PROGRESS® ORBIX®

Orbix E2A | Application Server Platform CORBA Migration and Interoperability Guide

Version 3.3, SP11 March 2012

PROGRESS software

BUSINESS MAKING PROGRESS™

Progress Orbix v3.3.11

Updated: 07-Mar-2012

Contents

Preface

This document explains how to migrate applications from the Orbix and OrbixWeb products, which conform to CORBA 2.1, to Orbix E2A v5.0, which conforms to CORBA 2.4.

Orbix documentation is periodically updated. New versions between releases are available at this site:

http://communities.progress.com/pcom/docs/DOC-105220

If you need assistance with Orbix or any other Progress products, go to http://www.progress.com/orbix/orbix-support.html.

If you want to provide any comments on Progress documentation, go to http://www.progress.com/en/about/contact.html.

Audience

This document is aimed at *C++ or Java programmers* who are already familiar with Progress Software's Orbix or OrbixWeb products and who now want to migrate all or part of a system to use Orbix E2A v5.0.

Parts of this document are relevant also to *administrators* familiar with Orbix and OrbixWeb administration. See "Administration" on page 93 and "Configuring for Interoperability" on page 103.

Organization of this Guide

This guide is divided as follows:

Part I “Overview of Migration”

This part briefly discusses the advantages of migrating and the options for your migration strategy.

Part II “Migrating to Orbix E2A”

This part explains how to migrate client and server source (in C++ or Java) to Orbix E2A. For each of the features that have been modified or removed from Orbix E2A, relative to the features supported by Orbix and OrbixWeb, this part discusses the replacement features offered by Orbix E2A.

Part III “Interoperability”

This part discusses the issues that affect a mixed deployment of interoperating Orbix, OrbixWeb and Orbix E2A applications. With appropriate customization of the ORB configuration, you can obtain an optimum level of compatibility between the various applications in your system.

Additional Related Resources

The Progress knowledge base contains helpful articles, written by experts, about the Orbix and other products. You can access the knowledge base at the following location:

http://progresscustomersupport-survey.force.com/OpenEdgeKB/KnowledgeHome

The Progress update center contains the latest releases and patches for Progress products:

http://www.progress.com/en/support/index.html

Typographical Conventions

This guide uses the following typographical conventions:

`Constant width`	Constant width (courier font) in normal text represents portions of code and literal names of items such as classes, functions, variables, and data structures. For example, text might refer to the `CORBA::Object` class. Constant width paragraphs represent code examples or information a system displays on the screen. For example: `#include <stdio.h>`
Italic	Italic words in normal text represent *emphasis* and *new terms*. Italic words or characters in code and commands represent variable values you must supply, such as arguments to commands or path names for your particular system. For example: `% cd /users/`*your_name* Note: some command examples may use angle brackets to represent variable values you must supply. This is an older convention that is replaced with *italic* words or characters.

Keying Conventions

This guide may use the following keying conventions:

No prompt	When a command's format is the same for multiple platforms, a prompt is not used.
`%`	A percent sign represents the UNIX command shell prompt for a command that does not require root privileges.
`#`	A number sign represents the UNIX command shell prompt for a command that requires root privileges.
>	The notation > represents the DOS, Windows NT, Windows 95, or Windows 98 command prompt.
`...` . . .	Horizontal or vertical ellipses in format and syntax descriptions indicate that material has been eliminated to simplify a discussion.
[]	Brackets enclose optional items in format and syntax descriptions.
{}	Braces enclose a list from which you must choose an item in format and syntax descriptions.
|	A vertical bar separates items in a list of choices enclosed in {} (braces) in format and syntax descriptions.

Part I

Overview of Migration

In This Part

This part contains the following chapter:

Introduction

The newest generation of Progress tools provide significant advances over the previous generation of products.

In This Chapter

This chapter discusses the following topics:

Orbix E2A Advantages

Upgrading Orbix

The recommended path for customers upgrading to a new version of Orbix is to move to Orbix E2A. Because Orbix E2A is a CORBA 2.4-compliant ORB, it offers many new features over previous versions of Orbix:

- Portable interceptor support.
- Codeset negotiation support.
- Value type support.
- Asynchronous method invocation (AMI) support.
- Persistent State Service (PSS) support.
- Dynamic any support.

Unique Features

Orbix E2A also offers some unique benefits over other commercial ORB implementations, including:

- ORB extensibility using Progress Software's patented Adaptive Runtime Technology.

  Orbix E2A has a modular structure built on a micro-kernel architecture. Required ORB modules, ORB plug-ins, are specified in a configuration file and loaded at runtime, as the application starts up. The advantage of this approach is that new ORB functionality can be dynamically loaded into an Orbix application without rebuilding the application.
- Improved performance.

  The performance of Orbix E2A has been optimized, resulting in performance that is faster than Orbix 3.x and OrbixWeb 3.x in every respect.
- Advanced deployment and configuration.

  Orbix E2A supports a flexible model for the deployment of distributed applications. Applications can be grouped into configuration domains and organized either as file-based configuration domains or as configuration repository-based configuration domains.

- Rapid application development using the Orbix code generation toolkit.

  The code generation toolkit is an extension to the IDL compiler that generates a working application prototype—based on your application IDL—in a matter of seconds.

Migration Resources

Overview of Resources

Progress is committed to assisting you with your migration effort to ensure that it proceeds as easily and rapidly as possible. The following resources are currently available:

- Progress Software's migration web page, including the latest news and links to further resources, is at:

  http://www.progress.com/en/downloads.html.
- This migration and interoperability guide.

  This technical document provides detailed guidance on converting source code to Orbix E2A. The document aims to provide comprehensive coverage of migration issues, and to demonstrate how features supported in earlier Orbix versions can be mapped to Orbix E2A features.
- Professional Services migration packages

  Progress Software's Professional Services organization has put together a set of consultancy packages that facilitate rapid migration to Orbix E2A. Details of Professional Services assessment and migration packages are available at:

  http://www.progress.com/en/services/services.html.

Migration Options

Overview

The possible migration options are:

- Migrating to Orbix E2A
- Mixed Deployment

In This Section

This section contains the following subsections:

Migrating to Orbix E2A

Overview

The CORBA 2.4 specification, on which Orbix E2A is based, standardizes almost every aspect of CORBA programming. Migrating your source code to Orbix E2A, therefore, represents a valuable investment because your code will be based on a stable, highly standardized programming interface.

Client Side

On the client side, the main issue for migration is that the Orbix `_bind()` function is not supported in Orbix E2A. The CORBA Naming Service is now the recommended mechanism for establishing contact with CORBA servers.

Server Side

On the server side, the basic object adapter (BOA) must be replaced by the portable object adapter (POA). This is one of the major differences between the CORBA 2.1 and the CORBA 2.4 specifications. The POA is much more tightly specified than the old BOA; hence server code based on the POA is well standardized.

Proprietary Features

Orbix 3.x and OrbixWeb 3.x support a range of proprietary features not covered by the CORBA standard-for example, the Orbix locator, filters, loaders, smart proxies, transformers and I/O callbacks. When migrating to Orbix E2A, the proprietary features must be replaced by standard CORBA 2.4 features. This migration guide details how each of the proprietary features can be replaced by equivalent Orbix E2A functionality.

Further Details

The details of migrating to Orbix E2A are discussed in Part II of this guide. See "Migrating to Orbix E2A" on page 25.

Mixed Deployment

Overview

Mixed Deployment is appropriate when a number of CORBA applications are in deployment simultaneously. Some applications might be upgraded to use Orbix E2A whilst others continue to use Orbix 3.x and OrbixWeb 3.x. This kind of mixed environment requires on-the-wire compatibility between the generation 3 products and Orbix E2A. Extensive testing has been done to ensure interoperability with Orbix E2A.

On-the-Wire Interoperability

Both Orbix 3.3 and Orbix E2A have been modified to achieve an optimum level of on-the-wire compatibility between the two products.

Further Details

Interoperability is discussed in Part III of this guide. See “Interoperability” on page 101.

Part II
Migrating to Orbix E2A

In This Part

This part contains the following chapters:

CHAPTER 2

IDL Migration

This chapter discusses the Orbix 3.x IDL features that are not available in Orbix E2A.

In This Chapter

This chapter discusses the following topics:

The context Clause

IDL Syntax

According to IDL grammar, a `context` clause can be added to an operation declaration, to specify extra variables that are sent with the operation invocation. For example, the following `Account::deposit()` operation has a `context` clause:

```
//IDL

interface Account {
    void deposit(in CashAmount amount)
             context("sys_time", "sys_location");
    //...
};
```

Migrating to Orbix E2A

The `context` clause is not supported by Orbix E2A. IDL contexts are generally regarded as type-unsafe. Orbix clients that use them need to be migrated, to transmit their context information using another mechanism, such as service contexts, or perhaps as normal IDL parameters.

The opaque Type

Migrating to Orbix E2A

The object-by-value (OBV) specification, introduced in CORBA 2.3 and supported in Orbix E2A, replaces opaques. .

The Principal Type

Not Supported in Orbix E2A

The CORBA specification deprecates the `Principal` IDL type; therefore the `Principal` IDL type is not supported by Orbix E2A.

Interoperability

Orbix E2A has some limited on-the-wire support for the `Principal` type, to support interoperability with Orbix 3.x applications.

See “Launch and Invoke Rights” on page 106.

CHAPTER 3

Client Migration

Migration of client code from Orbix 3 to Orbix E2A is generally straightforward, because relatively few changes have been made to the client-side API.

In This Chapter

The following topics are discussed in this chapter:

Replacing the _bind() Function

Overview

The `_bind()` function is not supported in Orbix E2A. All calls to `_bind()` must be replaced by one of the following:

- CORBA Naming Service.
- CORBA Trader Service.
- Object-to-string conversion.
- corbaloc URL.
- The `ORB::resolve_initial_references()` operation.

CORBA Naming Service

The naming service is the recommended replacement for `_bind()` in most applications. Migration to the naming service is straightforward on the client side. The triplet of `markerName`, `serverName`, and `hostName`, used by the `_bind()` function to locate an object, is replaced by a simple name in the naming service.

When using the naming service, an object's name is an abstraction of the object location and the actual location details are stored in the naming service. Object names are resolved using these steps:

1. An initial reference to the naming service is obtained by calling `resolve_initial_references()` with `NameService` as its argument.
2. The client uses the naming service reference to resolve the names of CORBA objects, receiving object references in return.

Orbix E2A supports the CORBA Interoperable Naming Service, which is backward-compatible with the old CORBA Naming Service and adds support for stringified names.

CORBA Trader Service

Orbix E2A Trader provides advanced capabilities for object location and discovery. Unlike the Orbix Naming Service where an object is located by name, an object in the Trading Service does not have a name. Rather, a server advertises an object in the Trading Service based on the kind of service provided by the object. A client locates objects of interest by asking the Trading Service to find all objects that provide a particular service. The client can further restrict the search to select only those objects with particular characteristics.

Object-to-String Conversion

CORBA offers two CORBA-compliant conversion functions:

```
CORBA::ORB::object_to_string()
CORBA::ORB::string_to_object()
```

These functions allow you to convert an object reference to and from the stringified interoperable object reference (stringified IOR) format. These functions enable a CORBA object to be located as follows:

1. A server generates a stringified IOR by calling `CORBA::ORB::object_to_string()`.
2. The server passes the stringified IOR to the client, for example by writing the string to a file.
3. The client reads the stringified IOR from the file and converts it back to an object reference, using `CORBA::ORB::string_to_object()`.

Because they are not scalable, these functions are generally not useful in a large-scale CORBA system. Use them only to build initial prototypes or proof-of-concept applications.

corbaloc URL

A *corbaloc URL* is a form of human-readable stringified object reference. If you are migrating your clients to Orbix E2A but leaving your servers as Orbix 3.3 applications, the corbaloc URL offers a convenient replacement for `_bind()`.

To access an object in an Orbix 3.3 server from an Orbix E2A client using a corbaloc URL, perform the following steps:

1. Obtain the object key, *ObjectKey*, for the object in question, as follows:
   i. Get the Orbix 3.3 server to print out the stringified IOR using, for example, the `CORBA::ORB::object_to_string()` operation. The result is a string of the form `IOR:00...`
   ii. Use the Orbix E2A `iordump` utility to parse the stringified IOR. Copy the string that represents the object key field, *ObjectKey*.
2. Construct a corbaloc URL of the following form:

   `corbaloc:iiop:1.0@`*DaemonHost*`:`*DaemonPort*`/`*ObjectKey*`%00`

   Where *DaemonHost* and *DaemonPort* are the Orbix daemon's host and port respectively. A null character, `%00`, is appended to the end of the *ObjectKey* string because Orbix 3.3 applications expect object key strings to be terminated by a null character.

3. In the source code of the Orbix E2A client, use the `CORBA::ORB::string_to_object()` operation to convert the corbaloc URL to an object reference.

The general form of a corbaloc URL for this case is, as follows:

`corbaloc:iiop:`*GIOPVersion*`@`*Host*`:`*Port*`/`*Orbix3ObjectKey*`%00`

Where the components of the corbaloc URL are:

- *GIOPVersion*—The maximum GIOP version acceptable to the server. Can be either `1.0` or `1.1`.
- *Host* and *Port*—The daemon's (or server's) host and port. The *Host* can either be a DNS host name or an IP address in dotted decimal format.

The *Orbix3ObjectKey* has the following general form:

`:\`*Host*`:`*SvrName*`:`*Marker*`::`*IFRSvrName*`:`*InterfaceName*`%00`

Where the components of the Orbix 3 object key are:

- *Host*—The server host. The *Host* can either be a DNS host name or an IP address in dotted decimal format.
- *SvrName*—The server name of the Orbix 3.3 server.
- *Marker*—The CORBA object's marker.
- *IFRSvrName*—Can be either `IR` or `IFR`.
- *InterfaceName*—The object's IDL interface name.

WARNING: Constructing an Orbix 3.3 object key directly based on the preceding format does *not* always work because some versions of Orbix impose extra restrictions on the object key format. Extracting the object key from the server-generated IOR is a more reliable approach.

If you encounter any difficulties with using corbaloc URLs, please visit http://www.progress.com/en/support/index.html.

ORB::resolve_initial_references()

The `CORBA::ORB::resolve_initial_references()` operation provides a mechanism for obtaining references to basic CORBA objects, for example the naming service, the interface repository, and so on.

Orbix E2A allows the `resolve_initial_references()` mechanism to be extended. For example, to access the `BankApplication` service using `resolve_initial_references()`, simply add the following variable to the Orbix E2A configuration:

```
# Orbix E2A Configuration File
initial_references:BankApplication:reference =
   "IOR:010347923849..."
```

Use this mechanism sparingly. The OMG defines the intended behavior of `resolve_initial_references()` and the arguments that can be passed to it. A name that you choose now might later be reserved by the OMG. It is generally better to use the naming service to obtain initial object references for application-level objects.

Callback Objects

POA Policies for Callback Objects

Callback objects must live in a POA, like any other CORBA object; hence, there are certain similarities between a server and a client with callbacks. The most sensible POA policies for a POA that manages callback objects are:

Table 1: *POA Policies for Callback Objects*

Policy Type	Policy Value
Lifespan	`TRANSIENT`[a]
ID Assignment	`SYSTEM_ID`
Servant Retention	`RETAIN`
Request Processing	`USE_ACTIVE_OBJECT_MAP_ONLY`

a. By choosing a `TRANSIENT` lifespan policy, you remove the need to register the client with an Orbix E2A locator daemon.

These policies allow for easy management of callback objects and an easy upgrade path. Callback objects offer one of the few cases where the root POA has reasonable policies, provided the client is multi-threaded (as it normally is for callbacks).

IDL-to-C++ Mapping

Overview

The definition of the IDL-to-C++ mapping has changed little going from Orbix 3.x to Orbix E2A (apart from some extensions to support valuetypes). Two notable changes are:

- the `CORBA:Any` type.
- the `CORBA:Environment` parameter.

The CORBA::Any Type

In Orbix E2A, it is not necessary to use the type-unsafe interface to `Any`. Recent revisions to the CORBA specification have filled the gaps in the IDL-to-C++ mapping that made these functions necessary. That is, the following functions are deprecated in Orbix E2A:

```
// C++
// CORBA::Any Constructor.
Any(
    CORBA::TypeCode_ptr tc,
    void* value,
    CORBA::Boolean release = 0
);

// CORBA::Any::replace() function.
void replace(
    CORBA::TypeCode_ptr,
    void* value,
    CORBA::Boolean release = 0
);
```

The `CORBA::Environment` Parameter

The signatures of IDL calls no longer contain the `CORBA::Environment` parameter. This parameter was needed for languages that did not support native exception handling. However, Orbix applications also use it for operation timeouts.

System Exception Semantics

Overview

Orbix and OrbixWeb clients that catch specific system exceptions might need to change the exceptions they handle when they are migrated to Orbix 2000.

System Exceptions

Orbix E2A follows the latest CORBA standards for exception semantics. The two system exceptions most likely to affect existing code are:

Table 2: *Migrated System Exceptions*

When This Happens	Orbix 3 and OrbixWeb Raise	Orbix E2A Raises
Server object does not exist	`INV_OBJREF`	`OBJECT_NOT_EXIST`
Cannot connect to server	`COMM_FAILURE`	`TRANSIENT`

Minor Codes

System exception minor codes are completely different between OrbixWeb 3.2 and Orbix E2A for Java. Applications which examine minor codes need to be modified to use Orbix E2A for Java minor codes.

Dynamic Invocation Interface

Proprietary Dynamic Invocation Interface

Orbix-proprietary dynamic invocation interface (DII) functions are not available in Orbix E2A. Code that uses `CORBA::Request::operator<<()` operators and overloads must be changed to use CORBA-compliant DII functions.

Note: Orbix E2A generated stub code consists of sets of statically generated CORBA-compliant DII calls.

Server Migration

Server code typically requires many more changes than client code. The main issue for server code migration is the changeover from the basic object adapter (BOA) to the portable object adapter (POA).

In This Chapter

This chapter discusses the following topics:

Function Signatures

Changes to the Signature

In Orbix E2A, two significant changes have been made to C++ function signatures:

- The `CORBA::Environment` parameter has been dropped.
- New types are used for `out` parameters. An `out` parameter of `T` type is now passed as a `T_out` type.

Consequently, when migrating C++ implementation classes you must replace the function signatures that represent IDL operations and attributes.

Object IDs versus Markers

C++ Conversion Functions

Orbix E2A uses a sequence of octets to compose an object's ID, while Orbix 3 uses string markers. CORBA provides the following helper methods:

```
// C++
// Converting string marker -----> ObjectId
PortableServer::ObjectId *
PortableServer::string_to_ObjectId(const char *);

// Converting ObjectId -----> string marker
char *
PortableServer::ObjectId_to_string(
    const PortableServer::ObjectId&
);
```

to convert between the two types; hence migration from marker dependencies to Object IDs is straightforward.

Java Conversion Functions

In Java, an object ID is represented as a byte array, `byte[]`. Hence the following native Java methods can be used to convert between string and object ID formats:

```
// Java
// Converting string marker -----> ObjectId
byte[]
java.lang.String.getBytes();

// Converting ObjectId -----> string marker
// String constructor method:
java.lang.String.String(byte[]);
```

CORBA Objects versus Servant Objects

Orbix 3

In Orbix 3 there is no need to distinguish between a CORBA object and a servant object. When you create an instance of an implementation class in Orbix 3, the instance already has a unique identity (represented by a marker) and therefore represents a unique CORBA object.

Orbix E2A

In Orbix E2A, a distinction is made between the identity of a CORBA object (its object ID) and its implementation (a servant). When you create an instance of an implementation class in Orbix E2A, the instance is a servant object, which has no identity. The identity of the CORBA object (represented by an object ID) must be grafted on to the servant at a later stage, in one of the following ways:

- The servant becomes associated with a unique identity. This makes it a CORBA object, in a similar sense to an object in a BOA-based implementation.
- The servant becomes associated with multiple identities. This case has no parallel in a BOA-based implementation.

The mapping between object IDs and servant objects is controlled by the POA and governed by POA policies.

BOA to POA Migration

Overview

It is relatively easy to migrate a BOA-based server by putting all objects in a simple POA that uses an active object map; however, this approach is unable to exploit most of the functionality that a POA-based server offers. It is worth while redesigning and rewriting servers so they benefit fully from the POA.

In This Section

The Orbix 3 BOA is replaced by the POA in Orbix E2A. This affects the following areas of CORBA application development:

Creating an Object Adapter

Creating a BOA in Orbix 3.x

In Orbix 3, a single BOA instance is used. All CORBA objects in a server are implicitly associated with this single BOA instance.

Creating a POA in Orbix E2A

In Orbix E2A, an application can create multiple POA instances (using the `PortableServer::POA::create_POA()` operation in C++ and the `org.omg.PortableServer.create_POA()` operation in Java). Each POA instance can be individually configured, using POA policies, to manage CORBA objects in different ways. When migrating to Orbix E2A, you should give careful consideration to the choice of POA policies, to obtain the maximum benefit from the POA's flexibility.

Defining an Implementation Class

Overview

There are two approaches to defining an implementation class in CORBA:

- "The Inheritance Approach"
- "The Tie Approach"

The Inheritance Approach

The most common approach to implementing an IDL interface in Orbix is to use the inheritance approach. Consider the following IDL fragment:

```
//IDL
module BankSimple {
    Account {
        //...
    };
};
```

The `BankSimple::Account` IDL interface can be implemented by defining a class that inherits from a standard base class. The name of this standard base class for Orbix 3 and Orbix E2A is shown in Table 3.

Table 3: *Standard Base Classes for the Inheritance Approach*

Description	Base Class Name
Orbix 3, C++ (BOA)	`BankSimple::AccountBOAImpl`
Orbix E2A, C++ (POA)	`POA_BankSimple::Account`
Orbix 3, Java (BOA)	`BankSimple._AccountImplBase`
Orbix E2A, Java (POA)	`BankSimple.AccountPOA`

Consider a legacy Orbix 3 application that implements `BankSimple::Account` in C++ as the `BankSimple_Account_i` class. The `BankSimple_Account_i` class might be declared as follows:

```
// C++
// Orbix 3 Version
// Inheritance Approach
class BankSimple_Account_i : BankSimple::AccountBOAImpl {
public:
    // Declare IDL operation and attribute functions...
};
```

When this implementation class is migrated to Orbix E2A, the `BankSimple::AccountBOAImpl` base class is replaced by the `POA_BankSimple::Account` base class, as follows:

```
// C++
// Orbix E2A Version
// Inheritance Approach
class BankSimple_Account_i : POA_BankSimple::Account {
public:
    // Declare IDL operation and attribute functions...
};
```

The Tie Approach

The tie approach is an alternative mechanism for implementing IDL interfaces. It allows you to associate an implementation class with an IDL interface using a delegation approach rather than an inheritance approach.

In Orbix E2A (C++) the tie classes are generated using C++ templates. When migrating from Orbix 3 to Orbix E2A, all `DEF_TIE` and `TIE` preprocessor macros must be replaced by the equivalent template syntax.

In Orbix E2A (Java) the tie approach is essentially the same as in Orbix 3. However, the names of the relevant Java classes and interfaces are different. For example, given an IDL interface, `Foo`, an Orbix E2A servant class implements the `FooOperations` Java interface and the associated Java tie class is called `FooPOATie`.

Creating and Activating a CORBA Object

Overview

To make a CORBA object available to clients, you should:

1. Create an implementation object. An implementation object is an instance of the class that implements the operations and attributes of an IDL interface. In Orbix 3, an implementation object is the same thing as a CORBA object. In Orbix E2A, an implementation object is a servant object, which is not the same thing as a CORBA object.
2. Activate the servant object. Activating a servant object attaches an identity to the object (a marker in Orbix 3 or an object ID in Orbix E2A) and associates the object with a particular object adapter.

Orbix 3

In Orbix 3, creating and activating an object are rolled into a single step. For example, in C++ you might instantiate a `BankSimple::Account` CORBA object using the following code:

```
// C++
// Orbix 3
// Create and activate a new 'Account' object.
BankSimple_Account_i * acc1 =
                        new BankSimple_Account_i("ObjectID");
```

This step creates the CORBA object and attaches the *ObjectID* identity to it (initializing the object's marker). The constructor automatically activates the CORBA object.

Orbix E2A

In Orbix E2A, creating and activating an object are performed as separate steps. For example, in C++ you might instantiate a `BankSimple::Account` CORBA object using the following code:

```
// C++
// Orbix E2A

// Step 1:  Create a new 'Account' object.
BankSimple_Account_i * acc1 = new BankSimple_Account_i();

// Step 2:  Activate the new 'Account' object.
PortableServer::ObjectId_var oid =
                  PortableServer::string_to_ObjectId("ObjectID");
// persistent_poa created previously
persistent_poa->activate_object_with_id(oid, acc1);
```

Activation is performed as an explicit step in Orbix E2A. The call to `PortableServer::POA::activate_object_with_id()` attaches the *ObjectID* identity to the object and associates the `persistent_poa` object adapter with the object.

CHAPTER 5

Migrating Proprietary Orbix 3 Features

Proprietary Orbix 3 feature are replaced by a range of standards-compliant Orbix E2A features.

In This Chapter

The following proprietary features of Orbix 3 are discussed in this chapter:

Orbix 3 Locator

Overview

The Orbix 3 locator is an Orbix-specific feature that is used in combination with `_bind()` to locate server processes. Because Orbix E2A does not support `_bind()`, it cannot use the Orbix 3 style locator.

Note: Orbix E2A has a feature called a locator, which is not related in any way to the Orbix 3 locator. The Orbix E2A locator is a daemon process, `itlocator`, that locates server processes for clients.

If your legacy code uses the locator, you must replace it with one of the following Orbix E2A features:

- High availability.
- The CORBA Naming Service.
- The CORBA Initialization Service.

High Availability

The Orbix E2A high availability feature provides fault tolerance—that is, a mechanism that avoids having a single point of failure in a distributed application. With the enterprise edition of Orbix E2A, you can protect your system from single points of failure through clustered servers.

A clustered server is comprised of multiple instances, or replicas, of the same server; together, these act as a single logical server. Clients invoke requests on the clustered server and Orbix routes the requests to one of the replicas. The actual routing to any replica is transparent to the client.

The CORBA Naming Service

If your legacy code uses the load-balancing feature of the Orbix 3 locator, you can replace this by the `ObjectGroup` feature of the Orbix E2A naming service. Object groups are an Orbix-specific extension to the naming service that allow you to register a number of servers under a single name.

Table 4 shows how the Orbix 3 locator maps to the equivalent naming service functionality.

Table 4: Replacing the Orbix 3 Locator by the Naming Service

Orbix 3-Locator	Orbix E2A-Naming Service
Entry in the locator file, mapping the server name, *`SrvName`*, to a single server host, `HostName:` *`SrvName`*`:HostName:`	Object binding in the naming service, mapping a *`name`* to a single object reference.
Entry in the locator file, mapping the server name, *`SrvName`*, to multiple host names: *`SrvName`*`:Host1,Host2,Host3:`	Object group in the naming service, mapping a name to multiple object references.
Overriding functionality of `CORBA::LocatorClass`.	Custom implementation of the `IT_LoadBalancing::ObjectGroup` interface.

The naming service is the preferred way to locate objects in Orbix E2A. It is a standard service and is highly scalable.

The CORBA Initialization Service

The initialization service uses the `CORBA::ORB::resolve_initial_references()` operation to retrieve an object reference from an Orbix E2A configuration file, `DomainName.cfg`.

Table 5 shows how the Orbix 3 locator maps to the equivalent initialization service functionality.

Table 5: Replacing the Orbix 3 Locator by the Initialization Service

Orbix 3-Locator	Orbix E2A-Initialization Service

Entry in the locator file, mapping the server name, *`SrvName`*, to a single server host, `HostName`:
 	
`SrvName``:HostName:`
	Entry in the `DomainName.cfg` file, mapping an *`ObjectId`* to a single object reference:
	
	`initial_references:ObjectId:`
	`reference = "IOR:00...";`

Entry in the locator file, mapping the server name, *`SrvName`*, to multiple host names: *`SrvName`*`:Host1,Host2,Host3:`	*No Equivalent*
Override functionality of `CORBA::LocatorClass`.	*No Equivalent*

The initialization service can only be used as a replacement for the Orbix 3 locator when a simple object lookup is needed.

Filters

Overview

Filters are a proprietary Orbix 3 mechanism that allow you to intercept invocation requests on the server and the client side.

Orbix E2A does not support the filter mechanism. Instead, a variety of Orbix E2A features replace Orbix 3 filter functionality.

Equivalents

Table 6 summarizes the typical uses of Orbix 3 filters alongside the equivalent features supported by Orbix E2A.

Table 6: Orbix E2A Alternatives to Filter Features

Orbix 3 Filter Feature	Orbix E2A Equivalent
Request Logging	Use portable interceptors.
Piggybacking data on a Request	Use portable interceptors.
Multi-threaded request processing	Use a multi-threaded POA and (optionally) a proprietary `WorkQueue` POA policy.
Accessing the client's TCP/IP details	*Not supported*
Security using an authentication filter	Full security support is provided in the Orbix E2A enterprise edition.

In This Section

The following topics are discussed in this section:

Request Logging

Using Portable Interceptors

In Orbix E2A, request logging is supported by the new portable interceptor feature. Interceptors allow you to access a CORBA request at any stage of the marshaling process, offering greater flexibility than Orbix filters. You can use them to add and examine service contexts. You can also use them to examine the request arguments.

Piggybacking Data on a Request

Piggybacking in Orbix 3

In Orbix 3, filters support a piggybacking feature that enables you to add and remove extra arguments to a request message.

Piggybacking in Orbix E2A

In Orbix E2A, piggybacking is replaced by the CORBA-compliant approach using *service contexts*. A service context is an optional block of data that can be appended to a request message, as specified in the IIOP 1.1 standard. The content of a service context can be arbitrary and multiple service contexts can be added to a request.

Multi-Threaded Request Processing

Orbix 3

In Orbix 3, concurrent request processing is supported using an Orbix thread filter. The mechanism is flexible because it gives the developer control over the assignment of requests to threads.

Orbix E2A

In Orbix E2A, request processing conforms to the CORBA 2.4 specification. Each POA can have its own threading policy:

- `SINGLE_THREAD_MODEL` ensures that all servant objects in that POA have their functions called in a serial manner. In Orbix E2A, servant code is called only by the main thread, therefore no locking or concurrency-protection mechanisms need to be used.
- `ORB_CTRL_MODEL` leaves the ORB free to dispatch CORBA invocations to servants in any order and from any thread it chooses.

Orbix E2A Request Processing Extensions

Because the CORBA 2.4 specification does not specify exactly what happens when the `ORB_CTRL_MODEL` policy is chosen, Orbix E2A makes some proprietary extensions to the threading model.

The multi-threaded processing of requests is controlled using the Orbix E2A work queue feature. Two kinds of work queue are provided by Orbix E2A:

- *Automatic Work Queue*: A work queue that feeds a thread pool. When a POA uses an automatic work queue, request events are automatically dequeued and processed by threads. The size of the thread pool is configurable.
- *Manual Work Queue*: A work queue that requires the developer to explicitly dequeue and process events.

  Manual work queues give developers greater flexibility when it comes to multi-threaded request processing. For example, prioritized processing of requests could be implemented by assigning high-priority CORBA objects to one POA instance and low-priority CORBA objects to a second POA instance. Given that both POAs are associated with manual work queues, the developer can write threading code that preferentially processes requests from the high-priority POA.

Accessing the Client's TCP/IP Details

Recommendations for Orbix E2A

Some Orbix 3 applications use Orbix-specific extensions to access socket-level information, such as the caller's IP address, in order to implement proprietary security features. These features are not available in Orbix E2A, because providing access to low-level sockets would considerably restrict the flexibility of CORBA invocation dispatch.

To provide security for your applications, it is recommended that you use an implementation of the security service provided with the Orbix E2A Enterprise Edition instead.

Security Using an Authentication Filter

Recommendations for Orbix E2A

Some Orbix 3 applications use authentication filters to implement security features. In Orbix E2A, it is recommended that you use the security service that is made available with the Orbix E2A Enterprise Edition.

Loaders

Orbix 3 Loader

The Orbix 3 loader provides support for the automatic saving and restoration of persistent objects. The loader provides a mechanism that loads CORBA objects automatically into memory, triggered in response to incoming invocations.

Servant Manager

The Orbix 3 loader is replaced by equivalent features of the Portable Object Adapter (POA) in Orbix E2A. The POA can be combined with a servant manager to provide functionality equivalent to the Orbix 3 loader. There are two different kinds of servant manager:

- *Servant activator*: Triggered only when the target CORBA object cannot be found in memory.
- *Servant locator*: Triggered for every invocation.

Servant Activator

Taking the `PortableServer::ServantActivator` class as an example, the member functions of `CORBA::LoaderClass` correspond approximately as shown in Table 7.

Table 7: Comparison of Loader with Servant Activator Class

CORBA::LoaderClass Member Function	ServantActivator Member Function
`save()`	`etherealize()`
`load()`	`incarnate()`
`record()`	*No equivalent function.* An Orbix E2A object ID (equivalent to an Orbix 3 marker) can be specified at the time a CORBA object is created. This gives sufficient control over object IDs.

Table 7: Comparison of Loader with Servant Activator Class

CORBA::LoaderClass Member Function	ServantActivator Member Function
`rename()`	*No equivalent function.* An Orbix E2A object ID (equivalent to an Orbix 3 marker) cannot be changed after a CORBA object has been created.

Servant Locator

A servant locator can also be used to replace the Orbix 3 loader. In general, the servant locator is more flexible than the servant activator and offers greater scope for implementing sophisticated loader algorithms.

Smart Proxies

Orbix 3

The Orbix 3 smart proxies feature is a proprietary mechanism for overriding the default implementation of the proxy class. This allows applications to intercept outbound client invocations and handle them within the local client process address space, rather than using the default proxy behavior of making a remote invocation on the target object. Smart proxies can be used for such purposes as client-side caching, logging, load-balancing, or fault-tolerance.

Orbix E2A

Orbix E2A does not support smart proxies. The primary difficulty is that, in the general case, it is not possible for the client-side ORB to determine if two object references denote the same server object. The CORBA standard restricts the client-side ORB from interpreting the object key or making any assumptions about it. Orbix 3 was able to avoid this limitation by making assumptions about the structure of the object key. This is neither CORBA-compliant nor interoperable with other ORBs.

At best, the ORB can only determine that two object references are equivalent if they have exactly the same server location (host and port in IIOP) and object key. Unfortunately, this may be an unreliable indicator if object references pass through bridges, concentrators, or firewalls that change the server location or object key.

In this case, it is possible for two object references denoting the same CORBA object to appear different to the ORB, and thus have two different smart proxy instances. Since smart proxies are commonly used for caching, having two smart proxy instances for a single CORBA object is unacceptable.

Replacing Smart Proxies with Equivalent Orbix E2A Features

Table 8 shows how smart proxy tasks can be mapped to equivalent features in Orbix E2A.

Table 8: Orbix E2A Alternatives to Smart Proxy Features

Orbix 3 Smart Proxy Task	Orbix E2A Equivalent Feature
Fault Tolerance	Orbix E2A high availability, based on server clusters.
Logging	Orbix E2A built-in logging facility or portable interceptors
Caching	Implement smart proxy-like functionality by hand.

Fault Tolerance

Fault tolerance is provided by the high availability feature of the Orbix E2A locator. See "High Availability" on page 52.

Logging

For logging that requires access to request parameters, portable interceptors can be used in Orbix E2A. Portable interceptors are similar to Orbix 3 filters, but they are more flexible in that they allow you to read request parameters.

Caching

A smart proxy that implements client-side caching of data cannot be mimiced by a standard Orbix E2A feature. In this case, you have no option but to implement smart proxy-like functionality in Orbix E2A, and this can be done as follows:

1. Create a local implementation of the object to be proxified, by writing a class that derives from the client-side stub class.
2. Every time the client receives an object reference of the appropriate type, wrap the object reference with a corresponding smart proxy object. Before wrapping the object reference, however, you must determine the target object's identity by making an invocation on the remote target object, asking it for a system-wide unique identifying name. This is the key step that avoids the object identity problem described previously.

Based on the system-wide unique identifying name, the application can then either create a new smart proxy, or reuse the target object's existing smart proxy. The client application should consistently use the smart proxy in place of the regular proxy throughout the application.

Transformers

Orbix 3

Transformers are a deprecated feature of Orbix 3 that allow you to apply customized encryption to CORBA request messages. This could be used to implement a primitive substitute for a security service.

Orbix E2A

In Orbix E2A, transformers are not supported. It is recommended, instead, that you use the security service that is made available with the enterprise edition of Orbix E2A.

I/O Callbacks

Overview

Orbix E2A does not allow access to TCP/IP sockets or transport-level information. This is incompatible with the Orbix E2A architecture, which features a pluggable transport layer. Using Orbix E2A, you can replace TCP/IP with another transport plug-in such as IP multicast (which is connectionless), simple object access protocol (SOAP), hypertext transfer protocol (HTTP), asynchronous transfer mode (ATM), and so on. For example, the shared memory transport (SHMIOP) does not use file descriptors or sockets.

Purposes for Using I/O Callbacks

Orbix 3 I/O Callback functionality is generally used for two main purposes:

- *Connection Management*—the number of TCP/IP connections that can be made to a single process is typically subject to an operating system limit. Some form of connection management is required if this limit is likely to be reached in a deployed system.
- *Session Management*—I/O Callback functionality can be used to implement an elementary session-tracking mechanism. The opening of a connection from a client defines the beginning of a session and the closing of the connection defines the end of the session.

Because Orbix E2A has no equivalent to the Orbix 3 I/O Callback functionality, you must migrate any code that uses it.

In This Section

This section contains the following subsections:

Connection Management

Active Connection Management

Orbix E2A provides an active connection manager (ACM) that allows the ORB to reclaim connections automatically, and thereby increases the number of clients that can concurrently use a server beyond the limit of available file descriptors.

ACM Configuration Variables

IIOP connection management is controlled by four configuration variables:

- `plugins:iiop:incoming_connections:hard_limit` sets the maximum number of incoming (server-side) connections allowed to IIOP. IIOP refuses new connections above this limit.
- `plugins:iiop:incoming_connections:soft_limit` determines when IIOP starts to close incoming connections.
- `plugins:iiop:outgoing_connections:hard_limit` sets the maximum number of outgoing (client-side) connections allowed to IIOP. IIOP refuses new outgoing connections above this limit.
- `plugins:iiop:outgoing_connections:soft_limit` determines when IIOP starts to close outgoing connections.

Closing Client Connections

The ORB first tries to close idle connections in least-recently-used order. If there are no idle connections, the ORB closes busy connections in least-recently-opened order.

Active connection management effectively remedies file descriptor limits that has constrained past Orbix applications. If a client is idle for a while and the server ORB reaches its connection limit, it sends a GIOP `CloseConnection` message to the client and closes the connection. Later, the same client can transparently reestablish its connection, to send a request without throwing a CORBA exception.

Note: In Orbix 3, Orbix tended to throw a `COMM_FAILURE` on the first attempt at reconnection; server code that anticipates this exception should be reevaluated against current functionality.

Default File Descriptor Limits

Orbix E2A is configured to use the largest upper file descriptor limit on each supported operating system. On UNIX, it is typically possible to rebuild the kernel to obtain a larger number. However, active connection management should make this unnecessary.

Session Management

Overview

Because Orbix E2A features a pluggable transport layer, it is not appropriate to relate the duration of a client session to the opening and closing of TCP/IP connections from clients. This type of session management, which is typically implemented using I/O callbacks in Orbix 3, has to be migrated to an alternative model.

Session Management in Orbix E2A

Support for session management in Orbix E2A is provided by a *lease plug-in*. The lease plug-in implements a scheme for automatically tracking client sessions, based on the idea that a client obtains a lease from the server for the duration of a client session.

Client Migration

Client applications can easily be modified to use session management. Just edit the Orbix E2A configuration to make the client load the lease plug-in. No changes to the client source code are required.

Server Migration

On the server side, the following changes are required to use session management in Orbix E2A:

- Edit the Orbix E2A configuration to make the server load the lease plug-in.
- Modify the server source code so that it uses the lease plug-in to track client sessions.

Further Details

See the *CORBA Session Management Guide* for details of how to program and configure the lease plug-in for session management at:

http://documentation.progress.com/output/Iona/orbix/6.3.5/CORBA%20Session%20Management,%20C++.pdf

and

http://documentation.progress.com/output/Iona/orbix/6.3.5/CORBA%20Session%20Management.pdf

Demonstration code for the lease plug-in is also provided with the Orbix E2A product.

CHAPTER 6

CORBA Services

Orbix includes several CORBA services, such as the interface repository, the naming service, the notification service, and the security service. Because these service are based mainly on the CORBA standard, there are not many changes between Orbix 3 and Orbix E2A.

In This Chapter

The following topics are discussed in this chapter:

Interface Repository

Migration

Migrating source code that uses the Interface Repository (IFR) to Orbix E2A is straightforward. Link the migrated application against the stub code derived from the Orbix E2A version of the interface repository. No further changes should be necessary.

Naming Service

Backward Compatibility

The Orbix E2A naming service is backward compatible with Orbix 3.x in two respects:

- *Source code backward compatibility*: source code that is written to use the standard naming service interfaces can be migrated to Orbix E2A without modification.
- *On-the-wire backward compatibility*: Orbix 3.x applications can interoperate with the Orbix E2A naming service. If you need to interoperate Orbix 3.x applications, it is recommended that you recompile the naming stub code from the Orbix E2A IDL files.

New Interface

Orbix E2A adds a new interface, `CosNaming::NamingContextExt`, which is defined by the CORBA Interoperable Naming Service specification. This interface adds support for using names in stringified format.

Load Balancing

The naming service load-balancing extensions provided in Orbix 3 are also present in Orbix E2A. The Orbix E2A load-balancing interfaces are only slightly different from Orbix 3, requiring small modifications to your source code.

Notification Service

Overview

The Orbix E2A notification service has undergone significant modifications since the OrbixNotification 3 generation of the notification service.

Many of the changes that impact application migration reflect changes in the CORBA standard and require minimal changes to legacy OrbixNotification 3 application code.

In This Section

The following topics are discussed in this section:

CORBA Specification Updates

Overview

The Orbix E2A notification service complies to both the CORBA 2.4 specification and the OMG's Notification Service Specification, approved in June of 2000. To achieve compliancy with these specifications several changes were made to the notification services IDL and API's.

These changes will require that any applications that use generation 3 code will need to be recompiled and re-linked, at the very least. Other minor changes may also need to be made to generation 3 code to accommodate the changes in the API's. Compiler warnings will warn you of most changes that need to be made.

_bind()

The Orbix E2A notification service clients do not use `_bind()` to contact the notification service. Instead, clients should call `resolve_initial_references("NotificationService")` to obtain an object reference to the notification service. See "Replacing the _bind() Function" on page 32 for more information.

Subscription and Publication Notification

Orbix E2A provides notification service clients greater flexibility over how they receive subscription and publication details from the notification channel. To accomplish this, an input parameter has been added to `obtain_offered_types()` and `obtain_subscription_types()`.

The Orbix E2A operation signatures are:

```
// IDL
CosNotification::EventTypeSeq obtain_subscription_types(
                                          in ObtainInfoMode mode);
CosNotification::EventTypeSeq obtain_offered_types(
                                          in ObtainInfoMode mode);
```

The new parameter is of type `ObtainInfoMode` which is an `enum` defined in `CosNotifyChannelAdmin` as:

```
// IDL
enum ObtainInfoMode
  {
    ALL_NOW_UPDATES_OFF,
    ALL_NOW_UPDATES_ON,
    NONE_NOW_UPDATES_OFF,
    NONE_NOW_UPDATES_ON
  };
```

Any generation 3 clients that call `obtain_offered_types()` or `obtain_subscription_types()` will need to add the parameter. `ALL_NOW_UPDATES_OFF` mimics generation 3 functionality. For more information on the other values, see the *CORBA Notification Service Guide*.

Unstructured Event Clients

Orbix E2A introduced unstructured event, any-style, client interfaces into the `CosNotifyComm` module. This allows any-style clients to support the enhanced subscription features and it standardizes notification service client development. Any-style clients developed for OrbixNotification 3 used the interfaces from `CosEventComm`.

In addition, the Orbix E2A any-style proxy interfaces, defined in `CosNotifyChannelAdmin`, inherit their client interfaces directly from `CosNotifyComm`. In OrbixNotification 3 any-style proxies inherited client interfaces from `CosNotifyComm:NotifyPublish` and `CosEventComm::PushConsumer`.

Note: The `connect()` operation's parameter is still an interface defined in `CosEventComm`.

Not updating legacy code will not generate any compiler errors. However, at runtime any-style clients using legacy code will not be able to contact the notification service.

TimeBase::TimeT

Orbix E2A supports the new OMG standard definition of `TimeBase::TimeT`. In OrbixNotification 3 `TimeBase::TimeT` was defined as a structure containing two `unsigned longs`. In Orbix E2A it is defined as a `CORBA::ULongLong`.

Any generation 3 clients that use the timing features of the service will need to be updated to support the new definition of `TimeBase::TimeT`. If they are not, the Orbix E2A notification service will generate mashalling errors at runtime.

Quality of Service Properties

Overview

Orbix E2A notification uses new several new Quality-of-Service (QoS) properties and has reimplemented others.

PacingInterval

`PacingInterval` is reimplimented as a `TimeBase::TimeT` in Orbix E2A and is specified in units of 10^{-7} seconds. In Orbix 3 it was a `TimeBase:UtcT` and was specified in milliseconds.

Orbix E2A QoS Properties

Table 9 lists the new Orbix E2A QoS properties. For more detailed information on Orbix E2A QoS properties, see the *CORBA Notification Service Guide*.

Table 9: *Orbix E2A QoS Properties (Sheet 1 of 2)*

QoS Property	Description
`MaxEventsPerConsumer`	Specifies the maximum number of undelivered events that a channel will queue for a consumer. It is set with a `long` and is valid for supplier proxies, consumer admins, and notification channels.
`MaxRetries`	Specifies the maximum number of times a proxy push supplier calls `push()` on its consumer before giving up or the maximum number of times a proxy pull consumer calls `pull()` or `try_pull()` on its supplier before giving up. It is set with a `CORBA::Ulong` and is valid for consumer admins and notification channels.
`RetryTimeout`	Specifies the amount of time that elapses between attempts by a proxy push supplier to call `push()` on its consumer. It is set with a `TimeBase::TimeT` and defaults to 1 second.
`MaxRetryTimeout`	Sets the ceiling for the calculated value of `RetryTimeout`. It is set with a `TimeBase::TimeT` and defaults to 60 seconds.

Table 9: *Orbix E2A QoS Properties (Sheet 2 of 2)*

QoS Property	Description
`RequestTimeout`	Specifies the amount of time a channel object has to perform an operation on a client. It is set using a `TimeBase::TimeT`.
`PullInterval`	Specifies the amount of time that elapses between attempts by a proxy pull consumer to call `pull()` or `try_pull()` on its consumer. It is specifies with a `long` and defaults to 1 second.
`RetryMultiplier`	Specifies the number used to calculate the amount of time between attempts by a proxy push supplier to call `push()` on its consumer. It is set with a `CORBA::double` and defaults to 1.0.

Channel Administration Properties

Orbix E2A has introduced two properties to control the administration of a notification channel. These properties can only be set on a notification channel. For more information, see the *CORBA Notification Service Guide*.

Table 10 describes the new properties.

Table 10: *Orbix E2A Administration Properties*

Property	Description
`MaxConsumers`	Specifies the maximum number of consumers that can be connected to a channel at a given time. It is set using a `long` and defaults to 0 (unlimited).
`MaxSuppliers`	Specifies the maximum number of suppliers that can be connected to a channel at a given time. It is set using a `long` and defaults to 0 (unlimited).

Configuration / Administration Changes

Centralized Configuration

The Orbix E2A CORBA platform has a centralized configuration mechanism. This means that the notification service is configured using the standard Orbix E2A CORBA platform configuration tools and the information is stored in the common Orbix E2A database.

Starting the Notification Service

The Orbix E2A notification service can be configured to start on system boot, on demand, or from the command line.

To start the notification service from the command line use:

```
itnotify run [-backround]
```

The `-background` flag is optional and starts the notification service to run as a background process.

Managing the Notification Service

The Orbix E2A notification service can be managed in one of two ways.

- The Orbix E2A `itadmin` tool. For more information, see the *CORBA Administrator's Guide*.
- The Orbix E2A notification console, `itnotifyconsole`. For more information on using the console, see the *CORBA Notification Service Guide*.

Configuration Variables

The Orbix E2A notification service uses a new set of configuration variables. See the *CORBA Administrator's Guide* for a detailed listing of the new configuration variables.

Deprecated Features

Overview

Orbix E2A has deprecated some proprietary features from OrbixNotification 3. Any notification clients that make use of these features will need to be updated.

HealthCheck

The OrbixNotification 3 HealthCheck feature allowed notification channels, and optionally notification clients, to monitor their connections. In Orbix E2A this feature is no longer supported.

Code Modification

To find code using the HealthCheck feature search for the following strings:

- `DO_HEALTHCHECK`
- `DO_GL_HEALTHCHECK`
- `initializeHealthCheck`
- `startHealthCheck`
- `stopHealthCheck`
- `HealthCheck.h`

This code must be removed before the clients can be compiled using the Orbix E2A libraries.

Simulating HealthCheck in Orbix E2A

HealthCheck-like functionality is implemented in Orbix E2A using the `MaxRetries` QoS property. If a `ProxyPushSupplier` or a `ProxyPullConsumer` fails to communicate with its associated client in `MaxRetries` attempts, the notification channel forces a disconnect and destroys all of the resources used to support the client.

String Events

Orbix E2A no longer supports string events. All generation 3 clients using string events must be rewritten to use a valid event type.

SSL/TLS Toolkit

Overview

This section describes how to migrate from OrbixSSL or Orbix 3.3 security to the Orbix E2A SSL/TLS security service. Orbix E2A SSL/TLS has a very similar set of features to Orbix 3.3 security and it supports interoperability with legacy Orbix applications (see "SSL/TLS Toolkit Interoperability" on page 147).

The programming interfaces and administration of security have, however, changed significantly between Orbix 3.3 and Orbix E2A. This section provides an overview of these changes.

In This Section

This section contains the following subsections:

Changes to the Programming Interfaces

Support for Security Level 2

The APIs for Orbix E2A SSL/TLS are based on the CORBA Security Level 2 interfaces. The programming interface is, therefore, based on the following standard IDL modules:

- `Security`
- `SecurityLevel1`
- `SecurityLevel2`

Note: Orbix E2A SSL/TLS does not implement every interface in the `SecurityLevel1` and `SecurityLevel2` modules. The CORBA security API is a mechanism-neutral API that can be layered over a variety of security toolkits. Some of the standard interfaces are more appropriately implemented by a higher level security layer.

CORBA Policy-Based API

In contrast to OrbixSSL 3.x, the Orbix E2A SSL/TLS product supports a *CORBA policy-based* approach to setting security properties. This represents a significant enhancement over OrbixSSL 3.x because the policy-based approach lets you set properties at a finer granularity than before.

For example, client policies can be set at the following levels:

- ORB
- Thread
- Object reference

Server policies can be set at the following levels:

- ORB
- POA

No Support for Certificate Revocation Lists

Orbix E2A SSL/TLS version 2.0 has no support for certificate revocation lists (CRL). Therefore, the following OrbixSSL 3.x interfaces have no Orbix E2A equivalent:

```
IT_CRL_List
IT_X509_CRL_Info
IT_X509_Revoked
IT_X509_RevokedList
```

If you require certificate revocation in Orbix E2A, you can programmatically implement any required revocation checks by registering a certificate validator policy, `IT_TLS_API::CertValidatorPolicy`.

Mechanism-Specific API

Orbix E2A SSL/TLS provides a number of value-added APIs that deal with the mechanism-specific aspects of the SSL/TLS toolkit. The extra IDL interfaces provide the facility to parse X.509 certificates and set Orbix-specific security policies.

The mechanism-specific API is defined by the following IDL modules:

- `IT_Certificate`
- `IT_TLS`
- `IT_TLS_API`

Migrating OrbixSSL 3.x Classes and Data Types

When migrating to Orbix E2A, most of the old C++ and Java classes from OrbixSSL 3.x are replaced by equivalent IDL interfaces. Table 11 shows how to replace the OrbixSSL classes and data types by equivalent Orbix E2A SSL/TLS types.

Table 11: *Mapping OrbixSSL 3.x Types to Orbix E2A SSL/TLS*

OrbixSSL 3.x Type	Orbix E2A SSL/TLS Equivalent
`IT_AVA`	`IT_Certificate::AVA`
`IT_AVAList`	`IT_Certificate::AVAList`
`IT_CertError`	`IT_Certificate::CertError`
`IT_CRL_List`	*No equivalent*
`IT_Extension`	`IT_Certificate::Extension`
`IT_ExtensionList`	`IT_Certificate::ExtensionList`

Table 11: *Mapping OrbixSSL 3.x Types to Orbix E2A SSL/TLS*

OrbixSSL 3.x Type	Orbix E2A SSL/TLS Equivalent
`IT_OID`	`IT_Certificate::ASN_OID`
`IT_OIDTag`	`IT_Certificate::OIDTag`
`IT_SSL`	Equivalent functionality provided by the `Security`, `SecurityLevel1`, `SecurityLevel2`, and `IT_TLS_API` IDL modules.
`IT_UTCTime`	`IT_Certificate::UTCTime`
`IT_ValidateX509CertCB`	Use a combination of the `IT_TLS::CertValidator` interface and the `IT_TLS_API::CertValidatorPolicy` interface.
`IT_X509_CRL_Info`	*No equivalent*
`IT_X509_Revoked`	*No equivalent*
`IT_X509_RevokedList`	*No equivalent*
`IT_X509Cert`	`IT_Certificate::X509Cert`
`IT_X509CertChain`	`IT_Certificate::X509CertChain`

Configuration and Administration

Enabling Security in Orbix E2A

Security in Orbix E2A is enabled by configuring an application to load the security plug-in, `iiop_tls`.This is a relatively simple procedure involving just a few changes in the Orbix E2A CORBA platform configuration file; although advanced applications might also need to use security APIs.

Because application security is controlled by editing the configuration file, you must ensure that access to the configuration file is restricted.

External Configuration Granularity

The external configuration granularity refers to the effective scope of security configuration settings that are made in a configuration file. The external configuration granularity is mapped as follows:

- In OrbixSSL 3.x, it is identified with a process.
- In Orbix E2A SSL/TLS, it is identified with a single ORB instance.

KDM Support

The key distribution management (KDM) is a framework that enables automatic activation of secure servers. Both OrbixSSL 3.x and Orbix E2A SSL/TLS provide a KDM and the functionality is similar in each.

There is one significant difference between the OrbixSSL 3.x KDM and the Orbix E2A KDM. Protection against server imposters is implemented differently in the two products:

- In OrbixSSL 3.x, a binary checksum is calculated from the contents of the server executable file. The server is launched only if the calculated checksum matches the cached value.
- In Orbix E2A SSL/TLS, the node daemon relies on the server executables being stored in a secured directory to prevent tampering. A different sort of checksum is calculated (based on the contents of the server activation record) to ensure that the node daemon cannot be fooled into launching a server from an insecure directory.

No CRL Support

Orbix E2A SSL/TLS does not support certificate revocation lists. Hence, there are no equivalents for the corresponding OrbixSSL 3.x configuration variables. See also “No Support for Certificate Revocation Lists” on page 84.

Migrating OrbixSSL 3.x Configuration

Most of the OrbixSSL 3.x configuration variables have direct equivalents in Orbix E2A, as shown in Table 12. In addition, many of the properties listed in Table 12 can also be set programmatically in Orbix E2A.

Table 12: *Mapping OrbixSSL 3.x Configuration Variables to Orbix E2A*

OrbixSSL 3.x Configuration Variable	Orbix E2A SSL/TLS Equivalent
`IT_CA_LIST_FILE`	`policies:trusted_ca_list`
`IT_AUTHENTICATE_CLIENTS`	`policies:target_secure_invocation_policy`
`IT_SERVERS_MUST_AUTHENTICATE_CLIENTS.`	`policies:target_secure_invocation_policy`
`IT_INVOCATION_POLICY`	`policies:target_secure_invocation_policy` `policies:client_secure_invocation_policy`

`IT_SECURE_REMOTE_INTERFACES`
`IT_SECURE_SERVERS`
`IT_INSECURE_REMOTE_INTERFACES`
`IT_INSECURE_SERVERS`
	These properties cannot currently be specified in the Orbix E2A configuration file.
	You can, however, set the properties programmatically using the following interfaces:
	`SecurityLevel2::EstablishTrustPolicy`
	`SecurityLevel2::QOPPolicy`

`IT_CIPHERSUITES`	`policies:mechanism_policy`
`IT_ALLOWED_CIPHERSUITES`	*No equivalent in Orbix E2A*
`IT_CERTIFICATE_FILE` `IT_CERTIFICATE_PATH`	Equivalent functionality provided by: `principal_sponsor:auth_method_data`
`IT_BIDIRECTIONAL_IIOP_BY_DEFAULT`	Not applicable, because Orbix E2A does not support bi-directional IIOP.

`IT_CACHE_OPTIONS`	`policies:session_caching_policy`
	`plugins:atli_tls_tcp:session_cache_validity_period`
	`plugins:atli_tls_tcp:session_cache_size`

`IT_DEFAULT_MAX_CHAIN_DEPTH`	`policies:max_chain_length`
`IT_MAX_ALLOWED_CHAIN_DEPTH.`	*No equivalent in Orbix E2A.*

`IT_DAEMON_POLICY`
`IT_DAEMON_UNRESTRICTED_METHODS`
`IT_DAEMON_AUTHENTICATES_CLIENTS`
`IT_ORBIX_BIN_SERVER_POLICY`
	In Orbix E2A, the Progress services are configured using standard Orbix E2A configuration variables such as the secure invocation policies.

Table 12: *Mapping OrbixSSL 3.x Configuration Variables to Orbix E2A*

OrbixSSL 3.x Configuration Variable	Orbix E2A SSL/TLS Equivalent
`IT_DAEMON_UNRESTRICTED_METHODS`	*No equivalent in Orbix E2A.* There is currently no concept of service authorization in Orbix E2A.
`IT_FILTER_BAD_CONNECTS_BY_DEFAULT`	Not needed in Orbix E2A.
`IT_ENABLE_DEFAULT_CERT`	Not needed in Orbix E2A. There is no need for this option because Orbix E2A supports security unaware applications,.
`IT_DISABLE_SSL`	Not needed in Orbix E2A. Configure your application not to load the security plug-in.

`IT_KDM_CLIENT_COMMON_NAMES`
`IT_KDM_ENABLED`
`IT_KDM_PIPES_ENABLED`
`IT_KDM_REPOSITORY`
`IT_KDM_SERVER_PORT`
	Equivalent functionality is provided by the KDM in Orbix E2A.
	See the *CORBA SSL/TLS Guide*.

`IT_CHECKSUMS_ENABLED` `IT_CHECKSUM_REPOSITORY`	*No equivalent in Orbix E2A.* There is no binary checksum functionality in Orbix E2A. Orbix E2A SSL/TLS relies on storing server executables in secured directories.

`IT_CRL_ENABLED`
`IT_CRL_REPOSITORY`
`IT_CRL_UPDATE_INTERVAL`
	No equivalent in Orbix E2A.
	There is no CRL functionality in Orbix E2A.

Migrating Certificate and Private Key Files

Overview

In OrbixSSL 3.x, a variety of certificate and private key formats are used in different parts of the product. Orbix E2A SSL/TLS is based on a unified certificate file format, the industry standard PKCS#12 format, and the PEM format for storing trusted CA certificates. This subsection describes how to convert each of the legacy formats to PKCS#12.

Certificate File Formats

The following certificate file formats are used by OrbixSSL 3.x and Orbix E2A SSL/TLS:

- *Privacy enhanced mail (PEM) format*—A PEM file typically contains a single certificate. OrbixSSL 3.x can use this format to hold peer certificates. Orbix E2A SSL/TLS *cannot* use this format for peer certificates.
- *PKCS#12 format*—A PKCS#12 file contains a peer certificate chain, concatenated with a private key at the end. Both OrbixSSL 3.x and Orbix E2A SSL/TLS can use this format for peer certificates.

Migrating Certificate Files

You can migrate OrbixSSL 3.x certificate files to Orbix E2A SSL/TLS as shown in Table 13.

Table 13: *Converting Certificate Files*

Source OrbixSSL 3.x File Format	Target Orbix E2A File SSL/TLS Format	How to Convert
PEM format	PKCS#12 format	Use the `openssl pkcs12` utility, specifying the complete peer cert chain, private key and pass phrase.
PKCS#12 format	PKCS#12 format	*No conversion needed.*

Private Key File Formats

The following private key file formats are used either by OrbixSSL 3.x and Orbix E2A SSL/TLS:

- *PKCS#1 format*—An unencrypted private key format. Orbix E2A SSL/TLS only supports this format programmatically.
- *PKCS#8 format*—An encrypted private key format. Orbix E2A SSL/TLS only supports this format programmatically.
- *OpenSSL proprietary private key format*—A proprietary encrypted format generated by the OpenSSL toolkit utilities.
- *IONA proprietary KEYENC format (deprecated)*—An encrypted private key format generated by the OrbixSSL 3.x `keyenc` utility. This format was formerly used by OrbixSSL 3.x Java applications and is now deprecated.

Migrating Key Files

You can migrate OrbixSSL 3.x private key files to Orbix E2A SSL/TLS as shown in Table 14.

Table 14: *Converting Private Key Files*

Source OrbixSSL 3.x File Format	Target Orbix E2A SSL/TLS File Format	How to Convert
PKCS#1 format	PKCS#12 format	Use the `openssl pkcs12` utility, specifying the complete peer cert chain, private key, and pass phrase.
OpenSSL proprietary encrypted private key format	PKCS#12 format	Convert as follows: 1. Decrypt using the `openssl rsa` command. 2. Encrypt as PKCS#12 using the `openssl pkcs12` utility, specifying the complete peer cert chain, private key, and pass phrase.
IONA proprietary `keyenc` format	PKCS#12 format	Convert as follows: 1. Decrypt using the `keyenc -d` command: 2. Encrypt as PKCS#12 using the `openssl pkcs12` utility, specifying the complete peer cert chain, private key, and pass phrase.

Trusted CA Certificate Lists

In both OrbixSSL 3.x and Orbix E2A SSL/TLS, a trusted CA certificate list file consists of a concatenated list of PEM certificates.

Note: The Orbix E2A SSL/TLS Java Edition product currently does not accept any extraneous text (comments and so on) in a trusted CA list file. The extra text must therefore be removed if you are using Orbix E2A SSL/TLS Java Edition.

Interoperability

In a mixed system containing Orbix 3.3 Java Edition and Orbix E2A SSL/TLS, the PKCS#12 format can be used for peer certificates because Orbix 3.3 Java Edition also accepts the PKCS#12 format.

CHAPTER 7

Administration

The administration of Orbix E2A has changed significantly from Orbix 3. This chapter provides a brief overview of the main changes in Orbix administration.

In This Chapter

The following topics are discussed in this chapter:

Orbix Daemons

Orbix E2A Daemons

To provide greater flexibility and scaling, Orbix E2A replaced the Orbix 3 daemon, `orbixd`, with two daemons:

- The locator daemon, `itlocator`, helps clients to find Orbix E2A servers.
- The node daemon, `itnode_daemon`, launches dormant Orbix E2A servers in response to a client's request for service.

POA Names

Administering POA Names

In Orbix 3, CORBA objects were associated with a named server. In Orbix E2A, CORBA objects are associated with named POAs. This means that Orbix E2A object references included an embedded POA name instead of a server name.

The Orbix E2A locator daemon locates the CORBA object using the object reference's embedded POA name. Hence, POA names play a major role in configuring the Orbix E2A locator daemon.

Command-Line Administration Tools

Overview

Orbix E2A unifies many of Orbix 3's command-line tools under a single utility, `itadmin`. Also, some of the Orbix 3 command line-tools have been deprecated.

General Command-Line Tools

Table 15 compares the Orbix 3 general purpose command-line tools with the Orbix E2A tools.

Table 15: *Comparison of Orbix 3 and Orbix E2A General Command-Line Tools*

Description	Orbix 3	Orbix E2A
Show implementation repository (IMR) entry.	`catit`	`itadmin process show`
Security commands.	`chownit, chmodit`	*No equivalent*
Show configuration	`dumpconfig`	`itadmin config dump`
Associate hosts into groups	`grouphosts`	*No equivalent*
C++ IDL compiler	`idl`	`idl`
CodeGen toolkit	`idlgen`	`idlgen`
Java IDL compiler	`idlj`	`idl`
Interface Repository (IFR)	`ifr`	`itifr`
Kill a server process	`killit`	`itadmin process stop`
List server	`lsit`	`itadmin process list`
Create a sub-directory in the IMR	`mkdirit`	*No equivalent*
Orbix daemon	`orbixd`	`itlocator` and `itnode_daemon`
Ping the Orbix daemon	`pingit`	*No equivalent*
List active servers	`psit`	`itadmin process list -active`
Add a definition to the IFR	`putidl`	`idl -R`

Table 15: *Comparison of Orbix 3 and Orbix E2A General Command-Line Tools*

Description	Orbix 3	Orbix E2A
Register a server in the IMR	`putit`	`itadmin process create`
Show an IFR definition	`readifr`	`itadmin ifr show`
Remove a sub-directory from the IMR	`rmdirit`	*No equivalent*
Unregister a server from the IMR	`rmit`	`itadmin process remove`
Remove a definition from the IFR	`rmidl`	`itadmin ifr remove`
Associate servers with groups	`servergroups`	*No equivalent*
Associate hosts with servers	`serverhosts`	*No equivalent*

Naming Service Command Line Tools

Table 16 compares the Orbix 3 naming service command-line tools with the Orbix E2A tools.

Table 16: *Comparison of Orbix 3 and Orbix E2A Naming Service Command-Line Tools*

Description	Orbix 3	Orbix E2A
Add a member to an object group	`add_member`	`itadmin nsog add_member`
Print the IOR of an object group	`cat_group`	*No equivalent*
Print the IOR of an object group's member	`cat_member`	`itadmin nsog show_member`
Print the IOR of a given name	`catns`	`itadmin ns resolve`
Remove an object group	`del_group`	`itadmin nsog remove`
Remove a member from an object group	`del_member`	`itadmin nsog remove_member`
List all object groups	`list_groups`	`itadmin nsog list`
List the members of an object group	`list_members`	`itadmin nsog list_member`

Table 16: *Comparison of Orbix 3 and Orbix E2A Naming Service Command-Line Tools*

Description	Orbix 3	Orbix E2A
List the bindings in a context	`lsns`	`itadmin ns list`
Create an object group	`new_group`	`itadmin nsog create`
Create an unbound context	`newncns`	`itadmin ns newnc`
Select a member of an object group	`pick_member`	*No equivalent*
Bind a name to a context	`putncns`	`itadmin ns bind -context`
Create a bound context	`putnewncns`	`itadmin ns newnc`
Bind a name to an object	`putns`	`itadmin ns bind -object`
Rebind a name to a context	`reputncns`	`itadmin ns bind -context`
Rebind a name to an object	`reputns`	`itadmin ns bind -object`
Remove a binding	`rmns`	`itadmin ns remove`

Activation Modes

Orbix 3

Orbix 3 process activation modes, *shared*, *unshared*, *per-method*, *per-client-pid* and *persistent*, are used for a variety of reasons. For example, they are used to achieve multi-threaded behavior in a single-threaded environment, to increase server reliability, and so on. The two most popular modes are:

- *Shared mode*—which enables all clients to communicate with the same server process.
- *Per-client-pid mode*—which enforces a 1-1 relationship between client process and server process, is sometimes used to maximize server availability.

Orbix E2A

Orbix E2A provides the following activation mode:

- `on_demand`—the process only activates when required.

Orbix E2A moved CORBA object association from the server to the POA. Because of this, all Orbix E2A processes are shared.

Migration

Migration of source code should be straightforward, because the choice of activation mode has almost no impact on BOA or POA-based server code.

Load Balancing

The additional activation modes provided by Orbix 3 are typically used to achieve some form of load-balancing that is transparent to the client. The Enterprise Edition of Orbix E2A includes transparent locator-based load balancing over a group of replica POAs. This answers the needs currently addressed by Orbix 3 activation modes.

Part III

Interoperability

In This Part

This part contains the following chapters:

CHAPTER 8

Configuring for Interoperability

This chapter describes the main configuration changes that must be made to facilitate interoperability between Orbix 3.x and Orbix E2A applications.

In This Chapter

This chapter discusses the following topics:

Interoperability Overview

Synopsis

Orbix E2A v5.0, C++ and Java Editions, has been tested for interoperability with the following Progress Software products: Orbix 3.0.1-82, OrbixWeb 3.2-15, and Orbix 3.3, C++ and Java Editions.

This Interoperability Guide describes how to configure applications that use a mixture of Progress products and any feature limitations that apply to such interoperating systems.

Patched Releases of Orbix 3.3

The following patched releases have been tested for interoperability with Orbix E2A v5.0:

- Orbix 3.0.1-82
- OrbixWeb 3.2-15
- Orbix 3.3.2 C++ Edition
- Orbix 3.3.2 Java Edition

It is recommend that you install these or later versions, if your applications need to interoperate with Orbix E2A.

The `_bind()` Function

Orbix E2A does not support the `_bind()` function for establishing connections between clients and servers. Neither Orbix 3.0.1-82, OrbixWeb 3.2-15, nor Orbix 3.3 clients can use the `_bind()` function to establish a connection to an Orbix E2A server. You must use a CORBA Naming Service instead. For example, you could use either the Orbix 3.3 naming service or the Orbix E2A naming service.

IDL Feature Support

Orbix E2A supports a larger set of IDL data types and features than Orbix 3.3. When developing IDL interfaces for use with Orbix E2A and other Progress products you need to restrict your IDL to a subset that is supported by all of the interoperating products.

In particular, the following articles describe IDL features that are subject to limitations or require special configuration:

- “Use of #pragma prefix” on page 112
- “Use of #pragma ID in IDL” on page 115
- “Fixed Data Type and Interoperability” on page 116
- “Use of wchar and wstring” on page 118
- “C++ Keywords as Operation Names” on page 119

Changed Exception Semantics

The semantics of some CORBA system exceptions are different in Orbix E2A, as compared with Orbix 3.0.1-82, OrbixWeb 3.2-15, or Orbix 3.3. If you have existing code written for Orbix 3.0.1-82, OrbixWeb 3.2-15, or Orbix 3.3, you should read the following articles:

- “Orbix 3.3 C++ Edition—System Exceptions” on page 122
- “Orbix 3.3 Java Edition—System Exceptions” on page 129

These articles describe how to configure your legacy application so that it is insulated from any differences in exception semantics.

Other Affected Features

It is not possible to use bi-directional IIOP when interoperating with Orbix E2A applications because this feature is not supported by Orbix E2A. See “Callbacks and Bi-Directional IIOP” on page 151.

If you want to use the new Orbix E2A interoperable naming service as the common naming service for your interoperating system, see “The Orbix E2A Interoperable Naming Service” on page 140.

The remaining articles in this guide describe miscellaneous issues that might affect interoperability in a mixed product environment.

Launch and Invoke Rights

Synopsis

When an Orbix E2A client attempts to open a connection to an Orbix 3.0.1-82, OrbixWeb 3.2-15, or Orbix 3.3 server you must make sure that the system is configured such that the Orbix E2A client has launch and invoke rights.

Role of Launch and Invoke Rights

In Orbix 3.3 the `orbixd` daemon process is responsible both for launching servers and for redirecting client requests to servers. These two functions are governed by *launch rights* and *invoke rights*, respectively.

Launch and invoke rights on Orbix 3.3 servers are based on the idea that the client *userID* is transmitted along with request messages. The field of the request message that contains the user ID is known as the Principal of the invocation.

If launch and invoke rights are not configured correctly, the Orbix E2A client raises a `CORBA::OBJECT_NOT_EXIST` system exception.

Setting Launch Rights

The launch rights associated with an Orbix 3.3 server specify which users are allowed to cause automatic launching of the server. Launch rights in Orbix 3.3 are granted with the following form of `chmodit`:

```
chmodit l+userID ServerName
```

Setting Invoke Rights

The invoke rights associated with an Orbix 3.3 server are used to determine which users are allowed to invoke on the server. Invoke rights are granted using:

```
chmodit i+userID ServerName
```

Orbix E2A and Orbix 3.3

The configuration must be altered for an Orbix E2A client invoking on an Orbix 3.3 server. There are two possible approaches to fix the launch and invoke rights:

- Alter the configuration of the Orbix E2A client.
- Relax the security on the `orbixd` daemon.

Alter the Configuration of the Orbix E2A Client

Two entries must be made (or changed) in the Orbix E2A configuration file:

```
# Orbix E2A Configuration File
policies:giop:interop_policy:send_locate_request = "false";
policies:giop:interop_policy:send_principal      = "true";
```

The `policies:giop:interop_policy:send_locate_request` option controls whether Orbix E2A sends `LocateRequest` messages before sending initial `Request` messages. This option must be set to false because `LocateRequest` messages do not contain a Principal field.

The `policies:giop:interop_policy:send_principal` option controls whether Orbix E2A sends Principal information containing the current user name in GIOP 1.0 and GIOP 1.1 requests. The user name is matched against the launch and invoke rights listed in the `orbixd` daemon to determine the permissions of the Orbix E2A client.

Relax the Security on the `orbixd` Daemon

Alternatively, you can relax the security on the orbixd daemon so that all clients have launch and invoke rights. For example, use the `chmodit` command line utility to change the launch and invoke rights:

`chmodit l+all` *ServerName*
`chmodit i+all` *ServerName*

These commands give permission for any client to invoke or launch the server *ServerName*. Permissions are granted even if the Principal value is left blank in the incoming requests.

GIOP Versions

GIOP Version of a Connection

The GIOP version used by a client-server connection is determined by the client. When a client is about to open a connection to a CORBA object, the client examines the version information in the object's IOR.

- If the GIOP version in the IOR is greater than or equal to the default GIOP version of the client, the client initiates a connection using the client's default GIOP version.
- Otherwise, the client initiates a connection using the GIOP version in the IOR.

Effect of GIOP Version

The GIOP version of a connection is important because some CORBA features are not supported in early GIOP versions. Table 17 shows the minimum GIOP version required for some CORBA features, according to the CORBA specification.

Table 17: *CORBA-Specified Minimum GIOP Versions*

CORBA Feature	CORBA-Specified Minimum GIOP Version
`fixed` type	1.1
`wchar` and `wstring` types	1.1
codeset negotiation (Orbix E2A only)	1.1

Orbix-Specific Minimum GIOP Versions

Notwithstanding the CORBA-specified minimum GIOP versions, Orbix allows some features to be used at a lower GIOP version (in some cases requiring specific configuration variables to be set). Table 18 shows the Orbix-specific minimum GIOP versions.

Table 18: *Orbix-Specific Minimum GIOP Versions*

CORBA Feature	Orbix-Specific Minimum GIOP Version
`fixed` type	1.0
`wchar` and `wstring` types	1.0
codeset negotiation (Orbix E2A only)	1.1

For more details on these CORBA features, see the following sections:

- "Fixed Data Type and Interoperability" on page 116.
- "Use of wchar and wstring" on page 118.
- "Codeset Negotiation for Narrow and Wide Characters" on page 158.

Table of Default GIOP Versions

Table 19 shows the default GIOP versions for different Orbix clients when opening a connection to a server.

Table 19: *Default GIOP Version Used by Orbix Clients*

Client Version	Default GIOP Version
Orbix 3.0.1-82	1.0
OrbixWeb 3.2-15	1.0
Orbix 3.3 C++ Edition	1.1
Orbix 3.3 Java Edition	1.0
Orbix 2000 C++ Edition (up to version 2.0)	1.1
Orbix 2000 Java Edition (up to version 2.0)	1.1

CHAPTER 9

IDL Issues

This chapter describes those features of IDL that affect interoperability between Orbix 3.x and Orbix E2A applications.

In This Chapter

This chapter discusses the following topics:

Use of #pragma prefix

Synopsis

Using the `#pragma prefix` preprocessor directive in your IDL affects the semantics of the `_narrow()` function. When an Orbix 3.0.1-82 or Orbix 3.3 C++ client attempts to `_narrow()` an object reference originating from an Orbix E2A server, a remote `_is_a()` call is implicitly made.

The `#pragma prefix` preprocessor directive is not fully supported in OrbixWeb 3.2-15 and Orbix 3.3 Java Edition. An OrbixWeb 3.2-15 or Orbix 3.3 Java application can, however, interoperate with Orbix E2A, with an implicit `is_a()` call being made by the Orbix runtime.

Effect of `#pragma prefix`

The `#pragma prefix` directive is used to add a prefix to the `RepositoryId` of all the IDL declarations that follow. For example:

```
//IDL
#pragma prefix "mydomain.com"

interface Foo {
    //Various operations and attributes (not shown)
    ...
};
```

The default `RepositoryId` of the `Foo` interface would be `IDL:Foo:1.0`. When used as above, `#pragma prefix` causes the `RepositoryId` of the interface `Foo` to change to `IDL:mydomain.com/Foo:1.0`.

C++ Code Example

Consider, a `Foo` object reference that is generated by an Orbix E2A server. The Orbix E2A server stringifies the object reference using the `CORBA::ORB::object_to_string()` operation and writes it to a temporary file.

An Orbix 3.3 C++ client then reads the stringified object reference from the temporary file and converts it back to a `Foo` object reference as follows:

```
//C++
...
//-------------------------------------------------------------
// The following variables are assumed to be initialized already:
//     'stringObj'- A stringified object reference of char * type
//     'orbV'     - A reference to an ORB object,
//                  of CORBA::ORB_var type
//
try {
    CORBA::Object_var objV = orbV->string_to_object(stringObj);

    // Attempt to 'narrow' the object reference to type 'Foo_ptr'
    Foo_var myFooV = Foo::_narrow(objV);

    if (CORBA::is_nil(myFooV) ) {
        cerr << "error: narrow to Foo failed" << endl;
        exit(1);
    }
}
catch (CORBA::SystemException& sysEx) {
    ...  // deal with exceptions
}
```

Semantics of the `_narrow()` Function

When `Foo::_narrow(objV)` is invoked the object's `RepositoryId` is checked to make sure that it really is of type `Foo`. There are two ways a client can check the type of an object when it performs a `_narrow()`:

- Check the type locally, using the information in the client stub code.
- Check the type remotely, by calling back to the Orbix E2A server. The `_is_a()` function is invoked on the remote `Foo` object.

Because the `Foo` object reference originates from an Orbix E2A server, the Orbix 3.3 C++ client is unable to check the `RepositoryId` using its local stub code. It must call back to the server instead. The implementation of `_narrow()` calls the remote operation CORBA::Object::_is_a() on the object reference objV. The _is_a() function returns TRUE if the object is really of type Foo, otherwise it returns FALSE.

Effect on the CORBA Naming Service

The naming service is affected because it uses a #pragma prefix directive:

```
//IDL for the CORBA Naming Service
#pragma prefix "omg.org"

module CosNaming {
    ...
    interface NamingContext {
       ...
    };
};
```

When used as above, #pragma prefix causes the RepositoryId of the interface NamingContext to change to IDL:omg.org/CosNaming/NamingContext:1.0. An Orbix 3.3 C++ client that uses the Orbix E2A naming service, therefore, implicitly makes a remote _is_a() invocation whenever it invokes _narrow() on a naming service object.

Orbix 3.3 C++ Edition and Orbix E2A

When Orbix 3.3 C++ Edition and Orbix E2A applications are mixed in the same system, you can use IDL that has a #pragma prefix directive but the semantic behavior of _narrow() is affected.

Orbix 3.3 Java Edition and Orbix E2A

If a #pragma prefix preprocessor directive appears in your IDL, it is ignored by the Orbix 3.3 IDL-to-Java compiler. The Java stub and skeleton code is generated as if the #pragma prefix was not there.

When Orbix 3.3 Java Edition and Orbix E2A applications are mixed in the same system, you can use IDL that has a #pragma prefix directive but implicit is_a() calls will be made by the Orbix runtime.

Use of #pragma ID in IDL

Synopsis

The `#pragma ID` directive is supported in Orbix E2A, but is not supported in Orbix 3.3.

Syntax of `#pragma ID`

The `#pragma ID` directive is used to associate an arbitrary repository ID with a given IDL type name. It has the following syntax:

```
#pragma ID TypeName "RepositoryID"
```

The *RepositoryId* must be of the form *Format*:*String* where no colon may appear in *Format*. For example, if the *Format* of the repository ID is `IDL`:

```
//IDL
module Example {
    interface Foo {};
#pragma ID Foo "IDL:ArbitraryFooId:1.1"
};
```

The default repository ID that would normally be associated with `Foo` is `IDL:Example/Foo:1.0`. By including the `#pragma ID` directive the repository ID becomes `IDL:ArbitraryFooId:1.1` instead.

Orbix 3.3 C++ Edition and Orbix E2A

IDL that makes use of the `#pragma ID` directive cannot be used interoperably between Orbix 3.3 C++ Edition and Orbix E2A applications.

Orbix 3.3 Java Edition and Orbix E2A

IDL that makes use of the `#pragma ID` directive cannot be used interoperably between Orbix 3.3 Java Edition and Orbix E2A applications.

Fixed Data Type and Interoperability

Synopsis

When interoperating between an Orbix 3.0.1-82/OrbixWeb 3.2-15 application and an Orbix E2A C++ application, it is necessary to change the configuration of Orbix E2A C++ to use the fixed-point IDL type.

Interoperating with Orbix E2A C++ Edition

To enable the fixed-point type to be sent between an Orbix 3.0.1-82 application and an Orbix E2A C++ Edition application, the following configuration entry must be made (or changed) in the Orbix E2A configuration file:

```
# Orbix E2A Configuration File
policies:giop:interop_policy:allow_fixed_types_in_1_0 = "true";
```

If set to `true`, Orbix E2A permits fixed-point types to be sent over GIOP 1.0. Defaults to `false`.

Interoperating with Orbix E2A Java Edition

Orbix E2A Java accepts fixed-point types through GIOP 1.0 and GIOP 1.1 connections. No special configuration is needed, therefore, when sending fixed-point types between Orbix E2A Java and legacy products such as Orbix 3.0.1-82 or Orbix 3.3.

Orbix 3.0.1-82 and Orbix E2A C++ Edition

Orbix 3.0.1-82 uses GIOP 1.0 by default and Orbix E2A C++ does not permit fixed-point types to be sent over GIOP 1.0. It is necessary, therefore, to reconfigure Orbix E2A C++ in this case by setting the `allow_fixed_types_in_1_0` variable to `true`.

Orbix 3.3 C++ Edition and Orbix E2A C++ Edition

Orbix 3.3 C++ Edition uses GIOP 1.1 by default and Orbix E2A C++ permits fixed-point types to be sent over GIOP 1.1. There is, therefore, no need to reconfigure Orbix E2A C++ in this case.

Orbix 3.3 Java Edition and Orbix E2A C++ Edition

To enable the fixed-point type to be sent between Orbix 3.3 Java Edition and Orbix E2A C++ applications, two alternative configurations can be used:

- Make, or change, the following configuration entry in the Orbix E2A configuration file:

  ```
  # Orbix E2A Configuration File
  policies:giop:interop_policy:allow_fixed_types_in_1_0 =
     "true";
  ```

  If set to `true`, Orbix E2A C++ permits fixed-point types to be sent over GIOP 1.0. Defaults to `false`.

- Alternatively, you can configure Orbix 3.3 Java Edition to use GIOP 1.1 using the `IT_DEFAULT_IIOP_VERSION` configuration variable. This configuration variable can be set in any of the ways described in the Orbix 3.3 Administrator's Guide. For example, you can set it in the `orbixweb3.cfg` file as follows:

  ```
  #File: 'orbixweb3.cfg'
  OrbixWeb {
      # Other options not shown
      # ...
      IT_DEFAULT_IIOP_VERSION = "11";
  };
  ```

  By setting the `IT_DEFAULT_IIOP_VERSION` configuration variable to `11` you ensure that Orbix 3.3 Java Edition uses GIOP 1.1 by default on connections to servers. Because GIOP 1.1 officially supports marshalling of fixed-point data, this enables you to use fixed-point data interoperably.

Note: Orbix 3.3 C++ Edition has a similarly named environment variable, `IT_IIOP_VERSION`. However, setting `IT_IIOP_VERSION` in Orbix 3.3 C++ Edition does not have the same effect as setting `IT_DEFAULT_IIOP_VERSION` in Orbix 3.3 Java Edition. The `IT_IIOP_VERSION` environment variable cannot be used to enable use of the fixed point type between Orbix 3.3 C++ Edition and Orbix E2A.

Use of wchar and wstring

Synopsis

Table 20 summarizes the support for the `wchar` and `wstring` IDL types in the Orbix 3.3 and Orbix E2A products:

Table 20: *Support for the wchar and wstring Types by Product*

Product	**Supports** `wchar`	**Supports** `wstring`
Orbix E2A C++	*Yes*[a]	*Yes*
Orbix E2A Java	*Yes*	*Yes*
Orbix 3.3 C++ Edition	No	No
Orbix 3.3 Java Edition	*Yes*	*Yes*

a. In Orbix E2A C++ release 5.0, there is currently a bug that affects the transmission and reception of `wchar` characters when interoperating with other ORB products. This bug does not affect the `wstring` type.

All of the products that support `wchar` and `wstring` types can interoperate with each other (with the exception, currently, of Orbix E2A C++ when transmitting `wchar`s to and from other ORB products).

C++ Keywords as Operation Names

Synopsis

Previously, if your IDL contained operation names that are the same as C++ keywords, Orbix 3.0.1-82 and Orbix 3.3 C++ Edition could not interoperate with Orbix E2A.

This problem is now fixed. Orbix 3.3 applications can now interoperate with Orbix E2A even when your IDL contains C++ keywords as operation names.

IDL Example

Consider the following IDL:

```
//IDL
interface CPlusPlusKeywords {
    void    for();
    boolean class();
};
```

C++ Stub Code

The Orbix 3.3 IDL-to-C++ compiler maps this interface to the following proxy class:

```
//C++
class CPlusPlusKeywords: public virtual CORBA::Object {
    ...
public:
    ...
    virtual void _for (...) ;
    virtual CORBA::Boolean _class (...) ;
    ...
};
```

The names of the functions in C++ have a leading underscore character, for example `_for` and `_class`, to avoid clashing with the C++ keywords `for` and `class`.

On-the-Wire Format for Operation Names

When an Orbix 3.3 C++ or Java client makes a remote invocation using the `_for()` and `_class()` functions, the operation names are marshalled as `"for"` and `"class"` respectively. This behavior complies with CORBA 2.4 and is compatible with Orbix E2A servers.

Exceptions

This chapter discusses the differences in the handling of CORBA exceptions between Orbix 3.x and Orbix E2A.

In This Chapter

This chapter discusses the following topics:

Orbix 3.3 C++ Edition—System Exceptions

Synopsis

The semantics of system exceptions in Orbix prior to Orbix 3.0.1-20 are different from the semantics in Orbix E2A. In Orbix 3.3, however, exception semantics have been altered to make them compatible with Orbix E2A. An environment variable `IT_USE_ORBIX3_STYLE_SYS_EXC` is introduced that enables you to insulate legacy code from the change.

New Semantics and Old Semantics

Some system exceptions in Orbix E2A have different semantics to the corresponding exceptions in Orbix prior to Orbix 3.0.1-20. The exception semantics used by Orbix E2A are referred to here as *new semantics*. The exception semantics used by Orbix prior to Orbix 3.0.1-20 are referred to here as *old semantics*.

The `IT_USE_ORBIX3_STYLE_SYS_EXC` Variable

The `IT_USE_ORBIX3_STYLE_SYS_EXC` variable affects three different aspects of Orbix 3.0.1-82 and Orbix 3.3 applications:

- System exceptions raised by the server.
- System exceptions raised by the client.
- Transformation of exceptions arriving at the client.

System exceptions are not only raised by servers, they can also be raised on the client side. If a client encounters an error before it sends a `Request` message to a server, or after it receives a `Reply` message from a server, the client raises a system exception. The `IT_USE_ORBIX3_STYLE_SYS_EXC` variable therefore affects both client and server applications.

System Exceptions Raised by the Server

System exceptions raised by an Orbix 3.0.1-82 and Orbix 3.3 server are influenced in the following way by `IT_USE_ORBIX3_STYLE_SYS_EXC`.

Table 21: *Effect of IT_USE_ORBIX3_STYLE_SYS_EXC on a Server*

`IT_USE_ORBIX3_STYLE_SYS_EXC`	**Orbix 3.3 Server - Exception Raising**
Not defined	Old semantics
`YES`	Old semantics
`NO`	New semantics

System Exceptions Raised by the Client

System exceptions raised by an Orbix 3.0.1-82 and Orbix 3.3 client are influenced in the following way by `IT_USE_ORBIX3_STYLE_SYS_EXC`.

Table 22: *Effect of IT_USE_ORBIX3_STYLE_SYS_EXC on a Client*

`IT_USE_ORBIX3_STYLE_SYS_EXC`	**Orbix 3.3 Client - Exception Raising**
Not defined	Old semantics
`YES`	Old semantics
`NO`	New semantics

Transformation of Exceptions Arriving at the Client

Transformation of exceptions arriving at an Orbix 3.0.1-82 and Orbix 3.3 client are influenced in the following way by `IT_USE_ORBIX3_STYLE_SYS_EXC`.

Table 23: *Transformation of Exceptions at the Client Side*

`IT_USE_ORBIX3_STYLE_SYS_EXC`	**Orbix 3.3 Client - Exception Raising**
Not defined	Transform to old semantics
`YES`	Transform to old semantics
`NO`	Transform to new semantics

Transformation is applied to system exceptions incoming from the network. This feature dynamically intercepts system exceptions arriving at the client and, if necessary, converts them to the type of system exception expected by the client (consistent with either new or old semantics). This is essential to ensure that the client can apply a consistent style of exception handling irrespective of the type of server it is talking to.

Difference between Orbix Prior to Orbix 3.0.1-82 and Orbix 3.3

The presence of the transformation feature means that there is a significant difference between Orbix clients prior to Orbix 3.0.1-20 and Orbix 3.0.1-82/Orbix 3.3 clients even when the variable `IT_USE_ORBIX3_STYLE_SYS_EXC` is not set (or set equal to YES). An Orbix 3.0.1-82 or Orbix 3.3 client that uses old semantics actively transforms incoming system exceptions to old semantics. A pre-Orbix 3.0.1-20 client does not.

In This Section

This section contains the following subsections:

The INV_OBJREF and OBJECT_NOT_EXIST Exceptions

Orbix E2A Semantics

In Orbix E2A the `INV_OBJREF` and `OBJECT_NOT_EXIST` system exceptions are raised under the following circumstances:

- The `INV_OBJREF` system exception is raised by `CORBA::ORB::string_to_object()` to indicate that the stringified object reference is malformed in some way.
- The `OBJECT_NOT_EXIST` system exception is raised by a server to indicate that a CORBA object does not exist.

Orbix 3.3 (New Semantics)

In Orbix 3.0.1-82 and Orbix 3.3 (new semantics) the `INV_OBJREF` and `OBJECT_NOT_EXIST` system exceptions are raised under the following circumstances:

- The `INV_OBJREF` system exception is raised for a variety of reasons. However, it is not raised to indicate that a CORBA object does not exist.
- The `OBJECT_NOT_EXIST` system exception is raised by a server to indicate that a CORBA object does not exist.

Pre-Orbix 3.0.1-20 (Old Semantics)

Prior to **Orbix 3.0.1-20** (old semantics) the `INV_OBJREF` and `OBJECT_NOT_EXIST` system exceptions are raised under the following circumstances:

- The `INV_OBJREF` system exception is raised for a variety of reasons. When raised by a server, with minor code `10101`, it indicates that a CORBA object does not exist.
- The `OBJECT_NOT_EXIST` system exception is never raised by pre-Orbix 3.0.1-20 applications.

The TRANSIENT and COMM_FAILURE Exceptions

Orbix E2A Semantics and Orbix 3.3 (New Semantics)

In Orbix E2A and in Orbix 3.0.1-82/Orbix 3.3 (new semantics) the `TRANSIENT` and `COMM_FAILURE` system exceptions are raised under the following circumstances:

- The `TRANSIENT` exception is raised if a client tries to send a message to a server but is unable to do so. In terms of the TCP/IP transport layer, this means an error occurred before or during an attempt to write to or connect to a socket.
- The `COMM_FAILURE` exception is raised if a client has already sent a message to a server but is unable to receive the associated reply. In terms of the TCP/IP transport layer, this means either the connection went down or an error occurred during an attempt to read from a socket.

Pre-Orbix 3.0.1-20 (Old Semantics)

Prior to Orbix 3.0.1-20 (old semantics) the `TRANSIENT` and `COMM_FAILURE` system exceptions are raised under the following circumstances:

- The `TRANSIENT` exception is never raised in pre-Orbix 3.0.1-20 applications.
- The `COMM_FAILURE` exception is raised in pre-Orbix 3.0.1-20 applications if an error occurs while writing to, reading from, or connecting on a TCP/IP socket.

Orbix 3.3 C++ Edition and Orbix E2A

Overview

There are three different ways of setting the `IT_USE_ORBIX3_STYLE_SYS_EXC` configuration value:

- Set an environment variable.
- Set a configuration variable.
- Use the `SetConfigValue()` function.

Set an Environment Variable

Set the environment variable, `IT_USE_ORBIX3_STYLE_SYS_EXC`, as follows:

Windows

`set IT_USE_ORBIX3_STYLE_SYS_EXC=`*yes_or_no*

UNIX

`export IT_USE_ORBIX3_STYLE_SYS_EXC=`*yes_or_no*

Where *yes_or_no* can be the string `YES` or `NO`.

Set a Configuration Variable

Set the configuration variable, `IT_USE_ORBIX3_STYLE_SYS_EXC`, by editing the Orbix 3.3 configuration file:

```
# Orbix 3.3 Configuration File
Orbix {
    IT_USE_ORBIX3_STYLE_SYS_EXC = "yes_or_no";
};
```

Use the `SetConfigValue()` Function

Use the `CORBA::ORB::SetConfigValue()` function:

```
// C++
orb_p->SetConfigValue(
           "Orbix.IT_USE_ORBIX3_STYLE_SYS_EXC",
           "yes_or_no"
       );
```

Where `orb_p` is a pointer to a `CORBA::ORB` instance.

Compatibility Matrix

Table 24 shows the compatibility matrix between Orbix 3.0.1-82/Orbix 3.3 and Orbix E2A.

Table 24: *System Exception Handling Compatibility between* Orbix 3.0.1-82/Orbix 3.3 *and Orbix E2A*

Client Application	Orbix 3.0.1-82/Orbix 3.3 **Server (Old Semantics)**	Orbix 3.0.1-82/Orbix 3.3 **Server (New Semantics)**	**Orbix E2A Server**
Orbix 3.0.1-82/Orbix 3.3 Client (Old Semantics)	*Yes*	*Yes*	*Yes*
Orbix 3.0.1-82/Orbix 3.3 Client (New Semantics)	*Yes*	*Yes*	*Yes*
Orbix E2A Client	*No*	*Yes*	*Yes*

A *Yes* entry in the above table indicates compatible exception semantics for that combination.

An Orbix 3.0.1-82/Orbix 3.3 application described in the table as old semantics has its `IT_USE_ORBIX3_STYLE_SYS_EXC` variable set equal to `YES`, or unset. An Orbix 3.0.1-82/Orbix 3.3 application described in the table as new semantics has its `IT_USE_ORBIX3_STYLE_SYS_EXC` variable set equal to `NO`.

Orbix 3.3 Java Edition—System Exceptions

Synopsis

The semantics of system exceptions in OrbixWeb prior to OrbixWeb 3.2-05 are different from the semantics in Orbix E2A. In OrbixWeb 3.2-15 and Orbix 3.3 Java Edition, however, exception semantics have been altered to make them compatible with Orbix E2A. An environment variable `IT_USE_ORBIX3_STYLE_SYS_EXC` is introduced that enables you to insulate legacy code from the change.

New Semantics and Old Semantics

Some system exceptions in Orbix E2A have different semantics to the corresponding exceptions in OrbixWeb prior to OrbixWeb 3.2-05. The exception semantics used by Orbix E2A are referred to here as new semantics. The exception semantics used by OrbixWeb prior to OrbixWeb 3.2-05 are referred to here as old semantics.

The `IT_USE_ORBIX3_STYLE_SYS_EXC` Variable

The `IT_USE_ORBIX3_STYLE_SYS_EXC` variable affects two aspects of OrbixWeb 3.2-15 and Orbix 3.3 Java Edition applications:

- System exceptions raised by the server.
- System exceptions raised by the client.

The `IT_USE_ORBIX3_STYLE_SYS_EXC` variable therefore affects both client and server applications.

Note: OrbixWeb 3.2-15 and Orbix 3.3 Java applications do not perform transformations on incoming system exceptions.

System Exceptions Raised by the Server

System exceptions raised by an OrbixWeb 3.2-15/Orbix 3.3 Java server are influenced in the following way by `IT_USE_ORBIX3_STYLE_SYS_EXC`.

Table 25: *Effect of IT_USE_ORBIX3_STYLE_SYS_EXC on a Server*

`IT_USE_ORBIX3_STYLE_SYS_EXC`	**Orbix 3.3 Java Server - Exception Raising**
Not defined	Old semantics
`TRUE`	Old semantics
`FALSE`	New semantics

System Exceptions Raised by the Client

System exceptions raised by an OrbixWeb 3.2-15/Orbix 3.3 Java client are influenced in the following way by `IT_USE_ORBIX3_STYLE_SYS_EXC`.

Table 26: *Effect of IT_USE_ORBIX3_STYLE_SYS_EXC on a Client*

`IT_USE_ORBIX3_STYLE_SYS_EXC`	**Orbix 3.3 Java Client - Exception Raising**
Not defined	Old semantics
`TRUE`	Old semantics
`FALSE`	New semantics

In This Section

This section contains the following subsections:

The INV_OBJREF and OBJECT_NOT_EXIST Exceptions

Orbix E2A Semantics and Orbix 3.3 Java Edition (New Semantics)

In Orbix E2A and OrbixWeb 3.2-15/Orbix 3.3 Java Edition (new semantics) the `INV_OBJREF` and `OBJECT_NOT_EXIST` system exceptions are raised under the following circumstances:

- The `INV_OBJREF` system exception is raised by `CORBA::ORB::string_to_object()` to indicate that the stringified object reference is malformed in some way.
- The `OBJECT_NOT_EXIST` system exception is raised by a server to indicate that a CORBA object does not exist.

Orbix 3.3 Java Edition (Old Semantics)

In OrbixWeb 3.2-15/Orbix 3.3 Java Edition (old semantics) the `INV_OBJREF` and `OBJECT_NOT_EXIST` system exceptions are raised under the following circumstances:

- The `INV_OBJREF` system exception, with minor code `10100`, is raised by a server to indicate that a CORBA object does not exist.
- The `OBJECT_NOT_EXIST` system exception is never raised in OrbixWeb 3.2-15/Orbix 3.3 Java Edition (old semantics).

The TRANSIENT and COMM_FAILURE Exceptions

Orbix E2A Semantics and Orbix 3.3 Java Edition (New Semantics)

In Orbix E2A and OrbixWeb 3.2-15/Orbix 3.3 Java Edition (new semantics) the `TRANSIENT` and `COMM_FAILURE` system exceptions are raised under the following circumstances:

- The `TRANSIENT` exception is raised if a client tries to send a message to a server but is unable to do so. In terms of the TCP/IP transport layer, this means an error occurred before or during an attempt to write to or connect to a socket.
- The `COMM_FAILURE` exception is raised if a client has already sent a message to a server but is unable to receive the associated reply. In terms of the TCP/IP transport layer, this means either the connection went down or an error occurred during an attempt to read from a socket.

Orbix 3.3 Java Edition (Old Semantics)

In OrbixWeb 3.2-15/Orbix 3.3 Java Edition (old semantics) the `TRANSIENT` and `COMM_FAILURE` system exceptions are raised under the following circumstances:

- The `TRANSIENT` exception can be raised in an OrbixWeb 3.2-15/Orbix 3.3 Java client when attempting to make a connection through Orbix Wonderwall, or when attempting to deal with a `LOCATION_FORWARD` Reply message.
- The `COMM_FAILURE` exception is raised in OrbixWeb 3.2-15/Orbix 3.3 Java Edition (old semantics) if an error occurs while writing to, reading from, or connecting on a TCP/IP socket.

Orbix 3.3 Java Edition and Orbix E2A

Setting the IT_USE_ORBIX3_STYLE_SYS_EXC Variable

The IT_USE_ORBIX3_STYLE_SYS_EXC variable can be set in any of the ways described in the OrbixWeb Administrator's Guide.

For example, to switch on new semantics you can make the following entry in the OrbixWeb3.cfg configuration file:

```
# Orbix 3.3 Configuration File
OrbixWeb.IT_USE_ORBIX3_STYLE_SYS_EXC = "FALSE";
```

Compatibility Matrix

Table 27 shows the compatibility matrix between OrbixWeb 3.2-15/Orbix 3.3 Java Edition and Orbix E2A.

Table 27: *System Exception Handling Compatibility between* OrbixWeb 3.2-15/*Orbix 3.3 Java Edition and Orbix E2A*

Client Application	OrbixWeb 3.2-15/**Orbix 3.3 Java Server (Old Semantics)**	OrbixWeb 3.2-15/**Orbix 3.3 Java Server (New Semantics)**	**Orbix E2A Server**
OrbixWeb 3.2-15/Orbix 3.3 Java Client (Old Semantics)	*Yes*	*No*	*No*
OrbixWeb 3.2-15/Orbix 3.3 Java Client (New Semantics)	*No*	*Yes*	*Yes*
Orbix E2A Client	*No*	*Yes*	*Yes*

A *Yes* entry in the above table indicates compatible exception semantics for that combination.

An OrbixWeb 3.2-15/Orbix 3.3 Java application described in the table as old semantics has its IT_USE_ORBIX3_STYLE_SYS_EXC variable set equal to TRUE, or unset. An OrbixWeb 3.2-15/Orbix 3.3 Java application described in the table as new semantics has its IT_USE_ORBIX3_STYLE_SYS_EXC variable set equal to FALSE.

FILTER_SUPPRESS Exception

Synopsis

The FILTER_SUPPRESS exception is a system exception specific to Orbix and OrbixWeb. If an Orbix 3.3 C++ server or an Orbix 3.3 Java server sends the FILTER_SUPPRESS exception to an Orbix E2A client, it is converted to the standard system exception CORBA::UNKNOWN.

Purpose of the FILTER_SUPPRESS Exception

Filters are a proprietary feature of Orbix 3.3 that enable you to read and manipulate all incoming and outgoing messages. Prior to the availability of a standard CORBA Security Service some applications used filters to implement a rudimentary security mechanism. These legacy applications could block the execution of an operation on the server side by raising the FILTER_SUPPRESS exception in a filter.

How Orbix E2A Handles a FILTER_SUPPRESS Exception

When a FILTER_SUPPRESS exception is sent back to an Orbix E2A client, the Orbix E2A client does not recognize the exception. A CORBA::UNKNOWN system exception is raised instead by the Orbix E2A client.

Dynamic Invocation Interface and User Exceptions

Synopsis

The dynamic invocation interface (DII) in Orbix 3.3 cannot handle CORBA user exceptions.

Orbix 3.3 and User Exceptions

If a user exception is received by an Orbix 3.3 invocation, the Orbix 3.3 runtime converts the exception into a `CORBA::UNKNOWN` system exception which is then thrown by the `CORBA::Request::invoke()` operation.

Handling User Exceptions in Orbix 3.3 C++ Edition

Given an initialized request object, `req`, the following example shows an outline of how to deal with user exceptions in the DII:

```
// C++ - Orbix 3.3
// Initialize DII Request object, req.
...
// Make the invocation
try {
    req.invoke();
}
catch (...) {
    // You will reach this point if a user exception is thrown.
    ...
}
```

Handling User Exceptions in Orbix 3.3 Java Edition

Given an initialized request object, `req`, the following example shows an outline of how to deal with user exceptions in the DII:

```
// Java - Orbix 3.3
// Initialize DII Request object, req.
...
// Make the invocation
try {
    req.invoke();
}
catch (java.lang.Exception) {
    // You will reach this point if a user exception is thrown.
    ...
}
```

Orbix E2A and User Exceptions

In the Orbix E2A DII, however, user exceptions are supported in the DII. The standard exception class `CORBA::UnknownUserException` holds a `CORBA::Any` which can then be parsed with the aid of the dynamic any module to obtain the contents of the user exception.

Dynamic Invocation Interface and LOCATION_FORWARD

Synopsis

The dynamic invocation interface (DII) in Orbix 3.3 C++ Edition is now able to handle reply messages that have the status `LOCATION_FORWARD`. Previously, `LOCATION_FORWARD` replies were not supported in Orbix C++ applications.

The DII in Orbix 3.3 Java Edition has always been able to handle reply messages that have the status `LOCATION_FORWARD`.

See also "Multiple LOCATION_FORWARD" on page 156.

Location Forwarding Mechanisms

The IIOP protocol features support for location forwarding. It is used to dynamically discover the location of CORBA objects. There are two distinct kinds of message exchange that form the basis of location forwarding:

- The client ORB can deliberately probe the location of a CORBA object by sending a `LocateRequest` message to the server (or agent). The server (or agent) responds with a `LocateReply` message containing details of the object's location.
- When a client sends a regular `Request` message the server (or agent) might respond with a special type of `Reply` message that has a reply status of `LOCATION_FORWARD`. This reply will have details of the object's location.

Support for Location Forwarding

The location forward mechanism is used by the Orbix 3.3 daemon and the Orbix E2A locator service to direct clients to the true location of a CORBA server.

- The first type of message exchange is a `LocateRequest` followed by `LocateReply`.
- The second type of message exchange is a `Request` followed by a `Reply` with status `LOCATION_FORWARD`.

Both kinds of message exchange are supported in Orbix 3.3.

Services

In a mixed system with Orbix 3.x and Orbix E2A applications, you generally have a choice between an Orbix 3.x or an Orbix E2A implementation of a CORBA service. This chapter discusses the viable configurations of CORBA services in a mixed system.

In This Chapter

This chapter discusses the following topics:

The Orbix E2A Interoperable Naming Service

Synopsis

The naming service provided with Orbix E2A is an implementation of the CORBA Interoperable Naming Service (INS) specification. This section explains how to set up Orbix 3.3 applications to use the Orbix E2A INS.

Old and New Naming Services

In an environment that mixes Orbix 3.3 and Orbix E2A applications you have a choice between using the old CORBA Naming Service (NS), provided with Orbix 3.3, or the new CORBA Interoperable Naming Service (INS), provided with Orbix E2A.

The `NamingContextExt` Interface

The main difference between the old and new naming services is that the INS adds a new IDL `CosNaming::NamingContextExt` interface.

```
// File: CosNaming.idl
#pragma prefix "omg.org"

module CosNaming {
    ...
    interface NamingContextExt : NamingContext {
        typedef string StringName;
        typedef string Address;
        typedef string URLString;

        StringName to_string (in Name n)
            raises (InvalidName);
        Name to_name (in StringName sn)
            raises (InvalidName);

        exception InvalidAddress {};

        URLString to_url (in Address addr, in StringName sn)
            raises (InvalidAddress, InvalidName);
        Object resolve_str (in StringName sn)
            raises (NotFound, CannotProceed, InvalidName);

    };
};
```

Stub Code

Applications that use the INS should preferably be built against the new naming stub (generated from the interoperable naming service IDL). This makes the new `NamingContextExt` interface accessible. However, the old naming stubs (generated from the old naming service IDL) can also be used.

Narrowing and Remote `_is_a()` Operation

When an Orbix 3.3 application invokes `CosNaming::NamingContext::_narrow()` on an Orbix E2A `NamingContext` it makes a remote `_is_a()` invocation on the INS. The `_is_a()` invocation is used to confirm the type of the `NamingContext` object reference. See “Use of #pragma prefix” on page 112.

Orbix 3.3 and Orbix E2A

You can configure Orbix 3.3 to use both the Orbix 3.3 naming service and the Orbix E2A INS. This section describes how to configure the CORBA Initialization Service to obtain a reference to either naming service using the `CORBA::ORB::resolve_initial_references()` function.

Configuring Orbix 3.3 to Use the Orbix E2A INS

To connect both to the Orbix 3.3 naming service and the Orbix E2A naming service from an Orbix 3.3 application you must first configure the initialization service. Edit the configuration file `common.cfg` and make the following entries in the scope `Common.Services`.

```
# Orbix 3.3 Configuration File
Common {
    Services {
       # This is the stringified IOR for the root 'NamingContext'
        # of the 'Orbix 3' naming service.
        # You can obtain this IOR by running the naming service
        # as follows:
        #     ns -I <iorfile>
        NameService = "IOR:1234......";

       # This is the stringified IOR for the root 'NamingContext'
        # of the 'Orbix E2A' Interoperable Naming Service.
        # You can obtain this IOR using the Orbix E2A admin
        # utility as follows:
        #     itadmin ns resolve
        INS = "IOR:4567......";
    };
};
```

Orbix 3.3 Configuration Variables

The following configuration variables are set in the `Common.Services` scope:

- The configuration variable `Common.Services.NameService` is set to a stringified IOR for a `NamingContext` in the Orbix 3 naming service.
- The configuration variable `Common.Services.INS` is set to a stringified IOR for a `NamingContext` in the Orbix E2A interoperable naming service.

Setting the `Common.Services.INS` Variable

For example, consider the following IOR string:

```
IOR:010000002f00000049444c3a696f6e612e636f6d2f49545f4e616d696e672
f49545f4e616d696e67436f6e746578744578743a312e30000001000000000000
006e000000010102000b00000031302e322e312e31313300008a1300003f00000
03a3e0232311744656661756c74204c6f636174696f6e20446f6d61696e185f64
656661756c745f69745f6e635f6578745f706f615f00080000000000000200000
10000000600000006000000010000003500
```

You can assign this IOR string to `Common.Services.INS` as follows:

```
# Orbix 3.3 Configuration File
Common {
    Services {
        INS =
   "IOR:010000002f00000049444c3a696f6e612e636f6d2f49545f4e616d69
   6e672f49545f4e616d696e67436f6e746578744578743a312e30000001000
   000000000006e000000010102000b00000031302e322e312e31313300008a
   1300003f0000003a3e0232311744656661756c74204c6f636174696f6e204
   46f6d61696e185f64656661756c745f69745f6e635f6578745f706f615f00
   0800000000000002000001000000060000000600000001000000 3500";
    };
};
```

Orbix 3.3 Client Code for Using Both Naming Services

The following C++ code extract shows how an Orbix 3.0.1-20 application can make an initial connection to both naming services.

```
// C++ - Orbix 3 Client Code
int
main (int argc, char *argv[])
{
    CORBA::ORB_var orbV;

    try
    {
        cout << "Initializing the ORB." << endl;
        orbV = CORBA::ORB_init(argc, argv, "Orbix");
        CosNaming::NamingContext_var orbix3RootContextV;
        CosNaming::NamingContext_var orbix2000RootContextV;
        CORBA::Object_var objV;

        try
        {
            objV =
    orbV->resolve_initial_references("NameService");
            orbix3RootContextV =
    CosNaming::NamingContext::_narrow(objV);

            objV = orbV->resolve_initial_references("INS");
            orbix2000RootContextV =
    CosNaming::NamingContext::_narrow(objV);
        }
        catch (CORBA::SystemException &sysEx)
        {
            cerr << &sysEx << endl;
            return 1;
        }
        ...
    ...
}
```

After this code runs, `orbix3RootContextV` holds a reference to an Orbix 3 `NamingContext` and `orbix2000RootContextV` holds a reference to an Orbix E2A `NamingContext`.

Orbix 3.3 Java Edition and Orbix E2A

This section describes how to configure Orbix 3.3 Java Edition to connect to both the Orbix 3.3 naming service and the Orbix E2A naming service.

1 Obtain the IOR for the root naming context of the naming service.

Start the Orbix E2A naming service and enter the following command:

```
itadmin ns resolve > Naming.ref
```

The output of this command is an IOR string that looks similar to this:

```
IOR:010000002f00000049444c3a696f6e612e636f6d2f49545f4e616d696e672
f49545f4e616d696e67436f6e746578744578743a312e3000000001000000000000
006e000000010102000b00000031302e322e312e31313300008a1300003f00000
03a3e0232311744656661756c74204c6f636174696f6e20446f6d61696e185f64
656661756c745f69745f6e635f6578745f706f615f00080000000000000200000
1000000060000000600000001000000 3500
```

This is the IOR string for the root naming context of the Orbix E2A naming service.

---

2 The following Java code shows how an Orbix 3.3 Java client connects to both the Orbix 3.3 naming service and the Orbix E2A naming service:

```
//Java
NamingContext OWrootContext = null;

try {
    org.omg.CORBA.Object ncOWeb =
        orb_wrapper.get_orb().resolve_initial_references(
                                    "NameService"
                                );
    OW32rootContext = NamingContextHelper.narrow(ncOWeb);

    // read the ART Naming IOR from the file:
    String objRef = null;
    BufferedReader br = null;

    try {
        br = new BufferedReader( new FileReader("Naming.ref") );
        objRef = br.readLine();
    } catch (IOException e) {
        System.err.println(
            "IOException caught: " + e.toString()
        );
        ioe = new IOException();
    } finally {
        try {
            br.close();
        } catch (IOException ignore) { }
    }
   org.omg.CORBA.Object objNaming =
   orb.string_to_object(objRef);
    O2KRootContext = NamingContextHelper.narrow(objNaming);
} catch (SystemException ex) {
    System.err.println ("Exception caught during bind : " +
   ex.toString());
    System.exit (1);
} catch (org.omg.CORBA.ORBPackage.InvalidName in) {
    System.err.println ("Exception during narrow of initial
   reference : "
                         + in.toString());
    System.exit (1);
}
```

This code reads the stringified IOR for the Orbix E2A naming service from the file `Naming.ref`. The stringified IOR is converted to an object reference, `O2KRootContext`, using the `org.omg.CORBA.ORB.string_to_object()` function. The `O2KRootContext` object reference is used to access the root `NamingContext` of the Orbix E2A naming service.

Interface Repository Interoperability

Synopsis

Significant changes were made to the IDL definition of the Interface Repository (IFR) between CORBA 2.2 and CORBA 2.3. The Orbix E2A IFR is written to conform to the CORBA 2.4 specification and it has many advantages over the Orbix 3.3 IFR.

If you have both Orbix 3.3 and Orbix E2A applications that use the IFR, it is recommended that you change the Orbix 3.3 applications to use the Orbix E2A IFR.

Modifying Orbix 3.3 Applications to Use the Orbix E2A IFR

To change an Orbix 3.3 C++ application to use the Orbix E2A IFR, perform the following steps:

1. Take the IDL for the Orbix E2A IFR and generate stub code from it using the Orbix 3.3 IDL compiler.
2. Modify the source code of your Orbix 3.3 application to be consistent with the IDL for the Orbix E2A IFR.
3. Link your Orbix 3.3 application with the IFR stub code generated in step 1.

SSL/TLS Toolkit Interoperability

Orbix 3.3 to Orbix E2A Interoperability

Orbix version 3.3 or later is recommended for secure interoperability with Orbix E2A SSL/TLS. Both C++ and Java editions of Orbix 3.3 have been tested with Orbix E2A SSL/TLS. There are no known SSL-related interoperability problems affecting this product combination.

Orbix E2A Interoperability with Orbix 2000

Orbix E2A SSL/TLS (both C++ and Java) has been tested for secure interoperability with Orbix 2000 versions 1.2 and 2.0. There are no known SSL-related interoperability problems.

High Availability and Orbix 3.3 Clients

Synopsis

High availability is a feature of Orbix E2A that provides fault tolerance by grouping servers into *server clusters*. Orbix 3.3 clients (C++ and Java Editions) are now able to interoperate with Orbix E2A server clusters.

Support for Multi-Profile IORs

In Orbix 3.3.2 the client ORB iterates over a multi-profiled IOR until it is able to establish a connection to a server. It always starts at the first profile when connecting or reconnecting to a server.

CHAPTER 12

Connection Management

There are some differences in connection management between Orbix 3.x and Orbix E2A applications. In most cases these differences are unimportant but a minority of applications might be affected.

In This Chapter

This chapter discusses the following topics:

Orbix E2A Active Connection Management

Synopsis

Orbix E2A has a feature called active connection management (ACM) that is used to limit the number of open connections on an Orbix E2A application.

The Orbix E2A ACM feature has been interoperably tested with Orbix 3.3 and found to be fully compatible.

Configuring the ACM

To configure ACM in Orbix E2A edit the configuration file, making the following additional entries:

```
# Orbix E2A Configuration File
plugins:iiop:incoming_connections:hard_limit = "InHardLimit";
plugins:iiop:incoming_connections:soft_limit = "InSoftLimit";
plugins:iiop:outgoing_connections:hard_limit = "OutHardLimit";
plugins:iiop:outgoing_connections:soft_limit = "OutSoftLimit";
```

A value of `-1` indicates that there is no limit on the number of connections.

Callbacks and Bi-Directional IIOP

Synopsis

Orbix E2A does not support bi-directional IIOP. When an Orbix 3.3 application communicates with an Orbix E2A application, therefore, the connection semantics automatically revert to standard IIOP instead of bi-directional IIOP.

Callbacks can work but they will use standard IIOP instead of bi-directional IIOP. This has implications for using IIOP through a firewall.

Motivation for Bi-Directional IIOP

Bi-directional IIOP was introduced in Orbix in order to overcome the limitations of standard IIOP in relation to using callback objects through a firewall.

Standard IIOP—Single Connection

A standard IIOP connection between client **C** and server **S** is illustrated in Figure 1.

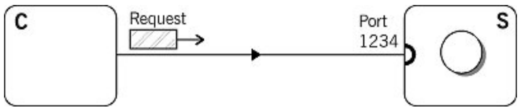

Figure 1: *Standard IIOP - Single Connection*

The server process supports one or more CORBA objects and listens on an IP port 1234. The client establishes a single TCP/IP connection to the server. Request messages flow from client to server, and reply messages from server to client.

Standard IIOP—Two Connections

Figure 2 shows the connections between a client and a server, where the client supports a callback object and bi-directional IIOP is switched off.

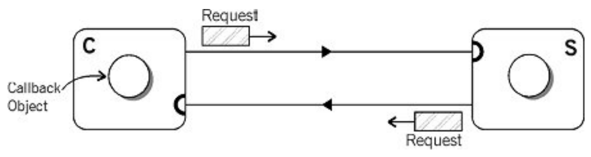

Figure 2: *Standard IIOP - Two Connections*

The callback object is the CORBA object implemented in the client. To begin with, the client opens a TCP/IP connection to the server on port 1234. This enables the client to invoke on the server object. At some point, the client passes an IOR for the callback object to the server (for example, as the parameter of an operation). Under the rules of standard IIOP, when the server tries to invoke on the callback object a new connection must be opened to the client's IP port 2345. Using this second connection, requests messages can flow in the reverse direction from server to client, and reply messages from client to server.

A second connection is required for callbacks because the IIOP specification requires request messages to be sent in only one direction through a given TCP/IP connection. However, there is no intrinsic limitation that prevents a TCP/IP connection from sending requests in either direction. The TCP/IP transport layer is intrinsically bi-directional.

Bi-Directional IIOP

Figure 3 illustrates bi-directional IIOP. Bi-directional IIOP takes advantage of the intrinsically bi-directional nature of TCP/IP by reusing the original connection to send requests in the reverse direction.

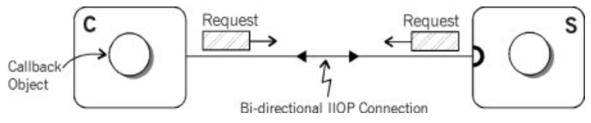

Figure 3: *Bi-Directional IIOP*

In this mode, request messages can be sent from client to server, or from server to client. There is no need to open a new TCP/IP connection from server to client: the server invokes on the callback object by sending request messages in the reverse direction along the original connection.

The reason for using bi-directional IIOP in Orbix 3.3 is because the standard IIOP approach to callbacks, opening a second connection from the server to the client, does not work when a firewall is interposed between the client and server. In this situation, reusing the original TCP/IP connection is the only way servers can make invocations on callback objects.

CORBA Firewall Specification

The limitations of standard IIOP when used in conjunction with a firewall are currently being addressed by the CORBA Firewall Specification. As soon as the firewall standard is adopted it will be implemented by Orbix E2A (including a standardized version of bi-directional IIOP mode). In the meantime, bi-directional mode is not available in Orbix E2A. Callbacks still work, however, using standard IIOP with two connections.

Setting the Listen Queue Size in Orbix 3.3 C++ Edition

Synopsis

A new configuration variable `IT_LISTEN_QUEUE_SIZE` is defined in Orbix 3.3 C++ Edition. It allows you to set the size of the queue associated with listening ports on an Orbix 3.3 C++ server. This is a useful optimization for a heavily loaded server that might receive many connection attempts in a short time.

Listen Queue Size

When an Orbix server wants to receive connections from clients it needs to call the `listen(int,int)` socket function. The second parameter of `listen()` sets the listen queue size associated with the socket. The listen queue size determines the maximum length that the queue of pending connections may grow to. In Orbix 3.3, the queue length is 5 by default.

The `IT_LISTEN_QUEUE_SIZE` Configuration Variable

Orbix 3.3 C++ Edition supports a new `IT_LISTEN_QUEUE_SIZE` configuration variable that enables you to configure the listen queue size. It can be set subject to the following constraints:

- The value should lie between 5 and 2000 (inclusive).
- If it is set to a value less than 5, the value 5 is used instead.
- If it is set to a value greater than 2000, the value 2000 is used instead.

Queue Size Hard Limit

The maximum queue size is subject to a hard limit that varies between platforms:

- *Solaris*—there is currently no limit.
- *HPUX*—the limit is 20.
- *Windows*—the limit is 5.

How to Set the Listen Queue Size

There are three different ways to set the `IT_LISTEN_QUEUE_SIZE` configuration value:

- Set the environment variable `IT_LISTEN_QUEUE_SIZE`:

  Windows

  ```
  set IT_LISTEN_QUEUE_SIZE=QueueSize
  ```

UNIX

`export IT_LISTEN_QUEUE_SIZE=`*QueueSize*

- Set the configuration variable `IT_LISTEN_QUEUE_SIZE` by editing the Orbix 3.3 configuration file as follows:

```
# Orbix 3.3 Configuration File
Orbix {
    IT_LISTEN_QUEUE_SIZE = "QueueSize";
};
```

- Use the `CORBA::ORB::SetConfigValue()` function:

```
// C++
orb_p->SetConfigValue(
           "Orbix.IT_LISTEN_QUEUE_SIZE",
           "QueueSize"
       );
```

Where `orb_p` is a pointer to a `CORBA::ORB` instance.

How to Query the Listen Queue Size

An application can query the value of `IT_LISTEN_QUEUE_SIZE` using the following code:

```
// C++
char* value = 0;
CORBA::Orbix.GetConfigValue("Orbix.IT_LISTEN_QUEUE_SIZE",value);
cout << endl << "Listen Queue size is " << value << endl;
// Caller is responsible for memory allocated
// in out parameter to GetConfigValue
//
delete[] value;
value = 0;
```

Multiple LOCATION_FORWARD

Synopsis

When an Orbix 3.3 C++ client attempts to connect to a server, it can deal with at most one `LOCATION_FORWARD` reply on a single request. In some cases, this limit might be exceeded when an Orbix 3.3 client attempts to connect to an Orbix E2A server.

An Orbix 3.3 Java client can deal with an infinite number of `LOCATION_FORWARD` replies on a single request.

Description

In a pure Orbix 3.3 environment the only time a `LOCATION_FORWARD` reply can be generated is when an Orbix 3.3 client contacts the Orbix daemon. In Orbix E2A, any server can generate a `LOCATION_FORWARD` reply. It is therefore possible that the limit of a single `LOCATION_FORWARD` could be exceeded when an Orbix 3.3 client attempts to connect to an Orbix E2A server.

Summary

Table 28 summarizes the handling of multiple `LOCATION_FORWARD` Reply messages.

Table 28: *Number of LOCATION_FORWARD Replies that Can Be Handled by Orbix Products*

Product	**Maximum Number of** `LOCATION_FORWARD` **Replies**
Orbix 3.3 C++ Edition	1
Orbix 3.3 Java Edition	*Infinity*
Orbix E2A C++ and Java	*Infinity*

CHAPTER 13

Codeset Negotiation

Codeset negotiation enables CORBA applications to agree on a common character set for transmission of narrow and wide characters.

In This Chapter

This chapter discusses the following topics:

Codeset Negotiation for Narrow and Wide Characters

Synopsis

Orbix 2000 (version 1.1 and later) and Orbix E2A support codeset negotiation, as defined by the CORBA 2.4 specification.

Neither Orbix 3.3 nor Orbix 2000 version 1.0 support codeset negotiation.

Description

The CORBA codeset conversion framework enables applications to ensure that they communicate using compatible character formats for both narrow characters, `char`, and wide characters, `wchar`.

Servers and Codeset Negotiation

A server that supports codeset negotiation appends a list of supported codesets (character formats) to the interoperable object references (IORs) it generates. The codesets are placed in standard `IOP::TAG_CODE_SETS` components in the IOR.

Clients and Codeset Negotiation

A client that supports codeset negotiation examines an IOR to check the list of codesets supported by the server. The client compares this list with its own list of supported codesets and, if a match is found, the client chooses the pair of transmission codesets (narrow character format and wide character format) to use for that particular connection.

When sending a `Request` message, the client appends an `IOP::CodeSets` service context that tells the server which codesets are used. The client continues to include an `IOP::CodeSets` service context in `Request` messages until the first `Reply` message is received from the server. Receipt of the first server `Reply` message implicitly indicates that codeset negotiation is complete. The same characters formats are used for subsequent communication on the connection.

Orbix E2A C++ Codesets

Table 29 shows the codesets supported by Orbix E2A C++.

Table 29: *Orbix E2A C++ Codesets*

Orbix E2A C++	Codeset
native codeset for char (NCS-C)	ISO-8859-1
conversion codesets for char (CCS-C)	*none*
native codeset for wchar (NCS-W)	UCS-2 or UCS-4
conversion codesets for wchar (CCS-W)	{UTF-16}

In Orbix E2A C++, the choice of native wide character codeset, UCS-2 or UCS-4, is based on the size of `CORBA::WChar` (either 2 or 4 bytes). On Windows UCS-2 is used and on most UNIX platforms UCS-4 is used.

Orbix E2A Java Codesets

Table shows the codesets supported by Orbix E2A Java.

Table 30: *Orbix E2A Java Codesets*

Orbix E2A Java	Codeset
native codeset for char (NCS-C)	ISO-8859-1
conversion codesets for char (CCS-C)	{UTF-8}
native codeset for wchar (NCS-W)	UTF-16
conversion codesets for wchar (CCS-W)	{}

Native codesets are used by the application to pass `char` and `wchar` data to the ORB.

Conversion codesets are used, where necessary, to facilitate interoperability with other ORBs or platforms.

Configuring Orbix E2A to Support Legacy Behavior

Default Behavior

By default the IOP::TAG_CODE_SETS tagged component is included in generated IORs and the transmission codesets are negotiated by clients and transmitted through an IOP::CodeSets service context. This is the CORBA defined behavior.

Legacy Behavior

Orbix E2A also provides legacy behavior, typically in the event that wide character data is communicated between Orbix E2A and Orbix 3.3 Java Edition.

Disabling Codeset Negotiation

The following configuration variable can be used to explicitly disable the codeset negotiation mechanism:

```
# Orbix E2A Configuration File
policies:giop:interop_policy:negotiate_transmission_codeset =
   "false";
```

The default is true.

This is a proprietary setting provided for interoperability with legacy implementations, such as Orbix 3.3 Java Edition. The native codeset for character data, ISO-8859-1 (Latin-1), is used and the overhead of full negotiation is avoided. In the event that wide character data is used, Orbix E2A reverts to the UTF-16 transmission codeset.

Enabling wchar Transmission on a GIOP 1.0 Connections

Passing wchar data over GIOP 1.0 can be enabled using the following configuration variable:

```
# Orbix E2A Configuration File
policies:giop:interop_policy:allow_wchar_types_in_1_0 = "true";
```

The default is false.

The transmission of wchar data is not legal in GIOP 1.0 by default.

Orbix E2A Codeset Negotiation Details

Codeset Negotiation Steps

The following steps are automatically performed by the ORB to negotiate codesets between a client and a server:

1 **Server Publishes IOR with Embedded Codesets**

An IOR is generated by the Orbix E2A server containing an embedded codeset that lists the character formats supported by the server.

2 **Client Reads IOR and Opens Connection to Server**

A client reads the IOR generated by the server in the previous step—typically, obtaining the IOR from the CORBA Naming Service.

Just before it opens a connection to the server, the client attempts to determine which character formats to use by looking for an embedded codeset in the IOR. What happens next depends on whether or not the client can find a codeset in the IOR:

- *Codeset not found*.

  The client reverts to using native format, ISO-8859-1, for narrow characters (char type). The format for wide characters is not defined at this point.

  Go to STEP 3.

- *Codeset found*.

  The client chooses a pair of character formats (for narrow and wide characters) by selecting the best match between its own codeset and the codeset advertised in the IOR. The choice of character formats is sent to the server by inserting an `IOP::CodeSets` service context into the initial GIOP `Request` message(s), until codeset negotiation is complete.

  This step presupposes that the connection being opened to the server is at least a GIOP 1.1 based connection—codeset negotiation is not supported prior to GIOP 1.1. If the IOR profile used to open the connection is IIOP version 1.1 or higher, and the client supports IIOP 1.1 or higher, full codeset negotiation is possible.

  Go to STEP 4.

3 **First Invocation Containing Wide Characters**

This step is performed only if codeset negotiation is not complete by the time the first invocation containing wide characters, `wchar`, is about to be made.

An Orbix E2A client considers codeset negotiation to be active if an `IOP::TAG_CODE_SETS` component is present in the IOR. If no such component exists and transmission of wide character data is subsequently attempted, the client raises a `CORBA::INV_OBJREF` exception.

If multiple IORs specify the same server address, the client can reuse the same connection to establish bindings for all of the IORs. Only the first IOR used on this connection is considered by the client when determining transmission codesets. Likewise on the server side, only the first codeset service context that is received by the server is considered when determining transmission codesets.

If an Orbix E2A server receives an invocation containing wide characters before codeset negotiation is complete, a `CORBA::BAD_PARAM` exception is raised.

4 **Subsequent Invocations**

Codeset negotiation is performed at most once for each client-server connection. When a pair of character formats has been agreed upon, the same character formats are used for all subsequent invocations on that connection.

Sample Codeset Conversion Scenarios

Orbix E2A C++ talking to Orbix E2A C++

When codeset negotiation is enabled, the following scenarios can occur involving the Windows and UNIX platforms. Table 31 shows the codesets used between an Orbix E2A C++ application on Windows and an Orbix E2A C++ application on UNIX.

Table 31: *Windows to UNIX Codesets*

Windows	On the Wire	UNIX
UCS-2	UTF-16	UCS-4

Table 32 shows the codesets used between two Orbix E2A C++ applications on Windows.

Table 32: *Windows to Windows Codesets*

Windows	On the Wire	Windows
UCS-2	UCS-2	UCS-2

Table 33 shows the codesets used between two Orbix E2A C++ applications on UNIX.

Table 33: *UNIX to UNIX Codesets*

UNIX	On the Wire	UNIX
UCS-4	UCS-4	UCS-4

Orbix E2A C++ talking to Orbix E2A Java

Table 34 shows the codesets used between an Orbix E2A C++ application (Windows or UNIX) and an Orbix E2A Java application when codeset negotiation is enabled.

Table 34: *Orbix E2A C++ Application to Orbix E2A Java Application*

Windows / UNIX	On the Wire	Java Platform
UCS-?	UTF-16	UTF-16

Using the Legacy Switch

When the Orbix E2A legacy switch is used to disable codeset negotiation, the UTF-16 codeset is used for all transmissions on the wire. Codeset negotiation is disabled by the following entry in the Orbix E2A configuration file:

```
# Orbix E2A Configuration File
policies:giop:interop_policy:negotiate_transmission_codeset =
    "false";
```

Table 35 shows the codesets used between an Orbix E2A C++ application (Windows or UNIX) and an Orbix E2A Java application when codeset negotiation is disabled.

Table 35: *Orbix E2A C++ Application to Orbix E2A Java Application*

Windows / UNIX	On the Wire	Java Platform
UCS-?	UTF-16	UTF-16

Table 36 shows the codesets used between an Orbix E2A Java application and an Orbix 3.3 Java application.

Table 36: *Orbix E2A Java Application to Orbix 3.3 Java Application*

Orbix E2A Java	On the Wire	Orbix 3.3 Java Edition
???-?	UTF-16	???-?

Index