The terms Tuxedo services/servers and eLink services/servers can be used interchangeably. Because this document is specifically addressing the eLink family, the terms “eLink service” and “eLink server” are used throughout.

The BEA eLink Platform complies with the Open Group's X/Open standards including support of the XA standard for two-phase commit processing, the X/Open **ATMI** API, and XPG standards for language internationalization. C, C++, COBOL, and Java are supported. The BEA eLink Platform connects to any RDBMS, OODBMS, file manager or queue manager, including a supplied XA-compliant queueing subsystem.

The following components operate with BEA eLink Platform:

- The **Information Integrator** translates data models used by different applications. It provides a cost-effective alternative to writing or generating programs to perform this function. It also handles complex translation with great power and scalability.
- The **Business Process Option** helps automate tasks in the distributed global business process and dynamically responds to business events and exceptions. The BPO is implemented by integrating eLink with technology based on InConcert workflow management software.
- An **eLink Adapter** provides the interface between the BEA eLink Platform and external applications with out-of-the-box functionality.

BEA eLink Adapter into WebLogic Enterprise CORBA Overview

The BEA eLink Adapter into WebLogic Enterprise CORBA (hereafter referred to as the eLink Adapter for CORBA) provides communication between WLE CORBA-based applications and applications integrated by the eLink Platform. Access to other CORBA environments is provided through WLE. eLink Adapter for CORBA uses the WLE CORBA IDL entries to generate local, WLE-based services. eLink-based applications make calls into a "remote" service that, in turn invokes a CORBA method and receives return information from the CORBA method.

Note: Third-party CORBA methods can be called indirectly provided that the foreign method is registered with WLE.

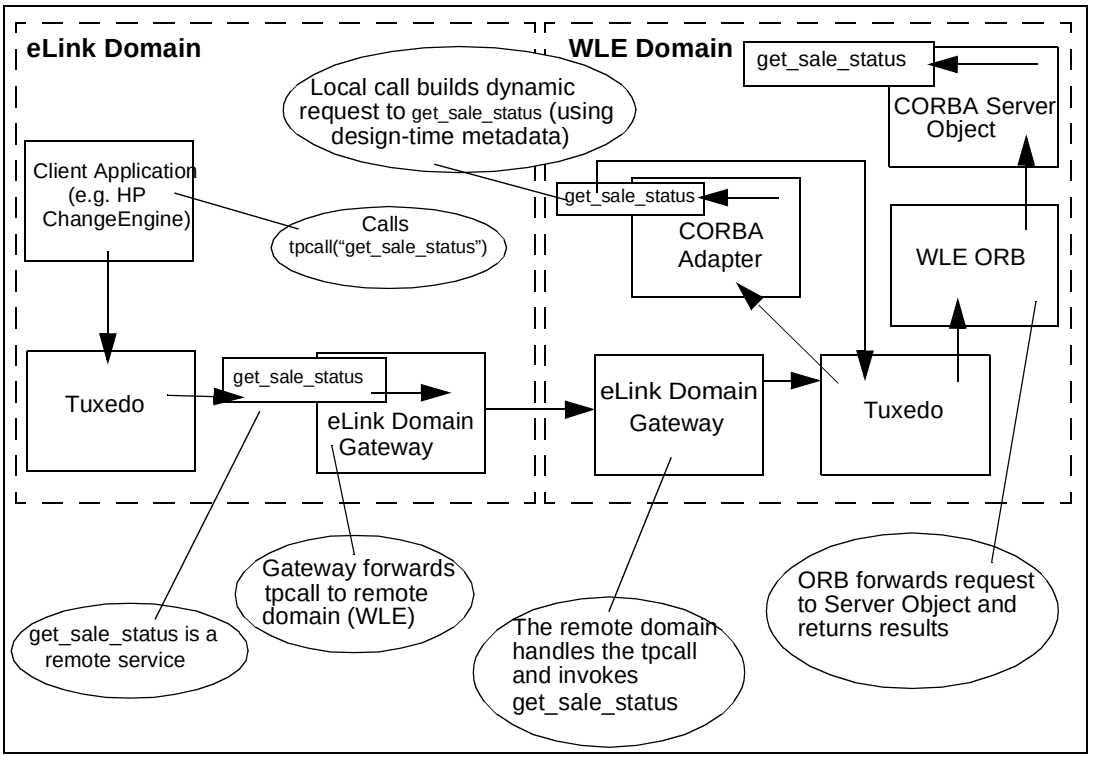
To achieve the eLink to CORBA integration, the eLink Adapter for CORBA provides a pre-built WLE object server that exposes the services necessary to represent one or more CORBA interfaces. This server translates an eLink service invocation from the eLink client into CORBA method invocation(s) on the correct CORBA object and returns any results from this invocation. The adapter uses FML32 buffers and supported data types to invoke CORBA object methods. Parameters consisting of structures or objects are not supported.

This communication is made possible using the existing eLink infrastructure and a set of application adapters. The eLink Adapter for CORBA runs within the WLE server application space and answers ATMI service requests initiated from an eLink Platform environment. These service requests can be initiated from within WLE or from a remote Tuxedo or eLink application using Domain Gateways. The eLink Adapter for CORBA provides a gateway to CORBA objects by transforming ATMI service requests to CORBA Object method invocations using the standard CORBA Dynamic Invocation Interface.

Using this adapter in conjunction with the Domains feature, ATMI service calls from anywhere on your TCP/IP network can be allowed access to CORBA objects. A domain is a collection of BEA eLink servers, services, interfaces, machines, and associated resource managers defined by a single `UBBCONFIG` (ASCII version) or `TUXCONFIG` (binary version) configuration file.

In the following illustration, a client running on the eLink Platform invokes a method of a CORBA object that returns customer order status.

Figure 1-2 eLink Adapter for CORBA Run-time Components

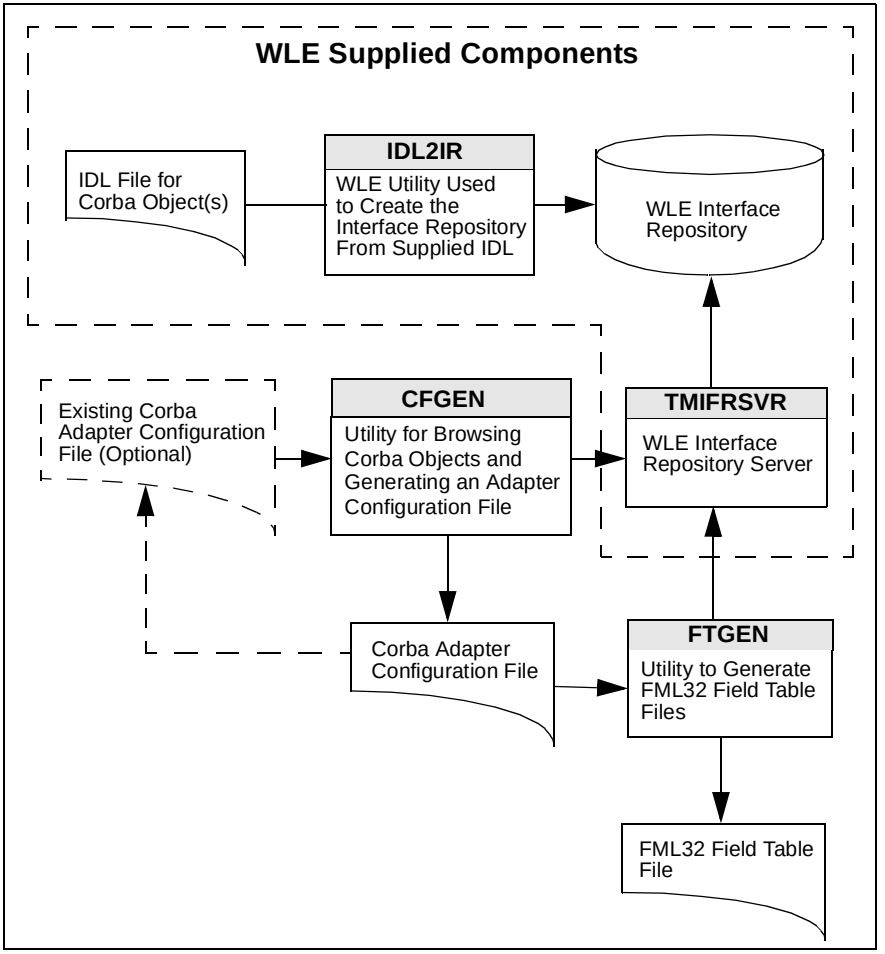


The eLink Adapter for CORBA determines the eLink business service to CORBA object method mapping via a configuration file. To facilitate creation of this file, the adapter includes command line configuration tools that browse the CORBA object information of the WLE Interface Repository and generate an adapter configuration file and an FML32 Field Table file.

- The CFGEN command line tool browses the CORBA object information in the WLE Interface Repository and generates a configuration file containing configuration parameters for object methods that can be accessed using the adapter.
- The FTGEN command line tool inputs an adapter configuration file along with the Interface Repository to create an FML32 Field Table file. The generated FML32 Field Table contains entries for all fields in the configuration file.

FML32 Field Table files are used to define the IDs and types of fields contained in an FML32 buffer.

Figure 1-3 BEA eLink Adapter for CORBA Configuration Components



2 Installing BEA eLink Adapter for CORBA

This section contains the following topics:

- Installation Prerequisites
- Installing on a UNIX-based Platform
- Installation Files and Directories

Installation Prerequisites

Complete the following tasks prior to installing eLink Adapter for CORBA:

- Read the *BEA eLink Adapter into WebLogic Enterprise CORBA Release Notes* for information on platform support, software requirements, product documentation access, and Customer Support contacts.
- Install and verify the operation of WebLogic Enterprise.

Installing on a UNIX-based Platform

BEA eLink Adapter for CORBA software runs on UNIX-based platforms. To install the UNIX-based software, you run a script that is supplied on the product CD-ROM and enter the needed information as the script progresses through the installation process. Refer to the following section for your installation instructions.

To install the eLink Adapter for CORBA software on a UNIX-based platform, you run the `install.sh` script. As the script runs, it asks you for the following information:

- Platform on which to install eLink Adapter for CORBA software
- Directory location of your WLE installation
- Whether you want to install the license file

Note: The platforms and file names shown in the following listings are examples only. These values are dependent on platform configurations for your system and may vary from the example.

Perform the following steps to run the script and install the eLink Adapter for CORBA on a supported UNIX platform:

1. Log on as root to install the eLink Adapter for CORBA.

```
$ su -
Pasword:
```

2. Access the CD-ROM device containing the BEA distribution media.

```
# ls -1 /dev/cdrom
total 0
brw-rw-rw  1 root  sys  22, 0 January 5 10:55 c1b0t010
```

3. Mount the CD-ROM.

```
# mount -r -F cdfs /dev/cdrom/c1b0t010 /mnt
```

4. Change the directory to your CD-ROM device.

```
# cd /mnt
```

5. List the CD-ROM contents.

```
# ls
```

```
install.sh  hp
```

6. Execute the installation script.

   ```
   # sh ./install.sh
   ```

7. The installation script runs and prompts you for responses as shown in Listing 2-1.

The following listing provide an example of running this script for the eLink Adapter for CORBA. The values in bold are supplied by you during installation.

Listing 2-1 eLink Adapter for CORBA Installation

```
cmadm@dalsun4:/cmhome/dist/altair-5 ls
hp          install.sh  sun5x
cmadm@dalsun4:/cmhome/dist/altair-5 sh install.sh

01) hp/hpux11        02) sun5x/sol26      03) sun5x/sol7


Install which platform's files? [01-     3, q to quit, l for list]: 2

** You have chosen to install from sun5x/sol26 **

BEA eLink Adapter into WebLogic Enterprise CORBA v1.0

This directory contains the BEA eLink Adapter into WebLogic
Enterprise CORBA System for
SunOS 5.6 (Solaris 2.6) on SPARC.


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     corba           BEA eLink Adapter into WebLogic Enterprise
                        CORBA

Select the package(s) you wish to install [?,??,q]: 1

BEA eLink Adapter into WebLogic Enterprise CORBA
(sparc) Release 1.0
```

```
Copyright (c) 2000 BEA Systems, Inc.
All Rights Reserved.
Distributed under license by BEA Systems, Inc.
BEA eLink is a trademark of BEA Systems, Inc.

Directory where eLink Adapter into WebLogic Enterprise CORBA
Adapter files are to be installed
(Enter your WebLogic Enterprise v5.1 directory path) [?,q]:
/work/cmadm/wle51


Using /work/cmadm/wle51 as the eLink Adapter into WebLogic
Enterprise CORBA base directory

Determining if sufficient space is available ...
986 blocks are required
2816896 blocks are available to /work/cmadm/wle51

Unloading /cmhome/dist/altair-5/sun5x/sol26/corba/CORBAW51.Z ...
bin/CFGEN
bin/FTGEN
bin/e2a
bin/lic.sh
eLink/catalogs/ELINKCORBA.text
eLink/elcorba/samples/e2aclient.c
eLink/elcorba/samples/e2aclient.flds
eLink/elcorba/samples/e2aclient.txt
eLink/elcorba/samples/e2alocal.dom
eLink/elcorba/samples/e2alocal.ubb
eLink/elcorba/samples/e2aremote.dom
eLink/elcorba/samples/e2aremote.ubb
eLink/elcorba/samples/makefile.nt
eLink/elcorba/samples/makefile.unix
eLink/elcorba/samples/setenv.sh
lib/libadk.so.1.10
udataobj/java/jdk/ConfigCorba.jar
960 blocks
... finished

Changing file permissions...
... finished

If your license file is accessible, you may install it now.
Install license file? [y/n]: n

Please don't forget to use lic.sh located in your product bin
directory to install the license file from the enclosed floppy.
Refer to your product Release Notes for details on how to do this.
```

```
Installation of BEA eLink Adapter into WebLogic Enterprise CORBA
was successful.

Please don't forget to fill out and send in your registration card
cmadm@dalsun4:/cmhome/dist/altair-5
```

Installation Files and Directories

The eLink Adapter for CORBA CD-ROM contains the libraries and executable programs listed in Table 2-1, Table 2-2, and Table 2-3. After installing the eLink Adapter into WebLogic Enterprise CORBA software, verify that these libraries and programs are installed on your system.

eLink Adapter for CORBA Installation

Verify that the following eLink Adapter for CORBA platform files are installed on your system.

Solaris

Verify that the following files are installed by the eLink Adapter for CORBA software for Soaris.

Table 2-1 Solaris Installation Files and Directories

Directory	Files
$TUXDIR/bin	e2a CFGEN FTGEN lic.sh
$TUXDIR/lib	libadk.so.1.10
$TUXDIR/eLink/catalogs	ELINKCORBA.text

Table 2-1 Solaris Installation Files and Directories

Directory	Files
$TUXDIR/eLink/elcorba/samples	e2aclient.c e2aclient.flds e2aclient.txt e2alocal.dom e2alocal.ubb e2aremote.dom e2aremote.ubb makefile.nt makefile.unix setenv.sh
$TUXDIR/udatobj/java/jdk	ConfigCorba.jar

HP-UX

Verify that the following files are installed by the eLink Adapter for CORBA software for HP-UX.

Table 2-2 HP-UX Installation Files and Directories

Directory	Files
$TUXDIR/bin	e2a CFGEN FTGEN lic.sh
$TUXDIR/lib	libadk.sl.1.10
$TUXDIR/eLink/catalogs	ELINKCORBA.text

Table 2-2 HP-UX Installation Files and Directories

Directory	Files
$TUXDIR/eLink/elcorba/samples	e2aclient.c e2aclient.flds e2aclient.txt e2alocal.dom e2alocal.ubb e2aremote.dom e2aremote.ubb makefile.nt makefile.unix setenv.sh
$TUXDIR/udataobj/java/jdk	ConfigCorba.jar

Sample Source Files and Directories

Verify that the following sample source files are installed by the eLink Adapter for CORBA software.

The following sample source files are contained in the `$TUXDIR/eLink/elcorba/samples` folder.

Table 2-3 Sample Source Files and Directories

Files	Content
`e2aclient.c`	Client source code.
`makefile.nt`	Windows NT makefile (MS VC++ 6)
`makefile.unix`	Unix makefile
`e2alocal.ubb`	UBB configuration file for e2a to run on UNIX and handle both local and remote requests from Windows NT.
`e2alocal.dom`	DM configuration file for e2a to run on UNIX and handle both local and remote requests from Windows NT.

Table 2-3 Sample Source Files and Directories

Files	Content
`e2aremote.dom`	DM configuration file for e2aclient to run on Windows NT and communicate with e2a on UNIX.
`e2aremote.ubb`	UBB configuration file for e2aclient to run on Windows NT and communicate with e2a on UNIX.
`setenv.sh`	Sample script in which environment variables are set.
`e2aclient.txt`	Adapter configuration file. This lists the services being advertised. This is the command line argument to e2a as specified in the CLOPT= line.
`e2aclient.flds`	Output from the FTGEN command line tool. Note that this file contains the FML32 field definitions that must manually added.

3 Understanding CORBA Objects

This section provides information on what a CORBA object is and the object terminology used throughout the eLink Adapter for CORBA information set.

Topics covered in this section include:

- Definition of a CORBA Object
- How a CORBA Object Comes Into Being
- Components of a CORBA Object
- How Object Invocation Works
- Understanding the CORBA Name Service

There are a number of variations of the definition of an object, depending on what architecture or programming language is involved. For example, the concept of a C++ object is significantly different from the concept of a CORBA object. Also, the notion of a COM object is quite different from the notion of a CORBA object.

The definition of a CORBA object is consistent with the definition presented by the Object Management Group (OMG). The OMG has a number of specifications and other documents that go into complete details on objects. For additional information, refer to the OMG web site located at HTTP://www.omg.org.

Definition of a CORBA Object

A CORBA object is a virtual entity that does not exist on its own until a client application refers to that object and requests a method on that object. The reference to the CORBA object is called an *object reference*. The object reference is the only means by which a CORBA object can be addressed and manipulated in a WLE system.

For more information about object references, see *Creating CORBA C++ Server Applications* or *Creating CORBA Java Server Applications* in the WebLogic Enterprise online documentation.

When the client or server application issues a request on an object via an object reference, the WLE server application instantiates the object specified by the object reference if the object is not already active in memory. (Note that a request always maps to a specific method invocation on an object.)

Instantiating an object typically involves the server application initializing the object's state, which may include having the object's state read from durable storage, such as a database.

The object contains all the data necessary to do the following:

- Execute the object's methods
- Store the object's state in durable storage when the object is no longer needed

How a CORBA Object Comes Into Being

The data that makes up a CORBA object may have its origin as a record in a database. The record in the database is the persistent or durable state of the object.

This record becomes accessible via a CORBA object in a WLE domain when the following sequence has occurred:

1. The server application's factory creates a reference for the object. The object reference includes information about how to locate the record in the database.

2. Using the object reference created by the factory, the client application issues a request on the object.
3. The object is instantiated. If the server application is implemented in C++, the object is instantiated by the TP Framework by invoking the `Server::create_servant` method, which exists in the Server object.

   If the server application is implemented in Java, the object is instantiated dynamically by the WLE system. (The `create_servant` method is not available to Server objects written in Java.)
4. The WLE domain invokes the `activate_object` method on the object, which causes the record containing state to be read into memory.

Whereas a language object exists only within the boundaries of the execution of the application, a CORBA object may exist across processes and machine systems. The WLE system provides the mechanism for constructing an object and for making that object accessible to the application.

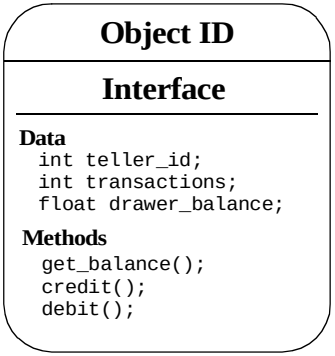
The WLE server application programmer is responsible for writing the code that initializes an object's state and the code that handles that object's state after the object is no longer active in the application. If the object has data in durable storage, this code includes the methods that read from and write to durable storage.

Components of a CORBA Object

CORBA objects typically have the following components, shown in the figure below:

- An ID, also known as an object ID, or OID

- An interface, which specifies the CORBA object's data and methods



The sections that follow describe each of these object components in detail.

The Object ID

The object ID (OID) associates an object, such as a database record,with its state and identifies the instance of the object. When the factory creates an object reference, the factory assigns an OID that may be based on parameters that are passed to the factory in the request for the object reference.

Note: The server application programmer must create the factories used in the WLE client/server application.The programmer is responsible for writing the code that assigns OIDs.

A WLE system can determine how to instantiate the object by using the following information:

- The OID
- Addressing data in the object reference
- The group ID in the object reference

The Object Interface

The object's interface, described in the application's OMG Interface Definition Language (IDL) file, identifies the set of data and methods that can be performed on an object. For example, the interface for a university teller object would identify:

- The data types associated with the object, such as a teller ID, cash in the teller's drawer; and the data managed by the object, such as an account
- The methods that can be performed on that object, such as obtaining an account's current balance, debiting an account, or crediting an account

One distinguishing characteristic of a CORBA object is the run-time separation of the interface definition from its data and methods. In a CORBA system, a CORBA object's interface definition may exist in a component called the Interface Repository. The data and methods are specified by the interface definition, but the data and methods exist in the server application process when the object is activated.

The Object's Data

The object's data includes all of the information that is specific to an object class or an object instance. For example, within the context of a bank application, a typical object might be a teller. The data of the teller could be:

- ID
- The amount of cash in the teller's drawer
- The number of transactions the teller has processed during a given interval, such as a day or month

You can encapsulate the object's data in an efficient way, such as by combining the object's data in a structure to which you can get access by means of an attribute. Attributes are a conventional way to differentiate the object's data from its methods.

The Object's Methods

The object's methods are the set of routines that can perform work using the object's data. For example, some of the methods that perform functions using teller object might include:

- `get_balance()`

- `credit()`
- `debit()`

In a CORBA system, the body of code you write for an object's methods is sometimes called the object implementation. You can think of the implementation as the code that defines the behavior of the object.

How Object Invocation Works

Since CORBA objects are meant to function in a distributed environment, OMG has defined an architecture for how object invocation works. A CORBA object can be invoked in one of two ways:

- By means of generated client stubs and skeletons - sometimes referred to as stub-style invocation
- By means of the dynamic invocation interface - referred to as dynamic invocation

For the purposes of this document, the following section describes stub-style invocation, which is simpler to use than dynamic invocation.

When you compile your application's OMG IDL file, one file that the complier generates is a source file called the client stub. The client stub maps OMG IDL method definitions for an object type to the methods in the server application that the system invokes to satisfy a request. The client stub contains code generated during the client application build process that is used in sending the request to the server application. Programmers should never modify the client stub code.

Another file produced by the IDL compiler is the skeleton, which is also a C++ or Java source file. The skeleton contains code used for method invocations on each interface specified in the OMG IDL file. The skeleton is a map that points to the appropriate code in the CORBA object implementation that can satisfy the client request. The skeleton is connected to both the object implementation and the WLE Object Request Broker (ORB).

When a client application sends a request, the request is implemented as an method on the client stub. When the client stub receives the request, the client stub sends the request to the ORB, which then sends the request through the WLE system to the skeleton. The ORB interoperates with the TP Framework and the Portable Object Adapter (POA) to locate the correct skeleton and object implementation.

Understanding the CORBA Name Service

The WebLogic Enterprise Name Service (referred to throughout this document as the CORBA Name Service) allows WebLogic Enterprise CORBA server applications to advertise object references using logical names. WebLogic Enterprise CORBA client applications can then locate an object by asking the CORBA Name Service to look up the name.

The CORBA Name Service provides:

- An implementation of the Object Management Group (OMG) Interoperable Name service (INS) specification.
- Application programming interfaces (APIs) for mapping object references into a hierarchical naming structure (referred to as a namespace).
- Commands for displaying bindings and for binding and unbinding naming context objects and application objects into the namespace.

The CORBA Name Service is a layered product. The CORBA Name Service is installed as part of the WebLogic Entrerprise product. For a complete description of the supported platforms and the installation procedure, see the *BEA WebLogic Enterprise Installation Guide*.

When using the CORBA Name Service:

1. WebLogic Enterprise CORBA server applications bind a name to one of its application objects or a naming context object within a namespace.
2. WebLogic Enterprise CORBA client applications can then use the namespace to resolve a name and obtain an object reference to the application object or the naming context object.

4 eLink Adapter for CORBA Configuration

This section contains instructions and information for configuring the eLink Adapter for CORBA to run in the WLE environment.

This section contains the following topics:

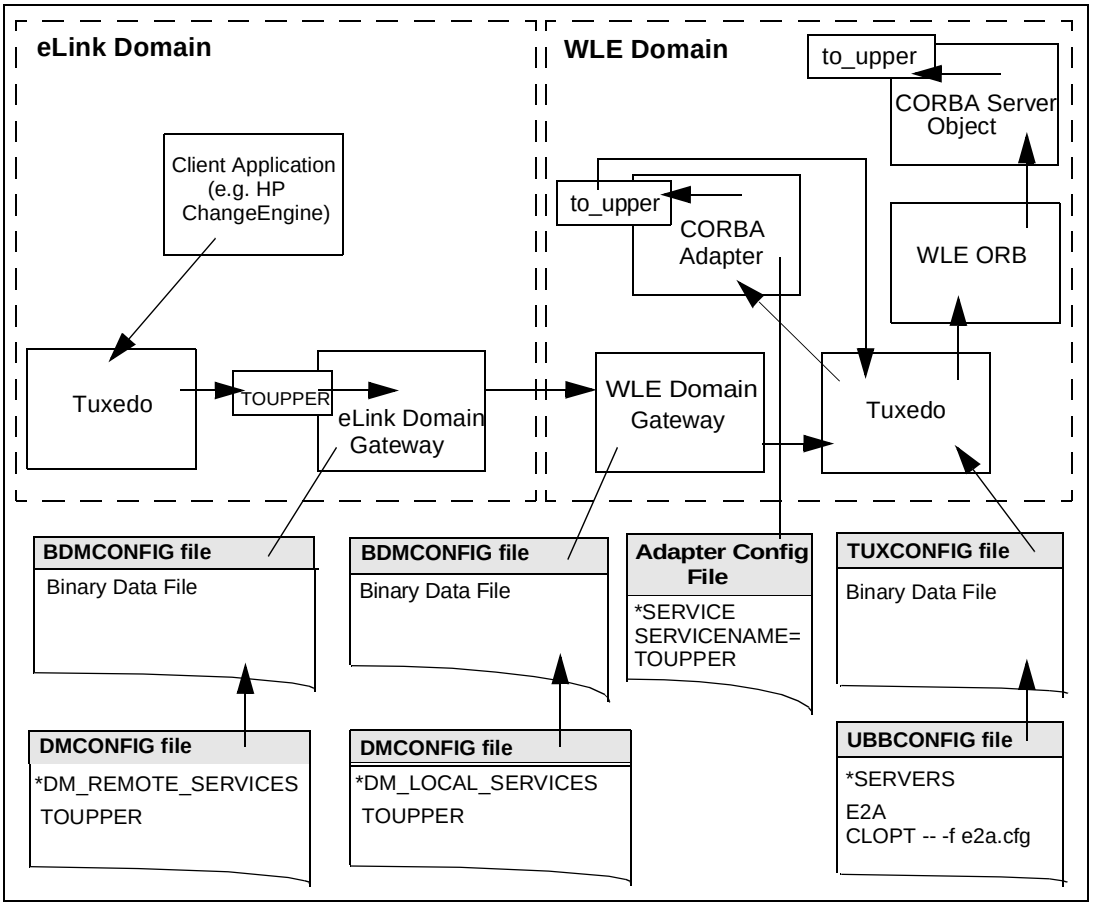
- eLink Adapter for CORBA Configuration Components
- eLink Adapter for CORBA Configuration Prerequisites
- Generating the Adapter Configuration File
- Generating an FML32 Field Table File
- Adding the e2a Server to the UBBCONFIG File
- Creating the DMCONFIG File
- Creating the FactoryFinder Configuration File

eLink Adapter for CORBA Configuration Components

Configuring the eLink Adapter for CORBA is a central task of the administrator. In configuring the adapter, you are defining the services to be made available by the adapter. In addition to adapter specific configuration, you must configure the FML32

fields to be used in a specific domain, define information necessary to boot the application you use, and describe the relationship between the local and remote domains being utilized. All this configuration is accomplished through the use of configuration files you will create, edit and compile. These configuration files work together to configure the eLink Adapter for CORBA. The following architectural diagram provides an overview of the configuration components for the eLink Adapter for CORBA and illustrates how they work together.

Figure 4-1 eLink Adapter for CORBA Configuration Components



The following files must be modified to configure the eLink Adapter into WebLogic Enterprise CORBA for use with the WebLogic Enterprise product. The order in which each component is described is the same order in which you should configure the eLink Adapter for CORBA.

- Adapter Configuration File

  The eLink Adapter for CORBA adapter configuration file defines each service that will be supported by an instance of the adapter. The file contains information on services to be made available to the eLink Adapter into WebLogic Enterprise CORBA. During initialization, the eLink Adapter into WebLogic Enterprise CORBA server reads the Adapter configuration file specified in the command line option (`CLOPT`) line in the `UBBCONFIG` file.

- FML32 Field Table File

  The FML32 Field Table files are used by the eLink Adapter for CORBA and WLE to manage FML32 field definitions. The Field Table file contains definitions of fields that will be used in a specific domain. Field definitions include the name of the FML32 field, a field identifier, and the data type of the field. The FTGEN configuration tool that generates this file uses the adapter configuration file and the Interface Repository as input to create the file. The FTGEN configuration tool also creates a list of fields used by the services listed in the adapter configuration file.

- `UBBCONFIG` file

  The `UBBCONFIG` file is an ASCII text version that contains the information necessary to boot the local application (an instance of the eLink Platform). The `UBBCONFIG` file also contains the name of the server used to advertise services that represent CORBA methods. The `UBBCONFIG` file is the file from which the `TUXCONFIG` file is generated.

- `TUXCONFIG` file

  The `TUXCONFIG` file is the binary version of the `UBBCONFIG` file and contains the information used by the `tmboot` command to start services and initialize the WLE application Bulletin Board in an orderly sequence. The `TUXCONFIG` file is accessed by all BEA WLE and eLink Platform processes for all configuration information.

- DMCONFIG file

  The DMCONFIG file describes the relationship between the local domain (the domain in which the DMCONFIG file resides) and the remote domain (any other domain). There is one DMCONFIG file per domain. The DMCONFIG file contains domain information for eLink Platform domains and for WLE domains. How multiple domains are connected and which services they make accessible to each other are defined in a Domains configuration file on each domain. The DMCONFIG file defines the following:

  - The remote domains with which the local domain can communicate
  - The local resources (such as services and queues) accessible to remote domains
  - The remote resources accessible to the local domain
  - Which local and remote resources are accessible through which gateways

- BDMCONFIG file

The DMCONFIG file is parsed and loaded into a binary version, called BDMCONFIG, by the dmloadcf utility. The dmadmin command uses BDMCONFIG (or a copy of it) for monitoring the run-time application.

One BDMCONFIG file is required on each domain in a multi-domain configuration in which the Domains feature is being used.

eLink Adapter for CORBA Configuration Prerequisites

The following configuration prerequisites must be met prior to beginning the eLink Adapter for CORBA configuration.

- Set environment variables
- Configure and load the WLE Interface Repository

Note: The eLink Adapter for CORBA configuration tools and the runtime server all utilize the WLE Interface Repository.

Setting Application Environment Variables

In order to function properly, the various eLink Adapter into WebLogic Enterprise CORBA components require certain environment variables concerning the operating system environment in which the servers reside. The `setenv` file that installs with your eLink Adapter into WebLogic Enterprise CORBA software is a sample script that sets environment variables.

For multiple-domain configurations, these environment variables must be set for both the local and remote domains. Additionally, if you modify the configuration of your eLink Adapter for CORBA system or the supporting software, you must also update the `setenv.sh` file to reflect those changes. The `setenv.sh` file is located in the samples directory, `$TUXDIR/eLink/elcorba/samples`.

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

- `TUXDIR` - The BEA WLE system root directory. For example, the following variable would work if WLE was installed in `/opt/tuxedo`.

  ```
  /opt/tuxedo
  ```

- `APPDIR` - The full pathname of the directory containing files used by the application (for example, configuration files).

  ```
  /work/apps
  ```

- `TUXCONFIG` - The full pathname of the `TUXCONFIG` file.

  ```
  $APPDIR/tuxconfig
  ```

- `BDMCONFIG` - The domain gateway configuration file.

  ```
  $APPDIR/dmconfig
  ```

- `PATH` - Must include the WLE installations `bin` directory.

  `$TUXDIR``/bin`

- The Load Library Path variable must include your WLE libraries so that servers will locate the dynamic libraries at runtime.

  - `SHLIB_PATH` for HP-UX.
  - `LD_LIBRARY_PATH` for Solaris

    `$TUXDIR``/lib`

- FLDTBLDIR32 - The path name of any directories that contain FML32 field tables.
- FIELDTBLS32 - File names of FML32 field tables used by this configuration.

Note: The FLDTBLDIR32 and FIELDTBLS32 environment variables are set during the FML32 Field Table file configuration.

The following listing provides an example of the environment variables.

Listing 4-1 Local Environment Variable Example

```
#!/bin/ksh
export TUXDIR="/opt/tuxedo"
export PATH=$PATH:$TUXDIR/bin
export SHLIB_PATH=$TUXDIR/lib
export TOBJADDR="//dalhp5:2468"
export APPDIR="/work/test/testcorba"
export TUXCONFIG="$APPDIR/tuxconfig"
export FLDTBLDIR32=$APPDIR:$TUXDIR/udataobj
export FIELDTBLS32=adapter.flds,Usysflds
```

Note: The TUXDIR, APPDIR, and TUXCONFIG variables must agree with their counterpart variables in the MACHINES section of the UBBCONFIG configuration file.

Understanding the Interface Repository

The BEA WebLogic Enterprise Interface Repository contains the interface descriptions of the CORBA objects. The CFGEN and FTGEN configuration tools that install as part of the eLink Adapter for CORBA access the Interface Repository to assist you in configuring the adapter. The Interface Repository also assists in runtime conversion of service calls to CORBA object invocations.

The BEA WebLogic Enterprise Interface Repository is based on the CORBA definition of an Interface Repository. It offers a proper subset of the interfaces defined by CORBA; that is, the APIs that are exposed to programmers are implemented as defined by the Common Object Request Broker: Architecture and Specification

Revision 2.2. However, not all interfaces are supported. In general, the interfaces required to read from the Interface Repository are supported, but the interfaces required to write to the Interface Repository are not. Additionally, not all `TypeCode` interfaces are supported.

The Interface Repository consists of two distinct components; the database and the server. The (`TMIFRSVR`) server performs operations on the database. The Interface Repository database is created and populated using the `idl2ir` administrative command. You must use the `idl2ir` administrative command to populate the Interface Repository database with any objects you plan to utilize using the eLink Adapter into WebLogic Enterprise CORBA. This administrative program is distributed with WLE.

Loading the Interface Repository

To create the Interface Repository and load it with interface definitions, you must use the `idl2ir` administrative program. If no repository file exists, this command creates it. If a repository file does exist, this command loads the specified interface definitions into it and, in effect, updates the file. When the file is updated, a new Interface Repository database file is created.

The Interface Repository server (`TMIFRSVR`) must be running in order for the eLink Adapter into WebLogic Enterprise CORBA to work properly. The `TMIFRSVR` server is included with WLE for accessing the Interface Repository. You can verify `TMIFRSVR` is running by using the `psr` command of the `tmadmin` utility.

The following provides an example of using the `idl2ir` command to load the Interface Repository.

Listing 4-2 Sample idl2ir Load Command

```
idl2ir -f e2aclient.ifr simple.idl
```

Verifying Interface Repository Load

To verify that the Interface Repository correctly loaded the objects you want, run the `ir2idl` program to populate your console window with a list of the objects loaded into the Interface Repository. You may optionally direct the outputto a file.

Listing 4-3 Sample ir2idl Command

```
ir2idl -f (filename)
```

For a description of the `idl2ir` and `ir2idl` programs and a listing of the `TMIFRSVR` server parameters, see the *Commands, System Processes, and MIB Reference* section of the BEA WebLogic Enterprise information set or *Managing Repositories* in the *T-Engine CORBA Administration* guide.

Generating the Adapter Configuration File

The eLink Adapter for CORBA configuration file defines each service that will be supported by an instance of the adapter. Each service is advertised by the adapter server so that it may be accessed from the eLink Platform. The CFGEN configuration tool generates an eLink Adapter configuration file. The generated file contains information on services available to the eLink Adapter for CORBA. However, some modifications are necessary prior to using the configuration. Registry and Factory parameters specify how objects are created and accessed by the adapter. Your CORBA administrator can provide this information to you. If you do not have a CORBA administrator that has this information, the application developer must provide you with the proper registry information on how the application object was registered within CORBA.

The CFGEN configuration tool runs against the CORBA Interface Repository and generates configuration parameters based upon the Interface Definition Language (IDL) interface information. The CFGEN configuration tool generates a new configuration or adds new services to an existing configuration. Service definitions are generated for CORBA object methods that accept or return data items that are compatible with the basic data types supported by FML32 (`string`, `double`, `float`, `char`, `short`, and `long`).

Refer to *Generating an FML32 Field Table File* for a list of data types supported by FML32.

The CFGEN configuration tool populates the following configuration parameters within the adapter configuration file:

- SERVICENAME
- OBJECTNAME
- METHOD
- PARAMS
- RETURN

Other parameters dealing with NameServices or Factories that are not available in the Interface Repository must be manually populated by you or a system administrator within the generated adapter configuration file. Refer to the *Editing the Adapter Configuration File* topic for instructions on editing this configuration file.

Invoking the CFGEN Configuration Tool

In order for the CFGEN configuration tool to execute properly, the TUXDIR and PATH environment variables must be set correctly. The $TUXDIR environment variable needs to point to the WLE installation. The PATH variable needs to include the directory where the CFGEN tool is installed ($TUXDIR/bin) along with the java runtime environment (JRE) bin directory.

Note: The CFGEN configuration utility must run while the WLE domain and TMIFRSVR server are booted in order to access CORBA and the Interface Repository.

Prior to running the CFGEN configuration tool, you need the following information:

- Host name where WLE is executing

  This name can be located in the ISL server definition in the UBBCONFIG file.

- Port WLE is using

  The port number can be located in the ISL server definition in the UBBCONFIG file.

- New and/or existing configuration file name(s)

- Names of any interfaces in the Interface Repository to be configured

Once the CFGEN tool is started, it prompts the user for information needed to run successfully. Optional parameters can be bypassed by pressing the Enter key at the prompt for the parameter without entering any data.

1. Start the CFGEN configuration tool by typing `CFGEN` at the command prompt.
2. *Enter the Host name where WLE is executing*: You are required to enter the Host name where WLE and its ISL server are executing.
3. *Enter the port WLE is using*: This is the port number specified for the ISL server. The Port number is required.
4. *Enter a new configuration file name*: A file name for the resulting adapter configuration file needs to be entered at this prompt. Enter the fully qualified file name to create a file in a directory other than the current working directory.
5. *To merge with an existing configuration, enter the existing configuration filename or Press Enter to continue*: If new service information needs to be added to information in an existing configuration file, enter the existing configuration file name at this prompt. Enter the fully-qualified file name to use a file located in a directory other than the current working directory.
6. *Enter a full or partial interface name to add a specific interface to the configuration or press enter to continue*: To add services for specific interfaces in the Interface Repository, enter a full or partial interface name at this prompt. (For example: to add services for an interface Customer, enter Customer at the prompt. To add services for interfaces beginning or ending with the word customer, enter Customer* or *Customer respectively). Multiple full or partial interface names may be entered. CFGEN will continue to prompt you until Enter is pressed with no interface name entered. If no full or partial interface name is entered, the CFGEN will browse the entire Interface Repository and generate service definitions for each eligible CORBA object method. This parameter is optional.
7. Once generation of the file is complete, the CFGEN configuration tool quits automatically.

The following listing is the console output of a CFGEN sample execution.

Listing 4-4 Sample CFGEN Execution

```
$CFGEN
Copyright (c) 2000 BEA Systems, Inc.
All Rights Reserved.
Distributed under license by BEA Systems, Inc.
```

```
       - BEA eLink Adapter into WebLogic Enterprise CORBA 1.0 -
            - - -     Configuration Generator      - - -

Enter the Host name where WLE is executing:
DALHP5

Enter the port WLE is using:
2468

Enter a new configuration file name:
config.txt

To merge with an existing configuration, enter the existing
configuration file name or press Enter to continue:
oldConfig.txt

Enter a full or partial interface name to add a specific interface
to the configuration or press Enter to continue:
Simp*

Enter a full or partial interface name to add a specific interface
to the configuration or press Enter to continue:

Loading existing configuration...

Adding services to configuration...

Configuration generation complete.

$
```

This sample execution of CFGEN executes against the Interface Repository located on DALHP5, port 2468. It will create a new configuration file (`Config.txt`) containing information from the existing configuration file (`OldConfig.txt`) and add new services for any interfaces starting with `Simp`. Error and Informational messages will be written to `CFGEN.log` in the current working directory.

Listing 4-5 Sample CFGEN Output

```
*SERVER
MINMSGLEVEL=
MAXMSGLEVEL=

*SERVICE
SERVICENAME=Simpleto_lower
REGISTERED=
```

```
REGISTRYTYPE=
FACTORYOBJECT=
FACTORYMETHOD=
FACTORYPARAMS=
OBJECTNAME=Simple
METHOD=to_lower
PARAMS=inVal
RETURN=Simpleto_lower_RTN

*SERVICE
SERVICENAME=Simpleto_upper
REGISTERED=
REGISTRYTYPE=
FACTORYOBJECT=
FACTORYMETHOD=
FACTORYPARAMS=
OBJECTNAME=Simple
METHOD=to_upper
PARAMS=inVal,outVal
RETURN=void
```

Editing the Adapter Configuration File

The file generated by the CFGEN configuration tool is the eLink Adapter for CORBA configuration file. During initialization, each server reads the configuration file specified in the command line option (`CLOPT`) line in the `UBBCONFIG` file. If no file is specified, the `e2a` server fails to boot.

Once you have created a configuration file, place a working copy of it in the eLink Platform application directory, `APPDIR`. Any changes to a configuration file require that the server(s) using that file be shutdown and restarted for the changes to take effect.

The adapter configuration file consists of a SERVER section that contains trace parameters and passwords and a SERVICE section for each service the eLink Adapter into WebLogic Enterprise CORBA wants to advertise to the eLink Platform environment. Each service maps to a CORBA method.

Each SERVER section requires the following parameters:

MINMSGLEVEL = *int*

Indicates the minimum level of tracing messages the adapter is to log. Valid values are 0 - 9.

MAXMSGLEVEL = *int*

Indicates the maximum level of tracing messages the adapter is to log. Valid values are 0 - 9.

Each SERVICE section requires the following parameters:

SERVICENAME = *name*

Identifies the name of the eLink Platform service to be advertised. The SERVICENAME created by CFGEN may be modified, but it must meet the following requirements. The SERVICENAME must not exceed 15 characters in length, cannot contain blank characters, and cannot be `NULL` or the `NULL` string (““). Service names beginning with the ‘.’ character are reserved for system servers. Servers specifying such services will fail to boot.

REGISTRYTYPE = FactoryFinder | NameService

Specifies the Registry location to be used for obtaining an object reference. Valid values are *FactoryFinder* and *NameService*. Using the FactoryFinder setting will cause the adapter to attempt to locate the value specified in the REGISTERED parameter using the FactoryFinder in WLE. Using the NameService setting will cause the adapter to attempt to locate the value specified in the REGISTERED parameter using the NameService in WLE.

REGISTERED = *registered name*

Identifies the registered name of the Object or Factory registered in WLE. Objects can be registered in the NameService, but Factories may be registered in either the NameService or FactoryFinder.

The registration location is specified in the REGISTRYTYPE parameter. If the REGISTERED name is a Factory, you must specify the method within the Factory to be used to retrieve the final Object.

FACTORYOBJECT= *object name*

The factory interface that contains FACTORYMETHOD (for example, `SimpleFactory`).

FACTORYMETHOD = *method name*

Specifies the CORBA method to be invoked on REGISTERED, if REGISTERED is a Factory. This method must return an OBJECTNAME object type. If REGISTERED is not a Factory object, this parameter must be left blank.

FACTORYPARAMS = *Factory method parameters*

A comma-separated FML32 Field list to match the invoked Factory method argument list. This list must include all of the method's arguments (types: in, out, and inout). Field names should not exceed the C preprocessor identifer restrictions (that is, it should contain only alphanumeric characters and the underscore character). Internally, the name is truncated to 30 characters, so names must be unique within the first 30 characters. If the method takes no arguments, the PARAMS parameter should be populated with the word "void".

OBJECTNAME = *object name*

Identifies the name of the CORBA Object containing the METHOD to be invoked.

METHOD = *method name*

Specifies the CORBA method to be invoked when this eLink Platform service is accepted by eLink Adapter into WebLogic Enterprise CORBA.

PARAMS = *method parameters*

A comma-separated FML32 Field list to match the invoked method argument list. This list must include all of the method's arguments (types: in, out, and inout). Field names should not exceed the C preprocessor identifer restrictions (that is, it should contain only alphanumeric characters and the underscore character). Internally, the name is truncated to 30 characters, so names must be unique within the first 30 characters. If the method takes no arguments, the PARAMS parameter should be populated with the word "void".

RETURN = *method return*

After creating or modifying the configuration file, the `RETURN` parameter specifies the FML32 Field name in which to return the invoked method's response. If the response type is a void type, the RETURN parameter should be populated with the word "`void`".

Note: FML32 values must be updated in the FML32 Field Table file using the FTGEN tool after all configuration file modifications are complete.

Generating an FML32 Field Table File

FML32 Field Tables are used by the eLink Adapter for CORBA and WLE to manage FML32 field definitions. The field table contains definitions of fields that will be used in a specific domain.

Field definitions include the name of the FML32 field, a field identifier, and the data type of the field. Following is a list of the CORBA data types, with the data types supported by eLink Adapter for CORBA indicated.

Data Type	Supported	FML32 Field Type
tk_abstract_interface	No	
tk_alias	No	
tk_any	No	
tk_array	No	
tk_boolean	Yes	FLD_CHAR
tk_char	Yes	FLD_CHAR
tk_double	Yes	FLS_DOUBLE
tk_enum	No	
tk_except,	No	
tk_fixed	No	
tk_float	Yes	FLD_FLOAT
tk_long	Yes	FLD_LONG
tk_longdouble	No	
tk_longlong	No	
tk_native	No	
tk_objref	No	

Data Type	Supported	FML32 Field Type
tk_octet	No	
tk_Principal	No	
tk_sequence	No	
tk_short	Yes	FLD_SHORT
tk_string	Yes	FLD_STRING
tk_string	No	
tk_struct	No	
tk_TypeCode	No	
tk_ulong	Yes	FLD_LONG
tk_ulonglong	No	
tk_union	No	
tk_ushort	Yes	FLD_SHORT
tk_value	No	
tk_value_box	No	
tk_void	Yes	-
tk_wchar	No	
tk_wstring	No	

The FTGEN configuration tool is a convenient way to generate FML32 field tables using the FML32 field names listed in the eLink Adapter for CORBA configuration file. It is not necessary to use this tool if FML32 fields are managed in another way. The FTGEN configuration tool also compiles a list of services contained in the configuration file and outputs them in a file named `tdomsvcs.txt`.

The FTGEN configuration tool accepts as input an eLink Adapter into WebLogic Enterprise CORBA configuration file and the Interface Repository. The FTGEN tool reads through the configuration file and extracts FML32 field names from the

FACTORYPARAMS, PARAMS, and RETURN parameters. It creates an FML32 field table containing these fields and associated type information. FML32 fields may exist multiple times in a configuration as long as the data types for all occurrences of the field have the same data type. This field name will only be defined once in the field table. Duplicate field names with different data types will be logged as an error and the second occurrence of the field will not be added to the generated field table.

Note: The eLink Platform allows the use of multiple Field Table files and the FTGEN command line tool can only ensure that FML32 field names are unique within the generated FML32 field table. If you experience conflicts, you may be able to resolve them by modifying the base value in the generated Field Table file.

In order for an eLink or WLE domain to use a new FML32 field table, environment variables must be updated to reference this table. The FLDTBLDIR32 environment variable must be updated to include the path of the new FML32 field table. The file name of the new field table must be included in the FIELDTBLS32 environment variable. The FML32 Field Table must be present in both environments and the environment variables must also be updated in both environments.

A list of services is also created by the FTGEN configuration tool in a file named tdomsvcs.txt. This list of services contains all services that are listed in the adapter configuration file. These service names should be added to domain definitions for the WLE and eLink environments that are utilizing the eLink Adapter for CORBA. The services should be included in the DM_LOCAL_SERVICES section of the WLE DMCONFIG file where the eLink Adapter into WebLogic Enterprise CORBA server component is executing and in the DM_REMOTE_SERVICES section of the eLink Adapter into WebLogic Enterprise CORBA DMCONFIG file that will be calling these adapter services.

Invoking the FTGEN Configuration Tool

In order for the FTGEN configuration tool to execute properly, the TUXDIR and PATH environment variables must be set correctly. The $TUXDIR environment variable needs to point to the WLE installation. The PATH variable needs to include the directory where the FTGEN tool is installed ($TUXDIR/bin) along with the java runtime environment (JRE) bin directory.

Prior to running the FTGEN configuration tool, you need the following information:

- Host name where WLE is executing

  This name is located in the ISL server definition in the `UBBCONFIG` file.

- Port WLE is using

  The port number can be located in the ISL server definition in the `UBBCONFIG` file.

- Existing adapter configuration file name

Once the FTGEN tool is started, it will prompt you for information needed to run successfully.

1. Start the FTGEN configuration tool by typing FTGEN at the command prompt.
2. *Enter the Host name where WLE is executing*: Enter the Host name where WLE and its ISL server are executing. This name can be located in the ISL server definition in the `UBBCONFIG` file.
3. *Enter the port WLE is using*: This is the port number specified for the ISL server. The port number can be located in the ISL server definition in the `UBBCONFIG` file.
4. *Enter the configuration file name*: A file name for the adapter configuration file that contains the FML32 field names needs to be entered at this prompt. You may enter the fully-qualified file name to use a file located in a directory other than the current working directory.
5. *Enter a file name for the new FML Field Table File*: Enter a new file name for the FML32 Field Table that is generated by FTGEN.
6. Once generation of the file is complete, the FTGEN configuration tool quits automatically.

The following listing is the console output of a FTGEN sample execution.

Listing 4-6 Sample FTGEN Execution

```
$FTGEN
Copyright (c) 2000 BEA Systems, Inc.
All Rights Reserved
Distributed under license by BEA Systems, Inc.
```

```
       - BEA eLink Adapter into WebLogic Enterprise CORBA 1.0 -
           - - -    FML32 Field Table Generator    - - -

Enter the hostname where WLE is executing:
DALHP5

Enter the port WLE is using:
2468

Enter the configuration file name:
config.txt

Enter a file name for the new FML Field Table File...
fmlFields.txt

Loading existing configuration...

Generating new FML Field Table file...

Generating Service List file...

FML Field Table generation complete.

$
```

This sample execution of FTGEN executes using the Interface Repository associated with the WLE ISL server executing on machine DALHP5, port 2468. It will input FML32 file names found in configuration file (`Config.txt`) and create FML32 Field Table file (`fmlFields.txt`) using this input. It also creates a `tdomsvcs.txt` file containing a list of services contained in the adapter configuration file. Informational and error messages will be written to `FTGEN.log` in the current working directory.

Listing 4-7 Sample FTGEN Field Table

```
#
# fml field table for eLink Adapter into WebLogic Enterprise CORBA
#
*base                1000
#  Name              FldID  Type   Flags         Comments
outVal               1      string               -      -
inVal                2      string               -      -
Simpleto_lower_RTN   3      string               -      -
```

The resulting FML32 Field Table contains relative field numbers starting at 1001. If this conflicts with existing field definitions, the Field Table should be edited to point to an unused range of field IDs. Modifying the base value will often be enough to eliminate conflicts. Base can be any value and ranges from 100 to 33554000.

The following listing is an example of the `tdomsvcs.txt` file containing a list of services contained in the adapter configuration file.

Listing 4-8 Sample FTGEN Service List (tdomsvcs.txt)

```
#
#      List of services configured for eLink Adapter for CORBA
#
#      Add these services to the *DM_LOCAL_SERVICES section of the
#      WLE DMCONFIG file and to the *DM_REMOTE_SERVICES section of
#      the eLink Platform DMCONFIG file.
#
Simpleto_upper
Simpleto_lower
```

Understanding FML32 Error Codes

The eLink Adapter for CORBA uses FML32 buffers to transport messages in the eLink Platform environment. The error code fields used by the eLink Adapter for CORBA must be installed in your environment to allow the eLink Adapter for CORBA to return error codes and messages.

If the eLink Adapter for CORBA is your first eLink Adapter, or you are not sharing Field Table Files between WLE and Tuxedo, you need to add the following two FML string fields to your FML32 field tables.

- `ELINK_ADAPTER_ERR`

  Stores the details of the errors specific to the eLink Adapter into WebLogic Enterprise CORBA

- `ELINK_ADAPTER_ERR_CODE`

  Stores the category code for the adapter-specific errors.

You must define the FML32 fields for the `e2a` server. The syntax for defining the response buffer fields is as follows.

Table 4-1 Syntax for Field Definition Table for eLink Adapter for CORBA

# Name	Number	Type	Flags	Comments
ELINK_ADAPTER_ERR	'n'	string	-	-
ELINK_ADAPTER_ERR_CODE	'n'	string	-	-

In order for eLink Adapter for CORBA to exchange data, identical field names must exist in both environments. The field names used to define a CORBA service or process that will be made available to eLink must be defined in the eLink environment.

Note: The actual field IDs are customer-defined; only the field names are reserved

Several examples of FML field tables can be found in the *FML Programmer's Guide.* Refer to this guide for more detailed instructions on transferring these environment variables into your environment.

Note: The environment variable `FIELDTBLS32` contains a list of table files required by eLink FML32 data handling.

The environment variable `FLDTBLDIR32` contains a list of directories to search when trying to locate `FIELDTBLS32` files.

Buffer Handling

All communication in the BEA eLink Platform system is transmitted through typed buffers. All buffers passed through the eLink Platform system have special headers, and must be allocated and freed through the eLink Platform ATMI (`tpalloc`(), `tprealloc`(), and `tpfree`()).

Adding the e2a Server to the UBBCONFIG File

In order for the eLink Platform to recognize and start the eLink Adapter for CORBA, it must be added to the UBBCONFIG file. The UBBCONFIG file is an ASCII text version of the binary data file TUXCONFIG. This UBBCONFIG file contains the information necessary to boot the eLink Adapter into WebLogic Enterprise CORBA. You can create and edit the UBBCONFIG with any text editor. You must create a UBBCONFIG file server entry for each new application. You can use the sample UBBCONFIG file as a starting point and edit it to meet the requirements of your particular application. Once you have updated the UBBCONFIG file, you must compile it into the TUXCONFIG file by running the tmloadcf command.

The TUXCONFIG file contains information used by tmboot to start services and initialize the Bulletin Board of a BEA WLE application in an orderly sequence. The tmadmin command line utility uses the configuration file (or a copy of it) to carry out monitoring activities. The tmshutdown command references the configuration file for information needed to shut the application down.

Note: When tmloadcf is executed, the TUXCONFIG environment variable must be set to the full pathname of the device or system file where TUXCONFIG is to be loaded.

Before running the eLink Adapter for CORBA, you must identify and add the e2a server entry to the UBBCONFIG file so that it is part of your WLE application. The e2a server advertises the services that represent CORBA methods. A sample UBBCONFIG file (named e2alocal.ubb) is provided on the installation CD-ROM. You can use this sample file as a base for creating your own UBBCONFIG file or edit an existing file.

Perform the following steps to edit your UBBCONFIG file.

1. Open the UBBCONFIG file.
2. In the SERVERS section, add an entry for the e2a server.

The following listing shows the syntax for the server definition in the UBBCONFIG file.

Listing 4-9 Syntax for the `e2a` Server Definition in the `UBBCONFIG` File

```
*SERVERS
        e2a     SRVGRP=APP_GRP
                RESTART=N
                SRVID=10
                CLOPT="-- -C e2aclient.txt"
```

3. Specify the command line option (`CLOPT`) parameter to indicate the name of the eLink Adapter for CORBA configuration file the adapter will use at runtime. See *Generating an Adapter Configuration File* for instructions on creating this configuration file.
4. Save and close the `UBBCONFIG` file.
5. Once all configuration files are created, compile the `UBBCONFIG` file to a `TUXCONFIG` file using the `tmloadcf` command.

   The following listing provides an example of the commands and parameters necessary to compile the `UBBCONFIG` file.

Listing 4-10 Compile Session Example

```
$ /work/testcorba
$ TUXCONFIG=/work/testcorba/tuxconfig; export TUXCONFIG
$ tmloadcf -y ubb.txt
```

The following SERVER section parameters are defined as follows:

`SRVGRP` = *`string_value`*

Specifies the name for the group in which the server is to run. The *`string_value`* must be the logical name associated with a server group in the Groups section. The *`string_value`* must be 30 characters or less. The GROUPS section also specifies the `GRPNO` for the server group and parameters to pass when the associated resource manager is opened. All server entries must have a server group parameter specified.

SRVID = *number*
: Specifies an integer that uniquely identifies a server within a group. Identifiers must be between 1 and 30,000 inclusive. This parameter must be present on every server entry.

CLOPT = "*servopts* -- *application_opts*"
: Specifies servopts options to be passed to the server when booted. "--" marks the end of system-recognized arguments and the start of arguments to be passed to a subroutine within the server. All eLink adapters will have exactly one argument after the --, the -C option followed by the name of the adapter-specific configuration file. All other configuration parameters specific to the adapter should appear in the adapter-specific configuration file.

RESTART = Y | N
: Specifies whether or not to restart the server upon abnormal termination.

For additional information about the SRVGRP, SRVID, CLOPT, and RESTART parameter syntax and definitions, refer to the *BEA Tuxedo Reference Manual*.

The following listing is a sample UBBCONFIG file. In this sample, the e2a server is defined in the SERVERS section with the required CLOPT parameter specified.

Note: If you have not already configured your TDomain, refer to the *Configuring the DMCONFIG File* topic for instructions on how to do so.

Listing 4-11 Sample UBBCONFIG File for the eLink Adapter for CORBA

```
*RESOURCES
       IPCKEY        55432
       DOMAINID      simpapp
       MASTER        SITE1
       MODEL         SHM
       LDBAL         N
*MACHINES
       "dalhp5"
              LMID          = SITE1
              APPDIR        = "/work/testcorba"
              TUXCONFIG     = "/work/testcorba/tuxconfig"
              TUXDIR        = "/work/wle51"
              MAXWSCLIENTS  = 10
*GROUPS
       SYS_GRP
              LMID          = SITE1
```

```
            GRPNO           = 1
      APP_GRP
            LMID            = SITE1
            GRPNO           = 2
*SERVERS
      DEFAULT:
            RESTART         = Y
            MAXGEN          = 5
      TMSYSEVT
            SRVGRP          = SYS_GRP
            SRVID           = 1
      TMFFNAME
            SRVGRP          = SYS_GRP
            SRVID           = 2
            CLOPT           = "-A -- -N -M"
      TMFFNAME
            SRVGRP          = SYS_GRP
            SRVID           = 3
            CLOPT           = "-A -- -N"
      TMFFNAME
            SRVGRP          = SYS_GRP
            SRVID           = 4
            CLOPT           = "-A -- -F"
      TMIFRSVR
            SRVGRP          = SYS_GRP
            SRVID           = 5
            CLOPT           = "-A -- -f repository.ifr"
      simple_server
            SRVGRP          = APP_GRP
            SRVID           = 1
            RESTART         = N
      cns
            SRVGRP          = SYS_GRP
            SRVID           = 6
            CLOPT           = "-A -- "
      ISL
            SRVGRP          = SYS_GRP
            SRVID           = 10
            CLOPT           = "-A -- -n//localhost:2468 -d/dev/tcp"
      e2a
            SRVGRP          = APP_GRP
            RESTART         = Y
            SRVID           = 10
            CLOPT           = "-- -C config.txt"

*SERVICES
```

Creating the DMCONFIG File

Many legacy applications, as well as various BEA software application adapters exist in the eLink environment. To provide a complete integration of eLink Platform and WebLogic Enterprise, you must provide the eLink Platform access to the ATMI service advertised by the eLink Adapter for CORBA. The ATMI services advertised by each eLink Adapter for CORBA `e2a` server can be exported to other domains including the eLink Platform domains using the TDOMAIN feature of WLE and eLink Platform.

Creating a `DMCONFIG` file is the first step to utilizing the eLink Platform and WLE TDOMAINS feature. A Domains configuration is a set of two or more domains (or applications) that can communicate and share services with the help of the BEA eLink Platform Domains feature. How multiple domains are connected and which services they make accessible to each other are defined in a Domains configuration (`DMCONFIG`) file on each domain. In a multi-domain application, a separate `DMCONFIG` file must be created for each participating domain. For your eLink Adapter into WebLogic Enterprise CORBA configuration to be successful, you must configure a new or existing `DMCONFIG` file for both the eLink Platform Domain and the WLE Domain. You must also define the Domains environment in the eLink `UBBCONFIG` file and the WLE `UBBCONFIG` file.

A `DMCONFIG` file defines the following:

- The remote domains with which the local domain can communicate
- The local resources (such as services and queues) accessible to remote domains
- The remote resources accessible to the local domain
- Which local and remote resources are accessible through which gateways

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

A BEA eLink Platform system domain application is defined as the environment described in a single TUXCONFIG file. A BEA eLink Platform system application can communicate with another BEA eLink Platform system or another TP application via a domain gateway group. In BEA eLink Platform 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 BEA eLink Platform system domain process that relays requests to another TP Domain and receives replies.

A *Local Domain* is 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 synonymously.

A *Remote Domain* is a remote application that is accessed through a Gateway Group. The remote application may be another BEA eLink Platform 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 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.

Accessing the DMCONFIG File

All domains configuration is stored in a binary file called BDMCONFIG. You can create and edit a text version of this file, DMCONFIG, with any text editor. You can update the compiled BDMCONFIG file while the system is running by using the dmadmin command when using Domains.

Perform the following steps to access, edit, save, and exit the DMCONFIG file.

Note: Because BEA eLink Adapter for CORBA 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 proper syntax for your platform when making command line entries.

1. Enter the platform command to gain access to the install directory.

```
cd install
```

2. Gain access to the `examples` subdirectory.

   ```
   cd examples
   ```

3. Enter the command to invoke the text editor and gain access to the `DMCONFIG` file.

   ```
   edit DMCONFIG
   ```

4. Edit the `DMCONFIG` file as necessary. Refer to the following section's parameter description for details about defining your BEA eLink Adapter for CORBA configuration.

5. When editing is complete, save and exit the `DMCONFIG` file.

   ```
   $exit
   ```

Defining Domain Parameters for the eLink Domain

For the eLink Domain gateway configuration file, only the required parameters are defined. Default settings are used for optional parameters. The following sections explain each `DMCONFIG` file section required for the eLink Domain configuration and the parameters associated with each.

Editing the DM_LOCAL_DOMAINS Section

The DM_LOCAL_DOMAINS section identifies the local domains and their associated gateway groups. This 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. The following are 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`* and the name of the gateway server group.

TYPE = *identifier*

is used for grouping local domains into classes. `TYPE` can be set to one of the following values: `TDOMAIN`, `SNAX`, `OSITP`, OR `TOPEND`. The `TDOMAIN` value

indicates that this local domain can only communicate with another BEA eLink Platform system.

DOMAINID = *string*

is used to identify the local domain. `DOMAINID` must be unique across both local and remote domains.

Editing the DM_REMOTE_DOMAINS Section

The DM_REMOTE_DOMAINS section identifies the known set of remote domains and their characteristics. This section has one entry that takes the following form:

`RDOM` required_parameters [*optional_parameters*]

where *`RDOM`* is an identifier value used to identify each remote domain known to this configuration. *`RDOM`* must be unique within the configuration. `TYPE` is used to classify the type of domains. `DOMAINSID` is a unique domain identifier.

Editing the DM_TDOMAIN Section

The DM_TDOMAIN section defines the addressing information required by domains of type `TDOMAIN`. This section should have one entry per local domain if requests from remote domains to local services are accepted on that local domain (gateway group), and one entry per remote domain accessible by the defined local domains. This section entry has the following form:

`DOM` required_parameters [*optional_parameters*]

where *`DOM`* is an identifier value is 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 section or *`RDOM`* in the DM_REMOTE_DOMAINS section. The following parameter is required:

NWADDR = *string*

specifies the network address associated with a local domain or a remote domain. If the association is with a local domain, the `NWADDR` is used to accept connections from other BEA eLink Platform system domains. If the association is with a remote domain, the `NWADDR` is used to initiate a connection on the remote domain.

Editing the DM_REMOTE_SERVICES Section

The DM_REMOTE_SERVICES section provides information about the services that are imported. These services are imported from the eLink Adapter for CORBA so that they can be accessed by clients in the local eLink domain. All services that are loaded in the adapter configuration file should be in this remote section. The `tdomsvcs.txt` file produced by the FTGEN utility contains all of these service names. Copy and paste them into this section of the `DMCONFIG` file.

Editing the DM_LOCAL_SERVICES Section

The DM_LOCAL_SERVICES section provides information about the services that are exported. This section of our sample file has no entries because no services are being exported.

Listing 4-12 eLink Domain DMCONFIG File Example

```
#
# lapp.dom
#
*DM_LOCAL_DOMAINS

ELINKDOM      GWGRP=LGWGRP
              TYPE=TDOMAIN
              DOMAINID="111111"

*DM_REMOTE_DOMAINS

WLEDOM        TYPE=TDOMAIN
              DOMAINID="222222"

*DM_TDOMAIN

ELINKDOM      NWADDR="//mach1:5000"

WLEDOM        NWADDR="//mach2:5000"

*DM_LOCAL_SERVICES

*DM_REMOTE_SERVICES

Simpleto_upper
Simpleto_lower
```

Defining Domain Parameters for the WLE Domain

For the WLE Domain gateway configuration file, only the required parameters are defined. Default settings are used for optional parameters. The following sections explain each DMCONFIG file section required for the WLE Domain configuration and the parameters associated with each.

Editing the DM_LOCAL_DOMAIN Section

The DM_LOCAL_DOMAIN section identifies the local domains and their associated gateway groups. This 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. The following are 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* and the name of the gateway server group.

TYPE = *identifier*

is used for grouping local domains into classes. TYPE can be set to one of the following values: TDOMAIN, SNAX, OSITP, OR TOPEND. The TDOMAIN value indicates that this local domain can only communicate with another BEA eLink Platform system.

DOMAINID = *string*

is used to identify the local domain. DOMAINID must be unique across both local and remote domains.

Editing the DM_REMOTE_DOMAINS Section

The DM_REMOTE_DOMAINS section identifies the known set of remote domains and their characteristics. This section has one entry that takes the following form:

RDOM required_parameters [*optional_parameters*]

where *RDOM* is an identifier value used to identify each remote domain known to this configuration. *RDOM* must be unique within the configuration. TYPE is used to classify the type of domains. DOMAINSID is a unique domain identifier.

Editing the DM_TDOMAIN Section

The DM_TDOMAIN section defines the addressing information required by domains of type TDOMAIN. This section should have one entry per local domain if requests from remote domains to local services are accepted on that local domain (gateway group), and one entry per remote domain accessible by the defined local domains. This section entry has the following form:

DOM required_parameters [*optional_parameters*]

where *DOM* is an identifier value is 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 section or *RDOM* in the DM_REMOTE_DOMAINS section. The following parameter is required:

NWADDR = *string*

specifies the network address associated with a local domain or a remote domain. If the association is with a local domain, the NWADDR is used to accept connections from other BEA eLink Platform system domains. If the association is with a remote domain, the NWADDR is used to initiate a connection on the remote domain.

Editing the DM_LOCAL_SERVICES Section

The DM_LOCAL_SERVICES section provides information about the services that are exported. These service are exported so that it can be accessed by clients in the remote domains. All services that are loaded in the adapter configuration file should be in this local section. The tdomsvcs.txt file produced by the FTGEN utility contains all of these service names. Copy and paste them into this section of the DMCONFIG file.

Editing the DM_REMOTE_SERVICES Section

The DM_REMOTE_SERVICES section provides information about the services that are imported. This section of our sample file has no entries because no services are being imported.

Listing 4-13 WLE Domain DMCONFIG File Example

```
#
# lapp.dom
#
*DM_LOCAL_DOMAINS

WLEDOM        GWGRP=LGWGRP
              TYPE=TDOMAIN
              DOMAINID="111111"

*DM_REMOTE_DOMAINS

ELINKDOM      TYPE=TDOMAIN
              DOMAINID="222222"

*DM_TDOMAIN

WLEDOM        NWADDR="//mach1:5000"

ELINKDOM      NWADDR="//mach2:5000"

*DM_LOCAL_SERVICES

Simpleto_upper
Simpleto_lower

*DM_REMOTE_SERVICES
```

Defining the Domains Environment in the eLink and WLE UBBCONFIG Files

In both the eLink and WLE `UBBCONFIG` files, you must define the following three servers.

- DMADM - The Domains administrative server enables run-time modification of the configuration information, required by domain gateway groups, that resides in the binary Domains configuration file. DMADM supports a list of registered gateway groups. There must be only one instance of DMADM per Domains application.
- GWADM - The gateway administrative server enables run-time administration of a particular domain gateway group. This server gets Domains configuration

information from the DMADM server. It also provides administrative functionality and transaction logging for the gateway group.

- GWTDOMAIN - The Domains gateway server enables access to and from remote Domains, allowing interoperability of two or more BEA eLink Platform domains. Information about the local and remote services it needs to export and import is included in the Domains configuration file. The Domains gateway server should always be configured with `REPLYQ=N`.

Compiling the UBBCONFIG and DMCONFIG Files

The local application configuration file (`UBBCONFIG`) contains the information necessary to boot the local application. Once you have updated this file, you must compile it into a binary data file by running the `tmloadcf` command.

The local domain gateway configuration file (`DMCONFIG`)contains the information used by the domain gateway for one domain for communication with other domains. You must compile this file into a binary data file by running the `dmloadcf` command.

To compile both configuration files, complete the procedure shown in the following sample session.

Listing 4-14 Compile Session Example

```
$ cd /home/lapp
$ TUXCONFIG=/home/lapp/lapp.tux; export TUXCONFIG
$ tmloadcf -y lapp.ubb
$ BDMCONFIG=/home/lapp/lapp.dom; export BDMCONFIG
$ dmloadcf -y lapp.dom
```

Once you create both the local and remote domains, you can then boot the WLE or eLink application using tmboot. The order in which two domains are booted does not matter. Monitor the applications with dmadmin. Once both applications are booted, a client in the local application can call the required service residing in the remote application.

Listing 4-15 Boot Command Example

```
$ tmboot -y
```

Creating the FactoryFinder Configuration File

A Remote Factory is a factory object that exists in a remote ORB that can be made available to the local system through a WebLogic Enterprise ORB.

Administrators are required to identify any factory objects that can be used in the current (local) /Domain, but that are resident in a different (remote) /Domain. You identify these factories in a FactoryFinder domain configuration file, also referred to as the `factory_finder.ini` file. This is an ASCII file that can be created and updated using a text editor.

The `factory_finder.ini` file can be used to identify remote CORBA factories and remote EJB Home interfaces that can be used in the local domain.

The `factory_finder.ini` file, installed as part of the WLE software, is the FactoryFinder configuration file for domains. This file is parsed by the WLE `TMFFNAME` service when it is started as a Master NameManager. The `factory_finder.ini` file contains information used by NameManagers to control the import and the export of object references for factory objects with other domains. To use the information in the `factory_finder.ini` file, you must specify the `factory_finder.ini` filename in the `-f` option of the `TMFFNAME` server process.

The format of the `factory_finder.ini` file is modeled after the syntax used to describe Domains, and is shown in the following sample.

Listing 4-16 Sample `factory_finder.ini` File

```
*DM_REMOTE_FACTORIES
“local_factory_id.factory_kind”
DOMAINID=”domain_id”
```

```
RNAME=”remote_factory_id.factory_kind”
...

[*DM_LOCAL_FACTORIES]
[“factory_id.factory_kind”]
```

A `factory_finder.ini` file applies to the domain in which it resides. It contains two sections: the `DM_REMOTE_FACTORIES` section and the `DM_LOCAL_FACTORIES` section. Either section can be absent or contain nothing.

The following sections provide more information on how to use the `DM_REMOTE_FACTORIES` section and the `DM_LOCAL_FACTORIES` section.

Note: The `factory_finder.ini` and `DMCONFIG` files must be coordinated, that is, if the `factory_finder.ini` file declares another domain to have accessible factories, there must be a way in `DMCONFIG` to get to that domain.

Editing the DM_REMOTE_FACTORIES Section

The `DM_REMOTE_FACTORIES` section provides information about the factory objects or EJB Home interfaces that are available in remote domains and that are imported so that applications in the local domain can use them. Identifiers for remote factory objects or EJB Home interfaces are listed in this section. The identifier, under which the object is registered, including a `kind` value of “FactoryInterface”, must be listed in this section.

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

Because the identifiers of factories in a multi-domain configuration may collide, the factory identifier and the `RNAME` parameters allow you to specify alternative identities, or “aliases,” in the local domain for remote factories. You can also assign multiple aliases to the same remote factory.

In a multi domain configuration, two different domains must not have a factory object or EJB Home interfaces with the same `factory_id.factory_kind` identifier.

If the same identifier, or name, is used in two domains, the software behavior varies according to the version of the BEA WebLogic Enterprise software.

Editing the DM_LOCAL_FACTORIES Section

The `DM_LOCAL_FACTORIES` section specified factory objects or EJB Home interfaces in the local domain that are available to be exported to other domains. This section can be used in the following ways:

- If the `DM_LOCAL_FACTORIES` section does not exist in a factory_finder.ini, or exists but is empty, all factory objects and EJB Home interfaces in the local domain are available to remote domains. This allows administrators an easy means to make local factory objects or EJB Home interfaces available to remote domains without having to provide an entry for every factory object or EJB Home interface in the local domain.
- If the `DM_LOCAL_FACTORIES` section exists in a factory_finder.ini file but contains the reserved keyword "`NONE`", none of the factory objects or EJB Home interfaces in the local domain are available to remote domains. This allows administrators to restrict access without having to provide an entry for every factory object or EJB Home interface in the local domain.

The identifier, or name, under which the factory object or EJB Home interface is registered, including a kind value of "`FactoryInterface`", must be listed in this section.

The factory_kind must be specified for the Namemanager to locate the appropriate factory object or EJB Home interface. An entry that does not contain a factory_kind value is not defaulted with a value of "`FactoryInterface`". This allows for the use of the CORBA NamingService.

The factory_finder.ini file specifies that the process of finding a factory can be exported to a remote domain by including a section beginning with "`*DM_REMOTE_FACTORIES`". In other words, including this section means that the local domain can find factories in a remote domain.

5 Running BEA eLink Adapter for CORBA

This section contains the following topics:

- Running the eLink Adapter for CORBA
- Shutting Down the eLink Adapter for CORBA

Running the eLink Adapter for CORBA

Once all configuration prerequisites have been completed successfully, you can bring up the eLink Adapter into WebLogic Enterprise CORBA using the tmboot command. Only the administrator who created the `TUXCONFIG` file can execute tmboot.

The adapter is generally booted from the machine designated as MASTER in the RESOURCES section of the `UBBCONFIG` file or the BACKUP acting as MASTER. For the tmboot command to find executables, eLink Platform system processes, such as the BBL, must be located in the `$TUXDIR/bin` directory. Application servers should be in the directory defined by the `APPDIR` variable, as specified in the configuration file.

When booting the eLink Adapter for CORBA, the tmboot command uses the `CLOPT` parameter from the `UBBCONFIG` file. Application servers are booted in the order specified by the SEQUENCE parameter, if SEQUENCE is not specified, servers are rebooted in the order in which they appear in the configuration file. The command line should look something like the following when using the `tmboot` command.

Listing 5-1 tmboot Command Example

```
$ tmboot [-g grpname] [-o seqeunce] [-s] [-A] [-y]
```

The `tmboot` command uses the following options:

-g *grpname*
: Boots all application servers in groups using this `grpname` parameter.

-o *sequence*
: Boots all servers in the order shown in the SEQUENCE parameter.

-s *server-name*
: Boots an individual server.

-A
: Boots all administrative servers for machines listed in the MACHINES section. This option ensures that the DBBL, BBL, and BRIDGE processes are started in the proper order.

-y
: Provides an automatic “yes” response to the prompt that asks whether all administrative and application servers should be booted. This prompt is displayed only if no options that limit the scope of the command (`-g grpname`, for example) are specified.

Shutting Down the eLink Adapter for CORBA

Use the `tmshutdown` command to shut down all or part of a BEA eLink Platform application. The rules for running this command are similar to those for running `tmboot`.

When the entire application is shut down, `tmshutdown` removes the interprocess communication (IPC) resources associated with the BEA eLink Platform system. The options used by `tmboot` for partial booting (`-A, -g, -I, -S, -s, -l, -M, -B`) are supported in `tmshutdown`.

The `tmshutdown` command does not shut down the administrative server BBL on a machine to which clients are attached. You can use the `-c` option to override this feature. You need this option for occassions when you must bring down a machine immediately and you cannot contact clients.

Running tmshutdown

Only the administrator who has written the `TUXCONFIG` file can execute tmshutdown. The application can be shut down only from the machine designated as `MASTER` in the configuration file. When the `BACKUP` acts as `MASTER`, it is considered to be the `MASTER` for shutdown purposes.

The order in which application servers are shut down is the reverse of the order specified by the `SEQUENCE` parameter for them, or the reverse order in which they are listed in the configuration file. If some servers have `SEQUENCE` numbers and others do not, the unnumbered servers are the first to be shut down, followed by the application servers with `SEQUENCE` numbers (in reverse order). Finally, administrative servers are shut down.

When an application is shut down, all the IPC resources allocated by the BEA eLink Platform system are removed. The `tmshutdown` command does not remove IPC resources allocated by the DBMS.

A Using the Sample Application

The BEA eLink Adapter into WebLogic Enterprise CORBA software includes a Sample Application. This Sample Application provides sample source code, make files, configuration files, and field definitions to help you build an eLink client. You may use this sample application to test your installation of BEA eLink Adapter into WebLogic Enterprise CORBA.

The following topics are discussed in this section:

- What is Included in the Sample Application
- Building and Running the Sample Application on Unix
- Building and Running the Client Program on Windows NT

What is Included in the Sample Application

The Sample Application provided with the BEA eLink Adapter into WebLogic Enterprise CORBA distribution is an eLink client program. The client program, `e2aclient.c`, makes an ATMI service call using the `tpcall()` function passing data in an FML32 buffer. The client program makes one of the two preconfigured service calls, `UPPER` or `LOWER`. `UPPER` converts a string of text to all upper case letters and `LOWER` converts a string of text to all lower case letters.

The sample configuration files provided as part of this distribution create an environment where the eLink Adapter for CORBA, e2a, is the only server that will respond to the `UPPER` and `LOWER` service calls. A complete list and description of all files provided as part of this sample can be found in Table A-1.

Table A-1 List of Sample Application Files

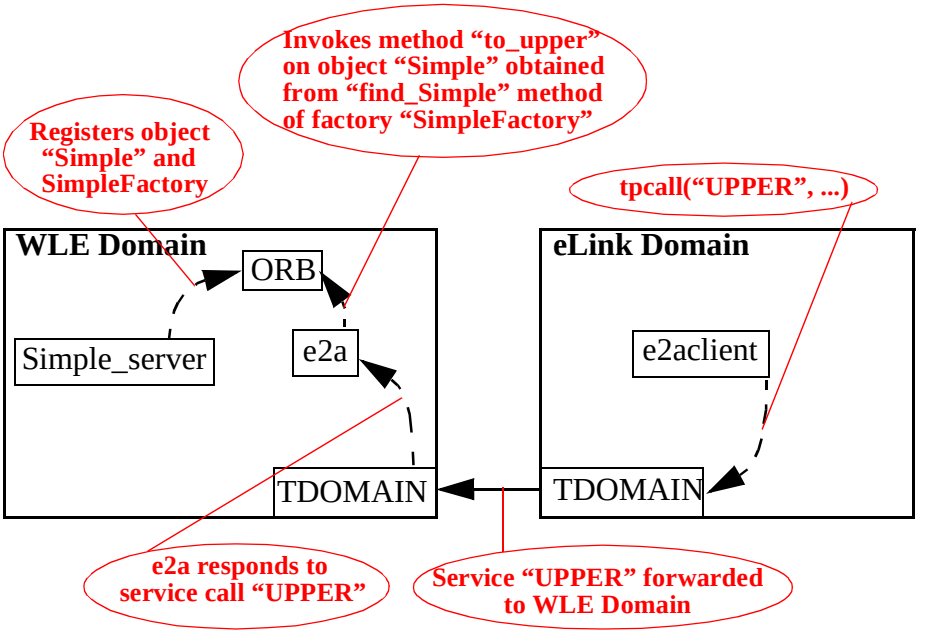
File	Description
readme.txt	A file containing the instructions included in this appendix
e2aclient.c	Client source code.
makefile.nt	NT makefile (MS VC++ 6).
makefile.unix	UNIX makefile.
e2alocal.dom	DM configuration file for e2a to run on UNIX and and handle both local and remote requests from NT.
e2alocal.ubb	UBB configuration file for e2a to run on UNIX and and handle both local and remote requests from NT.
e2aclient.txt	Adapter Configuration file. This lists the services being advertised. This is the command line argument to e2a as specified in the CLOPT= line.
e2aclient.flds	Output from the FTGEN Command Line Tool. Note that this file contains the FML32 field definitions that must be manually added.
e2aremote.dom	DM configuration file for e2aclient to run on NT and communicate with e2a on UNIX.
e2aremote.ubb	UBB configuration file for e2aclient to run on NT and communicate with e2a on UNIX.
setenv.sh	A sample script that sets environment variables.

The Sample Application provided with this distribution requires part of the CORBA Simpapp Application provided with the WebLogic Enterprise distribution. The binary server, `simple_server`, is created by building the `Simpapp` and is used as the CORBA application server. When the eLink Adapter for CORBA translates the service calls to method invocations, the CORBA object, Simple, is the target object

that responds to the method invocation(s). The adapter configuration, `e2aclient.txt`, maps the `UPPER` service call to the `to-upper` method and the `LOWER` service call to the `to-lower` method of the `Simple` object.

The following diagram shows an example of the Sample Application process with a `tpcall` to `UPPER` initiated from the eLink Domain.

Figure 5-1 Example of Sample Application Process



Building and Running the Sample Application on Unix

The Sample Application contains a single eLink client program, `e2aclient.c`. Review the following prerequisites and perform the following steps to build the client program and create the Sample Application domains, eLink (e2alocal) and WLE (e2aremote).

Prerequisites

Before you begin building the sample programs, the following prerequisites should be considered:

- Your environment must be properly configured for the eLink Adapter for CORBA. A `setenv.sh` file ia provided as a sample script for setting the environment variables. Refer to Chapter 4, "eLink Adapter for CORBA Configuration," for more information about configuring the environment.
- Review your WLE documentation and make sure you are familiar with the WLE Simpapp Application.
- The BEA eLink Adapter into WebLogic Enterprise CORBA is installed in your WLE installation directory, `$TUXDIR`.

Step 1 - Building the SIMPAPP Test Server

Since the eLink Adapter for CORBA only runs on Unix, you need only build the Unix version of the Test Server.

1. Compile the `Simpapp` application in the `$TUXDIR/samples/corba/simpapp` directory. The `Readme.txt` file in that directory contains build instructions. There is also a `runme` file that automates the process.

Note: The use of the Solaris 5.x compiler is required for Solaris.

2. Create a test directory, copy the `simple_server` executable and `simple.idl` file to your test directory, and add the `simple_server` entry under the `*SERVERS` section of the `UBB` file.

Step 2 - Creating the Interface Repository

Using the WLE utility function, `idl2ir`, create an Interface Repository (`.ifr`) file from the `simple.idl` file by entering the following command:

```
idl2ir -f e2aclient.ifr simple.idl
```

Edit the `CLOPT` line for the `UBB` file SERVER entry for `TMFIFRSVR` to use the Interface Repository (`ifr`) file created above.

Step 3 - Configuring the Adapter

Configuring the adapter requires to you to modify a adapter configuration file. A sample `e2aclient.txt` adapter configuration file installs as part of the product for the sample application and is available for use.

1. Use the `CFGEN` utility to create a skeleton Adapter configuration file. Refer to "Generating the Adapter Configuration File" for more information on creating a configuration file with `CFGEN`. The hostname and port input to this program can be found in the `CLOPT` line for the ISL server in the `UBB` configuration file (in the form `//hostname:port`) and the interface name input is the `object` name from the `simple.idl` file containing the methods to be called (for example, `Simple`).

Note: WLE (including `ISL` and `TMFIFRSVR`) must be booted before running the `CFGEN` utility.

2. Edit the skeleton file generated in the previous step as follows:

   *SERVER Section

   MINMSGLEVEL
   MAXMSGLEVEL
   Coupled together, these parameters control tracing. They can be left blank (which implies a setting of {0,0}) for minimal tracing or set to

{0,9} for maximum tracing. This affects the number of log messages placed in the ULOG file.

*SERVICE Section(s)

The following SERVICE entries need to be filled in for each of the methods that your client program accesses. Note that the entries filled in by CFGEN (SERVICENAME, OBJECTNAME, METHOD, PARAMS, and RETURN) should not be edited, since any changes to this file may require changes in other files. The data for the FACTORYMETHOD and FACTORYOBJECT fields can be found in the simple.idl file.

SERVICENAME

The advertised service name can be any name. The client program uses this name to request the service from e2a.

REGISTRYTYPE

The Registry location to be used for obtaining an object reference For this example, the REGISTRYTYPE is defined as FactoryFinder.

REGISTERED

The name of the registered factory. You can find the proper value for this field by running the ir2idl utility using as input the interface repository file you created in "Step 2 - Creating the Interface Repository".

For example:

```
idl2ir -f filename.ifr
```

The output consists of a number of lines; use the one that starts with #pragma ID SimpleFactory.

For example:

```
#pragma ID SimpleFactory "IDL:beasys.com/SimpleFactory:1.0"
```

METHOD

The CORBA method to be invoked when the service is called (for this client, to_lower and to_upper).

OBJECTNAME

The object name containing the method to be used (for example, Simple).

FACTORYMETHOD
: The factory method that returns an `OBJECTNAME` type object (for example, `find_simple`).

FACTORYOBJECT
: The factory interface that contains `FACTORYMETHOD` (for example, `SimpleFactory`).

FACTORYPARAMS
: Leave blank. `find_simple()` does not take arguments.

PARAM
: A comma-separated FML field list of arguments matching the prototypes of `METHOD` in the `IDL` file. The actual names can be anything.

RETURN
: An FML32 field name matching the `METHOD` return type. If void, use the key word 'void' and it will not be interpreted as a FML32 field.

  Notice that the two methods defined in `simple.idl` (`to_lower` and `to_upper`) have different function prototypes. While both take an input string, `to_upper` does the conversion in place and does not return a result. On the other hand, `to_lower` returns the converted string.

  The `PARAMS` and `RETURN` entries for `to_upper` should be set as follows:

  ```
  PARAMS=val
  RETURN=void
  ```

  The corresponding entries for `to_lower` should be set as follows:

  ```
  PARAMS=val
  RETURN=Simpleto_lower_RTN
  ```

3. Make sure the `CLOPT` entry for `e2a` in the WLE `UBBCONFIG` file uses the name of the Adapter Configuration file created in step 2 above.

4. Create the FML32 Field Table file. The `FTGEN` command line utility can be used to generate this file. Refer to "Generating an FML32 Field Table File" for more information on using this utility. The hostname and port input to this program can be found in the `CLOPT` line for the `ISL` server in the `UBB` configuration file (in the form `//hostname:port`) and the configuration file input is the file previously generated in step 1.

If e2a is your first eLink Adapter server, you will need to add the following entries to this file:

```
*base 2000
ELINK_ADAPTER_ERR           1       string  -       -
ELINK_ADAPTER_ERR_CODE      2       string  -       -
```

WLE locates this file using the following two environment variables that designate the path and file name of the FML32 Field Table File generated above:

`FIELDTBLS32` contains a list of the table file names.

`FLDTBLDIR32` contains a list of directories to search for the `FIELDTBLS32` files.

Step 4 - Building the Client

The following procedure assumes some familiarity with writing WLE or eLink Platform client applications. Refer to the "Programming" section in the BEA Tuxedo online documentation at /http://edocs.bea.com/tuxedo/tux65/ for more information on writing clients.

Note: All parameters to `e2a` are passed via FML32 buffers.

1. The FML32 Field Table File can be used to generate the `FLDID32 #defines` needed by the client program using the WLE `mkfldhdr32` utility:

   ```
   mkfldhdr32 e2aclient.flds
   ```

   The output file is `e2aclient.flds.h`, which you can `#include` in the client source code file.

Note: The names used in the `PARAMS` and `RETURN` entries in the adapter configuration file (see step 1) are listed in the output file from `FTGEN` tool and are defined as macros in the `.h` file created by `mkfldhdr32`.

2. In the client source code file, enter a `#include` line to include the `.h` output file you created from the `mkfldhdr32` program in the prior step.

   The FML32 buffer to be passed through `tpcall()` must contain all of the input parameters expected by the server program. When adding these fields to the FML32 buffer, the `fieldid` parameter of `Fadd32()` must be one of the

`#define FLDID32` macros listed in the `.h` file generated by the `mkfldhdr32` program.

3. The `svc` parameter to `tpcall()` requesting a service must be the `SERVICENAME` parameter used in the eLink Adapter configuration file created in the second task of Step 3 - Configuring the Adapter.
4. When extracting the returned data, make sure you access the proper return value by checking the `fieldid` parameter returned by `Fnext32` as you iterate through the returned FML32 field(s).
5. Compile the client program using the WLE command line utility, `buildclient`.
6. Run the program, making sure that it communicates with the server.

Building and Running the Client Program on Windows NT

The prior section described building and running the Client Program on a single Unix domain. While the BEA eLink Adapter for CORBA is available only for HP-UX and Solaris platforms, the sample client program can be built on Windows NT either as a Tuxedo or WLE client and configured to talk to a Unix-based WLE server. The sample `e2aclient.c` file compiles on both NT and Unix with no changes. The `UBB` and `DOM` files distributed with the sample application support running the client on NT with a remote connection to a Unix platform and are useful reference material.

Step 1 - Configuring the Server Program

1. Add `DMADM`, `GWADM` and `GWTDOMAIN` entries to the SERVERS group of the existing Server `UBBCONFIG` file.
2. Create a `DMCONFIG` file for the Server containing `DM_LOCAL_DOMAINS`, `DM_REMOTE_DOMAINS`, `DM_TDOMAIN` and `DM_LOCAL_SERVICES` sections. Refer to "Creating the DMCONFIG File" for information on these sections.

Step 2 - Running the Server Program

1. Compile the Server UBBCONFIG and DMCONFIG files and boot WLE by entering the following commands in order:

   a. tmloadcf -y e2alocal.ubb

   b. dmloadcf -y e2alocal.dom

   c. tmboot -y

2. Verify that the local connection still works by running the Unix client program.

Step 3 - Building the Client Program on NT

1. Copy the e2aclient.flds.h, created as part of "The Sample Application provided with this distribution requires part of the CORBA Simpapp Application provided with the WebLogic Enterprise distribution. The binary server, simple_server, is created by building the Simpapp and is used as the CORBA application server. When the eLink Adapter for CORBA translates the service calls to method invocations, the CORBA object, Simple, is the target object that responds to the method invocation(s). The adapter configuration, e2aclient.txt, maps the UPPER service call to the to-upper method and the LOWER service call to the to-lower method of the Simple object.", from Unix to Windows NT.

2. Compile the client program using the WLE command line utility, buildclient (see makefile.nt for an example of usage).

Note: The Microsoft Visual C++ 6.x compiler and nmake is required.

Step 4 - Configuring the Client Program

1. Create a UBBCONFIG file for the Client that contains server entries for DMADM, GWADM, and GWTDOMAIN. See "Adding the e2a Server to the UBBCONFIG File" and "Defining the Domains Environment in the eLink and WLE UBBCONFIG Files" for more information on the required server entries .

2. Create a DMCONFIG file for the client containing DM_LOCAL_DOMAINS, DM_REMOTE_DOMAINS, DM_TDOMAIN, and DM_REMOTE_SERVICES sections. Refer to "Creating the DMCONFIG File" for information on these sections.

Step 5 - Running the Client Program

1. Compile the client UBBCONFIG and DMCONFIG files and boot Tuxedo or WLE by entering the following commands in order:

   a. `tmloadcf -y e2aremote.ubb`

   b. `dmloadcf -y e2aremote.dom`

   c. `tmboot -y`

2. Verify that things are working properly by running the client program (e2aclient).

B Error and Information Messages

The CFGEN and FTGEN configuration tools provided with the eLink Adapter for CORBA issue error, warning, and informational messages to their log files and to the console during execution. The following sections describe the error and informational messages and the appropriate action to take.

Log File Messages for CFGEN and FTGEN Utilities

The CFGEN and FTGEN utilities issue the following log file error and informational messages:

101:INFO	**Method <methodName> for object <interfaceName> returns a data type not supported by FML, Service not Added**	
	DESCRIPTION	A CORBA object method defined in the Interface Repository returns a data type not compatible with FML32. Non FML data types are not supported by the Adapter. A Service is not added to the configuration for the CORBA object method.
	ACTION	None.

102:INFO	**Method <methodName> for object <interfaceName> takes an argument with a data type not supported by FML, Service not Added**	
	DESCRIPTION	A CORBA object method defined in the Interface Repository takes argument(s) of a data type not compatible with FML32. Non FML data types are not supported by the Adapter. A Service is not added to the configuration for the CORBA object method.
	ACTION	None.
103:INFO	**Adding Service <serviceName> to the configuration.....**	
	DESCRIPTION	Service <serviceName> has been added to the configuration file.
	ACTION	None.
104:INFO	**Service <svc.serviceName> already exists not added to configuration**	
	DESCRIPTION	Service <serviceName> is already on the configuration file. Duplicate Service names are not allowed on the configuration file.
	ACTION	None.
105:INFO	**FML Field Table <filename> created containing <number of fields> field definitions**	
	DESCRIPTION	An FML Field Table <filename> has been created containing <number of fields> field definitions
	ACTION	None.
201:WARN	**Unrecognized parameter : <configuration parameter>**	
	DESCRIPTION	An invalid parameter was encountered in the input configuration file.
	ACTION	Check the parameter name to ensure it is spelled correctly.

202:WARN	**Config file specifies more PARAMS than IR for Service : <serviceName> Type information not available for Argument : <argumentName>**	
	DESCRIPTION	The PARAMS parameter for <serviceName> has more arguments than the IDL loaded on the Interface Repository.
	ACTION	Verify the Interface in the Interface Repository and correct the PARAMS parameter to match.
203:WARN	**Config file specifies more FACTORYPARAMS than IR for Service : <serviceName> Type information not available for Argument : <argumentName>**	
	DESCRIPTION	The FACTORYPARAMS parameter for <serviceName> has more arguments than the IDL loaded on the Interface Repository.
	ACTION	Verify the Interface in the Interface Repository for the FACTORYOBJECT/FACTORYMETHOD and correct the FACTORYPARAMS parameter to match.
204:WARN	**Interface name matching <interface name> not found on Interface Repository.**	
	DESCRIPTION	An interface that matches the full or partial interface name entered during CFGEN initialization was not found on the Interface Repository.
	ACTION	Check spelling of the full or partial interface name entered to ensure it is correct. Verify that the desired interface exists on the Interface Repository. Make any necessary corrections and re-execute the CFGEN utility.
301:ERROR	**Service <serviceName> found multiple times in existing configuration**	
	DESCRIPTION	The input configuration file contains duplicate Service names.

	ACTION	Locate the duplicate Service names on the existing configuration and rename or delete one of the Services.
302:ERROR	**Argument <fieldName> not added, definition already exists for this Service <serviceName>**	
	DESCRIPTION	A Service definition contains multiple PARAMS or FACTORYPARAMS with the same argument name.
	ACTION	Ensure that each argument name for a Service definition is unique.
303:ERROR	**Existing argument <fieldName> appears multiple times with different data types**	
	DESCRIPTION	An argument name appears multiple times in the configuration file. Each occurrence does not have the same data type.
	ACTION	Verify that each occurrence of an argument name in the configuration defines arguments of the same data type.
304:ERROR	**Existing Configuration contains a Service with no Service Name**	
	DESCRIPTION	A Service definition in the configuration file has a blank SERVICENAME parameter.
	ACTION	Populate the SERVICENAME with a valid adapter service name.
305:ERROR	**Existing Configuration Service : <serviceName> has no Object Name**	
	DESCRIPTION	Service definition <serviceName> does not have a value for the OBJECTNAME parameter.
	ACTION	Populate the OBJECTNAME with the valid CORBA object name.
306:ERROR	**Existing Configuration Service : <serviceName> has no Method Name**	

DESCRIPTION	Service definition <serviceName> does not have a value for the METHODNAME parameter.
ACTION	Populate the METHODNAME with the valid CORBA method name.

Console Error Messages for CFGEN and FTGEN Utilities

The CFGEN and FTGEN utilities issue the following console error and informational messages:

401:ERROR	**Host name parameter is required to generate configuration**	
	DESCRIPTION	Host name was not entered when attempting to run CFGEN or FTGEN.
	ACTION	Enter the host name for the WLE domain that will be executing the Adapter server. This information can be found in the ISL server definition in the UBBCONFIG file for the WLE domain.
402:ERROR	**Port number parameter is required to generate configuration**	
	DESCRIPTION	Port number was not entered when attempting to run CFGEN or FTGEN.
	ACTION	Enter the port number for the WLE domain that will be executing the Adapter server. This information can be found in the ISL server definition in the UBBCONFIG file for the WLE domain.
403:ERROR	**New configuration file name is required to generate configuration**	
	DESCRIPTION	The new configuration file name was not entered when running CFGEN.
	ACTION	Enter a file name for the new configuration file generated by CFGEN at the prompt.
405:ERROR	**Configuration file name is required to generate FML Field Table**	
	DESCRIPTION	An input configuration file name is required when running FTGEN.

	ACTION	Enter the name of the configuration file containing the FML fields to be added to the FML Field Table.
406:ERROR	**FML Field Table name is required**	
	DESCRIPTION	A file name is required for the FML32 field table that is generated by FTGEN.
	ACTION	Enter a file name for the FML32 field table.
407:ERROR	**Unable to open output file <fileName> ; <errorDetail>**	
	DESCRIPTION	An Error was encountered when trying to open output file <filename>.
	ACTION	Verify that the path exists where the file name is to be created. Verify file permissions on the directory where the file is to be created.
408:ERROR	**Input file <configFile> access failed ; <errorDetail>**	
	DESCRIPTION	Unable to open or read the input file <configFile>.
	ACTION	Verify that the file exists in the current working directory or in the path specified. Verify file permissions to ensure read access.
409:ERROR	**Unable to connect to Interface Repository ; <errorDetail>**	
	DESCRIPTION	An error occurred when trying to connect to the CORBA Interface Repository.
	ACTION	Verify that the WLE domain that the Interface Repository belongs to is booted (and is running the TMIFRSVR and ISL servers). Verify the Host name and Port number used to connect to the Interface Repository.
410:ERROR	**Unable to access the Interface Repository ; <errorDetail>**	
	DESCRIPTION	An error occurred when trying to access to the CORBA Interface Repository.

	ACTION	Verify that the WLE domain that the Interface Repository belongs to is booted (and is running the TMIFRSVR and ISL servers). Verify the Host name and Port number used to connect to the Interface Repository.
411:ERROR	**Attempt to access Method <method> arguments failed for Object <object> ; <errorDetail>**	
	DESCRIPTION	An error occurred when trying to access to the CORBA Interface Repository.
	ACTION	Verify that the WLE domain that the Interface Repository belongs to is booted (and is running the TMIFSVR and ISL servers). Verify the Host name and Port number used to connect to the Interface Repository.
450:ERROR	**Unhandled exception encountered ; <errorDetail>**	
	DESCRIPTION	Execution Error.
	ACTION	Verify that WLE environment variables are set correctly (including TUXDIR and PATH).

Error Messages for eLink Adapter for CORBA Server

The eLink Adapter for CORBA server issues the following error and informational messages:

1001:INFO	**Server shutdown**	
	DESCRIPTION	The e2a server has terminated. Services advertised by this server are no longer available.
	ACTION	None
1002:INFO	**Received buffer type <buffer type> <subtype>**	
	DESCRIPTION	The buffer type and subtype received with the service request.
	ACTION	None
1003:INFO	**getField - Occurs - Field <num> time(s) -- Parameter <num> time(s)**	
	DESCRIPTION	The number of occurrences of each data item.
	ACTION	None
1004:INFO	**getField - FML('<fieldname>',<field type>) << value(<argument type>)**	
	DESCRIPTION	The FML field of name and type is being loaded from the CORBA argument of <argument type>
	ACTION	None
1005:INFO	**Factory Listing - Id <id> Kind**	
	DESCRIPTION	<id> and <kind> for each factory returned from FactoryFinder

	ACTION	None
1007:INFO	**C_Config:~C_Config() recovering resources**	
	DESCRIPTION	Configuration parameters have been unloaded from memory.
	ACTION	None
1008:INFO	**Service list built**	
	DESCRIPTION	All services from the configuration file have been parsed and loaded.
	ACTION	None
1010:INFO	**Found service '<service name>' in service map**	
	DESCRIPTION	A service request was received and a matching configuration SERVICENAME was located.
	ACTION	None
1011:INFO	**advertising service <num>) <service name>**	
	DESCRIPTION	<service name> was advertised as an ATMI service to BEA Tuxedo based on the configuration loaded.
	ACTION	None
1012:INFO	**Configuration loaded. Advertising services...**	
	DESCRIPTION	The configuration file has been parsed successfully. Valid services will be advertised to BEA Tuxedo.
	ACTION	None
1013:INFO	**Service '<service name>' factory method '<method name>' has <num> FML parameter(s)**	
	DESCRIPTION	Configuration <service name> is mapped to CORBA method <method name> with <num> parameters.
	ACTION	None

1014:INFO	**Factory OperationDef contains <num> parameter(s)**	
	DESCRIPTION	The factory method to be invoked calls for <num> parameters.
	ACTION	None
1015:INFO	**Service '<service name>' :method '<method name>'**	
	DESCRIPTION	Configuration <service name> is mapped to CORBA method <method name>
	ACTION	None
1016:INFO	**Service '<service name>':'<method name>' has <num> FML field(s) and <num> parameter(s)**	
	DESCRIPTION	Service to Method map with matching number of arguments in Interface Repository and Configuration file.
	ACTION	None
1017:INFO	**Default context obtained**	
	DESCRIPTION	CORBA default context will be used throughout server operation.
	ACTION	None
1018:INFO	**C_Service:~C_Service() recovering resources**	
	DESCRIPTION	A service's parameters have been unloaded from memory.
	ACTION	None
1019:INFO	**Located Object Interface in IR**	
	DESCRIPTION	Interface Repository lookup was successful
	ACTION	None
1020:INFO	**Located the operation '<operation name>' in IR**	
	DESCRIPTION	Interface Repository lookup was successful.
	ACTION	None

1021:INFO	**Located OperationDef in IR**	
	DESCRIPTION	OperationDef object was constructed successfully.
	ACTION	None
1022:INFO	**C_Service:getNVList() Getting empty NVList**	
	DESCRIPTION	An NVList was successfully created from the OperationDef.
	ACTION	None
1023:INFO	**factory result is type <type>**	
	DESCRIPTION	The OperationDef of the Factory method specifies a return type of <type>.
	ACTION	None
1024:INFO	**C_Service:getObject() '<object name>'**	
	DESCRIPTION	Attempting to acquire an Object reference to <object name> from WLE.
	ACTION	None
1025:INFO	**Creating request**	
	DESCRIPTION	Attempting to create a CORBA:Request object
	ACTION	None
1026:INFO	**Invoking one-way request**	
	DESCRIPTION	The CORBA method was defined as oneway. The request will be attempted and no results will be processed.
	ACTION	
1027:INFO	**Request complete**	
	DESCRIPTION	The request was successful. The CORBA method was invoked.
	ACTION	None

1028:INFO	**Invoking request**	
	DESCRIPTION	The CORBA method is about to be invoked.
	ACTION	None
1029:INFO	**Server raised Exception**	
	DESCRIPTION	The CORBA method raised an Exception.
	ACTION	None
1030:INFO	**setBuffer - Field('<fieled name>',<field type>) >> Parameter-<idx> ('<name>',<type>)**	
	DESCRIPTION	FML field <field name> is being used to populate method parameter <name>.
	ACTION	None
1031:INFO	**setBuffer - Occurs - Field <num> time(s) -- Parameter <num> time(s)**	
	DESCRIPTION	Specifies array depth.
	ACTION	None
1032:INFO	**setBuffer - Unknown type found**	
	DESCRIPTION	Method contained an argument of unknown or unsupported buffer type.
	ACTION	Consult documentation for supported buffer types.
1033:INFO	**setBuffer - SKIPPING parameter[<idx>] <argument name>**	
	DESCRIPTION	<argument name> is not an 'in' or 'inout' method argument.
	ACTION	None
1034:INFO	**return type - <type>**	
	DESCRIPTION	The OperationDef specifies a return argument of type <type>.
	ACTION	None

1035:INFO	**getBuffer - SKIPPING parameter[<idx>] <argument name>**	
	DESCRIPTION	<argument name> is not an 'out' or 'inout' method argument.
	ACTION	None
1036:INFO	**Locating Name Service <NameService name>**	
	DESCRIPTION	Using <NameService name> to lookup Objects.
	ACTION	None
1037:INFO	**Resolving object for <object name> in <registry>**	
	DESCRIPTION	Trying to locate <object name> using registry type <registry>.
	ACTION	None
1038:INFO	**Executing service <service name>**	
	DESCRIPTION	Attempting to invoke method(s) associated with <service name>.
	ACTION	None
3001:ERROR	**Configuration filename must be specified in the CLOPT**	
	DESCRIPTION	The '-C <filename>' parameter in the UBB CLOPT statement was not present.
	ACTION	Use tmunloadcf to verify that the e2a server has CLOPT= -- -C <filename> parameter.
3002:ERROR	**Invalid buffer type received '<buffer type>'**	
	DESCRIPTION	A service call was received that was not an FML32 buffer type.
	ACTION	The Adapter only accepts service calls that contain FML32 data buffers.
3003:ERROR	**Option requires an argument**	
	DESCRIPTION	The CLOPT -C argument was not followed by a filename.

	ACTION	Specify a valid filename after the -C in the Adapter CLOPT parameter.
3004:ERROR	**Unrecognized option**	
	DESCRIPTION	An invalid argument was specified in the UBB CLOPT.
	ACTION	Remove extraneous arguments from the CLOPT parameter of the server.
3005:ERROR	**Failed to initialize, correct CLOPT errors and retry tmboot**	
	DESCRIPTION	An error was detected trying to start the server.
	ACTION	Correct errors noted in previous ULOG messages.
3006:ERROR	**No services to advertise. Startup aborted**	
	DESCRIPTION	After parsing the configuration file, no valid services were detected.
	ACTION	Correct errors in service definitions or add valid service definitions to the configuration file.
3007:ERROR	**Unexpected CORBA Exception detected during initialization.**	
	DESCRIPTION	CORBA:Exception was received from WLE.
	ACTION	Troubleshoot problems with WLE.
3009:ERROR	**CORBA Exception <IDL exception name>**	
	DESCRIPTION	CORBA:Exception was received from WLE.
	ACTION	Attempt to diagnose the cause of the Exception generated from WLE.
3012:ERROR	**Unable to locate InterfaceRepository. Make sure that TMIFRSVR is running.**	
	DESCRIPTION	The Adapter was unable to access the InterfaceRepository in WLE.
	ACTION	Make sure that TMIFRSVR is running and has a valid .ifr database.

3013:ERROR	**Missing variable in Service '<service name>' factory object '<factory>' method '<method name>'**	
	DESCRIPTION	A required configuration parameter is missing.
	ACTION	Specify all required parameters in the *SERVICE section of the Adapter configuration file.
3014:ERROR	**SERVICE <service name> factory parameter count mismatch. config <num> != repository <num>**	
	DESCRIPTION	The factory method OperationDef argument count did not match the number of fields specified in the configuration file.
	ACTION	Make sure the number of FML Fields in the configuration file match the number of arguments in the IDL definition loaded in the repository.
3015:ERROR	**SERVICE <service name> parameter count mismatch. methods <num> != parameters <num>**	
	DESCRIPTION	The CORBA method OperationDef argument count did not match the number of fields specified in the configuration file.
	ACTION	Make sure the number of FML Fields in the configuration file match the number of arguments in the IDL definition loaded in the repository.
3018:ERROR	**SERVICE <service name> requires a RETURN field. Method return type is not void.**	
	DESCRIPTION	The CORBA method OperationDef result count did not match the number of fields specified in the configuration file.
	ACTION	Make sure the number of FML Fields in the configuration file match the number of arguments in the IDL definition loaded in the repository. Use void in the configuration file if the return type is void.

3019:ERROR	**SERVICE <service name> has a void return type and configuration specifies a RETURN.**	
	DESCRIPTION	The CORBA method OperationDef result count did not match the number of fields specified in the configuration file.
	ACTION	Make sure the number of FML Fields in the configuration file match the number of arguments in the IDL definition loaded in the repository. Use void in the configuration file if the return type is void.
3020:ERROR	**Invoking factory method < factory method> - CORBA Exception <exception name>**	
	DESCRIPTION	A CORBA:Exception occurred while trying to invoke the factory method.
	ACTION	Investigate CORBA exception from given factory method invocation.
3021:ERROR	**getField(<field name>) failed. Unable to retrieve parameter.**	
	DESCRIPTION	The FML buffer did not contain the field <field name>.
	ACTION	All fields specified in the configuration file for input arguments to methods must be specified in the FML buffer used when invoking the service.
3033:ERROR	**Unable to locate '<object name>' in '<registry type>'**	
	DESCRIPTION	The Object reference to <object name> could not be obtained.
	ACTION	Make sure the CORBA implementation of this object is loaded in the ORB.
3035:ERROR	**'<operation name>' is not a CORBA Operation**	
	DESCRIPTION	The method name specified in the configuration file is not defined this way in the Interface Repository.
	ACTION	Correct the file or database in error.

3036:ERROR	**'<interface name>' is not a CORBA Interface**	
	DESCRIPTION	The Object name specified in the configuration file is not defined this way in the Interface Repository.
	ACTION	Correct the file or database in error.
3037:ERROR	**Failed to locate Object '<object name>' in Interface Repository.**	
	DESCRIPTION	The <object name> could not be found in the Interface Repository.
	ACTION	Load the Interface Repository with the corresponding IDL.
3038:ERROR	**At least one method name must be specified**	
	DESCRIPTION	The method parameter in the configuration file was not found.
	ACTION	Fix the service definition in the configuration file that is in error.
3039:ERROR	**Advertising service '<service name>' - <tperrno> - <tpstrerror>**	
	DESCRIPTION	A BEA Tuxedo error has occurred.
	ACTION	See BEA Tuxedo System messages for proper action(s). tpadvertise failed with an error.

C Troubleshooting

This section contains information that will help you troubleshoot your BEA eLink Adapter into WebLogic Enterprise CORBA configuration. These tips are listed in the order it is suggested that you follow for troubleshooting, although it is not required that you follow this order.

The following topics are discussed:

- General Rules
- Checking Environment Variables
- Booting WLE
- Booting the CORBA Adapter with Maximum Tracing
- Verifying That All Expected Servers and Services are Running on All Platforms
- Verifying That the Client Program Runs
- Verifying That the Configuration Tools Run

General Rules

The following general rules will help you troubleshoot your system:

- With any error, always check the ULOG file. The ULOG file is a text file that contains error, warning, and informational messages generated by the eLink Platform and components like the eLink Adapter for CORBA.

The ULOG file name defaults to $APPDIR/ULOG.mmddyy. All but the extension can be changed by setting the ULOGPFX value in the MACHINES section of the UBB Configuration file.

For example, given the entry:

```
ULOGPFX="/work/logs/zlog"
```

The log file name on 4 July, 2000, would be:

```
/work/logs/zlog.070400
```

- While running WLE, use the following tmadmin commands:
  - **psr -** lists active servers.
  - **psc -** lists active services.
  - **q -** quits tmadmin.

Checking Environment Variables

Make sure the environment variables are set correctly by running the WLE buildserver and tmadmin command line utilities.

Try entering the following command lines:

- buildserver
  - If you receive a “not found” error, check the following:

    The TUXDIR environment variable is set and points to the WLE installation directory.

    The PATH environment variable includes $TUXDIR/bin.
  - If you receive a “Can't open shared library” error (Unix only), check that the SHLIB_PATH (HP) or LD_LIBRARY_PATH (Solaris) includes $TUXDIR/lib.
  - If you receive a message indicating an unlicensed version, make sure that you have a valid lic.txt file in $TUXDIR/udataobj.
- tmadmin

- If you receive a "TUXCONFIG environment variable not set" error, check that APPDIR and TUXCONFIG environment variables are properly set and TUXCONFIG contains a complete pathname. The usual setting for TUXCONFIG is $APPDIR/tuxconfig.
- Other errors may indicate that TUXCONFIG points to a non-existing or invalid tuxconfig file.
- If tmadmin loads, but you get a "No bulletin board exists." message, WLE is not active, you can access the utilities and enter q (quit) to exit tmadmin.

Booting WLE

If the command line utilities work, most of the global environment variables are set correctly. The next step is to attempt to boot WLE.

- In theMACHINES section of the WLE UBB configuration file, check that the hostname is correct and that the values listed for TUXDIR, TUXCONFIG, and APPDIR are correct and match the values for the global environment variables. Also check that all required servers have valid entries in the SERVERS section.

  Required servers are:

Table 5-1

WLE	TMSYSEVT, TMFFNAME, ISL (note that the TMFFNAME server is listed three times, with different CLOPT entries, all three are required)
NameService	cns (CORBA NameService)
e2a	TMIFRSVR, e2a
Sample	simple_server
Remote Access	DMADM, GWADM, GWTDOMAIN

Note: Detailed information on UBB Configuration files can be found in the **ubbconfig(5)** reference pages in the BEA Tuxedo Reference Manual (**http://e-docs.beasys.com/wle/tuxedo/refman/sect5/index.htm)**. Refer also to the sample UBB files that are distributed with WLE.

- Try entering the following command line:

  `tmloadcf UBBFileName`

  - If you receive a 'tmloadcf cannot run on a non-master node' error, check to see that the host name is correct.
  - If you receive a `TUXCONFIG` error, make sure the value for `TUXCONFIG` specified in the MACHINES section of the `UBB` file agrees with the global version.
  - If you receive 'file open errors' (for example, on attempting to open a `ULOG` file) make sure that `APPDIR` and `TUXDIR` are set correctly.

Booting the CORBA Adapter with Maximum Tracing

The next step is to boot the system with maximum tracing enabled by setting the `MINMSGLEVEL` and `MAXMSGLEVEL` values to 0 and 9 respectively in the Adapter configuration file (this is the file name specified in the `CLOPT` parameter for `e2a`).

Enter the following command line:

`tmboot`

- If nothing boots (not even the BBL), make sure that the `TUXDIR` and `APPDIR` are correct.
- If you get 'Cannot exec, executable file not found' error(s), make sure that the server names are correct and can be found either in the current working directory or in the `PATH`.
- If you get 'Application initialization failure' error(s), check the following entries in the `ULOG` file:
  - The `CLOPT` line for the server(s) that failed is correctly formed.
  - Any files specified in the `CLOPT` line actually exist.
- A message 'Process id=xxxx Assume started (pipe)' can indicate a problem with data, such as a `METHOD` value in the Adapter configuration file. Check both `psr`

and psc commands in tmadmin and verify that all servers and services are running.

Verifying That All Expected Servers and Services are Running on All Platforms

If tmboot seemed to work, use the tmadmin psr command to verify that all servers listed in the UBB Configuration file SERVERS section are running (note that psr will not list DEFAULT). Using the tmadmin psc command, also verify that all services listed in the Adapter Configuration file(s) SERVICE section(s) are running. Refer to the ULOG file for possible information about any missing servers or services.

On Windows NT, tmadmin should show the DMADM, GWADM, and GWTDOMAIN servers are running. The required services should show up with a program name of GWTDOMAIN. The following tmadmin output lists services and servers running after booting WLE with e2aremote.ubb.

Listing 5-2 Output from tmadmin for Windows NT

```
> psr
Prog Name      Queue Name  Grp Name      ID RqDone Load Done Current Service
---------      ----------  --------      -- ------ --------- ---------------
BBL.exe        55432       SITE1          0      0         0 (  IDLE )
DMADM.exe      00001.00001 GW_GRP         1      6       300 (  IDLE )
GWADM.exe      00001.00002 GW_GRP         2      0         0 (  IDLE )
GWTDOMAIN.exe  00001.00003 GW_GRP         3      0         0 (  IDLE )

> psc
Service Name Routine Name Prog Name  Grp Name  ID    Machine  # Done Status
------------ ------------ ---------  --------  --    -------  ------ ------
DMADMIN      DMADMIN      DMADM.exe  GW_GRP     1      SITE1       0 AVAIL
LCL_NT       GWS          GWADM.exe  GW_GRP     2      SITE1       0 AVAIL
GWA          GWA          GWADM.exe  GW_GRP     2      SITE1       0 AVAIL
Simpleto_up+ GWS          GWTDOMAIN+ GW_GRP     3      SITE1       0 AVAIL
Simpleto_lo+ GWS          GWTDOMAIN+ GW_GRP     3      SITE1       0 AVAIL
> q
```

On the Unix platform, `tmadmin` whould show that all servers, including `DMADM`, `GWADM`, and `GWTDOMAIN` are running and that all expected services are available. The following `tmadmin` output lists services and servers running after booting WLE with `e2alocal.ubb`.

Listing 5-3 Output from tmadmin for Unix

```
> psr
Prog Name      Queue Name  Grp Name      ID RqDone Load Done Current Service
---------      ----------  --------      -- ------ --------- ---------------
BBL            55432       SITE2          0     17       850 (  IDLE )
DMADM          00003.00001 GW_GRP         1      6       300 (  IDLE )
simple_server  00002.00001 APP_GRP        1      0         0 (  IDLE )
TMSYSEVT       00001.00001 SYS_GRP        1     20      1000 (  IDLE )
GWADM          00003.00002 GW_GRP         2      0         0 (  IDLE )
TMFFNAME       00001.00002 SYS_GRP        2      3       150 (  IDLE )
GWTDOMAIN      00003.00003 GW_GRP         3      0         0 (  IDLE )
TMFFNAME       00001.00003 SYS_GRP        3      1        50 (  IDLE )
TMFFNAME       00001.00004 SYS_GRP        4      0         0 (  IDLE )
TMIFRSVR       00001.00005 SYS_GRP        5     32      1600 (  IDLE )
cns            00001.00006 SYS_GRP        6      0         0 (  IDLE )
e2a            00002.00010 APP_GRP       10      0         0 (  IDLE )

> psc
Service Name Routine Name Prog Name  Grp Name  ID    Machine  # Done Status
------------ ------------ ---------  --------  --    -------  ------ ------
DMADMIN      DMADMIN      DMADM      GW_GRP     1      SITE2       0 AVAIL
REM_UX       GWS          GWADM      GW_GRP     2      SITE2       0 AVAIL
Simpleto_up+ INVOKE_METH+ e2a        APP_G+    10      SITE2       0 AVAIL
Simpleto_lo+ INVOKE_METH+ e2a        APP_G+    10      SITE2       0 AVAIL
>q
```

Verifying the Remote Connection

After ensuring that all servers and services are running on both platforms, try running the client program on Windows NT:

```
e2aclient upper "for all good men"
```

- If you receive an error message, "tpinit() failed - TPESYSTEM - internal system error", WLE (or Tuxedo) is not running on Windows NT.
- If you receive an error message, "Unable to request Service xxx, rc = 10 (TPESVCERR - server error while handling request), check the following:

1. If there is NOT a ULOG message on the Unix platform indicating a service request, check:

   a. The server is actually running on the Unix platform.

   b. The NWADDR in the NT DMCONFIG file for the server is correct.

2. If there is a ULOG message on the Unix platform with an error message "ERROR: Unable to obtain remote domain id (xxx) information from shared memory", or a similar message in the NT ULOG file, check to see that the DOMAINID fields in the DMCONFIG files match.

```
NT DMCONFIG                          Unix DMCONFIG

*DM_LOCAL_DOMAINS                    *DM_LOCAL_DOMAINS
LCL_NT DOMAINID="LOCAL_NT"           REM_UX  DOMAINID="REMOTE_UX"

*DM_REMOTE_DOMAINS                   *DM_REMOTE_DOMAINS
REM_UX DOMAINID="REMOTE_UX"          LCL_NT  DOMAINID="LOCAL_NT"
```

Verifying That the Client Program Runs

The last troubleshooting step is to see if the client program will run and talk to the server program through the eLink for CORBA Adapter.

Using the sample e2a client program, enter a sample command, such as:

```
e2aclient upper “now is the time”
```

- If you receive an error message, “Unable to request Service xxx , rc = 6 (TPENOENT - no entry found)”, make sure that you are requesting an advertised service by verifying that the service (see the `SERVICENAME` parameters in the Adapter configuration file) is actually being advertised (see the `ULOG` file and use the `tmadmin psc` command) and match the service alias you are using in the client program.

- If you receive an error message, “Unable to request Service xxx, rc = 11 (TPESVCFAIL - application level)”, verify that the `FIELDTBLS32` and `FLDTBLDIR32` environment variables are correctly defined.

Note: You must enter a `tmshutdown` and `tmboot` sequence for any change to these variables to take effect.

Verifying That the Configuration Tools Run

The eLink Adapter for CORBA includes two command line tools that are useful in configuring the Adapter. The CFGEN tool generates a skeleton configuration file using the Interface Repository.

The FTGEN tool generates an FML Field Table File using the configuration file and the Interface repository as input. The following troubleshooting examples apply to either the FTGEN or CFGEN utilities, CFGEN is used as an example.

Start the command line tool by typing `CFGEN.`

- If you get the message 'CFGEN: not found' on the console, verify that the directory where the CFGEN tool is installed (typically $TUXDIR/bin) is in the system PATH.
- If you get the message 'CFGEN[3]: java: not found' on the console verify that the java 1.2 JDK/bin or JRE/bin directory is in the system PATH.
- If you get the message 'Exception in thread “main” java.lang.NoClassDefFoundError: CFGEN on the console, verify that the TUXDIR environment variable is set and points to the directory where WLE is installed.
- If you get any other messages issued by the tools, refer to Appendix B, “Error and Information Messages.”

D BEA eLink Platform Reference

This section provides you with basic information about BEA eLink Platform and FML buffers. The following topics are covered:

- BEA eLink Platform Architecture
- ATMI Runtime Services
- FML32
- eLink Commands

BEA eLink Platform Architecture

The eLink Platform communications application programming interface, Application to Transaction Monitor Interface (**ATMI**), is a collection of runtime services that can be called directly by a C (or COBOL) application. These runtime services provide support for communications, distributed transactions, and system management.

The Management Information Base (**MIB**) maintains a virtual repository of all the configuration and operational information for a runtime eLink environment. The eLink services are implemented using a shared bulletin board (BB) that contains configuration information. This is the dynamic part of the eLink. **Servers** advertise their **services** in the Bulletin Board. The Bulletin Board Liaison (BBL) is an administrative eLink server that is the keeper of the Bulletin Board. There is a BBL on every machine participating in the integration infrastructure; the BBL coordinates

changes to the local copy of the MIB. The Distinguished Bulletin Board Liaison (DBBL) is responsible for propagating global changes to the MIB and is the keeper of the static part of the MIB. The MASTER node is the computer where the DBBL runs.

Administrators use an ASCII file to specify eLink system configuration. This file, called the **UBBCONFIG** file, is used as input by the configuration loading utility, `tmloadcf`. The `tmloadcf` utility generates a binary version of the configuration called the `tuxconfig` file. This binary file is used by the system to construct the Bulletin Board and contains the persistent part of the MIB.

eLink adapters are implemented as servers. They continually check their message queue for service requests. A service is the name of a server interface. Many servers can support a single service, thereby providing for load balancing and a fail-safe mechanism. The mapping of services to servers is recorded in the Bulletin Board. When a service request is made, the Bulletin Board forwards the request to a server (eLink Adapter) that advertises that service. An eLink server advertises a service by posting its name in the Bulletin Board. Services correspond to business functions in target applications such as SAP.

ATMI Runtime Services

The eLink Platform ATMI is a collection of runtime services that can be called directly by a C (or COBOL) application. The ATMI is a compact set of primitives used to open and close resources, begin and end transactions, allocate and free buffers, and provide the communication between adapters and other requestors or responders.

Following is a list of ATMI primitives for the C binding. See the *BEA Tuxedo Reference Guide* at http://edocs.bea.com/tuxedo/tux65/index.htm for detailed information on all the ATMI primitives.

Table 5-2 ATMI Primitives for the C Binding

API Group	C API Name	Description
Client Membership	tpchkauth	Check if authentication is needed
	tpinit	Used by a client to join an application
	tpterm	Used by a client to leave an application
Buffer Management	tpalloc	Create a message
	tprealloc	Resize a message
	tpfree	Free a message
	tptypes	Get a message type and subtype
Message Priority	tpgprio	Get the priority of the last request
	tpsprio	Set priority of the next request
Request/Response	tpcall	Synchronous request/response to **service**
	tpacall	Asynchronous request
	tpgetreply	Receive asynchronous response
	tpcancel	Cancel asynchronous request
Conversational	tpconnect	Begin a conversation with a **service**
	tpdiscon	Abnormally terminate a conversation
	tpsend	Send a message in a conversation
	tprecv	Receive a message in a conversation
Reliable Queueing	tpenqueue	Enqueue a message to an application queue
	tpdequeue	Dequeue a message to an application queue
Event-based	tpnotify	Send unsolicited message to a client
	tpbroadcast	Send message to several clients
	tpsetunsol	Set unsolicited message callback
	tpchkunsol	Check arrival of unsolicited message
	tppost	Post an event message
	tpsubscribe	Subscribe to event messages
	tpunsubscribe	Unsubscribe to event messages

Table 5-2 ATMI Primitives for the C Binding

API Group	C API Name	Description
Transaction Management	tpbegin	Begin a transaction
	tpcommit	Commit the current transaction
	tpabort	Rollback the current transaction
	tpgetlev	Check if in transaction mode
	tpsuspend	Suspend the current transaction
	tpresume	Resume a transaction
	tpscmt	Control commit return
Service Entry and Return	tpsvrinit	**Server** initialization
	tpsvrdone	**Server** termination
	tpreturn	End **service** function
	tpforward	Forward request
Dynamic Advertisement	tpadvertise	Advertise a **service** name
	tpunadvertise	Unadvertise a **service** name
Resource Management	tpopen	Open a resource manager
	tpclose	Close a resource manager

FML32

FML is a set of C language functions for defining and manipulating storage structures called fielded buffers, which contain attribute-value pairs called fields. The attribute is the field's identifier and the associated value represents the field's data content.

FML32 uses 32-bit values for the field lengths and identifiers. BEA eLink Adapters use FML32. FML32 allows for about 30 million fields, and field and buffer lengths of up to about 2 billion bytes. The definitions, types, and function prototypes for FML32 are located in `fml32.h` and functions are located in `-lfml32`. All definitions, types, and function names for FML32 have a "32" suffix (for example, `MAXFBLEN32`, `FLDID32`, `Fchg32`). Also the environment variables are suffixed with "32" (for example, `FLDTBLDIR32`, `FIELDTBLS32`).

FML Buffers

A fielded buffer is composed of field identifier and field value pairs for fixed length fields (for example, long, short), and field identifier, field length, and field value triples for varying length fields.

Figure 5-2 Example of a Fielded Buffer

A field identifier is a tag for an individual data item in a fielded buffer. The field identifier consists of the name of the field number and the type of data in the field. The field number must be in the range 1 to 33,554,431 inclusive for FML32, and the type definition for a field identifier is `FLDID32`.

Field numbers 1 to 100 are reserved for system use and should be avoided. The field types can be any of the standard C language types: `short`, `long`, `float`, `double`, and `char`. Two other types are also supported: `string` (a series of characters ending with a null character) and `carray` (character arrays). These types are defined in `fml32.h` as `FLD_SHORT`, `FLD_LONG`, `FLD_CHAR`, `FLD_FLOAT`, `FLD_DOUBLE`, `FLD_STRING`, and `FLD_CARRAY`.

For FML32, a fielded buffer pointer is of type `FBFR32 *`, a field length has the type `FLDLEN32`, and the number of occurrences of a field has the type `FLDOCC32`.

Fields are referred to by their field identifier in the FML32 interface. However, it is normally easier to remember a field name. There are two approaches to mapping field names to field identifiers. One is a compile-time mapping, the other is a run-time mapping.

Mapping Field Names to Field Identifiers

To avoid naming conflicts, BEA eLink Adapters must use the following run-time mapping method. Field name/identifier mappings can be made available to **FML32** programs at run-time through field table files. Field data types must be specifiable within field table files.

The FML32 interface uses the environment variables, `FLDTBLDIR32` to specify a list of directories where field tables can be found and `FIELDTBLS32` to specify a list of the files that are to be used from the table directories.

Note: The environment variables, `FLDTBLDIR32` and `FIELDTBLS32,` must be set prior to using FML32.

Within application programs, the FML32 function, `Fldid32,` provides for a run-time translation of a field name to its field identifier, and `Fname32` translates a field identifier to its field name. Type conversion should be performed implicitly via FML library functions. Implicit type conversion facilitates component reuse.

Use FML32 symbolic names and retrieve their values using `FLDID32`. The `Mkfldhdr32` function must not be used to build an eLink server because it may cause conflicts with other field IDs.

Any field in a fielded buffer can occur more than once. Many FML32 functions take an argument that specifies which occurrence of a field is to be retrieved or modified. If a field occurs more than once, the first occurrence is numbered 0 and additional occurrences are numbered sequentially. The set of all occurrences make up a logical sequence, but no overhead is associated with the occurrence number (that is, it is not stored in the fielded buffer). If another occurrence of a field is added, it is added at the end of the set and is referred to as the next higher occurrence. When an occurrence other than the highest is deleted, all higher occurrences of the field are shifted down by one (for example, occurrence 6 becomes occurrence 5, 5 becomes 4, etc.).

FML32 Primitives

Following is a summary of some of the FML32 primitives that are used for all eLink programs including general eLink services and adapters. This subset of FML32 primitives should be sufficient to create most eLink clients and servers. For more complete details and code examples, see the *BEA Tuxedo Reference Guide* at http://edocs.beasys.com/tuxedo/tux65/index.htm

Table 5-3 FML32 Primitives

FML Primitive	Description
`Fadd32`	Add new field occurrence
`Fchg32`	Change field occurrence value
`Ffind32`	Find field occurrence in buffer
`Fget32`	Get copy and length of field occurrence
`Fielded32`	Return true if buffer is fielded
`Finit32`	Initialize fielded buffer
`Fldid32`	Map field name to field identifier
`Fneeded32`	Compute size needed for buffer
`Fsizeof32`	Returns the size of an FML32 buffer

Warning: The `Falloc` function allocates FML buffers; however, buffers allocated using `Falloc` cannot be passed in a `tpcall`. **FML32** buffers that will be passed using the `tpcall` or `tpacall` **ATMI** primitives should be allocated by using a `tpalloc` with type parameter set to `FML32`.

Use FML32 symbolic names and retrieve their values using `Fldid32`. Field IDs must be determined dynamically at runtime or during initialization at boot time. The `Mkfldhdr32` function must not be used to build the adapter because it may cause conflicts with other field IDs.

eLink Commands

Commands are used to configure and administer the eLink runtime environment. Refer to *Administering the BEA Tuxedo System* for procedures and administrative tasks that are based on the command-line interface. For details about individual commands, refer to the *BEA Tuxedo Reference Manual*. Both documents may be found online at http://edocs.beasys.com/tuxedo/tux65/index.htm

Commonly Used Tuxedo Commands

Following is a list of the **Tuxedo** commands that are most commonly used.

Table 5-4 Commonly Used Tuxedo Commands

Tuxedo Commands	Description
`buildclient`	Constructs a BEA Tuxedo client module. This command combines the files supplied by the -f and -l options with the standard BEA Tuxedo libraries to form a load module and invokes the platform's default compiler to perform the build.
`buildserver`	Constructs a BEA Tuxedo server load module. This command generates a stub file containing a main() function and invokes the platform's default compiler to perform the build.
`tmadmin`	Invokes the BEA Tuxedo bulletin board command interpreter.
`tmboot`	Invokes a BEA Tuxedo application with a configuration defined by the options specified.
`tmloadcf`	Parses a `UBBCONFIG` file and load binary `TUXCONFIG` configuration file.
`tmshutdown`	Shuts down a set of BEA Tuxedo servers.

Table 5-4 Commonly Used Tuxedo Commands

Tuxedo Commands	Description
`ud32`	Runs the BEA Tuxedo ud32 client that reads a tab delimited text file, produces an FML32 buffer, and uses the buffer to make a request to a specified service.

Commonly Used tmadmin Commands

The `tmadmin` command allows you to inspect and dynamically configure your eLink application. There are many commands that can be invoked from `tmadmin`, probably the most important being help. Several of the most useful commands are summarized in the following table.

Table 5-5 Commonly Used tmadmin Commands

Command	Description
`help`	Prints help messages.
`quit`	Terminates the session.
`pclt`	Prints information for the specified set of client processes.
`psr`	Prints information for application and administrative servers.
`psc`	Prints information for application and administrative services.
`susp`	Suspends services.

For details about tmadmin commands, refer to the *BEA Tuxedo Reference Manual* at http://edocs.beasys.com/tuxedo/tux65/index.htm.

Glossary

A

application

In the WebLogic Enterprise system, a single computer program designed to do a certain type of work.

application programming interface (API)

The verbs and environment that exist at the application level to support a particular system software product. A set of well-defined programming interfaces (that it, entry points, calling parameters, and return values) by which one software program uses the services of another.

ATMI

Application to Transaction Monitor Interface. The application interface to the BEA eLink Platform system that includes transaction routines, message handling routines, service interface routines, and buffer management routines.

B

BEA eLink Platform application

One or more BEA eLink Platform domains cooperating to support a single business function.

BEA eLink Platform domain

A collection of BEA eLink Platform servers, services, interfaces, machines, and associated resource managers defined by a single `UBBCONFIG` (ASCII version) or `TUXCONFIG` (binary version) configuration file.

BEA eLink Platform System

The BEA eLink Platform software as the customer receives it from BEA Systems, Inc.

BEA WebLogic Enterprise Software

The BEA WebLogic Enterprise product as the customer receives it from BEA Systems, Inc.

BEA WebLogic Enterprise System

The BEA WebLogic Enterprise software and the hardware on which the WebLogic Enterprise software is running.

C

client

Any code that invokes an operation on a distributed object.

CORBA

Common Object Request Broker Architecture. A multivendor standard published by the Object Management Group for distributed object-oriented computing.

CORBA Interface

A description of an object type and its methods. At runtime, a CORBA Object implements an interface. The interface specifies methods sufficient to represent a given business entity and its behaviors.

CORBA Method

Referred to as an operation in the CORBA specification, this is a named function on an object. Methods operate only on their associated object and exist in a namespace unique to their class. Method names may be any length. CORBA objects (more specifically their interfaces) are specified using Interface Definition Language (IDL). IDL explicitly defines the data contract for a method (i.e. what arguments a method requires, and the types for those arguments).

CORBA Naming Service

The CORBA service that defines conventions for creating, deleting, copying, and moving objects.

CORBA Object

An instance of a CORBA interface that represents a single business entity and the implementation for that interface. Any number of objects may be instantiated for a given interface definition.

CORBA ORB

Any Object Request Broker (ORB) that complies with the CORBA standard. A CORBA ORB is a communications intermediary between client and server applications that typically are distributed across a network. The WebLogic Enterprise ORB is a CORBA ORB.

CORBA Services

A set of system services for objects that were developed for the programmer. These services, defined in OMG IDL by the OMG, can be used to create objects, control access to objects, track objects and object references, and control the relationship between types of objects. Programmers can call object service functions instead of writing and calling their own private object service functions.

D

Domain Configuration (`DMCONFIG`) File

The file that describes the relationship between the local domain (the domain in which the `DMCONFIG` file resides) and remote domains (any other domains). There is one `DMCONFIG` file per domain. The `DMCONFIG` file contains domain information for BEA eLink Platform domains and for WebLogic Enterprise domains.

E

EJB home interface

An EJB component interface that allows clients to look up and/or create EJBs.

eLink Application

A collection of operating system processes that together form a business application designed to accomplish a specific business task.

eLink Domain

Synonymous with eLink application.

eLink Service

A named function entry point which can perform some application function. A service invocation is the basic unit of work in an eLink application. Services are not associated with any particular object within a system and are global in scope. All services in an application exist within a single namespace. Service names are

limited to 15 characters in length. Services have no explicit data contract with a user, and pass data using typed buffers. The data contract is embodied in the actual implementation code for the service.

Enterprise Java Beans (EJB)

An API specification for building scalable, distributed, component-based, multitiered applications. EJBs leverage and extend the JavaBeans component model to provide rich, object-oriented transactional environment for developers creating enterprise applications.

F

factory

Any distributed CORBA object that returns an object reference to other distributed CORBA objects. A factory is located in the server application.

factory finder

A CORBA object that locates the factories that an application needs. Both client applications and server applications can use a factory finder. A factory finder object provides an implementation of the CORBAservices `COSLifeCycle.factoryFinder` interface, as well as the BEA `Tobj.FactoryFinder` interface.

factory_finder.ini file

The FactoryFinder configuration file for domains. This file is parsed by the TMFFNAME service when it is started as a Master NameManager. The file contains information used by NameManager to control the import and the export of object references for factory objects with other domains.

I

Interface Definition Language (IDL)

Specified in the CORBA specification, IDL defines the interfaces for all objects in a CORBA application. The interface definitions include object names, method names, method argument names and types, etc.

Interface Repository

An online database that contains the definitions of the interfaces that determine the CORBA contracts between client and server applications.

L

Local Factory

A factory object that exists in the local domain that is made available to remote domains through WebLogic Enterprise factory finder.

R

Remote Factory

A factory object that exists in a remote domain that is made available to the application through a WebLogic Enterprise factory finder.

T

TMFFNAME

A server application provided by BEA Systems, Inc. that runs the FactoryFinder and supporting NameManager services that maintain a mapping of application-supplied names to object references.

TMIFRSVR

A server application provided by BEA Systems, Inc. for accessing the Interface Repository API defined by CORBA.

Tuxedo

Middleware software that manages applications and transactions. Tuxedo also provides application development tools for writing distributed applications.

`TUXCONFIG` **file**

The binary version of the configuration file for a BEA WebLogic Enterprise or a BEA eLink Platform application. This file is accessed by all BEA WebLogic Enterprise and BEA eLink Platform processes for all configuration information.

U

`UBBCONFIG` **file**

eLink configuration file used to control the runtime behavior of an eLink application. This file describes the servers that will run in the application.

W

WebLogic Enterprise Domain

A collection of WebLogic Enterprise or BEA eLink Platform servers, services, interfaces, machines, and associated resource managers defined by a single `UBBCONFIG` (ASCII version) or `TUXCONFIG` (binary version) configuration.

Index

G

I

J

N

O

P

R

S

T