FUJITSU

FUJITSU Software
Interstage Application Server

Distributed Application Development Guide (CORBA Service Edition)

Windows/Solaris/Linux

B1WS-1085-03ENZ0(00)
April 2014

Preface

Purpose of this Document

This document provides information on how to develop applications using Interstage.

Note

Throughout this manual Interstage Application Server is referred to as Interstage.

Intended Readers

This document is intended for developers of distributed applications.

It is assumed that readers of this manual have a basic knowledge of:

- C
- C++
- COBOL
- OOCOBOL
- Java
- The Internet
- Object-oriented technology
- Distributed object technology (CORBA)
- Relational databases
- Basic knowledge of the OS used

Structure of This Document

The structure of this manual is as follows:

Chapter 1 Basic Knowledge for Developing CORBA Applications

This chapter outlines the development of CORBA applications.

Chapter 2 Notes on Developing CORBA Applications

This chapter provides notes on how to develop CORBA applications.

Chapter 3 CORBA WorkUnits

This chapter gives directs the user to the location of information on CORBA WorkUnits.

Chapter 4 C Programming Guide

This chapter explains how to develop CORBA applications in C.

Chapter 5 C++ Programming Guide

This chapter explains how to develop CORBA applications in C++.

Chapter 6 Java Programming Guide

This chapter explains how to develop CORBA applications in Java.

Chapter 7 COBOL Programming Guide

This chapter explains how to develop CORBA applications in COBOL.

Chapter 8 Naming Service Programming

This chapter explains the API (Application Programming Interface) and the programming that the Naming Service provides.

Chapter 9 Interface Repository Service Programming

This chapter explains the API (Application Programming Interface) and the programming provided by the Interface Repository Service.

CChapter 10 CORBA Programming

This chapter explains the technique of developing the CORBA application.

Chapter 11 CORBA Interface

This chapter explains CORBA interface that is used when programming application to handle object dynamically.

Chapter 12 Obtaining Naming Service Initial References

This chapter explains how to obtain initial references for the Naming Service.

Appendix A IDL

This appendix explains the IDL.

Appendix B Programs Provided

This appendix explains the programs that are provided by the CORBA Service.

Appendix C Importing and Exporting Interface Definition Information

This appendix explains how to move to the interface repository of the operating server the interface definition information that is required when a dynamic activation interface is used.

Appendix D Collection of Maintenance Information

This appendix explains how to collect maintenance information that is provided by the CORBA Service.

Appendix E Sample Programs (Windows®)

This appendix describes the sample Windows® programs that are provided.

Appendix F Sample Programs (Solaris(TM) Operating System/Linux)

This appendix describes the sample Solaris and Linux programs that are provided.

Appendix G Dynamic Skeleton Interface: DSI

This appendix explains Dynamic Skeleton Interface (DSI).

Appendix H COM/CORBA Linkage Programming

This appendix explains how to develop an application to link a CORBA server to an OLE2 automation controller on a personal computer.

Appendix I Example of Session Management using the Object to Process Bind Function

This appendix explains how to implement session management for an IDL definition using the Object to Process Bind function.

Conventions

Representation of Platform-specific Information

In the manuals of this product, there are parts containing content that relates to all products that run on the supported platform. In this case, an icon indicating the product platform has been added to these parts if the content varies according to the product. For this reason, refer only to the information that applies to your situation.

Windows32	Indicates that this product (32-bit) is running on Windows.
Windows64	Indicates that this product (64-bit) is running on Windows.
Windows32/64	Indicates that this product (32/64-bit) is running on Windows.
Solaris32	Indicates that this product (32-bit) is running on Solaris.
Solaris64	Indicates that this product (64-bit) is running on Solaris.
Solaris32/64	Indicates that this product (32/64-bit) is running on Solaris.

Linux32	Indicates that this product (32-bit) is running on Linux.
Linux64	Indicates that this product (64-bit) is running on Linux.
Linux32/64	Indicates that this product (32/64-bit) is running on Linux.

Abbreviations

Read occurrences of the following Components as their corresponding Service.

Service	Component
CORBA Service	ObjectDirector
Component Transaction Service	TransactionDirector

Export Controls

Exportation/release of this document may require necessary procedures in accordance with the regulations of the Foreign Exchange and Foreign Trade Control Law of Japan and/or US export control laws.

Trademarks

Trademarks of other companies are used in this documentation only to identify particular products or systems.

Product Trademarks/Registered Trademarks
Microsoft, Active Directory, ActiveX, Excel, Internet Explorer, MS-DOS, MSDN, Visual Basic, Visual C++, Visual Studio, Windows, Windows NT, Windows Server, Win32 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
Oracle and Java are registered trademarks of Oracle and/or its affiliates.

Other company and product names in this documentation are trademarks or registered trademarks of their respective owners.

Copyrights

April 2014 Third Edition

November 2012 First Edition

Contents

Chapter 1 Basic Knowledge for Developing CORBA Applications

This chapter provides the basic knowledge required to develop CORBA applications.

Note

This manual explains how to develop CORBA applications using multiple development languages. For details on how to create Java EE applications, refer to the Java EE Operator's Guide.

1.1 Basic Models for CORBA Applications

There are two basic models which are applicable when designing an Interstage system for CORBA applications:

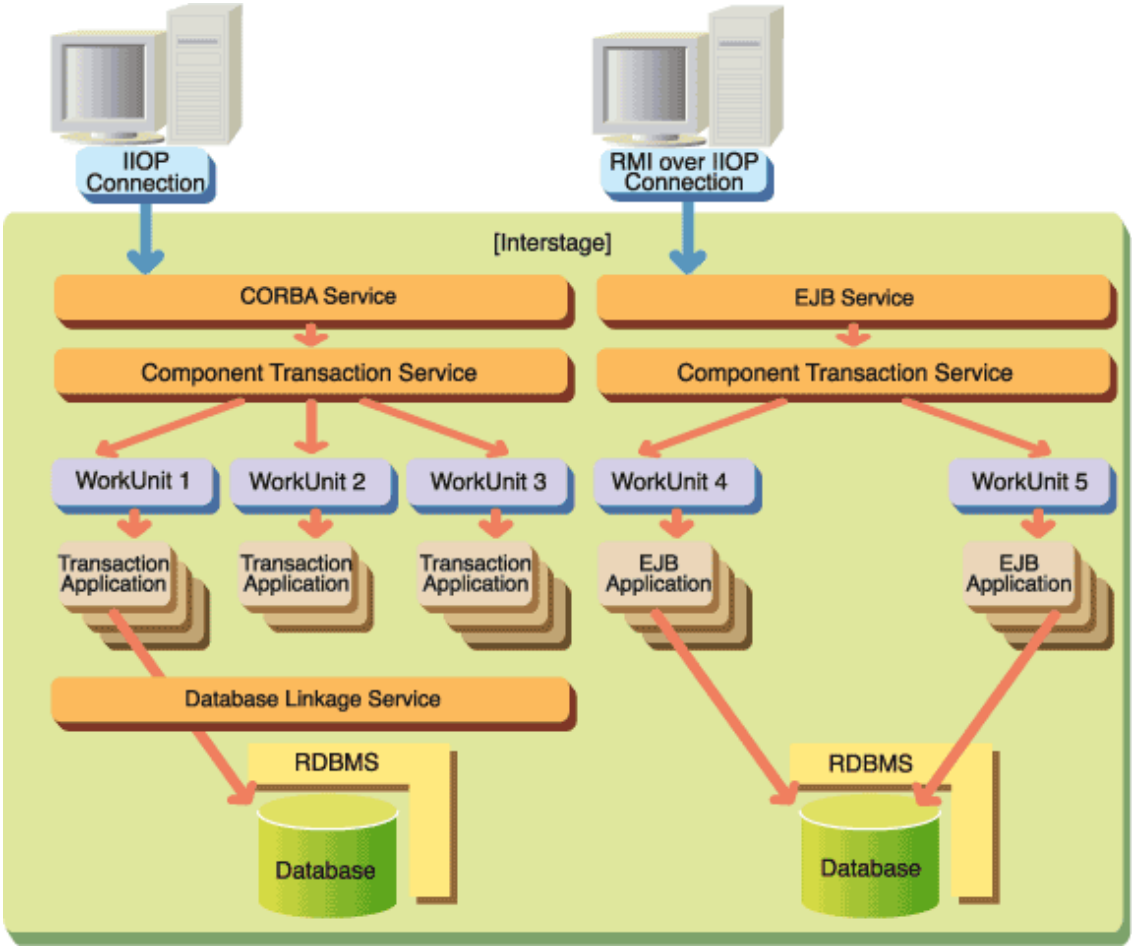
- 1.1.1 Two-layered Model
- 1.1.2 Three-layered Model

1.1.1 Two-layered Model

The two-layered model for clients and servers. The Object Request Broker (ORB) facilitates communication between client and server.

The following figure outlines the two-layered model.

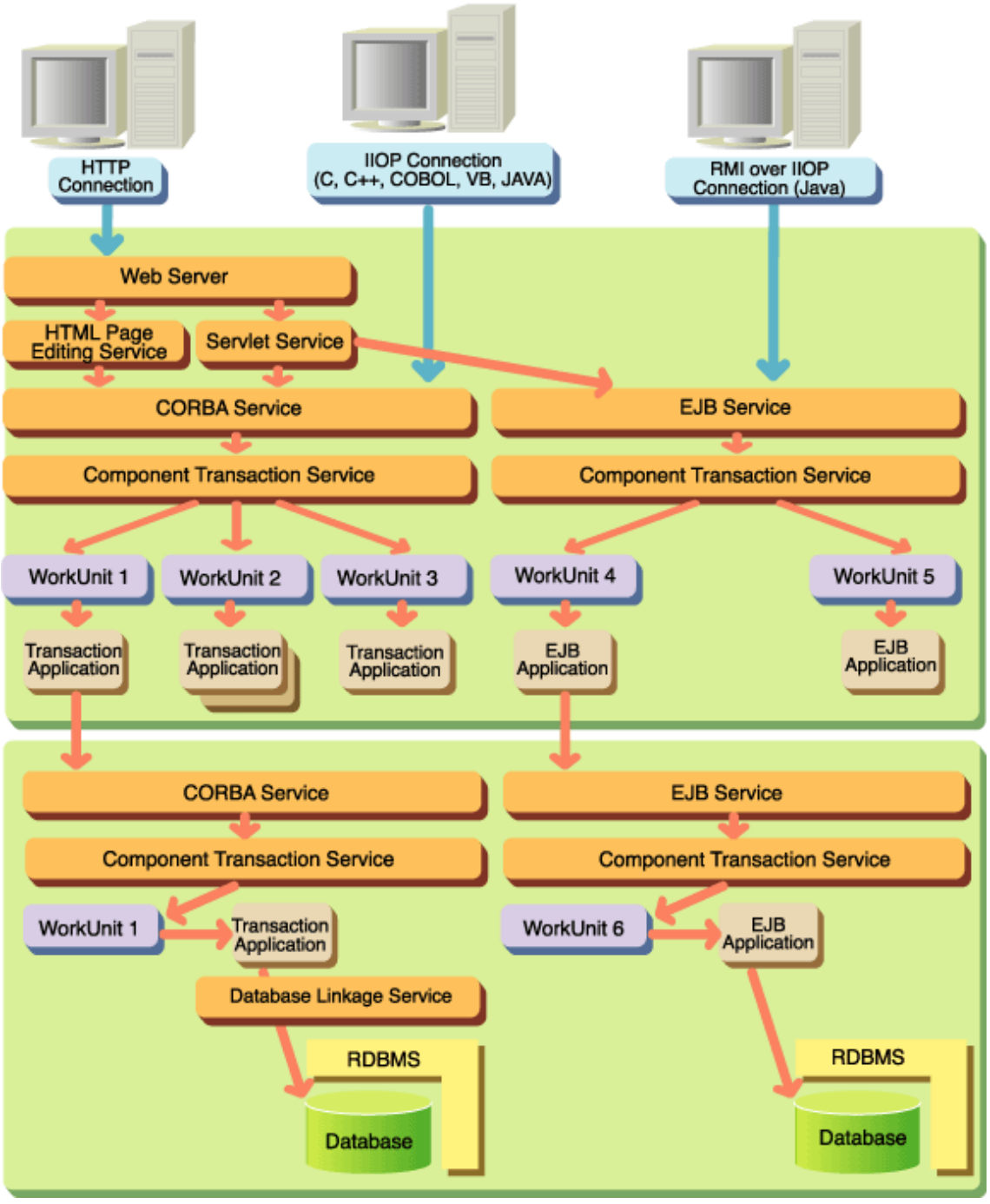
Figure 1.1 Two-layered Model



1.1.2 Three-layered Model

The three-layered model is developed by separating client servers and the server system into processing units such as application servers and database servers. That is, this system model is an extended version that separates the two-layered model servers. The following outlines the three-layered model.

Figure 1.2 Three-layered Model



Note

The HTML Page Editing Service can be used with the Windows(R) system and the Solaris(TM) Operating System.

In this document, Solaris(TM) Operating System is hereafter abbreviated as Solaris OS.

1.2 CORBA Service Operating Mode

This section describes the operating mode of CORBA applications.

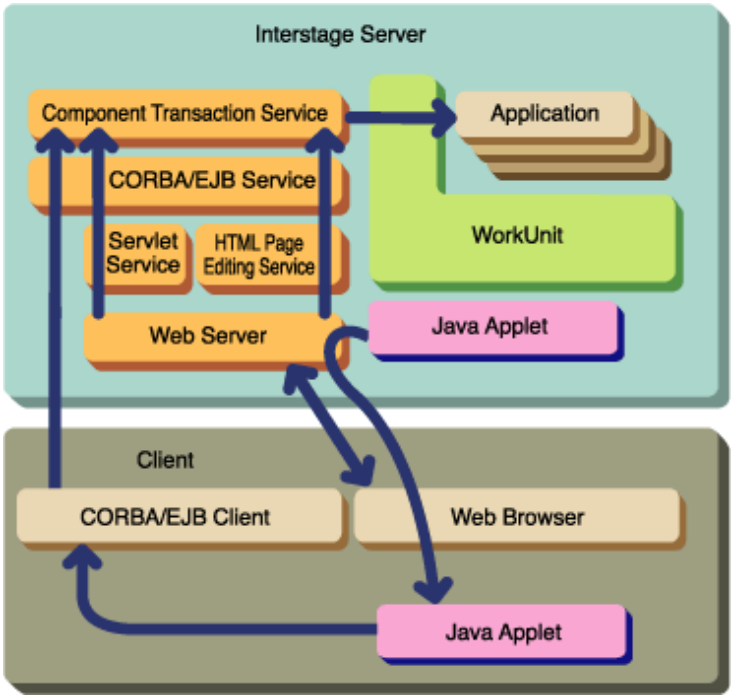
1.2.1 Operating Mode for Clients

There are two modes of client operation; the pre-installation type and the Portable-ORB type. When operating in a Java environment, you can use either type. This is not the case when a non-Java environment is used.

Pre-installation Mode (Available for all the Development Languages)

The Pre-installation mode installs the client runtime (a CORBA Service client) on the client machine prior to operation. Since pre-installation type has already been installed in the client machine, it doesn't need to download the client runtime when the application is run.

Figure 1.3 Pre-installation Operating Mode



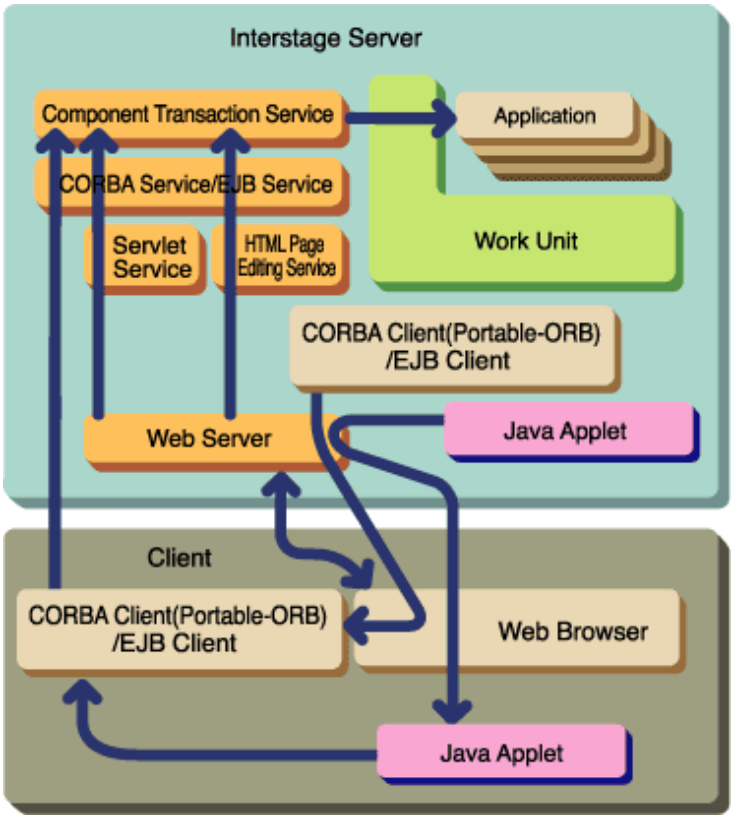
Note

The HTML Page Editing Service cannot be used with Linux systems.

Portable-ORB Mode (Available Only for Java)

In this operating mode, the client runtime (Portable-ORB) is installed on the client machine prior to operation, and the client runtime (Portable-ORB) is downloaded to server machine (the Web server) with the applet when the application is run. Because the Java runtime does not need to be installed on individual machines before operation in the Portable-ORB mode, the cost of application operation and maintenance can be reduced.

Figure 1.4 Portable-ORB Operating Mode



Note

The HTML Page Editing Service cannot be used with Linux systems.

1.2.2 Operating Mode for Servers

In server operations, the functions, including load balance for all systems other than Linux 64 bit, can be used by performing multi-server operation (that is, operation with more than one server). There are two methods of performing multi-server operation:

Server Object Mapping

Distribute server objects on multiple server machines. In this operation, if objects with high load are mapped on the same server machine, performance of the server machine may be affected on account of intense demand. Therefore, when designing a model, care must be taken to map objects so that load is equally distributed across servers.

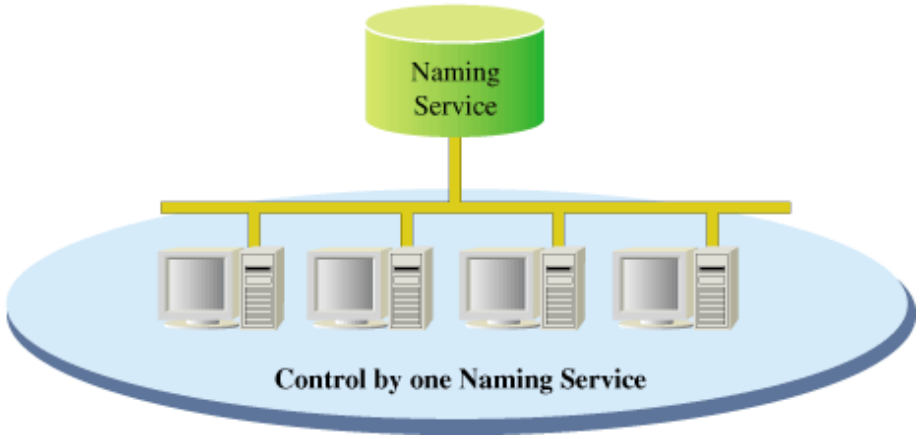
Naming Service Mapping

Distribute Naming Services on multiple server machines. In this operation, select the method appropriate to your system from the three mapping alternatives outlined below. Remember to take physical traffic and response due to load distribution and mapping into consideration when selecting a model.

Concentrated Type

A Naming Service controls all system objects. Because load is concentrated on one Naming Service, this model is suitable for small-scale systems.

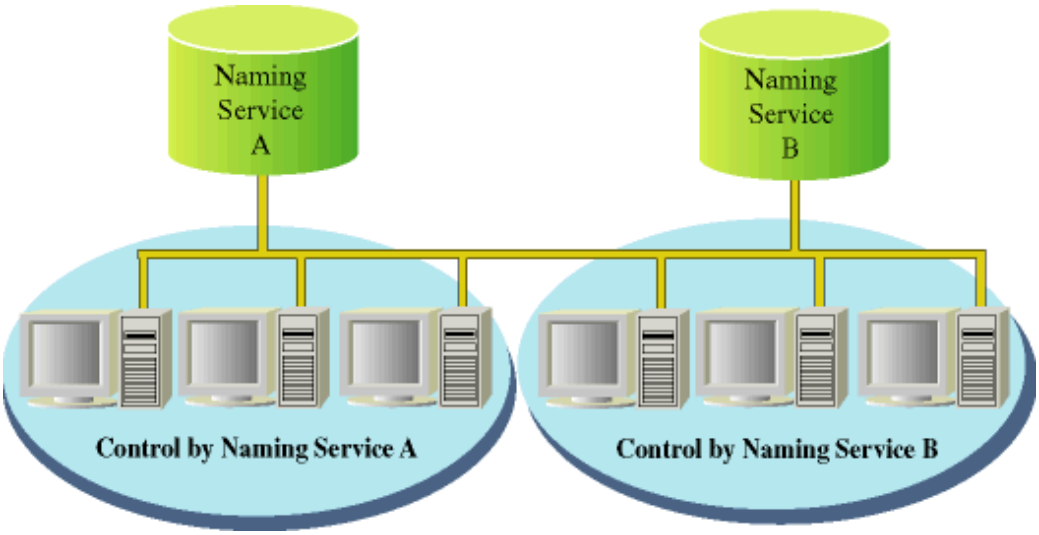
Figure 1.5 Concentrated Mapping Model
FOR A SMALL SCALE SYSTEM



Distributed Type

Multiple Naming Services control all system objects. Each Naming Service controls a group of objects, so that the load of access to a particular object is borne by a particular Naming Service. This model is suitable for medium-scale systems.

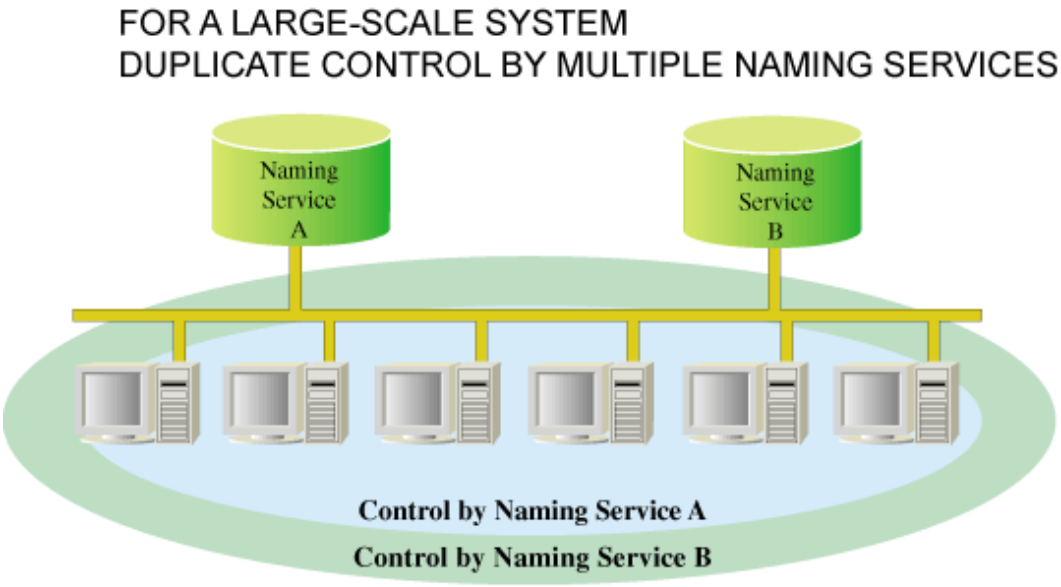
Figure 1.6 Distributed Mapping Model



Replication Type

Multiple Naming Services exists and controls all system objects. Each Naming Service can direct a request to any object. This model is suitable for large-scale systems with many clients.

Figure 1.7 Replication Mapping Model



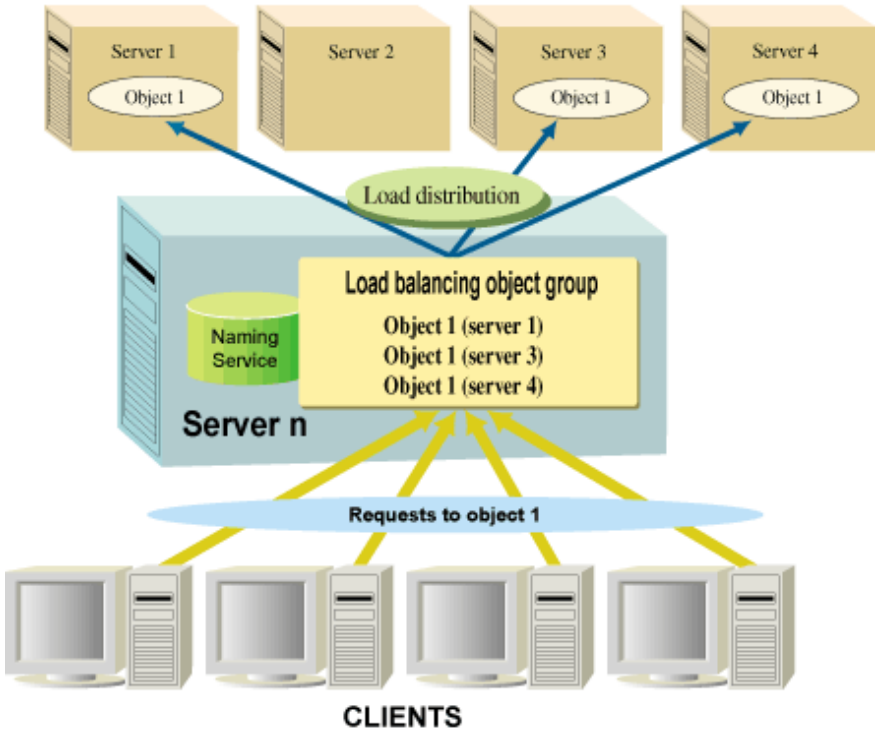
1.2.3 Operating Load Balance

This is not valid for Linux (64 bit).

Load balance provides the function to achieve load distribution in a level operating environment, linking with the Naming Service.

The load balance function maps multiple server objects (which have the same interface and provide the same functions) on multiple servers. If requests from clients are given to the objects, the load balance function adjusts the demand on these objects so that the load is equally distributed, and returns an Object Reference.

Figure 1.8 Naming Service used in the Load Balance Function



Load balance has the following two methods of operation, depending on the registration methods of the load balancing objects:

For Permanent Registration

Register the load balancing objects only once, when the system environment is developed. Use this way if the load balancing object is fixed.

When Registering Every Time the Server Object Starts

Register the load balancing objects every time a server object starts. Use this way if the load balancing object is variable.

1.3 Designing CORBA Applications

This section explains the requirements and limitations to be considered when designing a CORBA application.

1.3.1 Application Development Languages

When developing an application, use the following languages for both the client program and the server program:

- C
- C++
- Java
- COBOL

1.3.2 Server Application Startup Type

The processing of a server application is performed by selecting one of the following four startup types. Set the type when registering the application in the implementation repository, based on the system design.

shared

A server application processes more than one object at a time. The server objects process requests from more than one client.

unshared

A server application processes only one object. The server object processes requests from more than one client.

persistent

A server application processes more than one object at a time. The server objects process requests from more than one client.

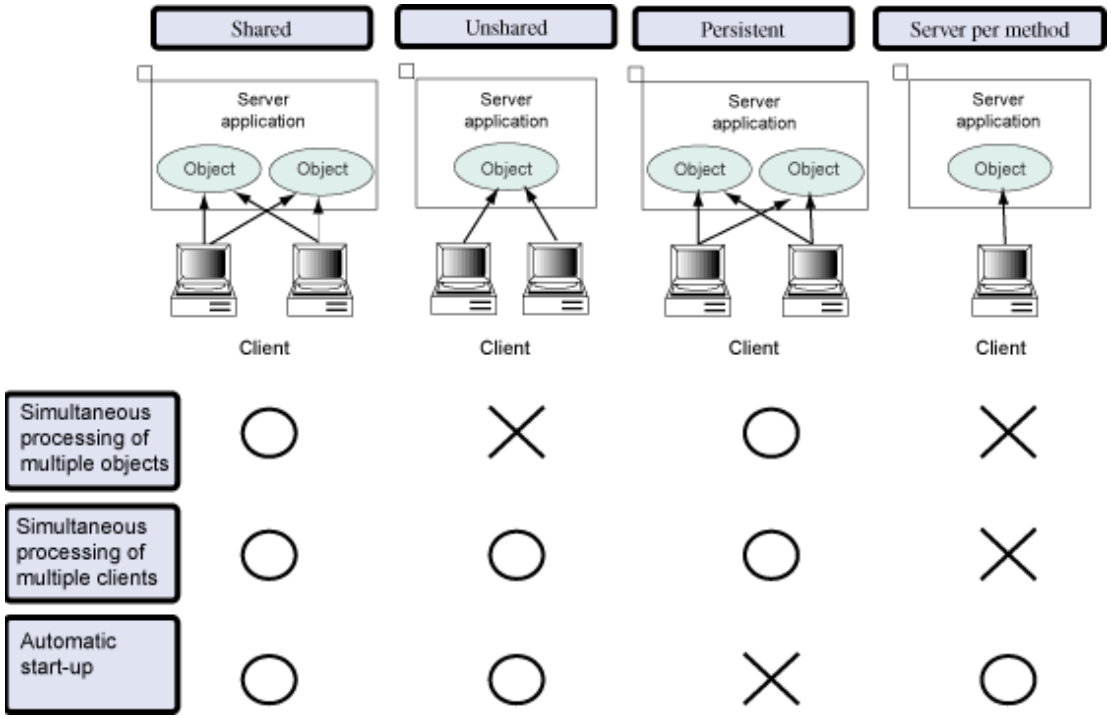
server per method

A server application processes only one object. The server object processes only one request from client.

When sharing the server application objects, specify "Shared" or "Persistent". When using the server application objects exclusively, specify "Unshared" or "Server per method". If a startup type other than "Persistent" is specified, the server application is automatically started when a request is issued, even if the application is not activated.

Select the type according to your object, referring to the following:

Figure 1.9 Server Application Startup Methods



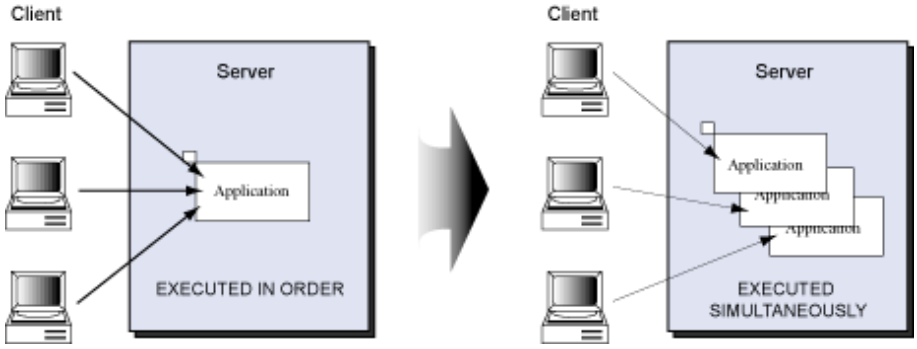
Note

Use of the persistent is recommended, unless there is a particular requirement for doing otherwise.

1.3.3 Concurrency Control

Using concurrency control, applications can process requests from multiple clients. Figure 1-11 illustrates the concurrency control operation.

Figure 1.10 Concurrency Control Operations



In the case of a CORBA application, concurrency control can be performed using the Implementation Repository.

For concurrency control operations on the same application, process and thread are available. By specifying maximum process concurrency and initial thread concurrency, requests from multiple clients can be processed.

Whether the concurrency can be set or not depends on the startup type of the server application, as illustrated in the following table.

Table 1.1 Server Application Startup and Multiplexing

Startup type of server application	Process multiplexing (Single)	Process multiplexing (Multi)	Thread multiplexing
shared	Can be specified	Cannot be specified	Can be specified
unshared	Can be specified	Cannot be specified	Can be specified
persistent	Can be specified	Can be specified	Can be specified
server per method	Can be specified	Cannot be specified	Cannot be specified

1.3.4 Instance Management System

Using the instance management function, the operation range of objects in each client application can be managed. Implement the Factory interface in the server application to dynamically generate the objects corresponding to each client application. This function is supported in C++ and in Java. For details, refer to the "CORBA Programming" chapter.

1.3.5 Client Application Interface

The client application can be categorized into two types of interfaces, depending on the server application invocation method. The two types are the static invocation interface and the dynamic invocation interface. Differences between the client and server side interfaces are not a problem.

Static Invocation Interface

The client application obtains interface information from the stub. Since a server object is identified at compilation, the interface can be created with no complication, thus achieving good performance.

Dynamic Invocation Interface

The client application obtains interface information dynamically from the Interface Repository. Since the client application identifies the server object at linkage, the client application can receive the latest information of the server object. Also, changes to operations can be made without disturbing the client. In this way, flexible system development can be achieved.

Note

Depending on the IDL definition file configuration, an area allocation variable and other entities specific to the interface may be defined in the stub. In this case, the application must be linked with the stub during compilation.

1.3.6 Server Application Interface

The server application can be categorized into two types of interfaces, depending on the method of opening interface information to the client. These types are static skeleton interfaces and dynamic skeleton interfaces. Differences between the client and server side interfaces are not a problem.

Static Skeleton Interface

Opens an interface via a skeleton. Linkage to the skeleton is generated from the IDL definition file and application fixes to the server object and server operations. This enables the simple creation of the interface and good performance.

Dynamic Skeleton Interface

Opens an interface via the Dynamic Skeleton Interface (DSI) function. When skeleton is not used and the server application is started, the DSI function is posted to ORB. This interface can dynamically decide to which server object the request is sent, corresponding to the client's request. This helps to achieve a flexible system.

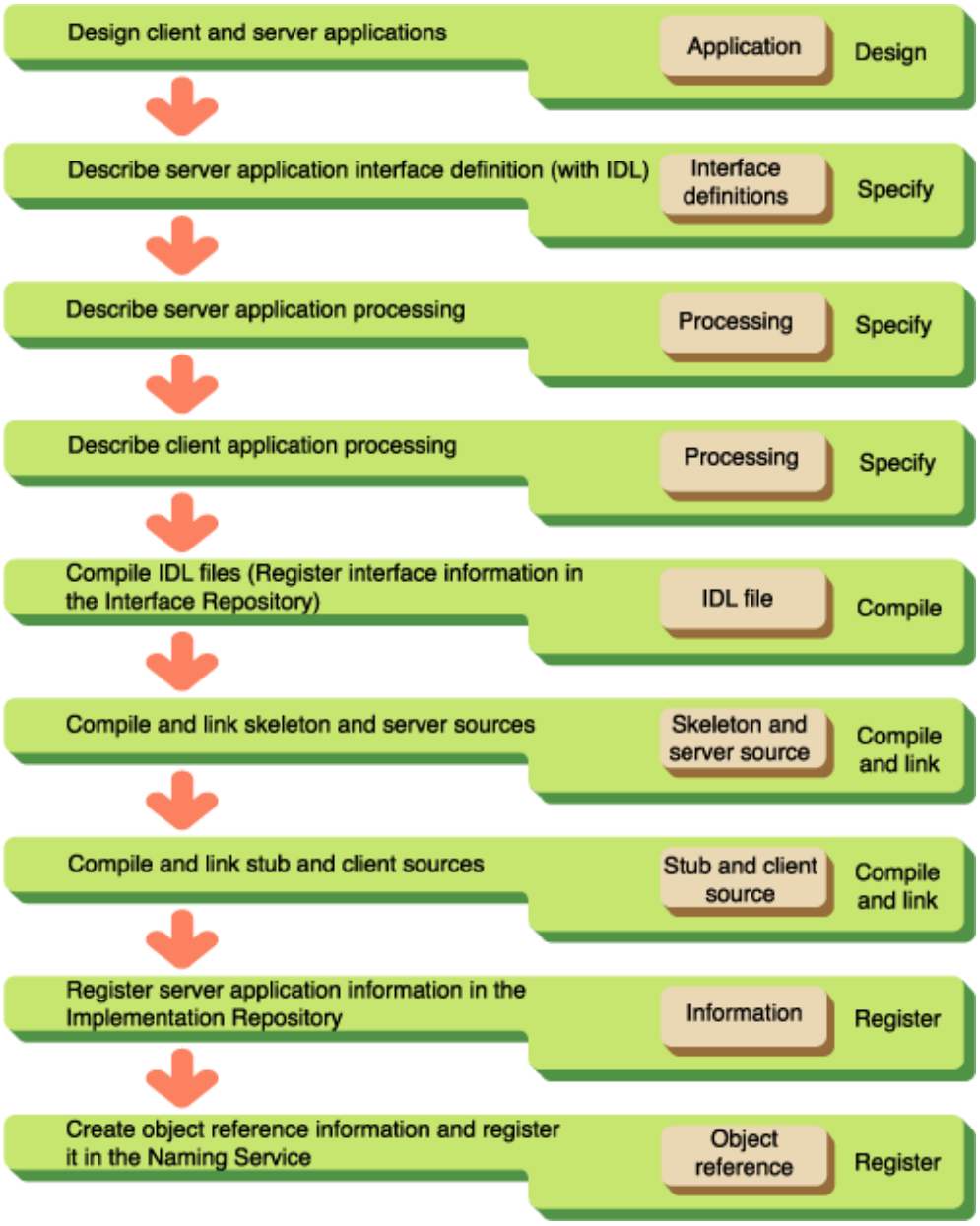
Note

Depending on the IDL definition file configuration, an area allocation variable and other entities specific to the interface may be defined in the skeleton. In this case, the application must be linked with the skeleton during compilation.

1.4 CORBA Application Development Procedure

The procedure for developing CORBA applications is outlined in the following figure.

Figure 1.11 CORBA Application Development Procedure



For details on the development procedure, refer to the following table for information specific to the development language.

Table 1.2 CORBA Development Procedure Language Reference

Development languages	Reference
C	C Programming Guide
C++	C++ Programming Guide
Java	Java Programming Guide
COBOL	COBOL Programming Guide

Chapter 2 Notes on Developing CORBA Applications

This chapter provides notes on how to develop CORBA applications.

Note

This manual explains how to develop CORBA applications using multiple development languages. For details on how to create Java EE applications, refer to the Java EE Operator's Guide.

2.1 Coding

This section explains the following aspects of coding:

Note

Windows64

- COBOL cannot be used.

Linux64

- COBOL can only be used in a Windows(R) client.

2.1.1 Data Area Allocation/Release Processing (C, C++, and COBOL)

To use the data types given below, the dedicated function for each data type must be used to perform area allocation. If the dedicated function is not used for the allocation, a memory access error may occur during application operation. When an area allocated dynamically at a client or stub is no longer required, an explicit release operation performed by the releasing function or method for each data type is required. Failure to perform releases will cause memory leakage.

- Character string type, wide character-string type, any type, sequence type, structure type, union type, and array.

  To release the data area together for the any and sequence types, the dedicated function must be used to set the release flag in advance. Unless the flag is explicitly set, the data area is not released by default; therefore, memory leakage may occur. For information about the release flag, refer to "Any Type" and "Sequence Type" in "Mapping Data Types" in the programming guide chapters ("C Programming Guide", "C++ Programming Guide", "Java Programming Guide" and "COBOL Programming Guide").

  When the data types given below are used, the NULL pointer cannot be used as the return value for the in and inout parameters for client applications, and the out and inout parameters for server applications. If the pointer is mistakenly set as the return value, a memory access error may occur.

- Character string type, sequence type, structure type, union type, and array

  In COBOL applications, use a dedicated API to set and retrieve data of the types given below. Refer to the corresponding description in the Reference Manual (API Edition) for details on each API.

Table 2.1 Data Setting API and Data Retrieving API

Data type	Data setting API	Data retrieving API
Character string type	CORBA-STRING-SET	CORBA-STRING-GET
Wide character-string type	CORBA-WSTRING-SET	CORBA-WSTRING-GET
Sequence type	CORBA-SEQUENCE-ELEMENT-SET	CORBA-SEQUENCE-ELEMENT-GET

See

- "Mapping Data Types" in the "C Programming Guide" chapter
- "Mapping Data Types" in the "C++ Programming Guide" chapter
- "Mapping Data Types" in the "COBOL Programming Guide" chapter

2.1.2 Signal Processing (C, C++, Java, and COBOL)

Solaris32/64

In a C language CORBA application, SIGCHLD signals can be received. In other languages, an application to receive signals cannot be created.

In the signal handler, only the "Asynchronous signals safe" (Async-Signal-Safe) function can be used. Refer to "attribute(5)" in the Solaris documentation for details on the Async-Signal-Safe function. Note that the function does permit the user to invoke standard C library functions such as printf, malloc, and free and CORBA APIs.

Linux32/64

Applications to receive signals cannot be created.

If a signal is received by several OS functions, EINTR is displayed in the error information and the functions fail. In the CORBA library, the OS function is reissued when the OS function fails and error information is EINTR. If the OS itself is changed, it might cause the OS function behavior to change when the signal is received. In this case, it might not be able to use the CORBA library control correctly when the signal is received.

2.1.3 Child Process/Thread Creation Exit Processing (C and C++)

Solaris32/64 Linux32/64

To terminate a child process created by a CORBA application, use _exit(), not exit(). If exit() is used, when the child terminates, the parent process is also mistaken as terminated and subsequent operation is not guaranteed.

When the runtime of another language (such as COBOL) is called, exit() may be used by a function/API of that language to close the process. Do not use the functions/API of another language to end a child process if the process image has not been replaced with a new process image using the 'exec' family function.

To use the CORBA function in the child process created by the CORBA application, change the process image using the 'exec' type function, issue the CORBA_ORB_init function (in C++ this is the CORBA::ORB::init function), and then perform initialization processing.

Solaris32/64

To create a child process from the process-mode server application, use fork(). For the thread-mode server application, use not fork() but fork1(). To further create a thread from the thread-mode server application, specify the THR_NEW_LWP and THR_BOUND flags as thr_create() arguments.

Windows32/64

When creating a child process after the CORBA_ORB_init function (or the CORBA::ORB::init function in C++) is invoked, do not use a _spawn function (e.g., _spawn or _spawnl). _spawn functions are designed so that the created child process will inherit the handle and other information from the parent process. If a _spawn function is used, the system becomes unable to appropriately control the CORBA service.

To create a process after ORB initialization, use the CreateProcess function.

Solaris32/64 Linux32/64

When the server application is running in SYNC_END mode, the request processing thread will not process the request after the CORBA_BOA_impl_is_ready function (or the CORBA::BOA::impl_is_ready function in C++) returns, but this does not guarantee that the thread is not still being processed by the operating system.

As a result, if functions in a library embedded dynamically by the dlopen function are registered in the end-of-thread destructor function by functions such as pthread_key_create, and then a library embedded dynamically after the CORBA_BOA_impl_is_ready function returns is released by the dlclose function, processing may end abnormally because the request processing thread tries to issue functions of the released library.

To avoid this problem, do not use the dlclose function to release a library containing the destructor function.

Linux32/64

If pthread_exit() is invoked from the initialization thread to close the thread, <defunct> is displayed when the ps command (from the Linux specification) is used to check process information. Also, it is not possible to get process information using the gcore and strace commands.

It is difficult to investigate the problem if these commands cannot be used to get process information. It is recommended that applications be developed which do not close the initialization thread. For example, for a server application in which operation mode is set to 'COMPATIBLE' in an implementation configuration, generally there is no need for an initialization thread following activation of the server application. In this case, set the operation mode to 'SYNC_END', and ensure that the process is closed when the activation function returns, or that there is an infinite sleep() loop in the initialization thread and the pthread_exit() is not invoked.

Resource before correction (Operation mode: COMPATIBLE)

```
int main( int argc, char *argv[] ){
    /* ORB initialization processing */
    CORBA_BOA_impl_is_ready(...); /* The activation function (arguments are omitted) */

    pthread_exit(NULL);
}
```

Resource after correction if SYNC_END is set for the operation mode

```
int main( int argc, char *argv[] ){
    /* ORB initialization processing */
    CORBA_BOA_impl_is_ready(...); /* The activation function (arguments are omitted) */

    return 0;
}
```

Resource after correction if the operation mode is not changed from COMPATIBLE

```
int main( int argc, char *argv[] ){
    /* ORB initialization processing */
    CORBA_BOA_impl_is_ready(...); /* The activation function (arguments are omitted) */

    while ( 1 ){
        sleep( 1000 );
    }
}
```

2.1.4 ORB Initialization Processing (C, C++, and COBOL)

As arguments to be passed to the ORB initialization function (CORBA_ORB_init for the C language), use argc and argv. They are to be passed to the main function unchanged.

Initialization Method

Windows32/64

Application	Initialization method
Console application	Pass the arguments argc and argv of the main() function to CORBA_ORB_init(). For details, refer to the sample program (for C, refer to simple_s.c, which is the source for simple_s).
Windows(R) application	Convert data to the format to be passed from the third argument of the WinMain() function to CORBA_ORB_init(). Refer to the program source msg_s.c in the sample program msg_s for details. Refer to arg.c for details on the data conversion routine get_arg().

Solaris32/64 Linux32/64

Application	Initialization method
System application	Pass arguments argc and argv of the main() function to CORBA_ORB_INIT(). Refer to the source simple_s.c of the sample program simple for details.
Windows(R) application	Convert data to the format to be passed from the third argument of the WinMain() function to CORBA_ORB_init().

In a Windows(R) application, CORBA_ORB_init() may be invoked from a location other than the EXE part or from a DLL to be invoked from Visual Basic. In this case, execute load processing for ODWIN.DLL in advance in the EXE part or Visual Basic processing and add 1 to the ODWIN.DLL load counter. When the application ends, the load counter incremented by 1 must be held.

In Windows(R) applications, CORBA_ORB_init() and other CORBA functions must not be issued in the DllMain function.

Invoke the ORB initialization function only once in a process using a CORBA function.

In the client application, do not invoke a server method before the ORB initialization function ends normally. If the server method is invoked before the ORB initialization processing is completed, the exception "UNKNOWN" is thrown.

2.1.5 oneway

If oneway is specified in the IDL definition operation declaration, the server does not notify the client of the execution result for the called method. If method call is used when oneway is specified, note the following points:

- There is no guarantee that requests will be processed in the server application.

  If there are insufficient resources in the server, or a network abnormality occurs, requests are not processed in the server application. At this time, the client cannot detect abnormalities.

- An exception might be notified to the oneway method because of a network abnormality.

  The abnormality that occurred in the server cannot be detected in the oneway method because receive processing is not performed. If an abnormality that occurs in request send processing is detected, however, the exception is also notified to the oneway method.

- There is a possibility that a large amount of server resources have been used up.

  The oneway method does not wait until the completion of processing in the server, so it might cause a large amount of server resources to be used up if the method is issued repeatedly.

2.1.6 Server Application Exception Processing (C, C++, and COBOL)

Do not use the CORBA_Environment structure (in C language) specified as an argument of a server method as that of another method to be invoked from that server method. Otherwise, the exception information of the server method is set to exception information generated in another method, and therefore the exception information of the original invoking method cannot be retrieved. To invoke another method from the server method, use another CORBA_Environment structure.

2.1.7 Server Application Exception Processing (C)

Linux64 Windows64

Do not use the CORBA_Environment structure specified as an argument of a server method as that of another method to be invoked from that server method. Otherwise, the exception information of the server method is set to exception information generated in another method, and therefore the exception information of the original invoking method cannot be retrieved. To invoke another method from the server method, use another CORBA_Environment structure.

2.1.8 Simultaneous Implementation of Server and Client Functions

When server and client functions are implemented in the application, the same function name may be created in a skeleton to be created from the IDL file of the server function and in a stub to be created from the IDL file of the client function. If these names are connected, a double definition is assumed; therefore, create the IDL file carefully.

2.1.9 Miscellaneous

For logical user mistakes in past instances where an application was put into no-response state, the following two causes can be assumed:

- The application stopped while processing sleep().
- Processing of the OS was slowed down because of a message queue (IPC resource) shortage.
- If the CORBA library (libOM.so for the Solaris/Linux version or odsv.dll/odwin.dll for the Windows version) is dynamically included into the dlopen function (or the LoadLibrary function for Windows) program, do not use the dlclose function (or the FreeLibrary function for Windows) to release the library.
- When CORBA's Java interface is used, do not use the Java method finalize for CORBA communication. Otherwise, a COMM_FAILURE exception may occur.
- In the pre-installed Java and the Java server, the C heap area that the CORBA Service acquires when the request is sent and received is released by the FullGC processing. Therefore, the interval when FullGC is processed becomes long, the C heap size is insufficient, and the memory may be insufficient, when a large size is specified for the Java heap. In the CORBA-Java application, call System.gc() according to the following timing, and process FullGC to prevent memory shortage.
  - After receiving the reply from the server
  - Fixed time

2.2 Compilation and Linkage

This section explains the following topics:

2.2.1 Thread Mode and Process Mode

Solaris32/64 Linux32/64

There are two modes for CORBA applications: thread and process mode. Note that the linkage method and required libraries differ with the mode.

Thread Mode

Mode in which an application process operates in multithread operation. Select this type for ordinary cases. Be sure to select this type when a library of multiple components to be invoked from the application is provided for multithreading.

To create a COBOL application for implementing intrinsic multithread operation, "PowerCOBOL97," which was explicitly designed for multithread operation, must be made available.

Set thr_conc_maximum in the definition file to 2 or a larger value. The following shows a sample definition file.

```
rep_id          = IDL:test1/intf1:1.0
type            = persistent
```

```
proc_conc_max  = 1
thr_conc_init  = 16
```

Note

Solaris32/64 Linux32/64

1. A server application connected to a library for process mode operates in process mode regardless of the above setting.
2. As the library of multiple components (language runtime and DBMS) to be linked, specify a thread-safe library supporting multithreading. If a thread-unsafe library not supporting multithreading is mistakenly linked, multithread operation cannot be guaranteed for the entire application process. Therefore, application operation becomes uncertain (it could run normally by chance, end abnormally, or suddenly malfunction). When the application is compiled or linked, extreme care must be taken to keep thread-safe and thread-unsafe libraries from being linked together. Use the *ldd* command to check the state of application linkage.
3. To run a thread-mode application, the environment variable LD_LIBRARY_PATH must not be set to /opt/FSUNod/lib/nt (for Solaris). Otherwise, the application may malfunction. To run process-mode and thread-mode applications together, verify that the environment variable LD_LIBRARY_PATH is not set to /opt/FSUNod/lib/nt (for Solaris) when the thread-mode application starts.

Process Mode

Mode in which an application process operates in a single thread. Select this type only when a library of multiple components to be invoked from the application is not provided for multithread operation. In ordinary cases, select the thread mode.

In the definition file, set thr_conc_maximum to 1 and proc_conc_max to 2 or more. The following shows a sample definition file. In process mode, the operation mode of CORBA_BOA_impl_is_ready() is SYNC_END regardless of the mode setting.

```
rep_id         = IDL:test1/intf1:1.0
type           = persistent
proc_conc_max  = 8
thr_conc_init  = 1
```

Solaris32/64 Linux32/64

Note

1. If a thread library is mistakenly specified at application linkage, the application may malfunction. Use the *ldd* command to check the application link state.
2. To operate a process-mode application, the environment variable LD_LIBRARY_PATH must have been set to /opt/FSUNod/lib/nt (for Solaris). Otherwise, the application may malfunction.

   - bsh system

     LD_LIBRARY_PATH = /opt/FSUNod/lib/nt:$LD_LIBRARY_PATH

     export LD_LIBRARY_PATH

   - ssh system

     setenv LD_LIBRARY_PATH /opt/FSUNod/lib/nt:$LD_LIBRARY_PATH

2.2.2 Common Notes on Creating CORBA Server Applications

Solaris32/64 Linux32/64

A CORBA application must be created as an executable file or shared library (object) that can dynamically be linked. At linkage, do not add the static link option (the -dn option for Solaris).

To link the application, all libraries required for the application, such as the library of multiple components (DMBS), must be linked. The storage destination for the linked library group must be set in the environment variable LD_LIBRARY_PATH for application execution. If any of the required libraries are not linked or any of the LD_LIBRARY_PATH settings are missing, an exception such as BAD_OPERATION is thrown.

To operate a CORBA server application using a dynamic skeleton interface, the interface information must be stored in the interface repository. If, during client-server communication, a mismatch occurs between the information stored in the interface repository and the information in the stub created by the IDLc command, a malfunction as explained below occurs.

- Request data that the server application received from the client application becomes invalid.
- A memory shortage occurs.
- A processing request returns no response.

To perform the operation, pre-check for a mismatch between the information stored in the interface repository and the information in the stub. Use the odlistir command to check the information stored in the interface repository. A function to check the interface information to automatically detect any mismatch is provided (this function is not enabled by default). To prevent the above problems, you should apply this function. Refer to "Operation Using a Function to Check the Interface Information" in the OLTP Server Operations Guide for details on this function.

2.2.3 Notes on Windows® Applications

Notes on Compiling Options

When compiling programs using Visual C++(R), use the following procedure to specify the options shown in the following table.

Microsoft(R) Visual C++(R) .NET or Microsoft(R) Visual Studio(R) 2005

Select Project | Properties | Configuration Properties | C/C++.

Table 2.2 Visual C++ Compilation Options

Category	Item	Setting (32 bit)	Setting (64 bit)
Code generation	CPU structure member alignment	4 bytes	8 bytes
	Runtime to be used	Multi-Threaded DLL	
Optimization		Default recommended	
Preprocessor	Preprocessor definitions	Add "OM_PC", "OM_WIN32_BUILD", and "_STDC_"(*1)	
Language	Treat wchar_t as built in type	No	
Details	Call rules	_cdecl	

Windows64

Microsoft Platform SDK for Windows Server 2003:

Add the following option: (*1)

- /D "OM_PC" /D "OM_WIN32_BUILD" /D "__STDC__" /MD /Zp8

1. If "__STDC__" is defined, an error may occur during compilation of source code that relies on Microsoft extensions. In this case, take the following steps:
2. Separate the source code for CORBA application processing from that which relies on Microsoft extensions.
3. Insert "#undef __STDC__" at the beginning of each file containing source code that relies on Microsoft extensions.

Configure the settings for the libraries to be linked as follows:

Microsoft(R) Visual C++(R) .NET or Microsoft(R) Visual Studio(R) 2005

Select Project | Properties | Configuration Properties | Linker | Input.

For details of the libraries to be linked, refer to "Programs Provided by the CORBA Service", "Libraries" in the "Programs Provided" appendix.

Register the INCLUDE and LIB directories located under the installation directory in Include and Library respectively, under Tools | Options | Directory.

Registration Examples Windows32

- Include Files

```
C:\MSDEV\INCLUDE;C:\MSDEV\MFC\INCLUDE;C:\Interstage\ODWIN\INCLUDE;
C:\Interstage\TD\INCLUDE;C:\Interstage\ots\INCLUDE;
C:\Interstage\ESWIN\INCLUDE
```

(when Visual C++ and Interstage are installed in C:\MSDEV and C:\Interstage respectively)

- Library Files

```
C:\MSDEV\LIB;C:\MSDEV\MFC\LIB;C:\Interstage\ots\LIB;
C:\Interstage\ODWIN\LIB;C:\Interstage\EXTP\LIB;C:\Interstage\TD\LIB;
C:\Interstage\ESWIN\LIB
```

(when Visual C++ and Interstage are installed in C:\MSDEV and C:\Interstage, respectively)

Registration Examples Windows64

Include Files

```
C:\MSDEV\INCLUDE;C:\MSDEV\MFC\INCLUDE;C:\Interstage\ODWIN\INCLUDE;
```

(when Visual C++ and Interstage are installed in C:\MSDEV and C:\Interstage respectively)

Library Files

```
C:\MSDEV\LIB\amd64;C:\MSDEV\MFC\LIB\amd64;C:\Interstage\ODWIN\LIB\x64
```

(when Visual C++ and Interstage are installed in C:\MSDEV and C:\Interstage, respectively)

Errors during Compilation

(a) Compile Errors

If a C++ application is compiled using Microsoft(R) Visual Studio(R) 2005, the following error message may be output and the compile may fail.

```
error C4980: '__value' : To use this keyword, the /clr:oldSyntax command line option is required
```

In this case, it is necessary to change the IDL file variable name. In the above example, change the "value" variable in the IDL file to another name.

The following is an example of changing the variable name.

IDL file before correction

```
module ODsample{
    interface intf{
        readonly attribute long value;
        ...
    };
};
```

IDL file after correction

```
module ODsample{
    interface intf{
        readonly attribute long value1;    /* value -> value1 */
       ...
    };
};
```

(b) Link Errors

When using ODSV.LIB or ODWIN.LIB to compile programs in Visual C++, the following error message may be displayed.

```
Linking...
odsv.lib(ODSV.dll) : error LNK2005: _CORBA_sequence_string_alloc has already been defined in
simple_skel.obj.
odsv.lib(ODSV.dll) : warning LNK4006:  _CORBA_sequence_string_alloc has already been defined in
simple_skel.obj. The second and subsequent definitions will be disregarded. _
```

If this error message appears, change the following settings in Visual C++, then re-execute build.

Microsoft(R) Visual C++(R) .NET or Microsoft(R) Visual Studio(R) 2005

Select Project | Properties | Configuration Properties | Linker | Command Line | Additional Options, then add the /FORCE option.

When ODSV.LIB or ODWIN.LOB is used and the build is executed in Visual C++, the following warning message may be displayed, but operations are not affected:

```
C:\Interstage\odwin\lib\odwin.lib : warning LNK4064: conflicting subsystem;
    image may not run.
```

Converting Applications into a DLL

This section describes how to create a DLL using files generated by an IDL compiler.

(a) IDL Compilation

If the source for generating a DLL is to be created, specify the -dy option before performing IDL compilation.

(b)Preprocessor Definition

Create preprocessor definitions in Visual C++ separately when creating and linking a DLL. Refer to the following table for details.

Microsoft(R) Visual C++(R) .NET or Microsoft(R) Visual Studio(R) 2005

Project | Properties | Configuration Properties | C/C++ | Preprocessor | Preprocessor Definitions

Table 2.3 DLL Preprocessor Definitions

Language	DLL Creation Linkage	Preprocessor Definition
C	At DLL creation	"OM_USRDLL_MAKE"
C	At DLL linkage	"OM_USRDLL_LINK"
C++	At skeleton DLL creation (Note 1)	"OM_SKDLL_MAKE"
C++	At skeleton DLL linkage	"OM_SKDLL_LINK"
C++	At stub DLL creation (Note 2)	"OM_STDLL_MAKE"
C++	At stub DLL linkage	"OM_STDLL_LINK"

Notes

1. When creating a DLL from a skeleton file and method implementation file
2. When creating a DLL from a stub file

(c)Precautions

As well as defining a preprocessor by setting a project as above, a preprocessor can also be specified in the #define statement in the source file in which IDL compiler-generated header files (*.h) are to be included.

When a DLL is to be created and linked using header files generated from multiple IDL files, specify as shown below.

Example

```
   #define OM_USRDLL_LINK
   #include "a.h" // IDL compiler-generated header
   #undef OM_USRDLL_LINK

   #define OM_USRDLL_MAKE
   #include "b.h" // IDL compiler-generated header
   #undef OM_USRDLL_MAKE
```

```
    // The following is source code.
    ...
```

Linking Libraries

Link either a server library or a client library in one application.

2.2.4 Notes on Solaris Applications

Solaris64

When compiling and linking the application, specify the option that corresponds to the 64-bit version. This option will vary depending on the compiler version used, so check the compiler manual.

2.2.5 Notes on Linux Applications

All applications must be created on an OS of the same distribution and version as the OS where that application is executed. When the OS version is upgraded, the application must be rebuilt in the OS environment of the new version.

To create an application using Linux for Intel64 (32-bit compatible) Interstage Application Server, the "-m32 -mtune=i386" option must be specified for the gcc/g++ command.

2.2.6 Examples of Server Application Compilation and Linkage (Solaris)

2.2.6.1 C Server Application (Thread Mode)

Example 1 (created as executable file)

odsvapl.c: Server application source

IDL_skel.c: Skeleton source

odsvapl: Server application executable file

```
% IDLc IDL.idl
% cc -c -D_REENTRANT -DNeedFunctionPrototypes -I/opt/FSUNod/include -o odsvapl.o odsvapl.c
% cc -c -D_REENTRANT -DNeedFunctionPrototypes -I/opt/FSUNod/include -o IDL_skel.o IDL_skel.c
% cc -lthread -lsocket -lnsl -L/opt/FSUNod/lib -lOM -o odsvapl odsvapl.o IDL_skel.o
```

Example 2 (created as shared object)

odsvapl.c: Server application source

IDL_A_B_skel.c: Skeleton source

libA_B.so: Server application shared object

```
% cat IDL.idl
  module A {
    interface B {
      :
    }
  }
% IDLc -dy IDL.idl
% cc -c -D_REENTRANT -DNeedFunctionPrototypes -I/opt/FSUNod/include -o odsvapl.o odsvapl.c
% cc -c -D_REENTRANT -DNeedFunctionPrototypes -I/opt/FSUNod/include -o IDL_A_B_skel.o IDL_skel.c
% cc -G -Kpic -lthread -lsocket -lnsl -L/opt/FSUNod/lib -lOM -o libA_B.so odsvapl.o IDL_A_B_skel.o
```

For information about usable compilers, refer to "Software Products Required for Application Development" in the "Supported Software" chapter in the Product Notes.

Specify a thread library (libthread.so) at the beginning of the other libraries. After linking, execute the *ldd* command to confirm that libthread.so is linked in the hierarchy over system library libc.so. Note that if libthread.so is linked under libc.so, the thread library operates abnormally, causing the application to malfunction.

```
% ldd odsvapl
libthread.so.1 =>  /usr/lib/libthread.so.1
    :                     :
libc.so.1 =>       /usr/lib/libc.so.1
    :                     :
```

2.2.6.2 C Server Application (Process Mode)

Example

odsvapl.c: Server application source

IDL_skel.c: Skeleton source

odsvapl: Server application executable file

```
% IDLc IDL.idl
% cc -c -DNeedFunctionPrototypes -I/opt/FSUNod/include -o odsvapl.o odsvapl.c
% cc -c -DNeedFunctionPrototypes -I/opt/FSUNod/include -o IDL_skel.o IDL_skel.c
% cc -lsocket -lnsl -L/opt/FSUNod/lib/nt -lOM -o odsvapl odsvapl.o IDL_skel.o
```

Refer to "Software Products Required for Application Development" in the "Supported Software" chapter of the Product Notes for information about usable compilers.

In a process mode application, do not add a thread library (libthread.so) during linkage. If it is added mistakenly, the application may malfunction. Use the ladd command to check whether thread libraries have been mistakenly mixed.

To run a process mode application, the environment variable LD_LIBRARY_PATH must have been set to /opt/FSUNod/lib/nt; otherwise, the application may malfunction.

2.2.6.3 COBOL Server Application (Using Thread Mode)

Example

odsvapl.cbl: Main program source (initialization processing part)

odsvapl_sub.cbl: Subprogram source (I/F implemented part)

IDL_A_B_skel.cbl: Skeleton source

IDL_cdr.cbl: CDR source

odsvapl: Server application executable file

libA-B.so: Server application shared object

```
% cat IDL.idl
  module A {
    interface B {
         :
    }
  }
% IDLc -cobol IDL.idl
% CORBA=/opt/FSUNod/include/COBOL
% export CORBA
% cobol -c -M -Tm odsvapl.cbl
% cobol -Tm -L/opt/FSUNod/lib -lOMcblMT -o odsvapl odsvapl.o
% cobol -G -Tm -o libIDL_cdr.so IDL_cdr.cbl
% cobol -G -Tm -o libIDL_A_B_skel.so IDL_A_B_skel.cbl
% cobol -G -Tm -L/opt/FSUNod/lib -lOMcblMT -lIDL_cdr -lIDL_A_B_skel -o libA-B.so odsvapl_sub.cbl
```

A CORBA application created by COBOL must have a dynamic link structure. Because the make-up of the dynamic program structure during operation is not guaranteed, do not specify the compiler option DLOAD.

When a COBOL source is compiled, do not specify the "-dy" option. Otherwise, the system will fail to invoke libraries for COBOL (libOMcblMT.so and libOMircblMT.so) provided by the CORBA service during application execution, causing the exception BAD_OPERATION.

2.2.6.4 COBOL Server Application (Using Process Mode)

Example

odsvapl.cbl: Main program source (initialization processing part)

odsvapl_sub.cbl: Subprogram source (I/F implemented part)

IDL_A_B_skel.cbl: Skeleton source

IDL_cdr.cbl: CDR source

odsvapl: Server application executable file

libA-B.so: Server application shared object

```
% cat IDL.idl
  module A {
    interface B {
         :
    }
  }
% IDLc -cobol IDL.idl
% CORBA=/opt/FSUNod/include/COBOL
% export CORBA
% cobol -c -M odsvapl.cbl
% cobol -L/opt/FSUNod/lib -lOMcbl -o odsvapl odsvapl.o
% cobol -G -o libIDL_cdr.so IDL_cdr.cbl
% cobol -G -o libIDL_A_B_skel.so IDL_A_B_skel.cbl
% cobol -G -L/opt/FSUNod/lib -lOMcbl -lIDL_cdr -lIDL_A_B_skel -o libA-B.so odsvapl_sub.cbl
```

A CORBA application created by COBOL must have a dynamic link structure. Because the make-up of the dynamic program structure during operation is not guaranteed, do not specify the compiler option DLOAD.

When a COBOL source is compiled, do not specify the "-dy" option. Otherwise, the system will fail to invoke libraries for COBOL (libOMcbl.so and libOMircbl.so) provided by the CORBA service during application execution, causing the exception BAD_OPERATION.

To operate a process-mode application, the environment variable LD_LIBRARY_PATH must have been set to /opt/FSUNod/lib/nt. Otherwise, the application may malfunction.

2.2.7 Compiling Java Applications

To compile the Java application, use the *javac* command of the JDK that is provided with Interstage Application Server.

Chapter 3 CORBA WorkUnits

Refer to Operating CORBA WorkUnits in the OLTP Server User's Guide for details on the operation procedure for the CORBA WorkUnit.

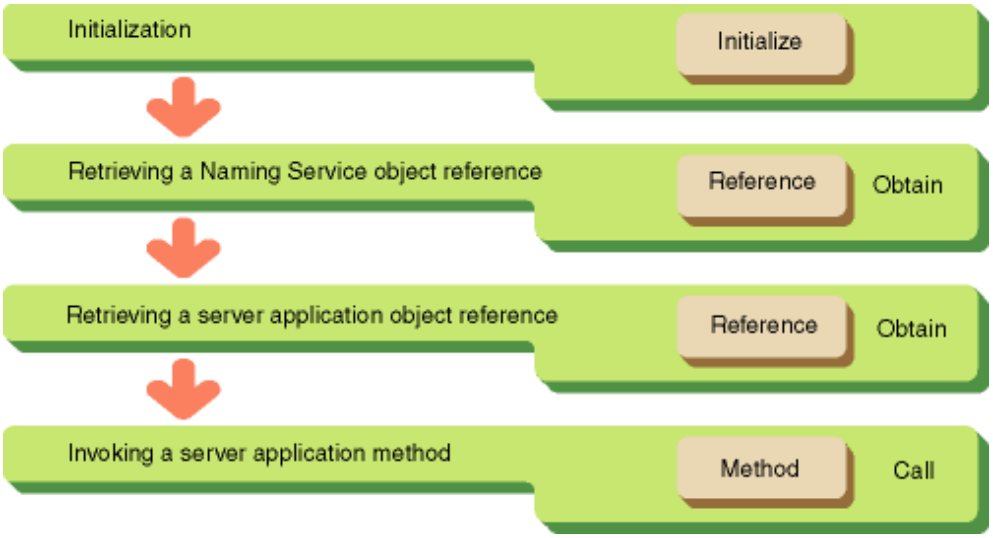
Chapter 4 C Programming Guide

This chapter explains how to develop CORBA applications in C.

4.1 Client Application Programming (Static Invocation Interface)

This section describes the client application processing flow in the static invocation interface as outlined in the following figure.

Figure 4.1 Client Application Processing Flow (Static Invocation Interface)



4.1.1 Initialization

To initialize, you must invoke the CORBA_ORB_init() method. An ORB object reference will be returned. This object reference is specified to use the ORB interface whenever it is invoked.

```
main( argc, argv )
     int     argc;
     char     *argv[];
{
     int                 current_argc = argc;
     CORBA_ORB           orb;     /* ORB object reference */
     CORBA_Environment   env;     /* Exception information */

     orb = CORBA_ORB_init(&current_argc,argv, FJ_OM_ORBid, &env );
```

4.1.2 Retrieving a Naming Service Object Reference

A Naming Service object reference is required to reference the object to be executed. The CORBA_ORB_resolve_initial_references() method (which retrieves the CORBA interface object reference) is used to retrieve a Naming Service object reference. You must specify CORBA_ORB_ObjectId_NameService as a parameter.

```
CosNaming_NamingContext  cos_naming;  /* NamingService object reference */
CosNaming_Name           name;        /* Information storage area to be retrieved */
CosNaming_NameComponent  name_component; /* Name component */

/* NamingService object reference acquired */
cos_naming = CORBA_ORB_resolve_initial_references (
             orb,
             CORBA_ORB_ObjectId_NameService,
```

```
            &env);
```

4.1.3 Retrieving a Server Application Object Reference

To retrieve a server application object reference, use the Naming Service method CosNaming_NamingContext_resolve(). You must specify the target object name as a method parameter.

```
CORBA_object                        obj;     /* Server application object reference */

name._length = name._maximum = 1;           /* Number of object names */
name._buffer = &name_component;             /* Object name storage area */
name_component.id = "ODdemo::calculator";   /* Object name */
name_component.kind = "";                   /* Object type */

/* Object reference for server application acquired */
obj = CosNaming_NamingContext_resolve(cos_naming, &name, &env );
```

4.1.4 Invoking a Server Application Method

To invoke a server program method, you must specify the method name. Concatenate the module name, interface name, and operation name using an underscore (_). In the following example, ODdemo, calculator, and calculate correspond to the module, interface and operation names. When invoking a method, you must specify the CORBA_Environment structure to which any exception information will be returned if an error occurs.

```
CORBA_long                  a,b;     /* Input parameter */
ODdemo_calculator_result    res;     /* Return value */

a = 100; b = 20;

/* Method call */
res = ODdemo_calculator_calculate( obj, a, b, &env );
```

4.2 Client Application Programming (Dynamic Invocation Interface)

The following figure outlines client application processing using a dynamic invocation interface.

Figure 4.2 Client Application Processing Flow (Dynamic Invocation Interface)

4.2.1 Initialization

Invoke CORBA_ORB_init() to initialize the operation. An ORB object reference is returned. This object reference will be used whenever an ORB interface is invoked.

```
main( argc, argv )
     int           argc;
     char          *argv[];
{
    int                current_argc = argc;
    CORBA_ORB          orb;           /* ORB object reference */
    CORBA_Environment  env;           /* Exception information */

    orb = CORBA_ORB_init(&current_argc,argv, FJ_OM_ORBid, &env );
```

4.2.2 Retrieving a Naming Service Object Reference

To retrieve a Naming Service object reference, use CORBA_ORB_resolve_initial_references(). This method retrieves the CORBA interface object reference and the Naming Service object reference. You must specify CORBA_ORB_ObjectId_NameService as a parameter.

```
CosNaming_NamingContext cos_naming; /* NamingService object reference*/
CosNaming_Name          name;       /* Information storage area to be retrieved */

/* NamingService object reference acquired */
cos_naming = CORBA_ORB_resolve_initial_references (
```

```
          orb,
          CORBA_ORB_ObjectId_NameService,
          &env);
```

4.2.3 Retrieving Server Application Information from the Interface Repository

The interface repository stores the IDL-defined module name, interface name, operation name, and parameter in a hierarchical structure. Retrieval of server application information is carried out as follows:

(1) Retrieving InterfaceDef Object Reference

The InterfaceDef object reference retrieves server application information from the interface repository. You must specify the server application object name as a parameter in order to obtain the server application object reference from the Naming Service. You must then use the CORBA_Object_get_interface() method to retrieve the InterfaceDef object reference.

```
CORBA_Object        obj;     /* Object reference for server application */
CosNaming_Name      name;    /* Information storage area to be referenced */
CORBA_InterfaceDef  intf;    /* InterfaceDef object reference */

name._length = name._maximum = 1;          /* Number of object names */
name._buffer = &name_component;            /* Object name storage area */
name_component.id = "ODdemo::calculator";  /* Object name */
name_component.kind = "";                  /* Object type */

/* Object reference for server application acquired */
obj = CosNaming_NamingContext_resolve (
          cos_naming
          &name,
          &env );

/* InterfaceDef object reference acquired */
intf = CORBA_Object_get_interface(obj, &env);
```

(2) Retrieving an OperationDef Object Reference

To retrieve specified parameter information from the interface repository, use CORBA_OperationDef_describe(). This method is used to retrieve server application information (parameter name, number of parameters, and parameter type). You must specify OperationDef object reference as a parameter.

```
CORBA_ContainedSeq       *intf_opr;

/* OperationDef object reference for method acquired */
intf_opr = (CORBA_OperationDef)CORBA_InterfaceDef_lookup_name(
     intf,
     "calculate",
     -1,
     CORBA_dk_Operation,           /* Operation information acquired */
     CORBA_FALSE,
     &env );
```

(3) Retrieving Parameter Information

To search the interface repository for parameter information on a method implemented in the server application, use CORBA_OperationDef_describe(). This information can include parameter names, number of parameters, and parameter types. You must specify the OperationDef object reference as a parameter.

```
/* Parameter information structure held by OperationDef object */
CORBA_Contained_Description     *description;

/* Parameter information structure acquired */
description = CORBA_OperationDef_describe( intf_opr->_buffer[0], &env );
```

4.2.4 Assembling Parameters

This section provides information on assembling parameters.

(1) Creating a Parameter List

To create a list object (NVList object) which stores the parameters to be passed to the server, use CORBA_ORB_create_list(). You must specify the number of parameters to be stored as a parameter. An NVList object reference is then returned. For further information on the NVList object interface, refer to "NVList Object" in the "CORBA Interface" chapter.

```
CORBA_OperationDescription     *opr_description;     /* Parameter information structure */
CORBA_any                     *tmp_any;            /* Work */
CORBA_ParDescriptionSeq       *params;             /* Parameter information */
CORBA_NVList                  arg_list;            /* List object */

/* Structure extracted from Contained Description structure */
tmp_any = &description->value;
opr_description = (CORBA_OperationDescription *)tmp_any->_value;

/* Parameter information extracted */
params = &opr_description->parameters;

/* List object generated */
CORBA_ORB_create_list( orb,
          params->_length,
          &arg_list,
          &env );
```

(2) Assigning Parameters to the List

To assign a parameter to the list to be passed to the server, use CORBA_NVList_add_item(). You must specify the CORBA_NVList object reference, the name of the server application, type, value, and length as parameters.

The order in which parameter information is set in the list object using CORBA_NVList_add_item() must be consistent with the IDL definition. If the first method argument is an in-parameter, the second argument is an out-parameter, and the third argument is an inout-parameter in the IDL definition, then parameter information must be set in the list object in the same order as in the IDL definition.

For details on how to set parameters, see "Acquiring/releasing parameters via a dynamic interface" in "Mapping to data type."

```
CORBA_long     a,b;

/* Parameter setting */
CORBA_NVList_add_item(
     arg_list,                          /* List object */
     params->_buffer[0].name,           /* Parameter name */
     TC_long,                           /* Parameter type */
     &a,                                /* Parameter value */
     sizeof( CORBA_long ),              /* Parameter length */
     CORBA_ARG_IN,                      /* Parameter type (in, out, inout) */
     &env);

CORBA_NVList_add_item(
     arg_list,                          /* List object */
     params->_buffer[1].name,           /* Parameter name */
     TC_long,                           /* Parameter type */
```

```
     &b,                                /* Parameter value */
     sizeof( CORBA_long ),              /* Parameter length */
     CORBA_ARG_IN,                      /* Parameter type (in, out, inout) */
     &env);
```

4.2.5 Creating a Request

To create a request object, use CORBA_Object_create_request(). You must specify an object reference for the server object, an NVList and NamedValue to store the returned value (described later) and the Invoking server as the request object. The object reference for the request object is then returned.

The request object acquired here must be deleted when it is no longer needed. For further information on object deletion, refer to "4.2.7 Deleting a Request".

```
CORBA_NamedValue     result;             /* Return */

result.argument._type = TC_ODdemo_calculator_result;
                                         /* Return parameter type */
result.len = sizeof( ODdemo_calculator_result );
                                         /* Return parameter length */
result.argument._value = NULL;

/* List object is generated */
CORBA_Object_create_request(
     obj,                                /* Object reference for server application */
     CORBA_OBJECT_NIL,                   /* context */
     "calculate",                        /* Method name */
     arg_list,                           /* Input parameter */
     &result,                            /* Return value */
     &request,
     CORBA_OUT_LIST_MEMORY,
     &env );
```

In the following example, the NamedValue is the location used to store return values from the server. The IDL file definition for NamedValue is as follows:

```
module CORBA
{
     struct NamedValue
     {
          identifier     name;        // Parameter name
          any            argument;    // Parameter value
          long           len;         // Parameter length
          Flags        arg_modes;     // Parameter passing method
                                      // (in, out, inout)
     };
};
```

Using C, NamedValue is defined as follows:

```
struct  CORBA_NamedValue
{
     CORBA_identifier   name;          /* Parameter name */
     CORBA_any          argument;      /* Parameter value */
     CORBA_long         len;           /* Parameter length */
     CORBA_Flags        arg_modes;     /* Parameter passing method (in, out, inout) */
};
```

4.2.6 Sending a Request

To send a request to server applications you can use either synchronous or asynchronous communication.

4.2.6.1 Synchronous Communication

To invoke the server application, use CORBA_Request_invoke() method. You must specify the object reference of the request object as a parameter.

```
CORBA_Request_invoke( request, NULL, &env );
```

Processing results of the server application are stored in the CORBA_NVList object (in this example, arg_list) set with CORBA_NVList_add_item().

4.2.6.2 Asynchronous Communication

To invoke the server application, use CORBA_Request_send(). You must specify the object reference of the request object as a parameter. Use CORBA_Request_get_response() method to receive the server return value. You must specify the object reference for the request object as a parameter.

```
CORBA_Request_send( request,         /* Processing request */
     (CORBA_Flags)NULL, env );
   ...
if ( CORBA_Request_get_response(     /* Processing result reception */
     request,
     CORBA_RESP_NO_WAIT,
     &env ) == CORBA_OK ) {          /* User-specified processing */
     .....
}
```

If the request from the CORBA_Request_get_response() server application is not completed successfully, the CORBA_Request_get_response() should be invoked again.

Processing results of the server application are stored in the CORBA_NVList object (in this example, arg_list) set with CORBA_NVList_add_item().

4.2.7 Deleting a Request

To delete a request object, use CORBA_Request_delete() method. You must specify the object reference of the request object as a parameter. To delete an NVList object, use CORBA_NVlist_free() with an object reference to the NVList object specified as a parameter.

When an NVList object is deleted, the out and inout parameter areas allocated on the server application side are automatically made free.

```
CORBA_Request_delete(request,&env);     /* Request object is deleted */
CORBA_NVList_free( arg_list, &env );    /* Deletes the NVList object for parameters */
```

4.3 Client Application Exception Handling

The server application processing result tells you whether the client application has terminated normally or abnormally. In the case of an abnormal termination, the processing result can also indicate whether the system or server application caused the application to terminate abnormally (known respectively as a system exception and a user exception).

Error information is stored in the CORBA_Environment structure specified when invoking the method. The CORBA_Environment structure is shown below.

```
typedef struct  {
     CORBA_enum     _major;
     CORBA_unsigned_long     _minor;
          .
```

```
        .
} CORBA_Environment;
```

The application must initialize the CORBA_Environment structure area before invoking a method.

```
CORBA_Environment env;

memset( &env, 0, sizeof(CORBA_ Environment));
```

The following values are set in _major:

```
CORBA_NO_EXCEPTION          : Normal termination
CORBA_SYSTEM_EXCEPTION      : System exception
CORBA_USER_EXCEPTION        : User exception
```

4.3.1 System Exception

The system exceptions shown in the following table are assigned. The data type notified for a system exception is the string type. To retrieve the string from the CORBA_Environment structure, use CORBA_exception_id().

Refer to the Reference Manual (API Edition) for the meaning of each exception.

Table 4.1 Exception Codes

Exception information	Exception code
BAD_CONTEXT	ex_CORBA_StExcep_BAD_CONTEXT
BAD_INV_ORDER	ex_CORBA_StExcep_BAD_INV_ORDER
BAD_OPERATION	ex_CORBA_StExcep_BAD_OPERATION
BAD_PARAM	ex_CORBA_StExcep_BAD_PARAM
BAD_QOS	ex_CORBA_StExcep_BAD_QOS
BAD_TYPECODE	ex_CORBA_StExcep_BAD_TYPECODE
CODESET_INCOMPATIBLE	ex_CORBA_StExcep_CODESET_INCOMPATIBLE
COMM_FAILURE	ex_CORBA_StExcep_COMM_FAILURE
CONTEXT	ex_CORBA_StExcep_CONTEXT
DATA_CONVERSION	ex_CORBA_StExcep_DATA_CONVERSION
FREE_MEM	ex_CORBA_StExcep_FREE_MEM
IMP_LIMIT	ex_CORBA_StExcep_IMP_LIMIT
INITIALIZE	ex_CORBA_StExcep_INITIALIZE
INTERNAL	ex_CORBA_StExcep_INTERNAL
INTF_REPOS	ex_CORBA_StExcep_INTF_REPOS
INV_FLAG	ex_CORBA_StExcep_INV_FLAG
INV_IDENT	ex_CORBA_StExcep_INV_IDENT
INV_OBJREF	ex_CORBA_StExcep_INV_OBJREF
INV_POLICY	ex_CORBA_StExcep_INV_POLICY
MARSHAL	ex_CORBA_StExcep_MARSHAL
NO_IMPLEMENT	ex_CORBA_StExcep_NO_IMPLEMENT
NO_MEMORY	ex_CORBA_StExcep_NO_MEMORY
NO_PERMISSION	ex_CORBA_StExcep_NO_PERMISSION

Exception information	Exception code
NO_RESOURCES	ex_CORBA_StExcep_NO_RESOURCES
NO_RESPONSE	ex_CORBA_StExcep_NO_RESPONSE
OBJ_ADAPTER	ex_CORBA_StExcep_OBJ_ADAPTER
PERSIST_STORE	ex_CORBA_StExcep_PERSIST_STORE
REBIND	ex_CORBA_StExcep_REBIND
TIMEOUT	ex_CORBA_StExcep_TIMEOUT
TRANSIENT	ex_CORBA_StExcep_TRANSIENT
UNKNOWN	ex_CORBA_StExcep_UNKNOWN
INVALID_TRANSACTION	ex_CORBA_StExcep_INVALID_TRANSACTION
TRANSACTION_MODE	ex_CORBA_StExcep_TRANSACTION_MODE
TRANSACTION_REQUIRED	ex_CORBA_StExcep_TRANSACTION_REQUIRED
TRANSACTION_ROLLEDBACK	ex_CORBA_StExcep_TRANSACTION_ROLLEDBACK
TRANSACTION_UNAVAILABLE	ex_CORBA_StExcep_TRANSACTION_UNAVAILABLE

A minor code is set in _minor in the CORBA_Environment structure when a system exception occurs. Refer to information on CORBA Service Minor Codes in the Reference Manual (API Edition) for details of minor code values.

4.3.2 User Exception

Strings are also used to report error information relating to user exceptions. To retrieve the string from the CORBA_Environment structure, use CORBA_exception_id(). The CORBA_exception_value is used to retrieve details from the CORBA_Environment structure on user-defined information defined by IDL.

4.3.3 Obtaining Exception Information

The following example shows how to obtain exception information.

```
/* Method invoked */
ret = ODdemo_calculator_calculate(obj, a, b, &env);

switch(env._major){
     case CORBA_SYSTEM_EXCEPTION:
          /* System exception error processing */
          id = CORBA_exception_id(&env);

          if ( strcmp (id, ex_CORBA_StExcep_COMM_FAILURE) == 0 ) {
              /*  Obtain minor code            */
               fprintf( stderr, "SystemException:COMM_FAILURE (minor = %x)\n", env._minor);
               .
               .
          }
               .
               .
     case CORBA_USER_EXCEPTION:
          /* User exception error processing */
          id  = CORBA_exception_id(&env);

          if ( strcmp (id, ex_exp_exc) == 0 ) {
               uexc  =  (exp_exc *)CORBA_exception_value(&env);
               .
               .
          }
```

```
.
.
```

4.4 Server Application Programming (Static Skeleton Interface)

A server application is composed of initialization and interface implementation components. The following figure outlines the initialization component.

Figure 4.3 Server Application Initialization Component

4.4.1 Initialization

Initialization is carried out by invoking CORBA_ORB_init(). An ORB object reference is then returned. This object reference is specified whenever the ORB interface is invoked.

```
main( argc, argv )
     int     argc;
     char    *argv[];
{
     int     current_argc = argc;
     CORBA_ORB             orb;     /* ORB object reference */
     CORBA_Environment     env;     /* Exception information */

     orb = CORBA_ORB_init( &current_argc,argv, FJ_OM_ORBid, &env );
```

The CORBA_ORB_BOA_init() method is used to initialize a basic object adapter.

```
     CORBA_BOA     boa;     /* BOA object reference */

     /* BOA is acquired */
     boa = CORBA_ORB_BOA_init(
                   orb, &current_argc, argv, CORBA_BOA_OAid, &env );
```

Invoke server application initialization as required.

4.4.2 Activating the Server

When initialization is complete, the server application notifies ORB. Once this instruction has been issued, ORB sends a request to the server from the client application. The activation method varies according to the server type, listed in the following table.

Table 4.2 Server Activation Method

Server Type	Method
shared server	CORBA_BOA_impl_is_ready
unshared server	CORBA_BOA_obj_is_ready
persistent server	CORBA_BOA_impl_is_ready
server per method	CORBA_BOA_impl_is_ready

(1) Retrieving the Implementation Repository Object Reference

To retrieve an Implementation Repository object reference, use CORBA_ORB_resolve_initial_references() to specify CORBA_ORB_ObjectId_ImplementationRepository as the second parameter.

(2) Searching for an ImplementationDef Object Reference

To obtain an ImplementationDef object reference, use FJ_ImplementationRep_lookup_id() to specify the ImplementationRep object for the server application for the first parameter.

(3) Activating a Server

To activate the server, use CORBA_BOA_impl_is_ready() or CORBA_BOA_obj_is_ready().

The method used in a server application depends on whether the server application is implemented using a process or a thread. If the server application is implemented using a thread, the method used depends on information assigned in the implementation repository. The differences between the methods are shown below.

Table 4.3 Differences in Server Application Methods

Condition		Operation
Implementation by process	mode=SYNC_END for *OD_impl_inst*	The method does not return control immediately but only when the operator instructs this object to stop (*1).
Implementation by thread	mode= SYNC_END for *OD_impl_inst*	Same as above
	mode= COMPATIBLE for *OD_impl_inst*	The method returns control immediately. Next, invoke thr_exit.

*1 After the method returns control, a deactivation process (refer to "4.4.4 Deactivating the Server") is required to close the file opened during initialization and to release the location.

```
CORBA_ImplementationDef     impl;       /* ImplementationDef object reference */
ImplementationRep           impl_rep;   /* ImplementationRep object reference */

/* Object reference for implementation information retrieved */
impl_rep = CORBA_ORB_resolve_initial_references(
     orb,
     CORBA_ORB_ObjectId_ImplementationRepository,
     &env);

/* ImplementationRep object reference retrieved */
impl = FJ_ImplementationRep_lookup_id(
     impl_rep,
     _IMPL_ODdemo_calculator,           /* Repository ID */
     &env );

/* Server activated */
CORBA_BOA_impl_is_ready(boa, impl, &env );
```

4.4.3 Interface Implementation Function

After initialization, interface implementation processing by the server application is described.

The object reference is mapped to the first parameter of the interface implementation function. The IDL-defined parameter and pointer to the CORBA_Environment structure are then mapped to the first parameter.

```
ODdemo_calculator_result              /* Return value type */
ODdemo_calculator_calculate(          /* Implementation function of calculate method */
     ODdemo_calculator          obj,  /* Server object reference */
     CORBA_long                 a,    /* Input value */
```

```
     CORBA_long                 b,    /* Input value */
     CORBA_Environment         *env ) /* Error value */
{
     ODdemo_calculator_result   res;  /* Return value */

     res.add_result = 0;
     res.subtract_result = 0;
     res.multiple_result = 0;
     res.divide_result = 0;

     if( b == 0 ){                    /* 0 division check */
          CORBA_BOA_set_exception(
               boa,
               CORBA_USER_EXCEPTION,
               ex_ODdemo_calculator_ZEROPARAM,
               NULL,
               env );
          return( res ) ;
     }
     res.add_result = a+b;                  /* Addition result setting */
     res.subtract_result = a-b;             /* Subtraction result setting */
     res.multiple_result = a*b;             /* Multiplication result setting */
     res.divide_result = (CORBA_float)a/b;  /* Division result setting */
     return res;
}
```

4.4.4 Deactivating the Server

If a user sends a request to stop, the server application notifies ORB that subsequent requests from the client should not be accepted. When this notification is received, ORB will no longer send the processing request from the client to the server application and will return an exception to the client. The deactivation method and the server type to which it relates, is listed in the following table.

Table 4.4 Server Deactivation Method

Server Type	Method
shared server	CORBA_BOA_deactivate_impl
unshared server	CORBA_BOA_deactivate_obj
persistent server	CORBA_BOA_deactivate_impl
server per method	Not issued

The CORBA_BOA_deactivate_impl() method specifies the ImplementationRep object for the server application as a parameter. CORBA_BOA_deactivate_obj() specifies the object reference as a parameter.

```
CORBA_BOA_deactivate_impl(boa, impl, &env );     /* Server deactivation */
```

If the isstopwu command or the Interstage Management Console is used to stop an operating WorkUnit, the server application notifies ORB that subsequent requests from the client should not be accepted. Client applications no longer need to issue the deactivation method.

If the odcntlque command is used to stop server applications when the CORBA application program is operated under control of the WorkUnit, client applications no longer need to issue the deactivation method.

4.5 Programming Server Application Exceptions

This section contains information on programming server application exceptions.

Setting Exception Information

If a user exception is to be assigned, CORBA_string_alloc is invoked. This function allocates the location to which the string identifying the user exception is assigned. The string is copied to this location. The alloc function generated by the IDL compiler is then used to allocate the location used for assigning a user exception. The alloc function name is specified in a format where the IDL-specified interface name, exception identifier, and alloc are connected by an underscore (_). Because IDL-defined exception information constitutes a structure in C, a value is assigned to the structure member variable. In the following example, user exception identification information is not assigned. CORBA_BOA_set_exception() is used to assign exception information in the CORBA_Environment structure.

```
ODdemo_calculator_result
ODdemo_calculator_calculate(
     ODdemo_calculator            obj,
     CORBA_long                   a,
     CORBA_long                   b,
     CORBA_Environment            *env )
{
     ODdemo_calculator_result     res;

             :
     if( b == 0 ){
          /* Error ZEROPARAM set in env */
          CORBA_BOA_set_exception(
               boa,                     /* BOA object reference */
               CORBA_USER_EXCEPTION,    /* User exception */
               ex_ODdemo_calculator_ZEROPARAM,
                                        /* Method that caused a user exception */
               NULL,                    /* User exception identification information */
               env );                   /* CORBA_Environment */
          return( 0 ) ;
     }
          :
     return res;
}
```

To call another method from the inside of a server method, use another CORBA::Environment structure. Do not use the CORBA::Environment structure passed to a parameter of a server method as a parameter for calling another method. If this is done, exception information for the inside of the server method that is generated by another method will be set as the exception information.

The following example shows an acceptable programming example and an incorrect programming example:

Correct Programming Example

```
ODdemo_calculator_result
ODdemo_calculator_calculate(
    ODdemo_calculator   obj,
    CORBA_long          a,
    CORBA_long          b,
    CORBA_Environment   *env )
{
    CosNaming_NamingContext  nc;
    CosNaming_Name  name;
    CORBA_Object  obj;
    CORBA_Environment  local_env; /*  */

    memset( &local_env, 0, sizeof(CORBA_Environment) );
        :

    /* The exception information for CosNaming_NamingContext_bind()
     * is set in local_env.  Thus it is not handled as the exception
     * information for ODdemo_calculator_calculate().
```

```
     */
    CosNaming_NamingContext_bind( nc, &name, obj, &local_env );
        :
}
```

Incorrect Programming Example

```
ODdemo_calculator_result
ODdemo_calculator_calculate(
    ODdemo_calculator   obj,
    CORBA_long          a,
    CORBA_long          b,
    CORBA_Environment   *env )
{
    CosNaming_NamingContext  nc;
    CosNaming_Name           name;
    CORBA_Object             obj;
        :

    /* When an exception is returned to CosNaming_NamingContext_bind(),
     * the exception information for CosNaming_NamingContext_bind()
     * is incorrectly handled as the exception information for
     * ODdemo_calculator_calculate().
     */
    CosNaming_NamingContext_bind( nc, &name, obj, env );
        :
}
```

Obtaining Exception Information

Obtaining exception information for server applications is the same as for client applications. Refer to "4.3 Client Application Exception Handling" for details.

4.6 Registering of a Server Application

Register the created server application to ImplementationRepository and NamingService.

4.6.1 Registering in the Implementation Repository

The OD_impl_inst command is used to register server application information in the implementation repository. The following is an example of registration processing using the OD_impl_inst command. The information to be specified is as follows:

Windows32/64

```
OD_impl_inst -a -r IDL:ODdemo/calculator:1.0 -t S -f  D:\server\simple_s.exe
```

Solaris32/64 Linux32/64

```
OD_impl_inst -a -r IDL:ODdemo/calculator:1.0 -t S -f /home/guest/simple_s -u user -g group
```

-a

Indicates that server application information is registered.

-r IDL:ODdemo/calculator:1.0

Specifies the implementation repository ID.

-t S

Specifies the server type with one uppercase letter. The types that can be specified include persistent (P), shared (S), unshared (U), and server per method (M).

Windows32/64

-f D:\server\simple_s.exe

Solaris32/64 Linux32/64

-f /home/guest/simple_s

Specifies the path name of the server application storage location.

Solaris32/64 Linux32/64

-u user

Specifies the User ID used when executing the server application.

Solaris32/64 Linux32/64

-g group

Specifies the Group ID used when executing the server application.

4.6.2 Registering in the Naming Service

To enable other applications to access a server application as an object, you must create an object reference to identify it. The object reference created is also registered in the Naming Service. You can create an object in the following way:

- Use the OD_or_adm command.
- In the server application, BOA functionality is used to create an object reference and register it in the Naming Service.

Using the OD_or_adm Command

Once an object reference has been created, use OD_or_adm command to register it in the Naming Service. The following example deals with registration using the OD_or_adm command and information that needs to be specified:

```
OD_or_adm -c IDL:ODdemo/calculator:1.0 -n ODdemo::calculator
```

-c IDL:ODdemo/calculator:1.0

Registers an object reference using the specified interface repository ID.

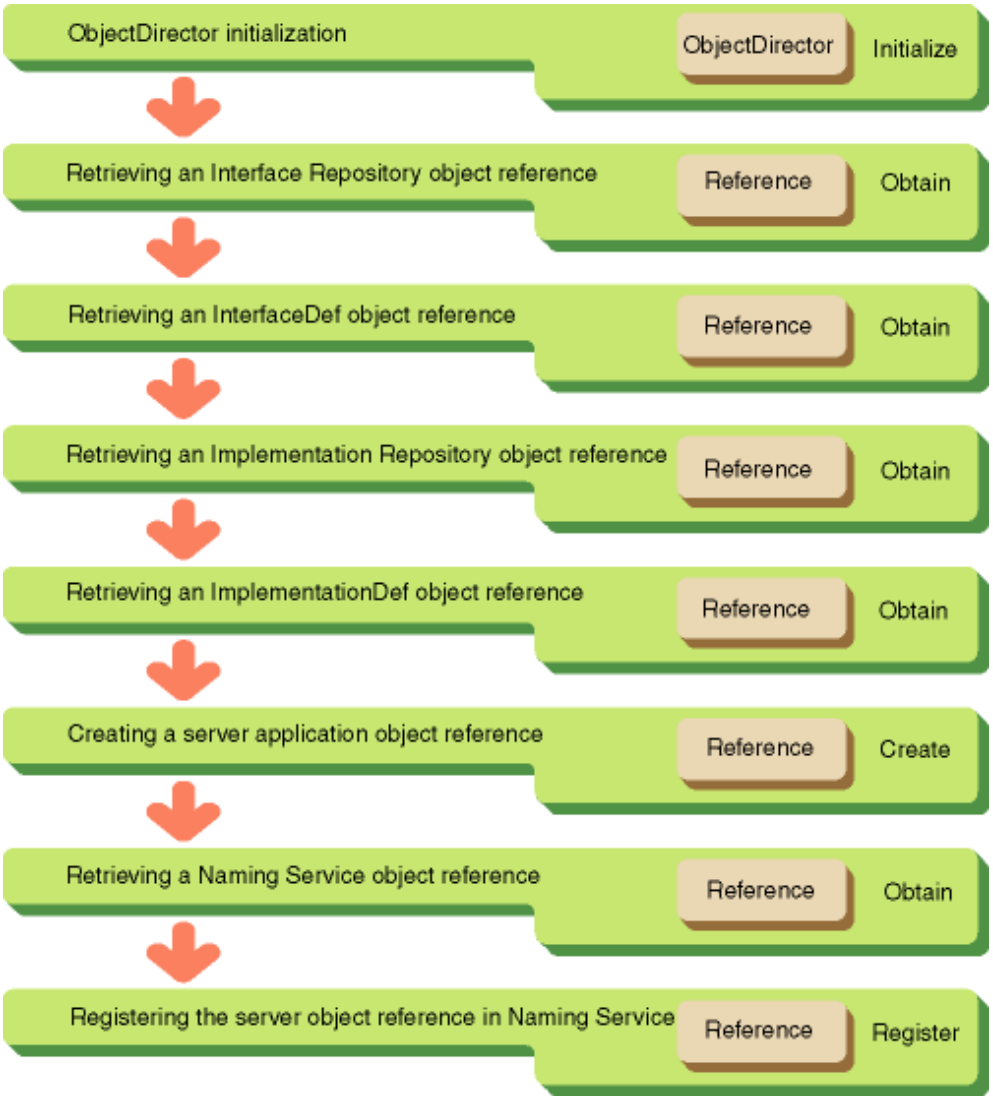
-n ODdemo::calculator

Specifies the name of the object to be registered in the Naming Service.

Creation Method in a Server Application

The following figure describes the process to create a server application object reference.

Figure 4.4 Creation Process in a Server Application




```
CORBA_ORB               orb;       /* Object returned from ORB_init */
CORBA_BOA               boa;       /* Object returned from ORB_BOA_init */
CORBA_Repository        intf_rep;  /* Object reference for interface repository */
CORBA_InterfaceDef      intf;      /* InterfaceDef for interface repository */
FJ_ImplementationRep    impl_rep;  /* Object reference for implementation repository */
CORBA_ImplementationDef impl;      /* ImplementationDef for implementation repository*/
CORBA_Object            new_obj;   /* Created object reference */
CORBA_ReferenceData     id;        /* ReferenceData storage area */
CosNaming_NamingContext cos_naming;/* Object reference for NamingService */
CosNaming_Name          name;      /* Information storage area to be retrieved */
CORBA_Environment       env;       /* Exception information */

/* ObjectDirector Initialization (omitted) */
   :

/* ReferenceData Initialization */
id._length  = 0;
id._maximum = 0;
id._buffer  = NULL;

/* Getting an Interface Repository object reference */
intf_rep = CORBA_ORB_resolve_initial_references(
          orb,
```

```
          CORBA_ORB_ObjectId_LightInterfaceRepository,
          &env );

/* Getting an InterfaceDef object reference */
intf = CORBA_Repository_lookup_id(
          intf_rep,
          "...",  /* Interface repository ID */
          &env );

/* Getting an Implementation Repository object reference */
impl_rep = CORBA_ORB_resolve_initial_references(
          orb,
          CORBA_ORB_ObjectId_ImplementationRepository,
          &env );

/* Getting an ImplementationDef object reference */
impl = FJ_ImplementationDef_lookup_id(
          impl_rep,
          "...", /* Interface repository ID */
          &env );

/* Creating an ObjectReference */
new_obj = CORBA_BOA_create( boa, &id, intf, impl, &env, );

/* NamingService reference is acquired  */
cos_naming = CORBA_ORB_resolve_initial_references(
          orb,
          CORBA_ORB_ObjectId_NameService,
          &env);

name._length = name._maximum = 1;             /* Number of object names */
name._buffer = &name_component;               /* Object name storage area */
name_component.id = "ODdemo::calculator";     /* Object name */
name_component.kind = "";                     /* Object type */

/* Addition to Naming Service */
CosNaming_NamingContext_bind (cos_naming, &name, new_obj, &env );
```

Note

CORBA_BOA_create() creates an ObjectReference with its default character set registered in Implementation Repository. The default character set is specified by OD_impl_inst or OD_set_env command.

4.7 Mapping Data Types

This section describes the data types to be used to create client-server applications in C.

4.7.1 Basic Data Types

The basic data types defined in CORBA may be used in programs, by defining them as shown in the following table.

Table 4.5 Definition of Basic Data Types for C

CORBA Data Types		Definition in C	Remarks
Integer type	long	CORBA_Long	(*1)
	short	CORBA_Short	(*2)
	unsigned long	CORBA_unsigned_long	

CORBA Data Types		Definition in C	Remarks
	unsigned short long long	CORBA_unsigned_short CORBA_long_long	
Floating-point type	float double long double	CORBA_float CORBA_double CORBA_long_double	Windows32/64 Solaris32/64
Character type	char wchar	CORBA_char CORBA_wchar	
Octet type	octet	CORBA_octet	
Boolean type	boolean	CORBA_boolean	
Character string type	string wstring	CORBA_string CORBA_wstring	Refer to "4.7.2 String Type" Refer to "4.7.3 Wide String Type"
Character string type	string	CORBA_string	Refer to "4.7.2 String Type"
Enumeration type	enum	CORBA_enum	
Character string type	string	CORBA_string	Refer to "4.7.2 String Type"
Any type	any	CORBA_any	Refer to "4.7.4 Any Type"
Object reference	Object	CORBA_Object	
Type code	TypeCode	CORBA_TypeCode	

1. *1 int in Linux (64 bit).
2. *2 unsigned int in Linux (64 bit).

4.7.2 String Type

This section describes string type data.

IDL Mapping

When "string" (character string type) is specified in IDL, data in C must be declared as a CORBA_string. Assume that the following definition was made in IDL:

```
typedef     char     *CORBA_string;     /* string definition*/
```

This is explained using the following IDL definition example.

IDL

```
module ODsample{
     interface     stringtest{
          string op1(in string str1, out string str2, inout string str3);
     };
};
```

C Language

```
CORBA_string
ODsample_stringtest_op1(
     ODsample_stringtest obj;          /* Object reference */
     CORBA_string str1,                /* "in" parameter */
     CORBA_string *str2,               /* "out" parameter */
     CORBA_string *str3,               /* "inout" parameter */
     CORBA_Environment *env )          /* Exception information */
```

Parameters Handled by Client Applications

The following table shows how the client application parameters are handled.

Table 4.6 Client Parameter Area (Character String Type)

Parameter	Parameter passed to server	Parameter passed from server
in	The area is allocated by CORBA_string_alloc().	_
inout	Same as the *in* parameter	The area is automatically allocated by the stub.
out return	_	Same as the *inout* parameter

When an area that has been allocated by the client and stub is no longer required, it must be released by CORBA_free().

The following is an example of client application processing.

```
CORBA_Environment  env;
CORBA_string     str1, str2, str3, ret;

str1 = CORBA_string_alloc(2);     /* Allocation of area for "in" parameter */
strcpy( str1 ,"IN" );             /* Setting of "in" parameter */
str3 = CORBA_string_alloc(7);     /* "Allocation of area for "inout" parameter */
strcpy( str3 ,"INOUT:1" );        /* Setting of "inout" parameter */
ret = ODsample_stringtest_op1(obj, str1, &str2, &str3, &env );

CORBA_free( ret );                /* Release of area for return value */
CORBA_free( str1 );               /* Release of area for "in" parameter */
CORBA_free( str2 );               /* Release of area for "out" parameter */
CORBA_free( str3 );               /* Release of area for "inout" parameter */
```

Parameters Handled by Server Applications

The following table shows how the server application parameters are handled.

Table 4.7 Server Parameter Area (Character String Type)

Parameter	Parameter passed from client	Parameter passed to client
in	The character string area is automatically allocated or released by the skeleton.	_
inout	The character string area is automatically allocated by the skeleton.	When a character string shorter than the passed parameter is to be returned, the character string is set in the passed character string area.

Parameter	Parameter passed from client	Parameter passed to client
		When a character string longer than the passed parameter is to be returned, the passed character string area is released once by CORBA_free() and re-allocated by CORBA_string_alloc(), The character string area is automatically released by the skeleton.
out return	-	The character string area is allocated by CORBA_string_alloc(), and is automatically released by the skeleton.

The following is an example of server application processing.

```
CORBA_string
ODsample_stringtest_op1(
     ODsample_stringtest obj;          /* Object reference */
     CORBA_string str1,                /* "in" parameter */
     CORBA_string *str2,               /* "out" parameter */
     CORBA_string *str3,               /* "inout" parameter */
     CORBA_Environment *env )          /* Exception information */
   {
          CORBA_string str;

          /* Processing of "out" parameter */
          *str2 = CORBA_string_alloc(3);
                                       /* Allocation of area for "out" parameter */
          strcpy( *str2, "OUT" );      /* Setting of "out" parameter */

          /* Processing of "inout" parameter */
          CORBA_free( *str3 );         /* Release of area passed from client */
          *str3 = CORBA_string_alloc(7);
                                       /* Allocation of area for output parameter */
          strcpy( *str3, "INOUT:2" );  /* Setting of output parameter */

          /* Processing of return value */
          str = CORBA_string_alloc(6);
                                       /* Allocation of area for return value */
          strcpy( str, "RETURN" );     /* Setting of return value */
          return( str );
     }
```

4.7.3 Wide String Type

This section describes wstring type data.

IDL Mapping

When "wstring" (character wide string type) is specified in IDL, data in C must be declared as a CORBA_wstring. Assume that the following definition was made in IDL:

```
type def   CORBA_wchar     *CORBA_wstring;
```

This is explained using the following IDL definition example.

IDL

```
module ODsample{
     interface     wstringtest{
          wstring op1(in wstring str1, out wstring str2, inout wstring str3);
```

```
    };
};
```

C Language

```
CORBA_wstring
ODsample_wstringtest_op1(
     ODsample_wstringtest obj;         /* Object reference */
     CORBA_wstring str1,               /* "in" parameter */
     CORBA_wstring *str2,              /* "out" parameter */
     CORBA_wstring *str3,              /* "inout" parameter */
     CORBA_Environment *env )          /* Exception information */
```

Parameters Handled by Client Applications

The following table shows how the client application parameters are handled.

Table 4.8 Client Parameter Area (Character String Type)

Parameter	Parameter passed to server	Parameter passed from server
in	The area is allocated by CORBA_wstring_alloc().	_
inout	Same as the *in* parameter	The area is automatically allocated by the stub.
out return	_	Same as the *inout* parameter.

Note

When an area that has been allocated by the client and stub is no longer required, it must be released by CORBA_free().

The following is an example of client application processing.

```
CORBA_wstring      str1, str2, str3, ret;
CORBA_Environment  env;

str1 = CORBA_wstring_alloc(2);     /* Allocation of area for "in" parameter */
str1[0] = ...;           /* Setting of "in" parameter (refer Setting a wstring) */

str3 = CORBA_wstring_alloc(5);     /* "Allocation of area for "inout" parameter */
str3[0] = ...;           /* Setting of "inout" parameter (refer Setting a wstring) */

ret = ODsample_wstringtest_op1(obj, str1, &str2, &str3, &env );

CORBA_free( ret );               /* Release of area for return value */
CORBA_free( str1 );              /* Release of area for "in" parameter */
CORBA_free( str2 );              /* Release of area for "out" parameter */
CORBA_free( str3 );              /* Release of area for "inout" parameter */
```

Parameters Handled by Server Applications

The following table shows how the server application parameters are handled.

Table 4.9 Server Parameter Area (Character String Type)

Parameter	Parameter passed from client	Parameter passed to client
in	The character string area is automatically allocated or released by the skeleton.	_

Parameter	Parameter passed from client	Parameter passed to client
inout	The character string area is automatically allocated by the skeleton.	When a character string shorter than the passed parameter is to be returned, the character string is set in the passed character string area. When a character string longer than the passed parameter is to be returned, the passed character string area is released once by CORBA_free() and re-allocated by CORBA_wstring_alloc(), The character string area is automatically released by the skeleton.
out return	_	The character string area is allocated by CORBA_wstring_alloc(), and is automatically released by the skeleton.

The following is an example of server application processing.

```
CORBA_wstring
ODsample_stringtest_op1(
     ODsample_wstringtest obj;         /* Object reference */
     CORBA_wstring wstr1,              /* "in" parameter */
     CORBA_wstring *wstr2,             /* "out" parameter */
     CORBA_wstring *wstr3,             /* "inout" parameter */
     CORBA_Environment *env )          /* Exception information */
   {
          CORBA_wstring wstr;

          /* Processing of "out" parameter */
          *wstr2 = CORBA_wstring_alloc(3);
                                        /* Allocation of area for "out" parameter */
          (*wstr2)[0] = ...;
                         /* Setting of "out" parameter (refer Setting a wstring)*/

          /* Processing of "inout" parameter */
          CORBA_free( *wstr3 );         /* Release of area passed from client */
          *wstr3 = CORBA_wstring_alloc(5);
                                        /* Allocation of area for output parameter */

          (*wstr3)[0] = ...;
                        /* Setting of output parameter (refer Setting a wstring) */

          /* Processing of return value */
          wstr = CORBA_wstring_alloc(4);
                                        /* Allocation of area for return value */
          wstr[0] = ...;
                         /* Setting of return value (refer Setting a wstring) */
          return( wstr );
     }
```

Setting a wstring

An example of setting a wstring is shown below. (The variable wstr is CORBA_wchar * type)

For EUC or ShiftJIS

Set each character separately (for EUC).

```
wstr[0] = 0xc6fc;
wstr[1] = 0xcbdc;
wstr[2] = 0xb8ec;
wstr[3] = 0x0000;      /* terminator */
```

For UNICODE on Windows®

Specify a string enclosed in "" with the prefix L. It is treated as UNICODE.

```
wstr = L" .... "
```

For UNICODE on Solaris/Linux Solaris32/64 Linux32/64

Set each character separately in UNICODE.

```
wstr[0] = 0x65e5;
wstr[1] = 0x672c;
wstr[2] = 0x8a9e;
wstr[3] = 0x0000;      /* terminator */
```

Note

When the wide string type is output to the console or a similar destination, the system may be unable to display all characters properly because the output processing depends on the operating system. For information on the output method, see the documentation for each operating system.

L "*" format strings cannot be used on Solaris or Linux systems.

4.7.4 Any Type

This section describes any type data.

IDL Mapping

When *any* type is specified in IDL, the data in C must be declared using CORBA_any structure. An allocation function (named with "module name", "interface name", and "structure name" concatenated with underscore ("_"). It is referred as XX_alloc function) for any data, _value area, is generated by IDL compiler. _type is the TypeCode used to identify the data type. _value is the pointer to the area in which the data is stored. The configuration of the CORBA_any structure is given below.

```
typedef struct any {
     CORBA_TypeCode        _type;        /* Data identification information */
     void                  *_value;      /* Data storage area */
} CORBA_any;
```

Assume that IDL any type is defined:

IDL

```
module ODsample{
     struct      sample1  {
                 long      para1;
                 string    para2;
     };
     struct      sample2  {
                 char      para1;
                 float     para2;
     };
     struct      sample3  {
                 char      para1;
                 double    para2;
     };
     interface   anytest  {
                 any       op1(in any any1, out any any2, inout any any3 );
     };
};
```

Parameters Handled by Client Applications

The following table shows how the client application parameters are handled.

Table 4.10 Client Parameter Area (Any Type)

Parameter	Parameter passed to server	Parameter passed from server
in	The CORBA_any_alloc() function or the XX_alloc function is used to dynamically allocate the area (CORBA_any structure area or _value area).	_
inout	Same as the *in* parameter	The area is automatically allocated by the stub.
out return	_	Same as the *inout* parameter

Note

When an area that has been allocated by the client and stub is no longer required, it must be released by CORBA_free(). Whether or not the area specified by _value is to be released can be set in the release flag.

The functions and flags shown below are used to reference and set the release flag.

Function

```
CORBA_boolean   CORBA_any_get_release(CORBA_any * );
      void      CORBA_any_set_release(CORBA_any *, CORBA_boolean );
```

Flag

CORBA_TRUE: When CORBA_free() is issued, the area specified by _value is also released .

CORBA_FALSE: When CORBA_free() is issued, the area specified by _value is not released . (Default)

The release flag of the parameter (out, return) allocated by the stub is set in CORBA_TRUE.

The following is an example of client application processing.

```
CORBA_Environment     env;
CORBA_Object          obj;
ODsample_sample1      *smp1;
ODsample_sample2      *smp2;
ODsample_sample3      *smp3;
CORBA_any             *any1, *any2, *any3, *any0;

any1 = CORBA_any_alloc();          /* Allocation of area for input parameter */
any1->_type = TC_ODsample_sample3; /* Setting of Typecode */
smp3 = any1->_value = ODsample_sample3_alloc();
smp3->para1 = 'a';                 /* Setting of input parameter */
smp3->para2 = 0.00001;             /* Setting of input parameters */
any3 = CORBA_any_alloc();          /* Allocation of area for input-output parameter */
any3->_type = TC_ODsample_sample2; /* Setting of Typecode */
smp2 = any3->_value = ODsample_sample2_alloc();
smp2->para1 = 'c';                 /* Setting of input-output parameter */
smp2->para2 = 0.0001;              /* Setting of input-output parameter */
any0 = ODsample_anytest_op1( obj, any1, &any2, any3, &env );
CORBA_free( any0 );                /* Release of area for return value */
CORBA_free( any1 );                /* Release of area for input parameter */
```

```
CORBA_free( any2 );                /* Release of area for output parameter */
CORBA_free( any3 );                /* Release of area for input-output parameter */
```

Parameters Handled by Server Applications

The following table shows how the server application parameters are handled.

Table 4.11 Server Parameter Area (Any Type)

Parameter	Parameter passed from client	Parameter passed to client
in	Each area (CORBA_any structure area or data area) is automatically allocated or released by the skeleton.	_
inout	Each area (CORBA_any structure area or data area) is automatically allocated by the skeleton.	The data area (_value) is released once by CORBA_free() and re-allocated by the XX_alloc function. The area is automatically released by the skeleton.
out return	_	The CORBA_any structure area or the data area (_value) is allocated by the CORBA_any_alloc() function or the XX_alloc function. The area is automatically released by the skeleton.

The following is an example of server application processing.

```
CORBA_any
*ODsample_anytest_op1(
     ODsample_anytest     obj,
     CORBA_any            *any1,
     CORBA_any            **any2,
     CORBA_any            *any3,
     CORBA_Environment    *env )
{
     CORBA_any            *p;
     ODsample_sample1     *smp1;
     ODsample_sample2     *smp2;
     ODsample_sample3     *smp3;

/* Processing of "out" parameter */
     p = *any2 = CORBA_any_alloc();
                                        /* Allocation of area for output parameter */
     p->_type = TC_ODsample_sample2;    /* Setting of Typecode */

/* Allocation of area for output parameter data */
     smp2 = p->_value = ODsample_sample2_alloc();
     smp2->para1 = 'x';                 /* Setting of output parameter */
     smp2->para2 = 0.001;               /* Setting of output parameter */
     CORBA_any_set_release( p, CORBA_TRUE );

/* Processing of "inout" parameter */
     any3->_type = TC_ODsample_sample3; /* Setting of "Typecode */
     CORBA_free( any3->_value );
     smp3 = any3->_value = ODsample_sample3_alloc();
     smp3->para1 = 'y';
     smp3->para2 = 0.0001;

/* Processing of return code */
     p = CORBA_any_alloc();
     p->_type = TC_ODsample_sample1;
     smp1 = p->_value = ODsample_sample1_alloc();
     smp1->para1 = 300;
```

```
     smp1->para2 = CORBA_string_alloc(4);
     strcpy( smp1->para2, "test" );
     CORBA_any_set_release( p, CORBA_TRUE );
     return( p );
}
```

4.7.5 Sequence Type

This section describes sequence type data.

IDL Mapping

When *sequence* (sequence type) is specified in IDL, the data is mapped to the structures (sequence structures) in C language. Functions for allocating the sequence structure area and the data area (_buffer) are also generated. (The sequence function name is generated using "module name_interface name_structure name_alloc", and the function is called XX_alloc. The _buffer function name is generated using "module name_interface name_structure name_allocbuf", and the function is called XX_allocbuf.)

The structure of a sequence structure is shown below.

```
struct
{
     CORBA_unsigned_long     _maximum;     /* Maximum length of sequence */
     CORBA_unsigned_long     _length;      /* Length of sequence */
     CORBA_Type              *_buffer;     /* Sequence data */
};
```

This is explained using the following IDL definition example.

IDL

```
module ODsample{
     interface       seqtest{
          typedef    sequence<long>     sampleseq;
          sampleseq  op1(in sampleseq seq1, out sampleseq seq2,
                         inout sampleseq seq3 );
     };
};
```

C Language

```
typedef struct
{
     CORBA_unsigned_long     _maximum;        /* Maximum length of sequence */
     CORBA_unsigned_long     _length;         /* Length of sequence */
     CORBA_long              *buffer;         /* Sequence data */
} sampleseq;
```

Parameters Handled by Client Applications

The following table shows how the client application parameters are handled.

Table 4.12 Client Parameter Area (Sequence Type)

Parameter	Parameter passed to server	Parameter passed from server
in	The XX_alloc function or the XX_allocbuf function is used to allocate the sequence structure area or the data area (_buffer).	_

Parameter	Parameter passed to server	Parameter passed from server
	Sets the maximum number of sequence arrays, the number of arrays to be used, and the address of the data area in the member.	
inout	Same as the *in* parameter	The area is automatically allocated by the stub. (Each member is also set.)
out return	_	Same as the *inout* parameter

Note

When an area that has been allocated by the client and stub is no longer required, it must be released by CORBA_free(). Whether or not the data area (allocated by the XX_allocbuf function) is to be released is set in the release flag.

The functions and flags shown below are used to reference and set the release flag.

Function

```
CORBA_boolean          CORBA_sequence_get_release( void * );
              void     CORBA_sequence_set_release( void *, CORBA_boolean );
```

Flag

CORBA_TRUE: When CORBA_free() is issued, the sequence data area is also released.

CORBA_FALSE: When CORBA_free() is issued, the sequence data area is not released. (Default)

The release flag of the parameter (out, return) allocated by the stub is set in CORBA_TRUE.

When using a sequence (of *sequence* or *any* type) with the *in* or *inout* parameters, set CORBA_TRUE in the release flag for the *sequence* or *any* type area that was allocated by the XX_allocbuf function.

When using a sequence with the *out*, *inout*, or *return value* parameters, where the *_buffer* type of the sequence is *sequence* or *any* type and *_buffer* is not NULL, set CORBA_TRUE in the release flag for the *_buffer* area before releasing the area using CORBA_free().

The following is an example of client application processing.

```
     CORBA_Environment           env;
     CORBA_Object                obj;
     ODsample_seqtest_sampleseq  *seq0, *seq1, *seq2, *seq3;
     CORBA_long  *smp;
     int            i;

/* Processing of "in" parameter */
     seq1 = ODsample_seqtest_sampleseq_alloc();
     seq1->_maximum = seq1->_length = 2;
     seq1->_buffer = ODsample_seqtest_sampleseq_allocbuf(2);
     for( i = 0; i < 2; i++ ){
          smp = &(seq1->_buffer[i]);
          *smp = i;
     }
     CORBA_sequence_set_release( seq1, CORBA_TRUE )

/* Processing of "inout" parameter */
     seq3 = ODsample_seqtest_sampleseq_alloc();
     seq3->_maximum = seq3->_length = 3;
     seq3->_buffer = ODsample_seqtest_sampleseq_allocbuf(3);
```

```
    for( i = 0; i < 3; i++ ){
         smp = &(seq3->_buffer[i]);
         *smp = i*10;
    }
    CORBA_sequence_set_release( seq3, CORBA_TRUE )

    seq0 = ODsample_seqtest_op1( obj, seq1, &seq2, seq3, &env );

    CORBA_free( seq0 );
    CORBA_free( seq1 );
    CORBA_free( seq2 );
    CORBA_free( seq3 );
```

Parameters Handled by Server Applications

The following table shows how the server application parameters are handled.

Table 4.13 Server Parameter Area (Sequence Type)

Parameter	Parameter passed from client	Parameter passed to client
in	Each area (sequence structure area or data area) is automatically allocated or released by the skeleton. When the _buffer type of the sequence type is sequence or any type and _buffer is not NULL, set CORBA_TRUE in the release flag for the _buffer area.	_
inout	Each area (sequence structure area or data area) is automatically allocated by the skeleton. When the _buffer type of the sequence type is sequence or any type and _buffer is not NULL, set CORBA_TRUE in the release flag for the _buffer area.	The data area (_buffer) is allocated once by CORBA_free() and re-allocated by the XX_allocbuf function. The area is automatically released by the skeleton. Set CORBA_TRUE in the release flag for the area of sequence or any type that was allocated by the XX_allocbuf function.
out return	_	The sequence structure area or data area (_buffer) is allocated by the XX_alloc function or the XX_allocbuf function. The area is automatically released by the skeleton. Set CORBA_TRUE in the release flag for the area of the sequence or any type that was allocated by the XX_allocbuf function.

The following is an example of server application processing.

```
ODsample_seqtest_sampleseq
*ODsample_seqtest_op1(
     ODsample_seqtest               obj,
     ODsample_seqtest_sampleseq     *seq1,
     ODsample_seqtest_sampleseq     **seq2,
     ODsample_seqtest_sampleseq     *seq3,
     CORBA_Environment              *env )
{
     ODsample_seqtest_sampleseq     *seq;
     CORBA_long                     *smp;
     int                            i;

/* Processing of "out" parameter */
```

```
     seq = *seq2 = ODsample_seqtest_sampleseq_alloc();
     seq->_maximum = seq->_length = 4;
     seq->_buffer = ODsample_seqtest_sampleseq_allocbuf(4);
     for( i = 0; i < 4; i++ ){
          smp = &(seq->_buffer[i]);
          *smp = i*100;
     }
     CORBA_sequence_set_release( seq2, CORBA_TRUE );

/* Processing of "inout" parameter */
     CORBA_free( seq3->_buffer );
     seq3->_maximum = seq3->_length = 5;
     seq3->_buffer = ODsample_seqtest_sampleseq_allocbuf(5);
     for( i = 0; i < 5; i++ ){
          smp = &(seq3->_buffer[i]);
          *smp = i*1000;
     }
     CORBA_sequence_set_release( seq3, CORBA_TRUE );

/* Processing of return value */
     seq = ODsample_seqtest_sampleseq_alloc();
     seq->_maximum = seq->_length = 6;
     seq->_buffer = ODsample_seqtest_sampleseq_allocbuf(6);
     for( i = 0; i < 6; i++ ){
          smp = &(seq->_buffer[i]);
          *smp = i*10000;
     }
     CORBA_sequence_set_release( seq, CORBA_TRUE );
     return( seq );
}
```

4.7.6 Structure Type

This section describes structure type data.

IDL Mapping

When "struct" (structure type) is specified in IDL, data in C must be declared using struct.

The IDL compiler generates a function for allocating the location of the structure. This function is called the XX_alloc function, and connects the module name, interface name, structure name, and alloc using underscore ("_").

IDL

```
module ODsample{
     struct samplefix {          /* Structure (fixed length) */
                       long         para1;
                       long         para2;
     };
     struct samplevar {          /* Structure (variable length) */
                       long         para1;
                       string       para2;
     };
     interface         structtest{
                       samplefix    op2(
                                    in samplefix str1,
                                    out samplefix str2,
                                    inout samplefix str3
                       );
                       samplevar    op1(
                                    in samplevar str1,
                                    out samplevar str2,
                                    inout samplevar str3
```

```
          );
     };
};
```

C Language

```
typedef struct ODsample_samplefix{          /* Structure (fixed length) */
     CORBA_long     para1;
     CORBA_long     para2;
} ODsample_samplefix;

typedef struct ODsample_samplevar{          /* Structure (variable length) */
     CORBA_long     para1;
     CORBA_char     *para2;
} ODsample_samplevar;
```

Fixed-length Parameters Handled by Client Applications

A client application does not need specific location allocation/release functions when processing in, out and inout parameters of a fixed-length structure. You can specify the structure address as a function parameter.

```
    ODsample_samplefix     fix0, fix1, fix2, fix3;
    CORBA_Environment      env;
    CORBA_object           obj;

    fix1.para1 = 10;       /* Setting of "in" parameter */
    fix1.para2 = 11;       /* Setting of "in" parameter */
    fix3.para1 = 20;       /* Setting of "inout" parameter */
    fix3.para2 = 21;       /* Setting of "inout" parameter */
    fix0 = ODsample_structtest_op2( obj, &fix1, &fix2, &fix3, &env );
```

Variable-length Parameters Handled by Client Applications

The following table shows how the client application parameters are handled.

Table 4.14 Client Parameter Area (Structure or Variable-Length Data Area)

Parameter	Parameter passed to server	Parameter passed from server
in	The XX_alloc function or the data area allocation function is used to allocate the structure area or the variable-length data area.	_
inout	Same as the *in* parameter	The area is automatically allocated by the stub.
out return	_	Same as the *inout* parameter

When an area that has been dynamically allocated by the client and stub is no longer required, it must be released by CORBA_free(). The variable-length data area is also released when CORBA_free() is issued.

The following is an example of client application processing.

```
ODsample_samplevar         *var1, *var, *var3, *ret;
     CORBA_Environment     env;
     CORBA_Object          obj;
```

```
    var1 = ODsample_samplevar_alloc();
                                    /* Allocation of area for "in" parameter */
    var1->para1 = 5;                /* Setting of "in" parameter */
    var1->para2 = CORBA_string_alloc(4);
                        /* Allocation of area for "in" parameter */
    strcpy( var1->para2, "test" );     /* Setting of "in" parameter */
    var3 = ODsample_samplevar_alloc();
                        /* Allocation of area for "inout" parameter */
    var3->para1 = 4;                /* Setting of "inout" parameter */
    var3->para2 = CORBA_string_alloc(3);
                                    /* Allocation of area for "inout" parameter */
    strcpy( var3->para2, "pro" );      /* Setting of "inout" parameter */
    var0 = ODsample_structtest_op1( obj, var1, &var2, var3, &env );
    CORBA_free( var0 );             /* Release of area for return value */
    CORBA_free( var1 );             /* Release of area for "in" parameter */
    CORBA_free( var2 );             /* Release of area for "out" parameter */
    CORBA_free( var3 );             /* Release of area for "inout" parameter */
```

Fixed-length Parameters Handled by Server Applications

A server application does not need specific location allocation/release functions when processing in, out and inout parameters of a fixed-length structure. When using the out or inout parameter, the processing results can be set in the structure member.

```
ODsample_samplefix
ODsample_structtest_op2(
     CORBA_Object     obj,
     ODsample_samplefix     *str1,
     ODsample_samplefix     *str2,
     ODsample_samplefix     *str3,
     CORBA_Environment      *env )
{
     ODsample_samplefix     fix;

/* Processing of "out" parameter */
     str2->para1 = 0;
     str2->para2 = 1;

/* Processing of "inout" parameter */
     str3->para1 = 2;
     str3->para2 = 3;

/* Processing of return value */
     fix.para1 = 4;
     fix.para2 = 5;
     return( fix );
}
```

Variable-length Parameters Handled by Server Applications

The following table shows how the server application parameters are handled.

Table 4.15 Server Parameter Area (Structure or Variable-length Data Area)

Parameter	Parameter area passed from client	Parameter area passed to client
in	The structure area or the variable-length data area is automatically allocated or released by the skeleton.	_
inout	The structure area or the variable-length data area is automatically allocated by the skeleton.	The variable-length data area is released once by CORBA_free() and re-allocated by the data area allocation function.

Parameter	Parameter area passed from client	Parameter area passed to client
		The area is automatically released by the skeleton.
out return	_	The structure area or the variable-length data area is allocated by the XX_alloc function or the data area allocation function. The area is automatically released by the skeleton.

The following is an example of server application processing.

```
ODsample_samplevar
*ODsample_structtest_op1(
     ODsample_structtest     obj,
     ODsample_samplevar      *str1,
     ODsample_samplevar      **str2,
     ODsample_samplevar      *str3,
     CORBA_Environment       *env )
{
     ODsample_samplevar      *str;

     print_strvar( str1 );
     print_strvar( str3 );

/* Processing of "out" parameter */
     *str2 = ODsample_samplevar_alloc();
     (*str2)->para1 = 12;
     (*str2)->para2 = CORBA_string_alloc( 16 );
     strcpy( (*str2)->para2, "(*str2)->para2" );

/* Processing of "inout" parameter */
     CORBA_free( str3->para2 );
     str3->para1 = 12;
     str3->para2 = CORBA_string_alloc(16);
     strcpy( str3->para2, "str3->para2" );

/* Processing of return value */
     str = ODsample_samplevar_alloc();
     str->para1 = 11;
     str->para2 = CORBA_string_alloc(16);
     strcpy( str->para2, "str->para2" );
     return( str );
}
```

4.7.7 Union Type

This section describes union type data.

IDL Mapping

When union (union type) is specified in IDL, the data is mapped to a structure (union type structure) consisting of data type discriminator information ("_d") and a union data area ("_u") in C language. A function for allocating the area for the union type structure is also generated. (The function name is generated using "module name_interface name_union name_alloc", and the function is called XX_alloc.)

The structure of a union type structure is shown below.

```
struct {                          /* Union type structure */
      CORBA_long  _d;             /* Data type discriminator information */
      union {                     /* Union data area */
             CORBA_long    xxx;
           :
```

```
          } _u;
     } ;
```

This is explained using the following IDL definition example.

IDL

```
     module ODsample{

     union samplefix switch(long){          /* Union (fixed length) */
          case 1:     long     para1;
          case 2:     long     para2;
     };
     union samplevar switch(long){          /* Union (variable length) */
          case 1:     long     para1;
          case 2:     string   para2;
     };
     interface     uniontest{
          samplefix   op2(
                      in samplefix uni1,
                      out samplefix uni2,
                      inout samplefix uni3
                          );
          samplevar   op1(
                      in samplevar uni1,
                      out samplevar uni2,
                      inout samplevar uni3
                          );
     };
};
```

C Language

```
typedef struct ODsample_samplefix{          /* Union (fixed length) */
     CORBA_long     _d;
     union {
          CORBA_long     para1;
          CORBA_long     para2;
     } _u;
} ODsample_samplefix;

typedef struct ODsample_samplevar{          /* Union (variable length) */
          CORBA_long     _d;
          union {
          CORBA_long     para1;
          CORBA_char     *para2;
     } _u;
} ODsample_samplevar;
```

Fixed-length Parameters Handled by Client Applications

It is not necessary to allocate or release the structure location. The data type identification value and corresponding data type value to be assigned must be set within the structure.

```
CORBA_ORB     orb;
CORBA_Environment      env;
ODsample_samplefix     unif0, unif1, unif2, unif3;

unif1._d = 2;               /* Setting of "in" parameter discriminator */
unif1._u.para2 = 'z';       /* Setting of "in" parameter value */
```

```
unif3._d = 2;               /* Setting of "inout" parameter discriminator */
unif3._u.para2 = 'y';       /* Setting of "inout" parameter value */
unif0 = ODsample_uniontest_op2( obj, &unif1, &unif2, &unif3, &env );
```

Variable-length Parameters Handled by Client Applications

The following table shows how the client application parameters are handled.

Table 4.16 Client Parameter Area (Union Type or Variable-length Data Area)

Parameter	Parameter passed to server	Parameter passed from server
in	The XX_alloc function or the data area allocation function is used to allocate the structure area or the variable-length data area. Sets the data type identification value and data.	_
inout	Same as the *in* parameter	The area is automatically allocated by the stub.
out return	_	Same as the *inout* parameter

When an area that has been dynamically allocated by the client and stub is no longer required, it must be released by CORBA_free(). The variable-length data area is also released when CORBA_free() is issued.

The following is an example of client application processing.

```
CORBA_ORB             orb;
CORBA_Environment     env;
ODsample_samplevar    *uni0, *uni1, *uni2, *uni3;

uni1 = ODsample_samplevar_alloc();
uni1->_d = 1;               /* Setting of "in" parameter discriminator */
uni1->_u.para1 = 10;        /* Setting of "in" parameter value */
uni3 = ODsample_samplevar_alloc();
                            /* Allocation of area for "inout" parameter */
uni3->_d = 2;               /* Setting of "inout" parameter discriminator */
uni3->_u.para2 = CORBA_string_alloc(11);
                            /* Allocation of area for "inout" parameter */
strcpy( uni3->_u.para2, "INOUT:para2" );
                            /* Setting of "inout" parameter value */

uni0 = ODsample_uniontest_op1( obj, uni1, &uni2, uni3, &env );
CORBA_free( uni0 );         /* Release of area for "in" parameter */
CORBA_free( uni1 );         /* Release of area for "out" parameter */
CORBA_free( uni2 );         /* Release of area for "inout" parameter */
CORBA_free( uni3 );         /* Release of area for return value */
```

Fixed-length Parameters Handled by Server Applications

You do not need to allocate or release the structure location when setting server-application processing results as out or inout parameters in the union. You must set the data-type identification value and corresponding data type value to be assigned.

```
ODsample_samplefix
ODsample_uniontest_op2(
     CORBA_Object     obj,
     ODsample_samplefix     *uni1,
     ODsample_samplefix     *uni2,
```

```
     ODsample_samplefix     *uni3,
     CORBA_Environment      *env )
{
     ODsample_samplefix     uni;

/* Processing of "out" parameter */
     uni2->_d = 1;
     uni2->_u.para1 = 10;

/* Processing of "inout" parameter */
     uni3->_d = 2;
     uni3->_u.para2 = 'c';

/* Processing of return value */
     uni._d = 1;
     uni._u.para1 = 100;
     return( uni );
}
```

Variable-length Parameters Handled by Server Applications

The following table shows how the server application parameters are handled.

Table 4.17 Server Parameter Area (Union or Variable-Length Data Area)

Parameter	Parameter area passed from client	Parameter area passed to client
in	The structure area or the variable-length data area is automatically allocated or released by the skeleton.	_
inout	The structure area or the variable-length data area is automatically allocated by the skeleton.	The variable-length data area is released once by CORBA_free () and re-allocated by the data area allocation function. The area is automatically released by the skeleton.
out return	_	The structure area or the variable-length data area is allocated by the XX_alloc function or the data area allocation function. The area is automatically released by the skeleton.

The following is an example of server application processing.

```
ODsample_samplevar
*ODsample_uniontest_op1(
     ODsample_uniontest     obj,
     ODsample_samplevar     *uni1,
     ODsample_samplevar     **uni2,
     ODsample_samplevar     *uni3,
     CORBA_Environment      *env )
{
     ODsample_samplevar     *uni;

/* Processing of "out" parameter */
     *uni2 = ODsample_samplevar_alloc();
     (*uni2)->_d = 2;
     (*uni2)->_u.para2 = CORBA_string_alloc(9);
     strcpy( (*uni2)->_u.para2, "OUT:param" );

/* Processing of "inout" parameter */
     if( uni3->_d == 2 )
          CORBA_free( uni3->_u.para2 );
```

```
     uni3->_d = 1;
     uni3->_u.para1 = 10;

/* Processing of return value */
     uni = ODsample_samplevar_alloc();
     uni->_d = 1;
     uni->_u.para1 = 30;
     return( uni );
}
```

4.7.8 Array

This section describes array type data.

IDL Mapping

When an array is specified in IDL, the data is also mapped to an array in C language. An array slice and an array slice area allocation function are generated. (With an array slice, dimension information is deleted from the head of the array. The array slice name is generated using "module name_interface name_array name_slice". The allocation function name is generated using "module name_interface name_array name_alloc", and the function is called XX_alloc.)

This is explained using the following IDL definition example:

IDL

```
module ODsample{
     interface     arraytest{
          typedef  long     fix[4][3][2];        /* Array (fixed length) */
          typedef  string   str[2][3][4];        /* Array (variable length) */
          fix      op1(in fix para1, out fix para2, inout fix para3 );
          str      op2(in str para1, out str para2, inout str para3 );
     };
};
```

C Language

```
typedef CORBA_long     ODsample_arraytest_fix[4][3][2];    /* Array (fixed length) */
typedef CORBA_long     ODsample_arraytest_fix_slice[3][2];

typedef CORBA_char     *ODsample_arraytest_str[2][3][4];   /*Array (variable length)*/
typedef CORBA_char     *ODsample_arraytest_str_slice[3][4];
```

Fixed-length Parameters Handled by Client Applications

When using the in, out and inout parameters, you do not need to release locations. For a return value, the array slice location is allocated in the stub using the XX_alloc function. When this location is no longer necessary, the client application must release it using the CORBA_free() function.

```
     ODsample_arraytest_fix           fix1, fix2, fix3;
     ODsample_arraytest_fix_slice     *ret;
     CORBA_Object                     obj;
     CORBA_Environment                env;
     int                              i, j, k;

     for( i = 0; i < 4; i++ )
     for( j = 0; j < 3; j++ )
     for( k = 0; k < 2; k++ ){
          fix1[i][j][k] = i;
          fix3[i][j][k] = i*10;
     }
```

```
    ret = ODsample_arraytest_op1( obj, fix1, fix2, fix3, &env );
    CORBA_free( ret );
```

Variable-length Parameters Handled by Client Applications

When using in and inout parameters, you do not need to allocate or release locations. For an out parameter or a return value, the array slice location is allocated in the stub using the XX_alloc function. When this location is no longer required, the client application must release it using the CORBA_free() function.

```
    ODsample_arraytest_str          str1, str3;
    ODsample_arraytest_str_slice    *str2, *ret;
    CORBA_Object                    obj;
    CORBA_Environment               env;
    int                             i, j, k;
    CORBA_string                    buf[];

    for( i = 0; i < 2; i++ )
    for( j = 0; j < 3; j++ )
    for( k = 0; k < 4; k++ ){
         sprintf( buf, "str1[i][j][k]", i, j, k );
         str1[i][j][k] = CORBA_string_alloc(strlen(buf));
         strcpy( str1[i][j][k], buf );
         sprintf( buf, "str3[i][j][k]", i, j, k );
         str3[i][j][k] = CORBA_string_alloc(strlen(buf));
         strcpy( str3[i][j][k], buf );
    }
    ret = ODsample_arraytest_op2( obj, str1, &str2, str3, &env );
    for( i = 0; i < 2; i++ )
    for( j = 0; j < 3; j++ )
    for( k = 0; k < 4; k++ ) {
         CORBA_free(str1[i][j][k]);
         CORBA_free(str3[i][j][k]);
    }
    CORBA_free( ret );
    CORBA_free( str2 );
```

Fixed-length Parameters Handled by Server Applications

When a server application is processing in, "inout," and out parameters of a fixed-length array, you do not need to use specific location allocation/release functions. When processing an out or inout parameter, the processing results can be set in the array member. To notify the return value, the required location must be allocated using the XX_alloc function. The return value is set in the location, then returned. The location is then released during skeleton processing.

```
ODsample_arraytest_fix_slice
*ODsample_arraytest_op1(
     ODsample_arraytest             obj,
     ODsample_arraytest_fix         para1,
     ODsample_arraytest_fix         para2,
     ODsample_arraytest_fix         para3,
     CORBA_Environment              *env )
{
     ODsample_arraytest_fix_slice   *para;
     int i, j, k;


     /* Processing of "out" parameter */
     for( i = 0; i < 4; i++ )
     for( j = 0; j < 3; j++ )
     for( k = 0; k < 2; k++ )
          para2[i][j][k] = i+j+k;
```

```
     /* Processing of "inout" parameter */
     for( i = 0; i < 4; i++ )
     for( j = 0; j < 3; j++ )
     for( k = 0; k < 2; k++ )
          para3[i][j][k] = (i+j+k)*10;

     /* Processing of return value */
     para = ODsample_arraytest_fix_alloc();
     for( j = 0; j < 3; j++ )
     for( k = 0; k < 2; k++ )
          para[j][k] = j*k;
     return( para );
}
```

Variable-length Parameters Handled by Server Applications

When in and inout parameters of a variable-length array are being processed by the server application, you do not need to use specific location allocation/release functions. When processing an out parameter or a return value, you must allocate the required location using the XX_alloc function, and data must be assigned to that location. When processing an inout parameter, the in-structure variable-length data location passed from the client must be released beforehand using the CORBA_free() function.

```
ODsample_arraytest_str_slice
*ODsample_arraytest_op2(
     CORBA_Object                    obj,
     ODsample_arraytest_str          para1,
     ODsample_arraytest_str_slice    **para2,
     ODsample_arraytest_str          para3,
     CORBA_Environment               *env )
{
     CORBA_string                    str;
     ODsample_arraytest_str_slice    *para;
     int                             i, j, k;
     char                            buf[120];

     /* Processing of "out" parameter */
     *para2 = ODsample_arraytest_str_alloc();
     for( j = 0; j < 3; j++ )
     for( k = 0; k < 4; k++ ){
          sprintf(buf, "(*para2)[%d][%d]", j, k );
          (*para2)[j][k] = CORBA_string_alloc(strlen(buf));
          strcpy( (*para2)[j][k], buf );
     }

     /* Processing of "inout" parameter */
     for( i = 0; i < 2; i++ )
     for( j = 0; j < 3; j++ )
     for( k = 0; k < 4; k++ )
          CORBA_free( para3[i][j][k] );
     for( i = 0; i < 2; i++ )
     for( j = 0; j < 3; j++ )
     for( k = 0; k < 4; k++ ){
          sprintf(buf, "para3[%d][%d][%d]", i, j, k );
          para3[i][j][k] = CORBA_string_alloc(strlen(buf));
          strcpy( para3[i][j][k], buf );
     }

     /* Processing of return value */
     para = ODsample_arraytest_str_alloc();
     for( j = 0; j < 3; j++ )
     for( k = 0; k < 4; k++ ){
```

```
          sprintf(buf, "para[%d][%d]", j, k );
          para[j][k] = CORBA_string_alloc(strlen(buf));
          strcpy( para[j][k], buf );
     }
     return( para );
}
```

4.7.9 Mapping Attribute Declaration (Attribute)

This section describes attribute declarations.

IDL Mapping

When an attribute declaration (attribute) is specified in IDL, a function is generated for setting or obtaining object data. (The function name is generated using "module name_interface name_set (get)_variable name", and the function is called "data set" or "data get".)

This is explained using the following IDL definition example:

IDL

```
module ODsample{
       interface  attrtest{
                           attribute long  para1;
                           attribute string para2;
                           readonly attribute long para3;
                          };
               };
```

C Language

```
ODsample_attrtest_set_para1( obj, 2, &env );
ODsample_attrtest_get_para1( obj, &env );

ODsample_attrtest_set_para2( obj, "test", &env );
ODsample_attrtest_get_para2( obj, &env );

ODsample_attrtest_get_para3( obj, &env );
```

Parameters Handled by Client Applications

The data area used to set data in the data set function is handled in the same way as the in parameter, and the data area used to obtain data in the data get function is handled in the same way as the "return" parameter. For variable-length data, the area which is no longer required must be released by CORBA_free().

```
     CORBA_ORB             orb;
     CORBA_Environment     env;
     CORBA_long            ret;
     CORBA_string          str;

     ODsample_attrtest_set_para1( obj, 2, &env );
     ret = ODsample_attrtest_get_para1( obj, &env );
     printf( "ODsample_attrtest_get_para1 returns [%d]\n",  ret );

     ODsample_attrtest_set_para2( obj, "test", &env );
     str = ODsample_attrtest_get_para2( obj, &env );
     env_check( "ODsample_attrtest_get_para2" );
     printf( "ODsample_attrtest_get_para2 returns [%s]\n", str );
     CORBA_free( str );

     ret = ODsample_attrtest_get_para3( obj, &env );
     printf( "ODsample_attrtest_get_para3 returns [%d]\n", ret );
```

Parameters Handled by Server Applications

When the "string" location allocated using the get function is no longer required, the client application must release the location using the CORBA_free() function.

When "attribute" is being processed by the server application and a basic type in parameter (e.g. long type) the set or get function is used. No special location allocation/release functions are required. For "string" processing, the location must be allocated for out parameter processing, released, and then reallocated for inout parameter processing. Data is set in the location, then returned. The allocated location is released during skeleton processing.

```
void
ODsample_attrtest_set_para1(
        ODsample_attrtest      obj,
        CORBA_long             para,
        CORBA_Environment      *env )
{
        para1 = para;
}

CORBA_long
ODsample_attrtest_get_para1(
        ODsample_attrtest      obj,
        CORBA_Environment      *env )
{
        return para1 ;
}

void ODsample_attrtest_set_para2(
        ODsample_attrtest      obj,
        CORBA_string           para,
        CORBA_Environment      *env )
{
        CORBA_free( para2 );
        para2 = CORBA_string_alloc( strlen( para ) );
        strcpy( para2, para );
        return;
}

CORBA_string
ODsample_attrtest_get_para2(
        ODsample_attrtest      obj,
        CORBA_Environment      *env )
{
        CORBA_string    ret;
        if( para2 == NULL ) {
                ret = CORBA_string_alloc(1);
                ret[0] = '\0';
                return( ret );
        }
        else {
                ret = CORBA_string_alloc( strlen(para2)+1 );
                strcpy( ret, para2 );
                return( ret );
        }
}

CORBA_long ODsample_attrtest__get_para3(
        ODsample_attrtest      obj,
        CORBA_Environment      *env )
{
        return( para3 );
}
```

4.7.10 Allocating and Releasing Parameter Area Using Dynamic Interface

This section describes how to create parameters using Dynamic Invocation Interface (DII). Set parameters by CORBA_NVList_add_item() function.

IN Mode

To pass in parameters to a server application, allocate parameter area in a client application, and set the pointer to the 4th parameter of CORBA_NVList_add_item().

```
CORBA_NVList_add_item(
     arg_list,
     name,                   /* Specify the parameter name defined in IDL */
     type,                   /* Specify the parameter's TypeCode */
     &param,                 /* Specify the pointer to the parameter area */
     sizeof( CORBA_long ),   /* Specify the parameter size */
     CORBA_ARG_IN,           /* Specify CORBA_ARG_IN */
     &env );
```

Areas allocated by a client application must be made available when they are no longer needed.

OUT Mode

To receive the server application processing result from an out parameter, the client application does not need to allocate an area. Specify CORBA_NVList_add_item() as follows.

```
CORBA_NVList_add_item(
     arg_list,
     name,                    /* Specify the parameter name defined in IDL */
     type,                    /* Specify the parameter's TypeCode */
     NULL,                    /* Specify NULL */
     0,                       /* Specify 0 */
     CORBA_ARG_OUT,           /* Specify CORBA_ARG_OUT */
     &env );
```

Data areas passed from the server application are automatically made available when the area for the NVList class is made free by CORBA_NVList_free(). Therefore, the out parameter area passed from the server application cannot be referenced after the area for the NVList class is made available. For further information on NVList object releasing, refer to "4.2.7 Deleting a Request".

INOUT Mode

To pass inout parameters to a server application, allocate parameter area in a client application.

```
CORBA_NVList_add_item(
     arg_list,
     name,                  /* Specify the parameter name defined in IDL */
     type,                  /* Specify the parameter's TypeCode */
     &param,                /* Specify the pointer to the parameter area */
     sizeof( CORBA_long ),  /* Specify the parameter size */
     CORBA_ARG_INOUT,       /* Specify CORBA_ARG_INOUT */
     &env );
```

After issuing the request, make sure the parameter areas allocated by the client application are made free when they are no longer needed.

Data areas passed from the server application are automatically made available when the area for the NVList class is made free by CORBA_NVList_free(). Therefore, the inout parameter area passed from the server application cannot be referenced after the area for the NVList class is made free. For further information on NVList object releasing, see 4.2.7 Deleting a Request.

RETURN

To receive the server application processing result from a return parameter, the client application does not need to allocate an area.

Data areas passed from the server application are automatically freed when the request object is deleted by CORBA_Request_delete().

4.7.11 Passing Parameters to a Server Application

The following table lists the valid data types (in C) for passing parameters.

Table 4.18 Valid Data Types (in C) for Passing Parameters

CORBA data type	In	Out	Inout	Return
long	long	long*	long*	long
short	short	short*	short*	short
unsigned long	unsigned long	unsigned long*	unsigned long*	unsigned long
unsigned short	unsigned short	unsigned short*	unsigned short*	unsigned short
long long	long long	long long*	long long*	long long
float	float	float*	float*	float
double	double	double*	double*	double
Windows32/64 Solaris32/64 long double	long double	long double*	long double*	long double
char	char	char*	char*	char
wchar	wchar	wchar*	wchar*	wchar
octet	octet	octet*	octet*	octet
boolean	boolean	boolean*	boolean*	boolean
enum	enum	enum*	enum*	enum
string	char*	char**	char**	char*
wstring	wchar*	wchar**	wchar**	wchar*
any	any*	any**	any*	any*
sequence	sequence*	sequence**	sequence*	sequence*
struct, fixed	struct*	struct*	struct*	struct
struct, variable	struct*	struct**	struct*	struct*
union, fixed	union*	union*	union*	union
union, variable	union*	union**	union*	union*
array, fixed	array	array	array	array slice*
array, variable	array	array slice**	array	array slice*
Object	CORBA_Object	CORBA_Object*	CORBA_Object*	CORBA_Object
TypeCode	CORBA_TypeCode	CORBA_TypeCode*	CORBA_TypeCode*	CORBA_TypeCode

Note

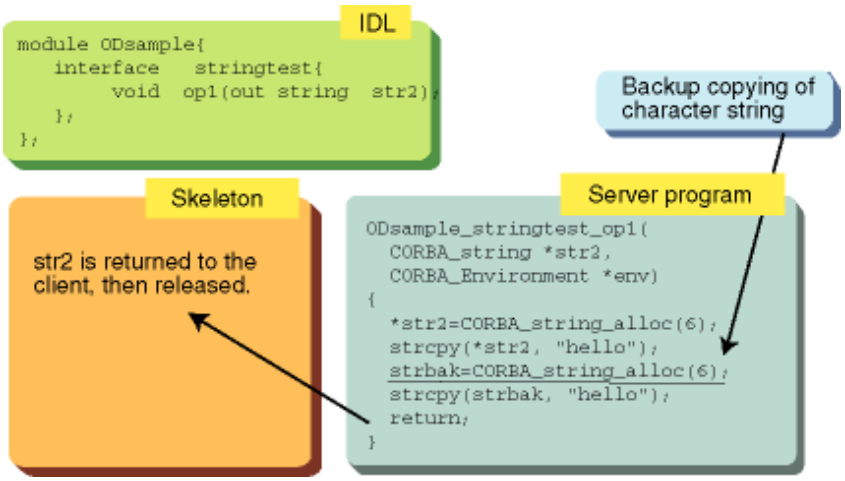
Assigning NULL pointer

NULL pointer cannot be assigned to out, inout and return parameters of server applications, and to in and inout parameters of client applications, for string, sequence, struct, union, and array types.

4.8 Any Type and Sequence Type Release Flags

Variable-length data is returned to a client, then automatically released during skeleton processing. Data not meant for release must therefore be copied to another safe location by the application. The following figure displays an example of "string" type data handling.

Figure 4.5 Releasing a String Type Data Location



The process of copying complex data (e.g., any/sequence type data) is complicated. To solve this problem, the Release flag may be set to select release or non-release processing.

The following table lists how the Release flag is set for *any* and *sequence* type variables.

Table 4.19 Release Flags for Any and Sequence Type Variables

Variable type	Setting
Any	Use CORBA_any_set_release. The default is CORBA_FALSE. (Refer to Example 1.)
Sequence	Use CORBA_sequence_set_release. The default is CORBA_FALSE. (Refer to Example 2.)

Example 1

```
CORBA_any  *any = CORBA_any_alloc();
CORBA_any_set_release(any, CORBA_TRUE);
```

Example 2

```
struct sequence<long>   data;
data = CORBA_data_alloc();
CORBA_sequence_set_release(data, CORBA_TRUE);
```

Note

When CORBA_TRUE is found: Released.

When CORBA_FALSE is found: Not released.

4.9 Notes on Application Development

Refer to the "Notes on Developing CORBA Applications" chapter.

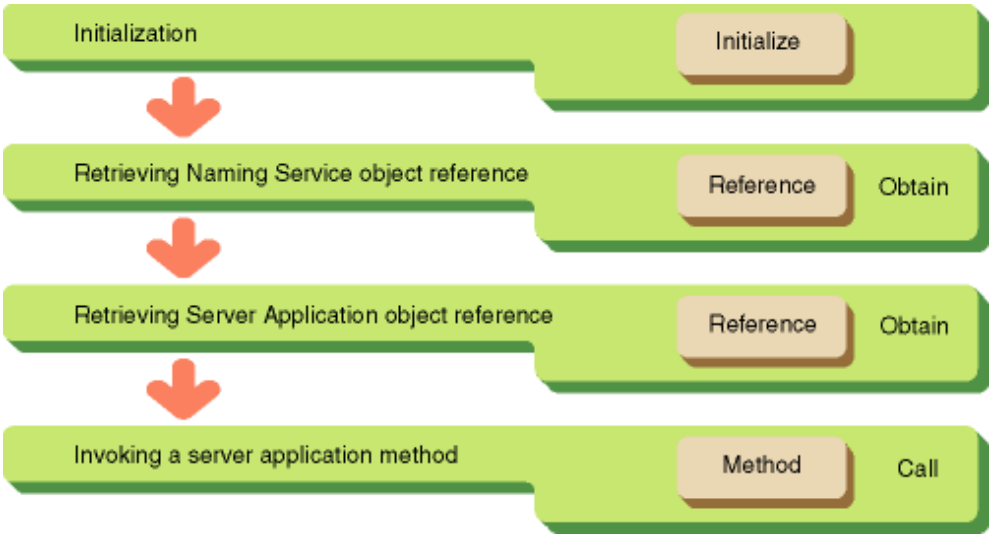
Chapter 5 C++ Programming Guide

This chapter explains how to develop CORBA applications in C++.

5.1 Client Application Programming (Static Invocation Interface)

The following figure outlines client application processing when using static invocation.

Figure 5.1 Client Application Processing when using Static Invocation



5.1.1 Initialization

To initialize an application, you must invoke *CORBA::ORB_init()*. An ORB object reference will be returned. This object reference will be used whenever an ORB interface is invoked.

```
main( int argc, char *argv[] )
{
  CORBA::ORB_ptr          orb;          // ORB ObjectReference
  CORBA::Environment_ptr  env;          // Exception
  int                     current_argc = argc;

  env = new CORBA::Environment;         // Exception
  orb = CORBA::ORB_init(current_argc, argv, FJ_OM_ORBid, *env);
```

5.1.2 Retrieving a Naming Service Object Reference

To retrieve a Naming Service object reference, the Object Reference of Naming Service is needed. Use the method in CORBA interface to retrieve Object References; *CORBA::ORB::resolve_initial_references()* to retrieve the Object Reference of Naming Service. To do that, specify *CORBA_ORB_ObjectId_NameService* as a parameter.

```
// Getting NamingService ObjectReference
CORBA::Object_ptr
obj = orb->resolve_initial_references( CORBA_ORB_ObjectId_NameService, *env );

// Casting to NamingContext class
CosNaming::NamingContext_ptr
NamingContext_obj = CosNaming::NamingContext::_narrow( obj );
```

5.1.3 Retrieving a Server Application Object Reference

To extract an Object Reference of the server application, use the Naming Service method; *CosNaming::NamingContext::resolve()*. Specify the target object name as a method parameter.

```
CosNaming::Name_ptr              name;                // CosNaming::Name
CosNaming::NameComponent_var     *name_component;     // Object name
CORBA::Object_ptr          obj;

// Getting CosNaming::NameComponent_var area
name_component = CosNaming::Name::allocbuf(1);

// Object name
name_component[0]->id = (const CORBA::Char *)"ODdemo::calculator";
// Object type
name_component[0]->kind = (const CORBA::Char *)"";

//Getting CosNaming::Name area
name = new CosNaming::Name(1,1,name_component,CORBA_TRUE );

//Getting an ObjectReference of the server application
obj = NamingContext_obj-> resolve( *name, *env );

// Casting to ODdemo::calculator class
ODdemo::calculator_ptr ap = ODdemo::calculator::_narrow( obj);
```

5.1.4 Invoking a Method Implemented in a Server Application

To invoke a method implemented in the server application, specify the method name with module names, interface names and operation names specified in IDL appended with "::"(two colons).

In the following example, those are ODdemo, calculator, and calculate respectively.

To handle an exception if it occurs between an ObjectReference retrieved from NamingService and the server application when invoking a method, specify *CORBA::Environment* structure to get it.

```
ODdemo::calculator::result     res;      // return value
CORBA::Long                    a, b;     // parameters

a = 10;
b = 5;

// Calling the method
res = ap->calculate( a, b, *env );
```

5.2 Client Application Programming (Dynamic Invocation Interface)

The following figure shows client application processing using a dynamic invocation interface.

Figure 5.2 Client Application Processing using a Dynamic Invocation Interface

5.2.1 Initialization

Invoke *CORBA::ORB_init()* to initialize the application. An ORB object reference is returned. This object reference will be used whenever an ORB interface is invoked.

```
main( argc, argv )
    int    argc;
    char   *argv[];
{
  int                  current_argc = argc;
  CORBA::Environment   env;                  // Exception
  CORBA::ORB           *orb;                 // ORB ObjectReference

  orb = CORBA::ORB_init(&current_argc, argv, FJ_OM_ORBid, env );
```

5.2.2 Retrieving a Naming Service Object Reference

To retrieve a target object to execute from Naming Service, an Object Reference of Naming Service is needed. To retrieve the Naming Service object reference use *CORBA::ORB::resolve_initial_references*. Specify *CORBA_ORB_ObjectId_NameService* as a parameter.

```
CORBA::Object_ptr        obj;                // NamingService ObjectReference

// Getting NamingService ObjectReference
obj = orb->resolve_initial_references(CORBA_ORB_ObjectId_NameService, *env);

// Casting to NamingContext class
```

```
CosNaming::NamingContext_ptr
NamingContext_obj = CosNaming::NamingContext::_narrow( obj );
```

5.2.3 Retrieving Server Application Information from the Interface Repository

Module names, interface names, operation names and parameters defined in IDL are stored hierarchically in the Interface Repository. To retrieve server application information from the Interface Repository, carry out the steps outline in this section.

(1) Retrieving an InterfaceDef Object Reference

To retrieve server application information from the Interface Repository, you need an InterfaceDef object reference. To obtain this, use the object name of the server application to search Naming Service for the server application object reference. Then, use *CORBA::Object::_get_interface()* to get the InterfaceDef object reference.

```
CORBA::Object_ptr             obj;
CosNaming::Name_ptr           name;
CosNaming::NameComponent_var  *name_component;

// Getting NamingService ObjectReference
obj = orb->resolve_initial_references(CORBA_ORB_ObjectId_NameService, *env );

CosNaming::NamingContext_ptr NamingContext
obj = CosNaming::NamingContext::_narrow( obj );

CORBA::release( obj );

// Allocating CosNaming::NameComponent_var area
name_component = CosNaming::Name::allocbuf(1);

name_component[0]->id = (const CORBA::Char *)"ODdemo::calculator";     // Object name
name_component[0]->kind = (const CORBA::Char *)"";                    // Object type
name = new CosNaming::Name(1,1,name_component,CORBA_TRUE );   // CosNaming::Name area

// Getting server application ObjectReference
obj = NamingContext_obj-> resolve( *name, *env );

// Getting InterfaceDef ObjectReference
CORBA::InterfaceDef_ptr intf = obj->_get_interface( *env );
```

(2) Retrieving an OperationDef Object Reference

To search for a specified method in the Interface Repository use *CORBA::InterfaceDef::lookup_name.* Specify a method name as a parameter in the server application. An OperationDef object reference will be returned, containing information about the specified method.

```
// Getting OperationDef method ObjectReference
CORBA::ContainedSeq_ptr intf_opr =  intf->lookup_name(
"calculate",                                  // Method name
-1,
CORBA::dk_Operation,                          // Getting operation information
CORBA_FALSE,
     *env );

// Casting to OperationDef class
CORBA::OperationDef_ptr OperationDef
     obj = CORBA::OperationDef::_narrow((*intf_opr)[0]);
```

(3) Retrieving Parameter Information

To search the Interface Repository for the parameter information of a method implemented in the server application, use *CORBA::OperationDef::describe()*. Parameter information includes parameter names, number of parameters, and parameter types. Specify the OperationDef object reference as a parameter.

```
// Retrieve method parameter information
CORBA::Contained::Description *description = OperationDef_obj->describe( *env );
```

5.2.4 Assembling Parameters

This section provides information on assembling parameters.

(1) Creating a Parameter List

To create a list object, which stores the parameters to be passed to the server, use *CORBA::ORB::create_list()*. You must specify the parameters to be stored as a parameter. An NVList object reference is then returned. For further information on the NVList object interface, refer to "NVList Object" in the "CORBA Interface" chapter.

```
CORBA::Any tmp_any =  *description->value;             // Extract parameter information
                                                       // in Contained_Description
CORBA::OperationDescription *opr_description =
        (CORBA::OperationDescription *)tmp_any.value();

// Extract parameter information
CORBA::ParDescriptionSeq    *params = &opr_description->parameters;

CORBA::NVList        *arg_list;                        // NVList

orb->create_list( params->length(), arg_list, *env );  // Generating list object
```

(2) Assigning Parameters to the List

To assign a parameter to the list to be passed to the server use *CORBA::NVList::add_value()*. You must specify the object reference for *CORBA::NVList*, the name of the server application, type, value, and length as parameters.

The order in which parameter information is set in the list object using add_value() must be consistent with the IDL definition. If the first method argument is an in-parameter, the second argument is an out-parameter and the third argument is an inout-parameter in the IDL definition, then parameter information in the list object must also be set in this order.

For details on how to set parameters, refer to "Allocating and Releasing Parameter Area Using Dynamic Interface" in "5.7 Data Type Mapping". For details on the "any" type, refer to "5.7.4 Any Type" in "5.7 Data Type Mapping".

```
CORBA::Any         p1, p2;
CORBA::Long        x = 10, y = 20;
p1 <<= x;
p2 <<= y;

// Setting a parameter
arg_list->add_value(
    ((*params)[0])->name,          // parameter name
    p1,                            // parameter date type, value
    CORBA::ARG_IN,                 // passing mode(in,out, inout)
    *env );

// Setting a parameter
 arg_list->add_value(
     ((*params)[1])->name,         // parameter name
     p2,                           // parameter data type, value
    CORBA::ARG_IN,                 // passing mode(in,out, inout)
    *env );
```

5.2.5 Creating a Request

To create a request object use *CORBA::Object::_create_request()*. You must specify an object reference for the target object, NVList and NamedValue to store the returned value. The object reference for the request object will be returned.

The request object acquired here must be deleted when it is no longer needed. For further information on object deletion, refer to "5.2.7 Deleting a Request".

```
CORBA::NVList_ptr arg_tmp;

// Generating list object for return parameter
orb->create_list(1, arg_tmp, *env);

ODdemo::calculator::result *res = NULL;
CORBA::Any any_tmp( _tc_ODdemo_calculator_result, res, CORBA_FALSE );

// Variable for result (CORBA::NamedValue type variable)
CORBA::NamedValue_ptr
result = arg_tmp->add_value(
                NULL,             // Setting result
                any_tmp,
                (CORBA::Flags)0,
                *env);

CORBA::Request_ptr     request;

// Generating request object
obj->_create_request(
        CORBA_OBJECT_NIL,         // context
        "calculate",              // method name
        arg_list,                 // input parameters
        result,                   // return value
        request,                  // area for output parameter
        CORBA::OUT_LIST_MEMORY,
        *env );
```

NamedValue is the location used to store return values from the server. The following is the IDL file definition for NamedValue.

IDL

```
module CORBA
{
    struct NamedValue
    {
        identifier     name;            // parameter name
        any            argument;        // value
        long           len;             // length
        Flags          arg_modes;       // passing mode(in,out,inout)
    };
};
```

C++

```
class CORBA
{
    class NamedValue
    {
        public:
        const char *name(CORBA::Environment &) const;     // parameter name
        Any *value(CORBA::Environment &) const;           // value
        Flags flags(CORBA::Environment &) const;          // passing mode(in,out,inout)
```

```
    };
};
```

5.2.6 Sending a Request

To send a request to server applications you can use either synchronous or asynchronous communication.

5.2.6.1 Synchronous Communication

To send a request to the server application with the synchronous communication method use *CORBA::Request::invoke()*. Specify the object reference of the request object as a parameter.

```
request->invoke( *env );
```

Asynchronous Communication

To send a request to the server application with the asynchronous communication method use *CORBA::Request::send_deferred()*. Specify the object reference of the request object as a parameter.

Then use *CORBA::Request::get_response()* to receive the response from the server application. Specify the object reference of the request object as a parameter.

```
request->send_deferred( *env );     // Sending a request
request->get_response( *env );      // Receiving a response
```

When it is found out that the request is not completed, again call CORBA::Request::get_response() method.

Processing Result Fetching

After the processing result is received, CORBA::NVList::item() is used to fetch the parameter information received from the server application. For details on parameter information fetching, refer to "Allocating and Releasing Parameter Area Using Dynamic Interface" in "5.7 Data Type Mapping".

```
CORBA::NamedValue_ptr nv;

// Processing result fetching
nv = arg_list->item(1, env);  // Fetches the 1st parameter.
CORBA::Any *ap = nv->value( env );
CORBA::Long *p = (CORBA::Long *)ap->value();

cout << *(CORBA::Long *)p << endl;   // Outputs the result.
```

5.2.7 Deleting a Request

To delete a request object use *CORBA::release()*. Specify the object reference of the request object as a parameter.

CORBA::release() is also used to delete an *NVList* object. To do this, specify the object reference of the *NVList* object.

When an NVList object is deleted, the out, inout and return parameter areas allocated on the server application side are automatically made available.

```
CORBA::release( request );          // Releasing request object
CORBA::release( arg_list);          // Releasing the NVList object for parameters
CORBA::release( arg_tmp );          // Releasing the NVList object for return parameters
```

5.3 Client Application Exception Handling

Exceptions are used to let the client application know if a request has failed. Exceptions are categorized into ORB (System) exceptions and server application (user) exceptions. System exception classes are listed in the following table.

Refer to the Reference Manual (API Edition) for the meaning of each exception.

Table 5.1 Exception Classes

Exception information	Exception class
BAD_CONTEXT	CORBA::StExcep::BAD_CONTEXT
BAD_INV_ORDER	CORBA::StExcep::BAD_INV_ORDER
BAD_OPERATION	CORBA::StExcep::BAD_OPERATION
BAD_PARAM	CORBA::StExcep::BAD_PARAM
BAD_QOS	CORBA::StExcep::BAD_QOS
BAD_TYPECODE	CORBA::StExcep::BAD_TYPECODE
CODESET_INCOMPATIBLE	CORBA::StExcep::CODESET_INCOMPATIBLE
COMM_FAILURE	CORBA::StExcep::COMM_FAILURE
CONTEXT	CORBA::StExcep::CONTEXT
DATA_CONVERSION	CORBA::StExcep::DATA_CONVERSION
FREE_MEM	CORBA::StExcep::FREE_MEM
IMP_LIMIT	CORBA::StExcep::IMP_LIMIT
INITIALIZE	CORBA::StExcep::INITIALIZE
INTERNAL	CORBA::StExcep::INTERNAL
INTF_REPOS	CORBA::StExcep::INTF_REPOS
INV_FLAG	CORBA::StExcep::INV_FLAG
INV_IDENT	CORBA::StExcep::INV_IDENT
INV_OBJREF	CORBA::StExcep::INV_OBJREF
INV_POLICY	CORBA::StExcep::INV_POLICY
MARSHAL	CORBA::StExcep::MARSHAL
NO_IMPLEMENT	CORBA::StExcep::NO_IMPLEMENT
NO_MEMORY	CORBA::StExcep::NO_MEMORY
NO_PERMISSION	CORBA::StExcep::NO_PERMISSION
NO_RESOURCES	CORBA::StExcep::NO_RESOURCES
NO_RESPONSE	CORBA::StExcep::NO_RESPONSE
OBJ_ADAPTER	CORBA::StExcep::OBJ_ADAPTER
PERSIST_STORE	CORBA::StExcep::PERSIST_STORE
REBIND	CORBA::StExcep::REBIND
TIMEOUT	CORBA::StExcep::TIMEOUT
TRANSIENT	CORBA::StExcep::TRANSIENT
UNKNOWN	CORBA::StExcep::UNKNOWN
INVALID_TRANSACTION	CORBA::StExcep::INVALID_TRANSACTION
TRANSACTION_MODE	CORBA::StExcep::TRANSACTION_MODE
TRANSACTION_REQUIRED	CORBA::StExcep::TRANSACTION_REQUIRED
TRANSACTION_ROLLEDBACK	CORBA::StExcep::TRANSACTION_ROLLEDBACK
TRANSACTION_UNAVAILABLE	CORBA::StExcep::TRANSACTION_UNAVAILABLE

CORBA::SystemException::minor() can be used to obtain minor codes when a system exception occurs. Refer to information on CORBA Service Minor Codes in the Reference Manual (API Edition) for details of minor code values.

You can identify exceptions by using the following methods:

- Using the try-catch method.
- Using information stored in the CORBA::Environment class, which was specified at the time of method invocation.

Normally, the try-catch method should be used for error judgment. If you specify the noex option for IDLc when creating a stub, information stored in the CORBA::Environment class will be used to check for the presence of an error when a method is invoked on the server application. For details on IDLc, see the Reference Manual (Command Edition).

Using try-catch

To handle exceptions using try-catch, define the corresponding catch blocks for every exception class. When an exception occurs, the specified catch block will be executed.

You specify the system exception class using *CORBA::SystemException*. Specific classes include: *CORBA::StExcep::UNKNOWN*, *CORBA::StExcep::BAD_PARAM*, etc.

For system errors, if you specify a *CORBA::SystemException* class in catch statement, the catch statement processes all system errors. If you don't need to identify system exceptions and user exceptions, specify the upper class, *CORBA::Exception* in catch statement then the catch statement catches all exceptions.

```
// Referring a user exception
CORBA::Environment    env;
CORBA::Long           l;
CORBA::ULong           minor;  //Minor code

try {
    obj->op(env);
} catch ( CORBA::SystemException &systemerr ) {
    minor = systemerr.minor(); // Obtaining minor code
    // Exception handling
} catch( ODdemo::calculator::ZEROPARAM &errcode ){
    // Exception handling
}

or

CORBA::Environment    env;
CORBA::Long           l;

try {
    obj->op(env);
} catch ( CORBA::Exception &e ) {
    // Exception handling
}
```

Using CORBA::Environment Class

To handle an exception using *CORBA::Environment class*, you must firstly identify the exceptions. *CORBA::Environment* class is shown below.

```
class Environment
{
public:
    void exception(Exception *);
    Exception *exception() const;
    void clear();
}
```

If an error occurs, query the member function exception(). If no exception has occurred, NULL is returned. If an exception has occurred one of the following exception class pointers will be assigned:

- **CORBA::SystemException**

  A system exception

- **CORBA::UserException**

  A user exception

System Exception

To identify a system exception, invoke the _narrow() method in the SystemException class, then check the result. If the result is TRUE, the exception is a System Exception class. To identify more detail exceptions, invoke _narrow() method of each detail class and check the result. If the result is TRUE, the exception is that detail class.

User Exception

To identify a user exception, invoke the _narrow() method in the UserException class, and check the result. If the result is TRUE, the exception is a UserException class. Then invoke _value() method and get detail information of the User Exception.

An example of obtaining exception information is shown below:

```
// Referring an exception
CORBA::Environment      *env;            // CORBA::Environment class
CORBA::Exception        *exc;            // CORBA::Exception class
CORBA::SystemException  *systemerr;      // CORBA::SystemException class
CORBA::UNKNOWN          *sys_unknown;    // unknown exception
CORBA::BAD_PARAM        *sys_bad_param;  // parameter exception
          .
          .
CORBA::UserException    *usererr;        // CORBA::UserException class
ODdemo::calculator::ZEROPARAM *errcode;  // exception detail
CORBA::Ulong             minor;          // minor code

env = new CORBA::Environment;
result = ap->divide(2,  3, *env);        // Calling a method
if( exc = env->exception() ){
    if( systemerr = SystemException::_narrow(exc)) {        // System exception
        minor = systemerr->minor();     // Obtaining minor code
        if( sys_unknown = CORBA::UNKNOWN::_narrow( systemerr) ){
          .               // Exception handling
          .
        }
        if( sys_bad_param = CORBA::BAD_PARAM::_narrow( systemerr) ){
          .               // Exception handling
          .
        }
    }
} else if( usererr = UserException::_narrow(exc)) {         // User exception
    if( errcode = ODdemo::calculator::ZERODIVIDE::_narrow( usererr ) ){
                          // Exception handling
    }
}
```

5.4 Server Application Programming (Static Invocation Interface)

This is not valid for Linux (64 bit).

The server application is composed of initialization and implementation components. The initialization component executes the processes shown in the following figure.

Figure 5.3 Server Application Initialization Component

5.4.1 Initialization

Initialization is carried out by invoking *CORBA::ORB_init()*. An ORB object reference is returned by this function. This object reference is specified whenever the ORB interface is invoked.

```
main ( int argc, char *argv[] )
{
CORBA::ORB_ptr            orb;          // ORB ObjectReference
CORBA::Environment_ptr    env;          // Exception information
int                       current_argc = argc;

env = new CORBA::Environment;
orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, *env );
```

Invoke *CORBA::ORB::BOA_init()*, and initialize the Basic Object Adapter.

```
CORBA::BOA_ptr            boa;          // BOA ObjectReference
boa = orb->BOA_init( current_argc, argv, CORBA_BOA_OAid, *env );
```

Then invoke application specific initialization at this point, if required.

5.4.2 Activating the Server

After initialization is complete, use *notifyORB* to communicate that the server is ready to accept requests. When ORB has been notified, it will begin to dispatch requests to the server.

Activation methods valid for various server types are listed in the following table.

Table 5.2 Activation Method of a Server

Server type	C++ method
shared server	CORBA::BOA::impl_is_ready
unshared server	CORBA::BOA::obj_is_ready
persistent server	CORBA::BOA::impl_is_ready
server per method	CORBA::BOA::impl_is_ready

(1) Retrieving the Implementation Repository Object Reference

To retrieve an Implementation Repository Object Reference, specify *CORBA_ORB_ObjectId_ImplementationRepository* as a method parameter.

(2) Searching for an ImplementationDef Object Reference

To get the ImplementationDef ObjectReference, invoke FJ::ImplementationRep::lookup_id() method. Specify the ImplementationRep ObjectReference of the server application as a parameter.

(3) Activating a Server

To activate the server, invoke *CORBA::BOA::impl_is_ready()* method or *CORBA::BOA::obj_is_ready()* method.

The method's action depends on whether the server application is implemented in single thread or multithread. If the server is implemented in multithread, information stored in *ImplementationRepository* also affects the method's action. The differences in actions are listed in the following table.

Table 5.3 Differences in Server Application Methods

Condition		Operation
Implementation by process	mode=SYNC_END for *OD_impl_inst*	The method does not return control immediately but only when the operator instructs this object to stop (*1).
Implementation by thread	mode= SYNC_END for *OD_impl_inst*	(same as above)
	mode= COMPATIBLE for *OD_impl_inst*	The method returns control immediately. Next, invoke thr_exit.

*1 After the method returns control, a deactivation process is required to close the file opened during initialization and to release the location.

```
CORBA::ImplementationDef_ptr impl;            // Implementation information
ImplementationRep_ptr      impl_rep;          // ImplementationRep ObjectReference
CORBA::Object_ptr            o;         // Pointer of Object Reference

// Search for Implementation information ObjectReference
o = orb->resolve_initial_references(
    CORBA_ORB_ObjectId_ImplementationRepository, *env );

// Casting to ImplementationRep class
impl_rep = FJ::ImplementationRep::_narrow(o);

// Search for the server ImplementationRep object
o = impl_rep->lookup_id( _IMPL_ODdemo_calculator, *env );

// Casting to ImplementationDef class
impl = CORBA::ImplementationDef::_narrow(o);

// Activating the server
boa->impl_is_ready( impl, *env );
```

5.4.3 Interface Implementation Functions

After initialization routine, implement interfaces in the server application. This section describes the two methods used to implement these interfaces:

- **Impl Class Method**

  This method implements interfaces as a method of a derived class, and has the following features:

  - The interface implementation class is generated by the IDL compiler. Because the user only needs to create the interface implementation function as a method of the interface implementation class, a server application is easily created.
  - Because the interface implementation class is generated and fixed by the IDL compiler, functions that cannot be coded in IDL cannot be added.

- **TIE Class Method**

  This method implements interfaces using template class functions, and has the following features:

  - The user creates the interface implementation class and interface implementation functions as its member functions.
  - The interface implementation functions are invoked via the TIE class generated by the IDL compiler.

- In addition to the interfaces (classes) defined in IDL, any function can be implemented concurrently in the interface implementation class.

To use the TIE class method, specify the -tie option when executing the *IDLc* command.

The implementation methods are explained using the following IDL definition.

```
module ODdemo{
    interface  calculator{
        exception ZEROPARAM {};
        struct result {
            long     add_result;
            long     subtract_result;
            long     multiple_result;
            float    divide_result;
        };
        result calculate( in long a, in long b )
            raises( ZEROPARAM );
    };
};
```

Impl Class Method

When the impl class is used, the Interface Implementation Function needs to be coded.

Interface Implementation Function

Code the interface implementation function as a member function of an interface implementation class that is defined using a name consisting of "module name_interface name_impl".

```
// A header generated from IDL
class ODdemo_calculator_impl {
public:
   ODdemo::calculator::result calculate(CORBA::Long, CORBA::Long,
          CORBA::Environment & )
      throw ( CORBA::Exception );
}

// Interface implementation
ODdemo::calculator::result                  // return type
ODdemo_calculator_impl::calculate(          // add method implementation
    CORBA::Long  a,                         // input value
    CORBA::Long  b,                         // input value
    CORBA::Environment & )                  // exception value
    throw( CORBA::Exception )
{
    if( b == 0 ){                           // check 0 division
        ODdemo::calculator::ZEROPARAM  *exc =
            new ODdemo::calculator::ZEROPARAM();
        throw( *exc );
    }

    ODdemo::calculator::result res;
    res.add_result = a+b;                   // addition
    res.subtract_result = a-b;              // subtraction
    res.multiple_result = a*b;              // division
    res.divide_result = (CORBA::Float)a/b;  // multiplication
    return res;
}
```

TIE Class Method

When the TIE class is used, code the following:

- Interface implementation class
- Interface implementation function
- Interface implementation class allocation function

Interface Implementation Class

The following definitions must be included as public attributes in the interface implementation class:

- Default constructor
- Destructor
- Operation and attribute implementation functions

The operation and attribute implementation functions have the same names as the interfaces defined in the header generated by the IDL compiler. Note that "throw (CORBA::Exception)" is not required.

Other member definitions and class names can be added as required.

```
class  CalculatorImpl {
  public:
      CalculatorImpl();           // constructor declaration
      ~CalculatorImpl();          // destructor declaration
      ODdemo::calculator::result
      calculate  (                // operation declaration
            CORBA::Long    a,
            CORBA::Long    b,
            CORBA::Environment  & );
  };
```

Interface Implementation Function

Define the interface implementation function as a member function of an interface implementation class.

No special processing needs to be coded for the default constructor and destructor, but independent processing can be coded.

```
CalculatorImpl::CalculatorImpl()
  {
  }

  CalculatorImpl::~CalculatorImpl()
  {
  }

  ODdemo::calculator::result                  // return value type
  CalculatorImpl::calculate(                  // method implementation
      CORBA::Long    a,                       // input value
      CORBA::Long    b,                       // input value
      CORBA::Environment   & )                // error value
{
  if( b == 0 ){                               // 0 division check
      ODdemo::calculator::ZEROPARAM  exc;
      throw( exc );
  }
  ODdemo::calculator::result res;
  res.add_result = a+b;                       // addition result
  res.subtract_result = a-b;                  // subtraction result
  res.multiple_result = a*b;                  // multiplication result
  res.divide_result = (CORBA::Float)a/b;      // division result
  return res;
}
```

Interface Implementation Class Allocation Function

The interface implementation class allocation function generates the interface implementation class and sets instances in the TIE class. This function is automatically invoked when a request is received from the client, and an interface implementation class instance is generated.

The names for the TIE class and the interface implementation class allocation function are as follows:

TIE class:

tie interface name

Interface implementation class allocation function:

interface class_ptr _ct _ interface name()

Instances need not be generated by a default constructor; they can be generated using a parameter-specified constructor, an operator, and so on. Only one constructor can be used to set instances in the TIE class. When the TIE class is allocated, the instances in the interface implementation class must be passed as arguments.

```
ODdemo::calculator_ptr  _ct_ODdemo_calculator()
  {
      CalculatorImpl  *impl = new CalculatorImpl();
      _tie_ODdemo_calculator<CalculatorImpl>    *tie
             = new _tie_ODdemo_calculator<CalculatorImpl>(*impl);
      return  tie;
  }
```

5.4.4 Deactivating the Server

When a server application receives a request to stop, it will notify ORB that it has stopped responding to requests. When the notification has been issued, ORB will stop dispatching requests to the server, and will return exceptions to clients. The following table lists deactivation methods for various server types.

Table 5.4 Server Deactivation Methods

Server type	C++ language
Shared server	CORBA::BOA::deactive_impl
Unshared server	CORBA::BOA::deactive_obj
Persistent server	CORBA::BOA::deactive_impl
Server per method	(none)

For *CORBA::BOA::deactivate_impl()*, specify the ImplementationRep object as a parameter. For *CORBA::BOA::deactivate_obj()*, specify the object reference for the server application.

```
boa->deactivate_impl( impl, *env );          // activating the server
```

If the *isstopwu* command or the Interstage Management Console is used to stop an operating WorkUnit, the server application notifies ORB that subsequent requests from the client should not be accepted. Client applications no longer need to issue the deactivation method.

If the *odcntlque* command is used to stop server applications when the CORBA application program is operated under control of the WorkUnit, client applications no longer need to issue the deactivation method.

5.5 Server Application Exception Handling

This is not valid for Linux (64 bit).

This section contains information on programming server application exceptions.

Setting Exception Information

The IDL compiler generates exception classes with identifiers in the syntax interface *name::exception* name. The exception class which is generated will store exception members as private data. In the following example, it is ODdemo, calculator, ZEROPARAM. To set an exception value, you must create a new exception instance, and then assign values to its members. Finally you must invoke *CORBA::Environment::exception()*,and set the exception to Environment class instance.

```
ODdemo::calculator::result             // return value
ODdemo_calculator_impl::calculate(     //divide method implementation
        CORBA::Long     a,
        CORBA::Long     b,
        CORBA::Environment & )
        throw( CORBA::Exception )
{
          :
    if( b == 0 ){
        // Throws an exception class
        ODdemo::calculator::ZEROPARAM  *exc =
            new ODdemo::calculator::ZEROPARAM();
        throw( *exc );
    }
        :

    return res;
}
```

Note

To call another method from the inside of a server method, use another CORBA::Environment class. Do not use the CORBA::Environment class passed to a parameter of a server method as a parameter for calling another method. If this is done, exception information for the inside of the server method that is generated by another method will be set as the exception information.

The following example shows an acceptable programming example and an incorrect programming example:

Correct Programming Example

```
ODdemo::calculator::result
ODdemo_calculator_impl::calculate(
      CORBA::Long        a,
      CORBA::Long        b,
      CORBA::Environment &env )
      throw( CORBA::Exception )
  {
    CosNaming::NamingContext_ptr  nc;
    CosNaming::Name               name;
    CORBA::Object_ptr             obj;
    CORBA::Environment  local_env; /* */
        :

    /* The exception information for CosNaming::NamingContext::bind()
     * is set in local_env.  Thus it is not handled as the exception
     * information for ODdemo::calculator_impl::calculate().
     */
    nc->bind( name, obj, local_env );
        :
}
```

Incorrect Programming Example

```
ODdemo::calculator::result
ODdemo_calculator_impl::calculate(
      CORBA::Long        a,
```

```
        CORBA::Long        b,
        CORBA::Environment &env )
        throw( CORBA::Exception )
    {
      CosNaming::NamingContext_ptr  nc;
      CosNaming::Name               name;
      CORBA::Object_ptr             obj;
          :

      /* When an exception is returned to CosNaming::NamingContext::bind(),
       * the exception information for CosNaming::NamingContext::bind()
       * is incorrectly handled as the exception information for
       * ODdemo::calculator_impl::calculate().
       */
      nc->bind( name, obj, env );
          :
  }
```

5.5.1 Obtaining Exception Information

Obtain exception information for server applications in the same way as for client applications. Refer to "5.3 Client Application Exception Handling" for details.

5.6 Registering a Server Application

This is not valid for Linux (64 bit).

Register the created server application to the Implementation Repository and the Naming Service.

5.6.1 Registering in the Implementation Repository

To register server application information to the Implementation Repository, use the *OD_impl_inst* command. The *OD_impl_inst* command syntax and a *def* file example are shown below.

```
OD_impl_inst -ax def
```

-ax def

Specify a definition file of server application information

Sample def File Contents Windows32/64

```
rep_id = IDL:ODdemo/calculator:1.0
lang   = CPP
type   = shared
binary = D:\server\simple_s.exe
mode   = SYNC_END
ior    = 1.1
```

Sample *def* File Contents Solaris32/64 Linux32/64

```
rep_id = IDL:ODdemo/calculator:1.0
lang   = CPP
type   = shared
binary = /home/guest/simple_s
uid    = user
gid    = group
mode   = SYNC_END
ior    = 1.1
```

rep_id = IDL:ODdemo/calculator:1.0

Specify the Implementation Repository ID

lang = CPP

Specify the language type of the server application. For a C++ application, specify CPP.

type = shared

Specify the server type. Available types are: persistent, shared, unshared, server per method.

binary = D:\server\simple_s.exe Windows32/64

binary = /home/guest/simple_s Solaris32/64 Linux32/64

Specify a pathname of shared library for the server application.

uid = user Solaris32/64 Linux32/64

Specify the User ID used when executing the server application.

gid = group Solaris32/64 Linux32/64

Specify the Group ID used when executing the server application.

mode = SYNC_END

Specify a mode when the server application issues *CORBA_BOA_impl_is_ready()* function. Available modes are: COMPATIBLE and SYNC_END.

ior = 1.1

Specify IOR version. Available versions are 1.0 and 1.1.

5.6.2 Registering in the Naming Service

To make a server application available as an object to other applications, you must create an object reference to identify it. The object reference must also be registered in the Naming Service.

You can create an object by:

- Using the *OD_or_adm* command
- Using BOA functionality to create an object reference and register it in Naming Service.

Using the OD_or_adm Command

Once an object reference has been created, use the *OD_or_adm* command to register it in the Naming Service. The following example deals with registration using the *OD_or_adm* command, and information that needs to be specified:

```
OD_or_adm -c IDL:ODdemo/calculator:1.0 -n ODdemo::calculator
```

-c IDL:ODdemo/calculator:1.0

Registers an object reference using the specified interface repository ID.

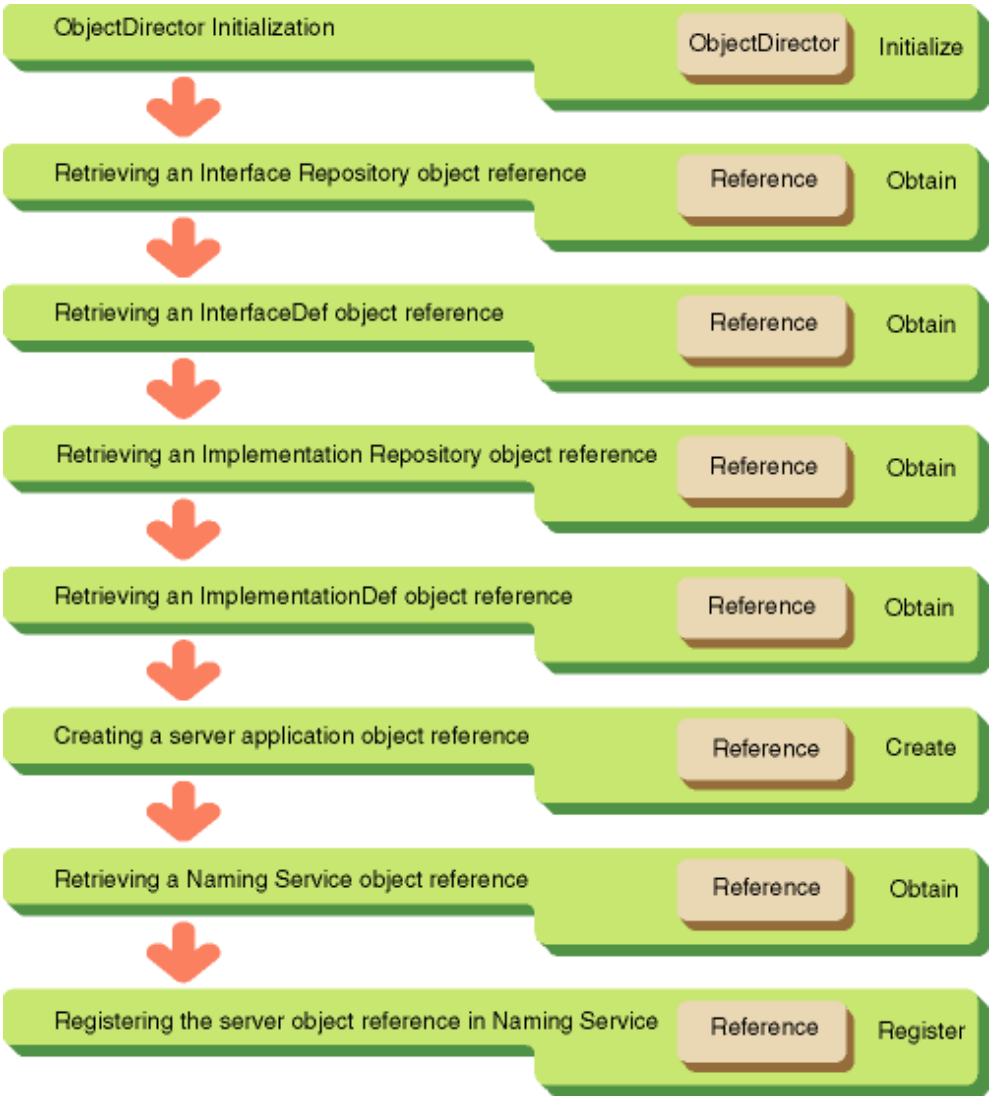
-n ODdemo::calculator

Specifies the name of the object to be registered in the Naming Service.

Creation Method in a Server Application

The following figure outlines this process.

Figure 5.4 Creation Method in a Server Application




```
CORBA::ORB_ptr               orb;     // Object returned from ORB_init
CORBA::BOA_ptr               boa;     // Object returned from ORB_BOA_init
CORBA::Repository_ptr        intf_rep;// Object reference for interface repository
CORBA::InterfaceDef_ptr      intf;    // InterfaceDef for interface repository
FJ::ImplementationRep_ptr    impl_rep;// Object reference for implementation repository
CORBA::ImplementationDef_ptr impl;   // ImplementationDef for implementation repository
CORBA::Object_ptr            new_obj; // Server application ObjectReference
CORBA::ReferenceData         id;      // ReferenceData area
CORBA_Environment  *env = new CORBA::Environment();     // Exception information

// ObjectDirector Initialization (omitted)
                :

// Getting an Interface Repository object reference
intf_rep = orb->resolve_initial_references(
    CORBA_ORB_ObjectId_LightInterfaceRepository,
    *env );

// Getting an InterfaceDef object reference
intf = intf_rep->lookup_id(
    "...",                              // Interface repository ID
    &env );
```

```
// Getting an Implementation Repository object reference
impl_rep = orb->resolve_initial_references(
    CORBA_ORB_ObjectId_ImplementationRepository,
    *env );

// Getting an ImplementationDef object reference
impl = impl_rep->lookup_id(
    "...",                              // Interface repository ID
    *env );

// Creating an ObjectReference
new_obj = boa->create( id, intf, impl , *env );

CORBA::Object_ptr obj;                // NamingService ObjectReference

// Getting NamingService ObjectReference
obj = orb->resolve_initial_references( CORBA_ORB_ObjectId_NameService, *env );

// Casting to NamingContext class
CosNaming::NamingContext_ptr NamingContext_obj =
    CosNaming::NamingContext::_narrow( obj );

CosNaming::Name_ptr          name;             // CosNaming::Name instance
CosNaming::NameComponent_var *name_component;  // object name area

// Allocating CosNaming::NameComponent_var area
name_component = CosNaming::Name::allocbuf(1);

name_component[0]->id = (const CORBA::Char *)"ODdemo::calculator";     // object name
name_component[0]->kind = (const CORBA::Char *)"";                     // object kind

// Getting CosNaming::Name area
name = CosNaming::Name(1,1,name_component,CORBA_TRUE );

// Registering server application ObjectReference
NamingContext_obj-> bind( *name, new_obj, *env );
```

CORBA_BOA_create() creates an Object Reference with its default character set registered in Implementation Repository. The default character set is specified by the *OD_impl_inst* or *OD_set_env* command.

5.7 Data Type Mapping

This section describes C++ data types used in the ObjectDirector application.

5.7.1 Basic Data Types

To use CORBA specified data types, define them as listed in the following table.

Table 5.5 Defining Basic Data Types (C++)

CORBA data	Type	C++	Comments
integer	long	CORBA::Long	
	short	CORBA::Short	
	unsigned long	CORBA::Ulong	
	unsigned short	CORBA::UShort	

CORBA data	Type	C++	Comments
	long long	CORBA::LongLong	
float	float double long double	CORBA::Float CORBA::Double CORBA::LongDouble	Windows32/64 Solaris32/64
character	char wchar	CORBA::Char CORBA::WChar	
octet	octet	CORBA::Octet	
boolean	boolean	CORBA::Boolean	
string	string wstring	CORBA::Char * CORBA::WChar *	Refer to "5.7.2 String Type" Refer to "5.7.3 Wide String Type"
enumeration	enum	enum	
any	any	CORBA::Any	Refer to "5.7.4 Any Type"
Object Reference	Object	CORBA::Object_ptr	
Type code	TypeCode	CORBA::TypeCode_ptr	

5.7.2 String Type

This section describes string type data.

IDL Mapping

String types in IDL are mapped to *CORBA::Char** in C++.

This is explained using the following IDL definition example.

IDL

```
module ODsample {
     interface     stringtest {
          string op1(in string str1, out string str2, inout string str3);
     };
};
```

C++ language

```
CORBA::Char *
ODsample::stringtest::op1(
     const CORBA::Char *str1,          /* in parameter */
     CORBA::Char  *&str2,              /* out parameter */
     CORBA::Char  *&str3,              /* inout parameter */
     CORBA::Environment  &env )        /* exception information */
```

Parameters Handled by Client Applications

The following table shows how the client application parameters are handled.

Table 5.6 Client Parameter Area (Character String Type)

Parameter	Parameter passed to server	Parameter passed from server
in	The area is allocated by CORBA::string_alloc().	_
inout	Same as the *in* parameter	The area is automatically allocated by the stub.
out	_	Same as the *inout* parameter

Parameter	Parameter passed to server	Parameter passed from server
return		

Note

When an area that has been allocated by the client and stub is no longer required, it must be released by *CORBA::string_free()*.

The following is an example of client application processing.

```
CORBA::Environment      env;
CORBA::Char     *str1, *str2, *str3, *ret;

str1 = CORBA::string_alloc(3);          /* allocate in parameter area */
strcpy( str1 ,"IN" );                   /* set in parameter */
str3 = CORBA::string_alloc(8);          /* allocate inout parameter area */
strcpy( str3 ,"INOUT:1" );              /* set inout parameter */
ret = obj->op1( str1, str2, str3, env );

CORBA::string_free( ret );              /* function to allocate return */
CORBA::string_free( str1 );             /* function to allocate in parameter */
CORBA::string_free( str2 );             /* function to allocate out parameter */
CORBA::string_free( str3 );             /* function to allocate inout parameter */
```

Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

The following table shows how the server application parameters are handled.

Table 5.7 Server Parameter Area (Character String Type)

Parameter	Parameter passed from client	Parameter passed to client
in	The area is automatically allocated or released by the skeleton.	_
inout	The area is automatically allocated by the skeleton.	When the character string to be returned is shorter than the passed parameter, the character string is set in the passed area. When the character string to be returned is longer than the passed parameter, the passed area is released by CORBA::string_free() and a new area is allocated by CORBA::string_alloc(). The passed area is automatically released by the skeleton.
out return	_	The area is allocated by CORBA::string_alloc(). This area is automatically released by the skeleton.

The following is an example of server application processing.

```
CORBA::Char *
ODsample_stringtest_impl::op1(
       const CORBA::Char   str1,     /* in parameter */
       CORBA::Char         *&str2,   /* out parameter */
       CORBA::Char         *&str3,   /* inout parameter */
       CORBA::Environment  &env )    /* exception  */
       throws( CORBA::Exception )
{
/* manipulate out parameter */
```

```
     str2 = CORBA::string_alloc(4);  /* allocate out parameter area */
     strcpy( str2, "OUT" );          /* set out parameter */

/* manipulate inout parameter */
     CORBA::string_free( str3 );     /* release in parameter */
     str3 = CORBA::string_alloc(8);  /* allocate out parameter */
     strcpy( str3, "INOUT:2" );      /* set out parameter */

/* manipulate return */
     str = CORBA::string_alloc(7);   /* allocate return area */
     strcpy( str, "RETURN" );        /* set return */
     return( str );
}
```

5.7.3 Wide String Type

This section describes wstring type data.

IDL Mapping

wstring types in IDL are mapped to *CORBA::WChar ** in C++.

This is explained using the following IDL definition example.

IDL

```
module ODsample {
     interface     wstringtest {
          string op1(in wstring str1, out wstring str2, inout wstring str3);
     };
};
```

C++ language

```
CORBA::WChar *
ODsample::wstringtest::op1(
     const CORBA::WChar *str1,         /* in parameter */
     CORBA::WChar  *&str2,             /* out parameter */
     CORBA::WChar  *&str3,             /* inout parameter */
     CORBA::Environment  &env )        /* exception information */
```

Parameters Handled by Client Applications

The following table shows how the client application parameters are handled.

Table 5.8 Client Parameter Area (Character Wide String Type)

Parameter	Parameter passed to server	Parameter passed from server
in	The area is allocated by CORBA::wstring_alloc().	_
inout	Same as the *in* parameter	The area is automatically allocated by the stub.
out return	_	Same as the *inout* parameter

Note

When an area that has been allocated by the client and stub is no longer required, it must be released by *CORBA::wstring_free()*.

The following is an example of client application processing.

```
CORBA::Environment      env;
CORBA::WChar     *str1, *str2, *str3, *ret;

str1 = CORBA::wstring_alloc(2);         /* allocate in parameter area */
str1[0] = ...;                     /* set in parameter (refer Setting a wstring)*/

str3 = CORBA::wstring_alloc(5);         /* allocate inout parameter area */
str3[0] = ...;                     /* set inout parameter (refer Setting a wstring)*/

ret = obj->op1( str1, str2, str3, env );

CORBA::wstring_free( ret );             /* function to allocate return */
CORBA::wstring_free( str1 );            /* function to allocate in parameter */
CORBA::wstring_free( str2 );            /* function to allocate out parameter */
CORBA::wstring_free( str3 );            /* function to allocate inout parameter */
```

Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

The following table shows how the server application parameters are handled.

Table 5.9 Server Parameter Area (Character Wide String Type)

Parameter	Parameter passed from client	Parameter passed to client
in	The area is automatically allocated or released by the skeleton.	_
inout	The area is automatically allocated by the skeleton.	When the character string to be returned is shorter than the passed parameter, the character string is set in the passed area. When the character string to be returned is longer than the passed parameter, the passed area is released by CORBA::wstring_free() and a new area is allocated by CORBA::wstring_alloc(). The passed area is automatically released by the skeleton.
out return	_	The area is allocated by CORBA::wstring_alloc(). This area is automatically released by the skeleton.

The following is an example of server application processing.

```
CORBA::WChar *
ODsample_wstringtest_impl::op1(
       const CORBA::WChar  str1,     /* in parameter */
       CORBA::WChar        *&str2,   /* out parameter */
       CORBA::WChar        *&str3,   /* inout parameter */
       CORBA::Environment  &env )    /* exception  */
       throws( CORBA::Exception )
{
/* manipulate out parameter */
     str2 = CORBA::wstring_alloc(3); /* allocate out parameter area */
     str2[0] = ...;                  /* set out parameter (refer Setting a wstring)*/
       :

/* manipulate inout parameter */
     CORBA::wstring_free( str3 );    /* release in parameter */
     str3 = CORBA::wstring_alloc(5); /* allocate out parameter */
```

```
     str3[0] = ...;                 /* set out parameter (refer Setting a wstring)*/

/* manipulate return */
     str = CORBA::wstring_alloc(4);  /* allocate return area */
     str[0] = ...;                  /* set return (refer Setting a wstring)*/
     return( str );
}
```

Setting a wstring

An example of setting a wstring is shown below. (The variable wstr is CORBA::WChar * type)

For EUC or ShiftJIS

Set each character separately (for EUC).

```
wstr[0] = 0xc6fc;
wstr[1] = 0xcbdc;
wstr[2] = 0xb8ec;
wstr[3] = 0x0000;    /* terminator */
```

For UNICODE on Windows®

Specify a string enclosed in "" with the prefix L. It is treated as UNICODE.

```
wstr = L" .... "
```

For UNICODE on Solaris/Linux Solaris32/64 Linux32/64

Set each character separately in UNICODE.

```
wstr[0] = 0x65e5;
wstr[1] = 0x672c;
wstr[2] = 0x8a9e;
wstr[3] = 0x0000;    /* terminator */
```

Note

When the wide string type is output to the console or a similar destination, the system may be unable to display all characters properly because the output processing depends on the operating system. For information on the output method, see the documentation for the operating system.

L "*" format strings cannot be used on Solaris and Linux systems.

5.7.4 Any Type

This section describes "any" type data.

IDL Mapping

When the any type is specified in IDL, the data is mapped to a CORBA::Any class that has the data type identification TypeCode (_tc), the data area address (_value), and the release flag (_release) as private data in C++ language. CORBA::Any class instances are generated by the new operator (C++). A CORBA_Any class method is used to access private data. The instance generation coding format is shown below.

```
CORBA::Any  *data = new CORBA::Any(
          CORBA::TypeCode_ptr  _tc,        // data type identification TypeCode
          void                 *_value,    // data area address
          CORBA::Boolean       release );  // release flag
```

IDL

```
Class CORBA {
     class Any {
                public:
          Any();                                        // default constructor
          Any( const Any & );                           // copy constructor
          Any( TypeCode_ptr tc, void *value,
Boolean release = CORBA_FALSE );
// constructor (for values which does not define type)
          ~Any();                                       // destructer
          Any     &operator=( const Any & );            // assign operator
// left shift assign operator
          void     operator<<=( Short );                // Short
          void     operator<<=( UShort );               // UShort
          void     operator<<=( Long );                 // Long
          void     operator<<=( ULong );                // Ulong
          void     operator<<=( LongLong );             // LongLong
          void     operator<<=( Float );                // Float
          void     operator<<=( Double );               // Double
          void     operator<<=( LongDouble );           // LongDouble Windows32/64 Solaris32/64
          void     operator<<=( const Any & );          // Any
          void     operator<<=( const Char * );         // Char * (String)
          void     operator<<=( const WChar * );        // WChar * (WString)

// right shift assign operator
          Boolean  operator>>=( Short & ) const;        // Short
          Boolean  operator>>=( UShort & ) const;       // UShort
          Boolean  operator>>=( Long & ) const;         // Long
          Boolean  operator>>=( ULong & ) const;        // Ulong
          Boolean  operator>>=( LongLong & ) const;     // LongLong
          Boolean  operator>>=( Float & ) const;        // Float
          Boolean  operator>>=( Double & ) const;       // Double
          Boolean  operator>>=( LongDouble & ) const;   // LongDouble Windows32/64 Solaris32/64
          Boolean  operator>>=( Any & ) const;          // Any
          Boolean  operator>>=( Char *& ) const;        // Char *
          Boolean  operator>>=( WChar *& ) const;       // WChar *
                    // helper to set boolean, octet, char
          struct     from_boolean {
               from_boolean( Boolean b ) : val(b) {}
               Boolean     val;
          };
          struct     from_octet {
               from_octet( Octet o ) : val(o) {}
               Octet     val;
          };
          struct     from_char {
               from_char( Char c ) : val(c) {}
               Char     val;
          };
          void     operator<<=( from_boolean );
          void     operator<<=( from_octet );
          void     operator<<=( from_char );
// helper to extract boolean, octet, char
          struct     to_boolean {
               to_boolean( Boolean &b ) : ref(b) {}
               Boolean     &ref;
          };
          struct     to_octet {
               to_octet( Octet &o ) : ref(o) {}
               Octet     &ref;
          };
          struct     to_char {
```

```
            to_char( Char &c ) : ref(c) {}
            Char     &ref;
        };
        Boolean  operator>>=( to_boolean ) const;
        Boolean  operator>>=( to_octet ) const;
        Boolean  operator>>=( to_char ) const;

        void     replace( TypeCode_ptr tc, void *value,
                  Boolean release = CORBA_FALSE );

        TypeCode_ptr     type() const;
        const     void     *value() const;

    private:
    TypeCode_ptr     _tc;              // TypeCode
        void         *_value;          // data value
        Boolean      release;          // storage management flag
    };
};
```

A coding example for a client/server application is shown below, using an *any* type IDL definition example.

IDL

```
module ODsample{
     struct sample1 {
          long       para1;
          string     para2;
     };
     struct sample2 {
          char       para1;
          float      para2;
     };
     struct sample3 {
          char       para1;
          double     para2;
     };
     interface       anytest{
          any        op1(in any any1, out any any2, inout any any3 );
     };
};
```

Parameters Handled by Client Applications

The following table shows how client application parameters are handled.

Table 5.10 Client Parameter Area (Any Type)

Parameter	Parameter passed to server	Parameter passed from server
in	When the area is to be allocated dynamically, an instance is generated by the new operator.	_
inout	Same as the *in* parameter.	The area is automatically allocated by the stub.
out return	_	Same as the *inout* parameter.

Note

When an area that has been allocated by the client and stub is no longer required, it must be released by the delete operator (C++). Whether or not the data area (*_value*) is to be released is specified in the release flag. The release flag is specified in the third parameter when an instance is generated in the *CORBA::Any* class.

- CORBA_TRUE: When delete is issued, the *_value* area is also released.
- CORBA_FALSE: When delete is issued, the *_value* area is not released. (Default)

The release flag for the parameter (*inout, out, return*) allocated by the stub is set in CORBA_TRUE. For details on the release flag, refer to "5.8 Any Type and Sequence Type Release Flags".

The following is an example of client application processing.

```
CORBA::Environment     env;
ODsample::anytest_ptr  obj;
ODsample::sample1      *smp1;
ODsample::sample2      *smp2;
ODsample::sample3      *smp3;
CORBA::Any             *any1, *any2, *any3, *ret;

ODsample::sample2 *smp2 = new ODsample::sample2;
                           /* allocate ODsample::sample2 */
smp2->para1 = 'z';         /* set a parameter */
smp2->para2 = 0.001;       /* set a parameter */
any1 = new CORBA::Any( _tc_ODsample_sample2, smp2, CORBA_TRUE );
                           /* allocate an in parameter and set a value */
ODsample::sample3 *smp3 = new ODsample::sample3;
                           /* allocate ODsample::sample3 */
smp3->para1 = 'x';         /* set a parameter */
smp3->para2 = 0.0001;      /* set a parameter */
any3 = new CORBA::Any();   /* allocate an inout parameter */
any3->replace( _tc_ODsample_sample3, smp3, CORBA_TRUE);
                           /* set an inout parameter */

any0 = obj->op1( *any1, any2, *any3, env );

delete ret;                /* release return area */
delete any1;               /* release an input parameter */
delete any2;               /* release an out parameter */
delete any3;               /* release an inout parameter */
```

Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

The following table shows how server application parameters are handled.

Table 5.11 Server Parameter Area (Any Type)

Parameter	Parameter passed from client	Parameter passed to client
in	The area is automatically allocated or released by the skeleton.	_
inout	The area is automatically allocated by the skeleton.	Data is replaced using the replace method. This area is automatically released by the skeleton.
out return	_	The area is allocated by the new operator. This area is automatically released by the skeleton.

The following is an example of server application processing.

```
CORBA::Any*
ODsample_anytest_impl::op1(
     const CORBA::Any       &any1,
     CORBA::Any             *&any2,
     CORBA::Any             &any3,
     CORBA::Environment     &env )
     throw( CORBA::Exception )
{
/* out parameter */
     ODsample::sample1 *smp1 = new ODsample::sample1;
                                     /* allocate ODsample::sample1 */
     smp1->para1 = 100;              /* set a value */
     CORBA::Char *str = CORBA::string_alloc(4);
     strcpy( str, "OUT" );
     smp1->para2 = str;              /* set a value */
     any2 = new CORBA::Any( _tc_ODsample_sample1, smp1, CORBA_TRUE);
                                     /* allocate an out parameter and set a value */

/* inout parameter */
     ODsample::sample2 *smp2 = new ODsample::sample2;
                                     /* allocate ODsample::sample2 */
     smp2->para1 = 'x';              /* set a value */
     smp2->para2 = 0.01;             /* set a value */
     any3->replace( _tc_ODsample_sample2, smp2, CORBA_TRUE );
                                     /* set in and out parameters */

/* return value */
     ODsample::sample3 *smp3 = new ODsample::sample3;
                                     /* allocate ODsample::sample3 */
     smp3->para1 = 'z';              /* set a value */
     smp3->para2 = 0.001;            /* set a value */
     CORBA::Any *any = new CORBA::Any( _tc_ODsample_sample3, smp3,
               CORBA_TRUE );         /* allocate return and set a value */
     return( any );
}
```

Using CORBA::*Any* Class Methods

The following table shows a description of methods defined in *CORBA::Any*

Table 5.12 CORBA::Any Class Methods

CORBA::Any Member	Description
default constructor	Creates a new instance and initializes _tc to 0, _value to 0. The release flag is set to CORBA_FALSE.
copy constructor	When an instance is created, the copy constructor allocates a data area for the instance and then copies the values passed in parameters into the data area. The release flag is set to CORBA_TRUE.
constructor	Creates a new instance and initializes _tc and _value to the specified values by copying. The release flag is set to a specified value. If no value is specified for the release flag, CORBA_FALSE is set.
destructor	If the release flag is CORBA_TRUE, releases an instance and releases _tc and _value.
assign operator	If the release flag is CORBA_TRUE, the assign operator frees the area corresponding to the current _tc and _value values. It then allocates a data area and sets the values passed in parameters. The release flag is set to CORBA_TRUE.

CORBA::Any Member	Description
left shift assign operator	If the release flag is CORBA_TRUE, the left shift assign operator frees the area corresponding to the current _tc and _value values. It then allocates a data area and sets the values passed in parameters. The release flag is set to CORBA_TRUE.
right shift assign operator	Retrieves value from _value.
helper	Identifies as Boolean, char, or octet.
left shift assign operator (for Boolean, char, octet)	Releases current_value and assigns a new value to _value.
right shift assign operator (for Boolean, char, octet)	Retrieves value from _value.
replace	If the release flag is CORBA_TRUE, the replace function frees the area corresponding to the current _value value. It then assigns the values passed in parameters to _tc and _value without allocating a new area. The release flag is set to a specified value. If no value is specified for the release flag, CORBA_FALSE is set.
type	Returns an object reference of _tc in TypeCode.
*value	The value function returns a void* pointer to the data area corresponding to the _value value.

Default Constructor

The default constructor initializes _to to 0 (*TypeCode::_nil()*), _value to 0, and the release flag to *CORBA_TRUE*. An example of the default constructor is shown below.

```
// Usage
CORBA::Any a;                       // initialize with default constructor

otherwise

CORBA::Any *b = new CORBA::Any;     // initialize with default constructor

// Initialization in default constructor
Any::Any()
{
      _tc = TypeCode::_nil();
      _value = 0;
      _release = CORBA_TRUE;
};
```

Copy Constructor

An example of copy constructor is shown below.

```
//Example
CORBA::Any a;          // initialize with default constructor

CORBA::Long x = 3;
a <<= x;               /* left shift assign operator. Set tc_long to _tc,
                       3 to _value, CORBA_TRUE to release flag */

CORBA::Any *b = new CORBA::Any(a);
     /* copy private data (_tc,_value) of a
        and set CORBA_TRUE to release flag */

CORBA::Any c(*b);      //copy constructor

// Actions in copy constructor
```

```
Any::Any( const Any &a )
{
     _tc = TypeCode::_duplicate(a.type());
     _value = ...      // Allocates an area and copies _value data passed in a.
     _release = CORBA_TRUE;
};
```

Assign Operator

An example of the assign operator is shown below.

```
// Usage
CORBA::Any a1;                    // initialize with default constructor
CORBA::Any a2;                    // initialize with default constructor

CORBA::Long x = 10;
a1 <<= x;                         // The left shift assign operator assigns _tc_long to _tc,
                                  // allocates an area for CORBA::Long in _value
                                  // and sets a value of 10,
                                  // and then sets the release flag to CORBA_TRUE.

// Actions in assign operator
Any &
Any::operator=( const Any &r )
{
     if( _release && _value )
          delete value;           // When release flag is CORBA_TRUE,
                                  // release current data
     CORBA::release( _tc );

     TypeCode::_release _tc;
     _tc = TypeCode::_duplicate(a.type());
     _value = ...                 // Allocates an area and copies the _value value passed in r.
     _release = CORBA_TRUE;
     return this;
}
```

Destructor

An example of destructor is shown below.

```
// Usage
void X()
{
     CORBA::Any *a = new CORBA::Any();

     CORBA::Long x = 3;
     (*a) <<= x;               // left shift assign operator

     delete a;                 // destructor
}

// Actions in destructor
Any:: ~Any( )0
{
     TypeCode::_release _tc;
     if( _release && _value )
          delete _value;       // When release flag is CORBA_TREU, release it
     CORBA::release( _tc );
}
```

Left Shift Assign Operator

An example of the left shift assign operator is shown below

```
// Usage
CORBA::Long value = 42;
Any a;
a <<= value;            // set tc_long to_tc, 42 to _value, CORBA_TRUE to release flag

// Actions in left shift assign operator
void CORBA::Any::operator<<=( Long v )
{
     if( _release && _value )
          delete _value;       // When release flag is CORBA_TRUE, release_value
     CORBA::release( _tc );

     _tc = .... (set _tc_long)
     _value = new Long(v);     // set v to _value
     _release = CORBA_TRUE;
}
```

Right Shift Assign Operator

An example of the right shift assign operator is shown below.

```
// Usage
CORBA::Long value;
Any a;
a <<= CORBA::Long(42);           // set _tc_long to _tc, _42 to _value
                                 // set CORBA_TRUE to release flag
if(a >>= value ) {               // When _tc indicates _tc_long, return CORBA_TRUE and
                                 // set _value to value
}

// right shift assign operator
CORBA::Boolean
CORBA::Any::operator>>=( Long &r )const
{
     if( _tc->kind != tk_long )  // check if _tc indicates tc_long
          return CORBA_FALSE
     r = ...                     // set _value
     return CORBA_TRUE;
}
```

In the case of strings and arrays, to ensure that object size limits are not exceeded when an application uses values retrieved from *any* type, it needs to check the *any* type *TypeCode*.

Helper Function

Boolean, char, and octet in IDL are mapped to unsigned char in C++. A compilation error occurs if attempts are made to assign to, or to retrieve from, the Any function Boolean, octet, or char values.

```
Octet oct = 040;
Any any;
any <<= oct;                    // compilation error
```

To identify Boolean, octet or char values you must use the helper function, the operator <<+, and the operator >>+. An example of the helper function is given below.

```
CORBA::Boolean b = CORBA_TRUE;
Any any;
any <<= Any::from_boolean(b);   // set _tc_boolean to _tc, b to _value.
```

```
  ...
if( any >>= Any::to_boolean(b)){
                                // _value of any is CORBA::Boolean
                                // assign _value of any to  b (CORBA_TRUE).
}
```

Constructor (Any Type, for Values which are not Type Defined)

This constructor copies tc of TypeCode, specified as a parameter, and assigns a value pointer to _value. If you invoke destructor when the release flag is CORBA_TRUE, _value will be released. When the release flag is CORBA_FALSE, an application must explicitly release _value.

```
//Usage
Long l = 1;
CORBA::Any *a = new CORBA::Any(tc_long, &l, CORBA_TRUE);
                     // the following constructor is invoked

// assign operator
Any::Any( TypeCode_ptr tc, void *value, Boolean release = CORBA_FALSE)
{
     _tc = ... (_assign tc_long)
     _value = value;
     _release = release;
}
```

Replace Function

The replace function assigns tc and value of TypeCode, specified as a parameter to _tc and _value. _value will be released if you invoke destructor when the release flag is CORBA_TRUE. When the release flag is CORBA_FALSE, an application must explicitly release _value.

```
// Usage
Long l = 1;
CORBA::Any *a = new CORBA::Any();

a->replace(_tc_long, &l, CORBA_FALSE);   // the following function is cinvoked

//  replace function processing
Any::Any::replace( TypeCode_ptr tc, void *value, Boolean release = CORBA_FALSE)
{
      if( _release && _value )
          delete value;  // Deletes the existing data if the release flag is CORBA_TRUE.
      CORBA::release( _tc );

      _tc = tc;
      _value = value;
      _release =  release;
}
```

Type() Function

The type function returns the *CORBA::TypeCode* object associated with *Any*.

```
CORBA::Any *a = new CORBA::Any();
CORBA::Long x;
(*a) <<= x;
  ...
CORBA::TypeCode_ptr tc = a->type();     // extract _tc
if( tc->kind() == tk_long ){
```

```
    ...                                 // execute this code
}
```

Value() Function

The *value()* function returns *_value* of *Any* type as *void** type. When no value is specified in *_value*, the value function returns a NULL pointer.

```
CORBA::Any any;
CORBA::Long x;
x = 3;
any <<= x;       //left shift assign operator assigns _tc_long to _tc, 3 to _value.

CORBA::Long *p = (CORBA::Long *)any.value();
                 // extract _value. Cast void* to CORBA::Long *.

count << "(*p)=" << *p << endl;
```

5.7.5 Sequence Type

This section describes sequence type data.

IDL Mapping

When the sequence type (*sequence*) is specified in IDL, the data is allocated to a sequence class that has the maximum number of arrays (*_maximum*), the number of arrays to be used (*_length*), the data area address (*_buffer*), and the release flag (*_release*) as private data in C++ language. A function for allocating the data area (*_buffer*) is also generated. (The function name is generated using "module name::interface name::sequence name::allocbuf", and the function is called *XX::allocbuf*.)

Sequence class instances are generated by the new operator (C++). Specify the maximum number of arrays, the number of arrays to be used, the data area address (allocated using the *XX::allocbuf* function), and the release flag in the parameters.

This is explained using the following IDL definition example.

IDL

```
module Odsample {
     interface      seqtest {
          typedef   sequence<long> seampleseq;
          sampleseq op1(in sampleseq seq1, out sampleseq seq2,
                    inout sampleseq seq3);
          };
};
```

This is rewritten in C++ as follows.

C++

```
class sampleseq
{
     public:
          sampleseq();                    //default constructor
          sampleseq( CORBA::ULong max);   //maximum constructor
          sampleseq( CORBA::ULong max, CORBA::ULong length,
CORBA::Long *data,
CORBA::Boolean release = CORBA_FALSE );
//T *data constructor
          sampleseq( const sampleseq &s );
                                          // copy constructor
          ~sampleseq();                   // destructor

          static CORBA::Long *allocbuf( CORBA::ULong );
                                          // allocbuf
```

```
          static void freebuf( CORBA::Long* );
                                        // freebuf

          sampleseq &operator=( const sampleseq &s );
                                        // assign operator

          CORBA::ULong maximum() const;
                                        // maximum access function

          void length( CORBA::ULong );
                                        // length access function
          CORBA::ULong length() const;
                                        // length access function

          CORBA::Long &operator[]( CORBA::ULong index );
                                        // get an indexth value in _buffer
          const CORBA::Long &operator[]( CORBA::ULong index )
                                             const;
                                       // get an indexth value in _buffer

     private:
          CORBA::ULong _maximum;         // maximum number of array element
          CORBA::ULong _length;          // number of array element
          CORBA::Long *_buffer;          // array value
          CORBA::Boolean _release;       //release flag
};

// Left shift assign operator
void operator<<=( CORBA::Any&, const ODsample::seqtest::sampleseq& );
void operator<<=( CORBA::Any&, ODsample::seqtest::sampleseq* );
void operator<<=( CORBA::Any_var&, const ODsample::seqtest::sampleseq& );

// Right shift assign operator
CORBA::Boolean operator>>=( const CORBA::Any&, ODsample::seqtest::sampleseq*& );
CORBA::Boolean operator>>=( const CORBA::Any_var&, ODsample::seqtest::sampleseq*& );
```

Parameters Handled by Client Applications

The following table shows how the client application parameters are handled.

Table 5.13 Client Parameter Area (Sequence Type)

Parameter	Parameter passed to server	Parameter passed from server
in	The new operator and the XX::allocbuf function are used to dynamically allocate the area.	_
inout	Same as the *in* parameter	The area is automatically allocated by the stub. (The structure members are also set.)
out return	_	Same as the *inout* parameter

Note

When an area that has been allocated by the client and stub is no longer required, it must be released by the delete operator (C++). Whether or not the data area (*_buffer*) is to be released is specified in the release flag. The release flag is specified in the fourth parameter when an instance is generated in the sequence class.

- CORBA_TRUE: When delete is issued, the *_buffer* area is also released.
- CORBA_FALSE: When delete is issued, the *_buffer* area is not released. (Default)

The release flag for the parameter (*out*, *return*) allocated by the stub is set in CORBA_TRUE. For details on the release flag, refer to " 5.8 Any Type and Sequence Type Release Flags".

The following is an example of client application processing.

```
ODsample::sampleseq *seq = new ODsample::sempleseq( CORBA::ULong maximum,
               CORBA::Ulong minimum,
               CORBA::LOng *data,
               CORBA::Boolean release = CORBA_FALSE);

CORBA::Environment     env;
ODsample::seqtest_ptr  obj;
ODsample::sampleseq    *seq0, *seq1, *seq2, *seq3;

/* in parameter */
CORBA::Long *p = ODsample::seqtest::sampleseq::allocbuf(3);
          // allocate area
for( int i = 0; i < 3; i++ )
     p[i] = i*10;
          // set value
seq1 = new ODsample::sampleseq( 3, 3, p, CORBA_TRUE );
          // create an instance with values

/* inout parameter */
CORBA::Long *q = ODsample::seqtest::sampleseq::allocbuf(5);
          // allocate area
for( i = 0; i < 5; i++ )
     q[i] = i*100;
          // set value
seq3 = new ODsample::sampleseq( 5, 5, q, CORBA_TRUE );
          // create an instance with values

seq0 = obj->op1( obj, *seq1, seq2, *seq3, env );

delete seq0;          //return return
delete seq1;          //release in parameter
delete seq2;          //release out parameter
delete seq3;          //release inout parameter
```

Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

The following table shows how the server application parameters are handled.

Table 5.14 Server Parameter Area (Sequence Type)

Parameter	Parameter passed from client	Parameter passed to client
in	The area is automatically allocated or released by the skeleton.	_
inout	The area is automatically allocated by the skeleton.	Data is copied using the assign operator. This area is automatically released by the skeleton.
out return	_	The sequence class and data area are allocated using the new operator and the XX::allocbuf function. This area is automatically released by the skeleton.

The following is an example of server application processing.

```
ODsample::seqtest::sampleseq*
ODsample_seqtest_impl::op1(
     const ODsample::seqtest::sampleseq     &seq1,
     ODsample::seqtest::sampleseq           *&seq2,
     ODsample::seqtest::sampleseq           &seq3,
     CORBA::Environment                     &env )
     throw( CORBA::Exception )
{
/* out parameter */
     CORBA::Long *p = ODsample::seqtest::sampleseq::allocbuf(2);
     for( int i = 0; i < 2; i++ )
          p[i] = i*1000;
       seq2 = new ODsample::seqtest::sampleseq( 2, 2, p, CORBA_TRUE );

/* inout parameter */
CORBA::Long *q = ODsample::seqtest::sampleseq::allocbuf(3);
     for( i = 0; i < 3; i++ )
          q[i] = i;
     seq3 = ODsample::seqtest::sampleseq( 3, 3, q, CORBA_TRUE );

/* return value */
     CORBA::Long *r = ODsample::seqtest::sampleseq::allocbuf(1);
     r[0] = 0;
     ODsample::seqtest::sampleseq_ptr seq =
          new ODsample::seqtest::sampleseq( 1, 1, r,
CORBA_TRUE);
     return seq;
}
```

Using Sequence Class Methods

The following table shows method descriptions in sequence class.

Table 5.15 Sequence Class Methods

Sequence member	Description
default constructor	Creates an instance with _length 0. For bounded sequences, _maximum is initialized to 0. Set the release flag to CORBA_FALSE.
maximum constructor	Creates an instance. For bounded sequence, _maximum is initialized to 0. Set the release flag to CORBA_FALSE.
T * data constructor	When an instance is created, the T*data constructor assigns the sequence length, maximum length, buffer pointer and release flag values passed in parameters, to _length, _maximum, _buffer and _release, with and without a size specification. If the release flag specification is omitted, the constructor sets the flag to CORBA_FALSE.
allocbuf function	Allocates array location and returns it to T *data.
freebuf function	Releases a location allocated with allocbuf function.
copy constructor	Creates an instance and sets _length, _maximum, and _buffer with specified values. Set the release flag to CORBA_TRUE.
destructor	If the release flag is CORBA_TRUE, the destructor frees the area corresponding to the _buffer value.
assign operator	If the release flag is CORBA_TRUE, the assign operator frees the area corresponding to the _buffer value. It then allocates a data area and copies the data from the assignment source. The release flag is set to CORBA_TRUE.
index operator	Returns element corresponding to the index.

Sequence member	Description
maximum() access function	The current available size is returned for unbounded sequences. The size specified in IDL is returned for bounded sequence.
length() access function	length(ULong) assigns sequence length and returns it.
Left shift assign operator	The left shift assign operator assigns a sequence to Any or Any_var. The release flag for the Any variable is set to CORBA_TRUE.
Right shift assign operator	The right shift assign operator fetches a pointer to the sequence from Any or Any_var.

Default Constructor

Default constructor initializes the sequence length as 0 for both bounded and unbounded sequences. In a bounded sequence, the maximum length is initialized using the member initializer, and cannot be changed in an application. In an unbounded sequence the default constructor initializes the maximum length as 0. In both cases the release flag is initialized as CORBA_FALSE. An example of default constructor is shown below.

```
//IDL
typedef sequence< long > LongSeq1;          //unbounded sequence
typedef sequence< long, 10 > LongSeq2;      // bounded sequence (size is 10)

//Usage
longSeq1 *seq1 = new Longseq1();
        // the following constructor, LongSeq1() is invoked and
        // initialize seq1
longSeq2 *seq2 = new LongSeq2();
        // the following default constructor LongSeq2() is invoked and
        // initialize seq2.

// default constructor (size is not specified)
LongSeq1::LongSeq1()
{
    _length = 0;
    _maximum = 0;
    _release = CORBA_FALSE;
}
//  default constructor ( size  is specified)
LongSeq2::LongSeq2() : _maximum(10)         //size specified in IDL
{
    _length = 0;
    _buffer = LongSeq2::allocbuf(10);
    _release = CORBA_FALSE;
}
```

Maximum Constructor

In an unbounded sequence, a constructor allocates a _buffer location of maximum size. You are able to specify this maximum size. Maximum constructor assigns CORBA_FALSE as the release flag. An example of maximum constructor is shown below.

```
//IDL``````````````````
typedef sequence< long > LongSeq;

//Usage
longSeq seq(10);     // maximum constructor is invoked and
                     // initialize _maximum of seq with 10

// maximum constructor
LongSeq::LongSeq(ULong max)
{
     _maximum = max;
```

```
     _buffer = LongSeq::allocbuf(max);
     release = CORBA_FALSE;
}
```

T *data Constructor

T *data constructor assigns specified parameters to private data for both bounded and unbounded sequences. Even in an unbounded sequence, you are able to assign the maximum length.

When you specify CORBA_FALSE as the release flag in T *data constructor, a new instance is explicitly released by an application when it is no longer required.

When you specify CORBA_TRUE as the release flag in T *data constructor, a new instance is automatically released when the sequence instance is released. An example of T *data constructor is shown below.

```
//Usage
long data[] = {1, 2, 3, 4, 5};
LongSeq seq = new LongSeq(10, 5, data);
     // T *data initializes seq.

// T *data constructor
LongSeq::LongSeq( ULong max, ULong length, Long *value,
                  Boolean release = CORBA_FALSE )
{
     _length = length;
     _maximum = max;
     if( release ) {
          _buffer = allocbuf( _maximum );
          for( int i = 0; i < _length; i++ )
               *(  _buffer + i ) = *( value + i );
     }
     else`
          _buffer = value;
     _release = release;
}
```

To create a string sequence, specify String_var* for the value parameter in T *data constructor. To create an object sequence, specify *Object_var**.

Copy Constructor

Copy constructor creates a new sequence with the same maximum and length values as the specified sequence, and copies sequence data. Copy constructor sets CORBA_TRUE as the release flag. An example of copy constructor is shown below.

```
//Usage
long data[] = {1, 2, 3, 4, 5};
LongSeq s1(10, 5, data);
LongSeq s2(s1)                    // copy constructor copies s1 to s2.

//copy constructor
LongSeq::LongSeq( const LongSeq &s )
{
     _length = s.length();
     _maximum = s.maximum();
     _buffer = allocbuf( _maximum );
     for( int i = 0; i < _length; i++ )
                                // Copy s  to  *( _buffer + i )
     _release = CORBA_TRUE;
}
```

Assign Operator

When you set the release flag to CORBA_TRUE, the assign operator releases the area pointer from _buffer, allocates a new location and copies the new sequence data. The assign operator sets CORBA_TRUE as the release flag. An example of assign operator is shown below.

```
//Usage
long data[] = {1, 2, 3, 4, 5};
LongSeq s1(10, 5, data);
LongSeq s2;

LongSeq s2 = s1;     // assign operator copies s1 to s2.
// assign operator
LongSeq &
LongSeqV1::operator=( const LongSeqV1 &s )
{
     if( _release )
          LongSeq::freebuf( _buffer );
     _length = s.length();
     _maximum = s.maximum();
     _buffer = LongSeq::allocbuf( _maximum );
     for( int i = 0; i < _length; i++ )
          // Copy *( _buffer + i )  to  s[i]
     _release =CORBA_TRUE;
     return *this;
}
```

Index Operator

The index operator returns contents that correspond to the index. An example of the index operator is shown below.

```
//Usage
Long l;
long data[] = {1, 2, 3, 4, 5};
LongSeq seq(10, 5, data,CORBA_TRUE);

l = seq[3];        // l is 4
```

When the sequence element is a string, String_var is returned. When the sequence element is an object reference, *Object_var* is returned.

Maximum() Access Function

Maximum() returns the available buffer size for an unbounded sequence. Various elements can then be assigned to the sequence buffer. Maximum() returns the bounded size in IDL for a bounded sequence. An example of maximum() is shown below.

```
//IDL
typedef sequence< long > LongSeq1;
typedef sequence< long, 10 > LongSeq2;

//Example
Long l;
LongSeq1 s1(5);
LongSeq2 s2;

l = s1.maximum();            // l  is 5
l = s2.maximum();            // l  is 10
```

Length() Access Function

Length() assigns a specified value to the sequence length, and allocates a location according to the size of length. Length() returns the length of the sequence. An example of length() is shown below.

```
//Usage
Long l;
```

```
long data[] = {1, 2, 3, 4, 5};
LongSeq s1 = new LongSeq(10, 5, data,CORBA_TRUE);

l = s1->length();          // l is 5
s1->length(6);             // set 5 to sequence length
```

Allocbuf and Freebuf Functions

The allocbuf function allocates a location in which to store sequence elements to be passed to T *data constructor. An example is shown below.

```
//Usage
Long *data = LongSeq::allocbuf(3);

data[0] = 0;
data[1] = 1;
data[2] = 2;

LongSeq *seq = new LongSeq(10, 3, data, CORBA_TRUE);
```

When allocbuf fails to allocate a location, it returns a NULL pointer. If the release flag is CORBA_TRUE, when the sequence is released by destructor a location allocated by allocbuf function is released automatically. If the release flag is CORBA_FALSE, an application must explicitly release it using freebuf.

An application uses new to allocate a sequence and delete to release it.

Left Shift Assign Operator

The left shift assign operator performs two tasks:

- Assign a copy of of the sequence to "_value of Any"
- Assign a pointer to the sequence to "_value of Any". The Any release flag is set to CORBA_TRUE.

A copy of the sequence can also be assigned to Any_var. An example is provided below.

```
//Usage
long data1[] = {1, 2, 3, 4, 5};
ODsample::seqtest::sampleseq *seq1 = new ODsample::seqtest::sampleseq( 10, 5, data1 );
CORBA::Any a1;
a1 <<= *seq1;  // Assigns a copy of seq1 to the _value of a1.
               // Sets TypeCode to _tc_ODsample_seqtest_sampleseq and the release flag to CORBA_TRUE.

long data3[] = {1, 2, 3};
ODsample::seqtest::sampleseq *seq3 = new ODsample::seqtest::sampleseq( 10, 3, data3 );
Any a3;
Any_var a3_v( &a3 );
a3_v <<= *seq3; // Assigns a copy of seq3 to the _value of a3.
               // Sets TypeCode to _tc_ODsample_seqtest_sampleseq and the release flag to CORBA_TRUE.
```

Right Shift Assign Operator

The right shift assign operator fetches the sequence specified in _value of Any. The operator is passed a pointer to the sequence as its argument. If the TypeCode of Any is equal to the TypeCode passed as an argument, the operator assigns the _value value to the argument, and returns with CORBA_TRUE.

The right shift assign operator can also be used for Any_var. An example is provided below.

```
//Usage
Any a;
a <<= *seq1; // Assigns a copy of seq1 to the _value of a.
```

```
            // Sets TypeCode to _tc_ODsample_seqtest_sampleseq and the release flag to CORBA_TRUE.

ODsample::seqtest::sampleseq *seq2, *seq3;
if( a >>= seq2 ){      // If the TypeCode of a is _tc_ODsample_seqtest_sampleseq,
                       // returns CORBA_TRUE and assigns the value of _value (seq1) to seq2.
}

Any_var a_var( &a );
if( a_var >>= seq3 ){  // If TypeCode of a is _tc_ODsample_seqtest_sampleseq,
                       // returns CORBA_TRUE and assigns the _value value (seq1) of a to seq3.
}
```

5.7.6 Structure

This section describes structure type data.

IDL Mapping

Struct in IDL is mapped to struct in C++. An IDL example of struct is defined as follows.

IDL

```
module ODsample{
     struct samplefix {     /* structure (fixed length) */
          long     para1;
          long     para2;
     };
     struct samplevar {     /* structure (variable length) */
          long     para1;
          string     para2;
     };
     interface     structtest{
          samplefix     op2(
               in samplefix str1,
               out samplefix str2,
               inout samplefix str3
          );
          samplevar     op1(
               in samplevar str1,
               out samplevar str2,
               inout samplevar str3
          );
     };
};
```

C++

```
struct samplefix{        /* structure (fixed length) */
     CORBA::Long         para1;
     CORBA::Long         para2;
} ;

samplevar{               /* structure (variable length) */
     CORBA::Long         para1;
     CORBA::String_var   *para2;
} ;
```

Fixed-length Parameters Handled by Client Applications

A client application has no special allocation/deallocation functions for *in*, *out*, or *inout* parameters. You must specify the structure address as a parameter.

```
ODsample::samplefix         strf0, strf1, strf2, strf3;
CORBA::Environment          env;
ODsample::structtest_ptr    obj;

strf1.para1 = 10;           /* in parameter */
strf1.para2 = 11;           /* in parameter */
strf3.para1 = 20;           /* inout parameter */
strf3.para2 = 21;           /* inout parameter */
fix0 = obj->op2( fix1, fix2, fix3, env );
```

Variable-length Parameters Handled by Client Applications

The following table shows how the client application parameters are handled.

Table 5.16 Client Parameter Area (Structure or Variable-length Data Area)

Parameter	Parameter passed to server	Parameter passed from server
in	The new operator and the data area allocation function are used to dynamically allocate the area.	_
inout	Same as the *in* parameter	The area is automatically allocated by the stub.
out return	_	Same as the *inout* parameter

Note

When an area that has been dynamically allocated by the client and stub is no longer required, it must be released by the delete operator (C++). When delete is issued, the variable-length area in the structure is also released.

The following is an example of client application processing.

```
ODsample_samplevar          *str0, str1, *str2, str3;
CORBA::Environment          env;
ODsample::structtest_ptr    obj;

str1.para1 = 5;                             /* in parameter */
str1.para2 = (const CORBA::Char *)"xxxx";   /* in parameter */
str3.para1 = 6;                             /* inout parameter */
CORBA::Char *str = CORBA::string_alloc(6);
strcpy( str, "yyyyy" );
str3.para2 = str;                           /* inout parameter */

str0 = obj->op1( str1, str2, str3, env );

delete( str0 );                             /* release return */
delete( str2 );                             /* release out parameter */
```

Fixed-length Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

A server application has no special allocation/release functions for *in*, *out*, or *inout* parameters. You must assign result values to structure members for these types.

```
ODsample::samplefix
ODsample_structtest_impl::op2(
     const ODsample::samplefix     &str1,
     ODsample::samplefix           &str2,
```

```
     ODsample::samplefix           &str3,
     CORBA::Environment            &env )
     throw( CORBA::Exception )
{
/* out parameter  */
     str2.para1 = 1;
     str2.para2 = 2;

/* inout parameter */
     str3.para1 = 11;
     str3.para2 = 12;

/* return */
     ODsample::samplefix fix;
     fix.para1 = 21;
     fix.para2 = 22;
     return( fix );
}
```

Variable-length Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

The following table shows how the server application parameters are handled.

Table 5.17 Server Parameter Area (Structure or Variable-length Data Area)

Parameter	Parameter area passed from client	Parameter area passed to client
in	The area is automatically allocated or released by the skeleton.	_
inout	The area is automatically allocated by the skeleton.	The new data is set in the passed parameter. (The old area is released and replaced with the new data.)
out return	_	The structure area or variable-length data area is allocated using the new operator or the data area allocation function. This area is automatically released by the skeleton.

The following is an example of server application processing.

```
ODsample_samplevar *
ODsample_structtest_impl::op1(
     const ODsample_samplevar     &str1,
     ODsample::samplevar          *&str2,
     ODsample::samplevar          &str3,
     CORBA::Environment     &env )
     throw( CORBA::Exception )
{
/* out parameter */
     str2 = new ODsample::samplevar;             /* allocate out parameter */
     str2->para1 = 12;                           /* set value */
     str2->para2 = (const CORBA::Char *)"aaa";   /* set value */

/* inout paramter */
     str3->para1 = 4;                            /* set value */
     str3->para2 = ( const CORBA::Char *)"bbb";  /* set value */
/* return */
     ODsample::samplevar *smp = new ODsample::samplevar;
/* allocate return */
     smp->para1 = 5;
     smp->para2 = ( const CORBA::Char *)"bbb";   /* set value */
```

```
    return( smp );
}
```

5.7.7 Union

This section describes union type data.

IDL Mapping

Union in IDL is mapped to a structure in C++ that has a discriminator (*_d*) and a pointer (*_ptr*). The discriminator is used to identify data type and the pointer points to data. An example of IDL union definition is shown below.

IDL

```
module ODsample{

     union samplefix switch(long){     /* union (fixed length) */
          case 1:     long     para1;
          case 2: long     para2;
     };
     union samplevar switch(long){     /* union (variable length ) */
          case 1:     long     para1;
          case 2: string     para2;
     };
     interface     uniontest{
          samplefix     op2(
               in samplefix uni1,
               out samplefix uni2,
               inout samplefix uni3
          );
          samplevar     op1(
               in samplevar uni1,
               out samplevar uni2,
               inout samplevar uni3
          );
     };
};
```

C++

```
class samplefix
{
     public:
          samplefix();                                  //default constructor
          samplefix( const samplefix& );                // copy constructor
          ~samplefix();                                 // destructor

          samplefix &operator=( const samplefix & );
// assign operator

          void     _d( CORBA::Long );                   // set discriminator value
          CORBA::Long     _d() const;                   // get discriminator value

          void     para1( CORBA::Long );                // set para1 data
          CORBA::Long     para1() const;                // get para1 data

          void     para2( CORBA::Long );                // set para2 data
          CORBA::Long     para2() const;                // get para2 data

     private:
          CORBA::Long     __d;                          // discriminator
          void          *_ptr;                          // data area
};
```

```
class samplevar
{
     public:
          samplevar();                                  // default constructor
          samplevar( const samplevar& );                // copy constructor
          ~samplevar(); // destructor

          samplevar &operator=( const samplevar & );
                    // assign operator
          // discriminator access function
          void     _d( CORBA::Long );                   // set discriminator
          CORBA::Long     _d() const;                   // get discriminator
          // Union member access function
          void     para1( CORBA::Long );                // set para1 data
          CORBA::Long     para1() const;                // get para1 data

          void     para2( CORBA::Char* );               // set para2 data
          void     para2( const CORBA::Char* );         // set para2 data
          void     para2( const CORBA::String_var& ); // get para2 data

          CORBA::Char *     para2() const;              // get para2 data

     private:
          CORBA::Long     __d;                          // discriminator
          void          *_ptr;                           // data area
};
```

Fixed-length Parameters Handled by Client Applications

The client sets the discriminator value and corresponding data value using each parameter member function. The client does not explicitly allocate/release locations.

```
ODsample::uniontest_ptr    obj;
ODsample::samplefix        unif0, unif1, unif2, unif3;
CORBA::Environment         env;

unif1.para2(100);          /* set data */
unif3.para1(200);          /* set data */
unif0 = obj->op2( unif1, unif2, unif3, env );
```

Variable-length Parameters Handled by Client Applications

The following table shows how the client application parameters are handled.

Table 5.18 Client Parameter Area (Union Type or Variable-length Data Area)

Parameter	Parameter passed to server	Parameter passed from server
in	The new operator is used to dynamically allocate the variable-length data area.	_
inout	Same as the *in* parameter	The area is automatically allocated by the stub.
out return	_	Same as the *inout* parameter

Note

When an area that has been dynamically allocated by the client and stub is no longer required, it must be released by the delete operator (C++). When delete is issued, the variable-length data area in the structure is also released.

The following is an example of client application processing.

```
ODsample::uniontest_ptr    obj;
ODsample::samplevar        *uni0, uni1, *uni2, uni3;
CORBA::Environment         env;

uni1.para1(10);                                    /* set data */

uni3.para2( (const CORBA::Char *)"INOUT::para2" );  /* set data */

uni0 = obj->op1(  uni1, uni2, uni3, env );
CORBA_free( uni0 );                                /* release return */
CORBA_free( uni2 );                                /* release the out parameter */
```

Fixed-length Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

The server sets the discriminator value and corresponding data using individual member functions for *out* and *inout* parameters. The server does not explicitly allocate/release locations.

```
ODsample::samplefix
ODsample_uniontest_impl::op2(
     const ODsample::samplefix     &uni1,
     ODsample::samplefix           &uni2,
     ODsample::samplefix           &uni3,
     CORBA::Environment            &env
 )
throw( CORBA::Exception )
{
/* out parameter */
     uni2.para1(10);               /* set data */

/* inout parameter */
     uni3.para2(100);              /* set data */

/* return value */
     ODsample::samplefix uni;
     uni.para1(100);               /* set data */
     return( uni );
}
```

Variable-length Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

The following table shows how the server application parameters are handled.

Table 5.19 Server Parameter Area (Structure or Variable-length Data Area)

Parameter	Parameter area passed from client	Parameter area passed to client
in	The area is automatically allocated or released by the skeleton.	_
inout	The area is automatically allocated by the skeleton.	The new data is set using the member access function. (The old area is automatically released.) With variable-length data, the data area is allocated using the data area allocation function. This area is automatically released by the skeleton.
out	_	The union area or variable-length data area is allocated using the new operator or the data area allocation function.

Parameter	Parameter area passed from client	Parameter area passed to client
return		This area is automatically released by the skeleton.

The following is an example of server application processing.

```
ODsample_samplevar *
ODsample_uniontest_impl::op1(
     const ODsample::samplevar     &uni1,
     ODsample::samplevar          *&uni2,
     ODsample::samplevar          &uni3,
     CORBA::Environment          &env )
{
/* out parameter */
     uni2 = new ODsample::samplevar;          /* allocate out parameter */
     uni2->para2( (const CORBA::Char *)"OUT::param" );     /* set data */

/* inout parameter */
     uni3.para1(10);          /* set data */

/* return */
     ODsample::samplevar *uni = ODsample::samplevar;
/* allocate return value*/
     uni.para1(30);          /* set data */
     return( uni );
}
```

Using Union Class Methods

The following table shows method descriptions in union class.

Table 5.20 Union Class Method

Union member	Description
default constructor	Creates an instance and initializes _ptr as 0.
copy constructor	Creates an instance and initializes _ptr, _d with specified values.
assign operator	Sets specified values to _ptr, _d.
destructor	Releases an instance and releases _ptr and _d.
discriminator access function	_d(Long) sets the specified value to discriminator. _d() returns discriminator.

Default Constructor

Default constructor initializes the data pointer *_ptr* as 0. An example of default constructor is shown below.

```
//IDL
union UnionSmp switch(long){
     case 1: long para1;
     case 2: string para2;
};

// Usage

UnionSmp *uni1 = new UnionSmp();          // UnionSmp() initializes uni1.
UnionSmp  uni2;                           // UnionSmp() initializes uni2.

// default constructor
```

```
UnionSmp::UnionSmp()
{
     _ptr = (void *)0;
}
```

Copy Constructor

Copy constructor copies a discriminator (*_d*) and a pointer (*_ptr*). An example of copy constructor is shown below.

```
//IDL
union UnionSmp switch(long){
     case 1: long para1;
     case 2: string para2;
};

// Usage
UnionSmp *uni1 = new UnionSmp();
uni1->para1(10);                          // union member access function
UnionSmp uni2 = new UnionSmp(*uni1);      // copy constructor set 1 to _d, 10 to _ptr.

//copy constructor
         UnionSample::UnionSmp(const UnionSmp &_UnionSmp )
         {
switch( _UnionSmp.__d ) {
   case 1:
          para1( _UnionSmp.para1() );
                    break;
             case 2:
                para2( (const CORBA::Char *)_UnionSmp.para2);
                break;
          }
          __d = _UnionSmp._d();
          }
```

Destructor

Destructor releases the location pointed to by *_ptr*. An example of destructor is shown below.

```
//IDL
union UnionSmp switch(long){
     case 1: long para1;
     case 2: string para2;
};

// Usage
UnionSmp *uni1 = new UnionSmp();
CORBA::Char *str = CORBA::string_alloc(5);
strcpy( str, "data");
uni1->para2(str);           //union member function. Str is set to _ptr

delete  uni1;               // destructor releases _ptr(str).
```

Discriminator Access Function

This accesses the *_d* discriminator to identify the data type. An example of the discriminator access function is shown below.

```
//IDL
union UnionSmp switch(long){
     case 1: long para1;
     case 2: string para2;
};
```

```
//Usage
UnionSmp *uni1 = new UnionSmp();
uni1->para1(10);
CORBA::Long l = uni1->_d();     // 1 is 1
uni1->_d(2);                    // d is 2
l = uni1->_d()                  // 1 is 2
```

Member Access Function

For each member in a union class there is an access function and a modify function, with the same name as the member. An example of an access function is shown below.

```
//IDL
typedef long Bytes[64];
struct S {
     long len;
};

union UnionSmp switch(long){
     case 1: long x;
     case 2: Bytes y;
     case 3: string z;
     case 4: S w;
};

// Usage
UnionSmp *uni1 = new UnionSmp();
uni1->x(10);                    // Union member access function. __d is 1, _ptr is 10

     UnionSmp *uni2 = new UnionSmp();
     Bytes data;
     for ( int i = 0; i < 64; i++ )
          data[i] = i;
     uni2->y(data);             // __d is 2, _ptr is data

     UnionSmp uni3(*uni2);
     Bytes_slice *slice = uni3->y();     // return _ptr

     UnionSmp uni4;
     CORBA::Char *str = CORBA::string_alloc(4);
     strcpy( str, "ZZZ" );
     uni4.z( str );             // __d is 3, _ptr is str.
     uni4.z( (const CORBA::Char *)"XXX" );
                                // release _ptr and copy data "XXX".
                                // (with const: copy data, without const:
                                // assign pointer)

     UnionSmp  uni5;
     S  str_data;
     str_data.len = 5;
     uni5.w( data_len );        // __d is, _ptr is data_len
     S  str_data2;
     str_data2 = uni5.w();      // str_data2 is set to _ptr
```

5.7.8 Array

This section describes arrays.

IDL Mapping

Array in IDL is mapped to array in C++. An allocation function for array data(a function named "module name::interface name::array name_alloc". It is called *XX::alloc()* function) is generated by IDL compiler. The index begins at 0 and the last index is <array size -1 >. An example of array in IDL is shown below.

IDL

```
module ODsample{
     interface     arraytest{
          typedef long     fix[4][3][2];          /* array (fixed length) */
          typedef string str[2][3][4];            /* array (varaiable length) */
          fix     op1(in fix para1, out fix para2, inout fix para3 );
          str     op2(in str para1, out str para2, inout str para3 );
     };
};
```

C++

```
typedef CORBA::Long     fix[4][3][2];              /* array (fixed length) */
typedef CORBA::Long fix_slice[4][3];
typedef CORBA::String_var     str[2][3][4];        /* array (variable length) */
typedef CORBA::String_var str_slice[2][3];
```

Fixed-length Parameters Handled by Client Applications

For *in*, *out*, and *inout* parameters, a client does not allocate/release locations, or specify addresses of automatic variables. For return values, a stub allocates a location using the *XX_alloc* function, and releases it using *XX_free* when it is no longer required.

```
CORBA::Environment                env;
ODsample::arraytest_ptr           obj;
ODsample::arraytest::fix_slice    *fix0;
ODsample::arraytest::fix           fix1, fix2, fix3;
int i, j, k;

for( i = 0; i < 4; i++ )
for( j = 0; j < 3; j++ )
for( k = 0; k < 2; k++ ) {
     fix1[i][j][k] = i;
     fix3[i][j][k] = i*10;
}
fix0 = obj->op1(  fix1, fix2, fix3, env );
ODsample::arraytest::fix_free(fix0);
```

Variable-length Parameters Handled by Client Applications

For *in*, and *inout* parameters, the client does not allocate/release locations, or specify addresses of automatic variables. For *out* and *return* values, a stub allocates a location using the *XX_alloc* function, and releases it with *XX_free* when it is no longer required.

```
CORBA::Environment                env;
ODsample::arraytest_ptr           obj;
ODsample::arraytest::str_slice    *str0, *str2;
ODsample::arraytest::str          str1, str3;
int i, j, k;
char buf[120];

for( i = 0; i < 2; i++ )
for( j = 0; j < 3; j++ )
for( k = 0; k < 4; k++ ) {
     sprintf( buf, "str1[%d][%d][%d] ", i, j, k );
     str1[i][j][k] = (const CORBA::Char *)buf;
     sprintf( buf, "str3[%d][%d][%d] ", i, j, k );
     str3[i][j][k] = (const CORBA::Char *)buf;
}
```

```
str0 = obj->op2( str1, str2, str3, env );
ODsample::arraytest::str_free( str0 );
ODsample::arraytest::str_free( str2 );
```

Fixed-length Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

For *in*, *out*, and *inout* parameters, the server does not allocate/release a location. For *out* and *inout* parameters, the server assigns values to parameters. For the return value, the server allocates location using the *XX_alloc* function, assigns a value and returns it. The allocated location is released within the skeleton.

```
ODsample::arraytest::fix_slice *
ODsample_arraytest_impl::op1(
ODsample::arraytest::fix     para1,
ODsample::arraytest::fix     para2,
ODsample::arraytest::fix     para3,
CORBA::Environment     &env,
throw( CORBA::Exception )
{
int i, j, k;

/* OUT parameter */
for( i = 0; i < 4; i++ )
for( j = 0; j< 3; j++ )
for( k = 0; k < 2; k++ )
 para2[i][j][k] = i+j+k;

/* inout parameter */
for( i = 0; i < 4; i++ )
for( j = 0; j < 3; j++ )
for( k = 0; k < 2; k++ )
para3[i][j][k] = (i+j+k)*10;

/* return value */
ODsample::arraytest::fix_slice *fix =
ODsample::arraytest::fix_alloc();
for( i = 0; i < 4; i++ )
for( j = 0; j < 3; j++ )
for( k = 0; k < 2; k++ )
fix[i][j][k] = i*j*k;

return( fix );
}
```

Variable-length Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

For *in* and *inout* parameters, a server does not allocate/release a location. For an *out* parameter and return value, the server allocates a location using the *XX_alloc* function and assigns a value to it. For an *inout* parameter, the server generates new data and assigns it to the parameter. Old data from the client is released and replaced with the new data.

```
ODsample::arraytest::str_slice *
ODsample_arraytest_impl::op2(
     ODsample::arraytest::str           para1,
     ODsample::arraytest::str_slice     *&para2,
     ODsample::arraytest::str           para3,
     CORBA::Environment                 &env,
     throw( CORBA::Exception )
{
     int i, j, k;
     char     buf[120];
```

```
    /* OUT parameter */
    str2 = ODsample::arraytest::str_alloc();
    for( i = 0; i < 2; i++ )
    for( j = 0; j< 3; j++ )
    for( k = 0; k < 4; k++ ){
         sprintf( buf, "str2[%d][%d][%d]", i, j, k );
         para2[i][j][k] = (const CORBA::Char *)buf;
    }

    /* inout parameter */
    for( i = 0; i < 2; i++ )
    for( j = 0; j < 3; j++ )
    for( k = 0; k < 4; k++ ){
         sprintf( buf, "str3[%d][%d][%d]", i, j, k );
         para3[i][j][k] = (const CORBA::Char *)buf;
    }

/* return value */
ODsample::arraytest::str_slice *str =
ODsample::arraytest::str_alloc();
    for( i = 0; i < 4; i++ )
    for( j = 0; j < 3; j++ )
    for( k = 0; k < 2; k++ ){
         sprintf( buf, "str[%d][%d][%d]", i, j, k );
         str[i][j][k] = (const CORBA::Char *)buf;
    }
    return( str );
}
```

5.7.9 Mapping Interface Declaration (Interface)

The following topics are discussed in this section:

- IDL Mapping
- Using Interface Methods

IDL Mapping

Interface declarations specified in IDL are mapped to a class in C++ language.

This is explained using the following IDL definition example:

IDL

```
interface A {
    ...              // operation definition
};
```

C++ language

```
type def A    *A_ptr;
class A : public virutual CORBA::Object {
    public:
        static A_ptr     _duplicate( A_ptr );
        static A_ptr     _narrow( CORBA::Object_ptr );
        static A_ptr     _nil();

                         // operation definition
       ...
};
```

CORBA::Object is inherited. The member functions in *CORBA::Object* are also available.

Using Interface Methods

The following table lists and describes the interface class methods associated with C++.

Table 5.21 Interface Class Methods

Interface class members	Description
_duplicate	Makes a copy of the specified object in the parameter.
_narrow	Makes a copy of the specified object information in the parameter, converts it to interface object type and returns it.
_nil	Returns NIL object.
is_nil (inherited from CORBA::Object class)	Checks if specified object is NIL.
release (inherited from CORBA::Object class)	Releases the specified object.

_duplicate()

Makes a copy of the specified object in the parameter.

```
A_ptr  obj;
obj ...                // get an object
A_ptr obj_copy = A::_duplicate( obj );
```

_narrow()

Makes a copy of the specified object information in the parameter, changes it to an interface object type and returns it.

```
CORBA::Object_ptr obj = NamingContext_opj->resolve( *name , env );
     // get an object from NamingService
A_ptr target = A::_narrow( obj ); // cast CORBA::Object_ptr to A_ptr
```

_nil()

Returns a NIL object.

```
A_ptr obj;
obj = A::_nil();          // set NIL
```

is_nil()

Checks if the specified object is NIL. If NIL, it returns CORBA_TRUE. Otherwise it returns CORBA_FALSE.

```
A_ptr     obj;
CORBA::Environment env;
...
obj->is_nil(env);
```

release()

Releases the specified object.

```
A_ptr     obj;
CORBA::Environment env;
...
obi->release(env);
```

5.7.10 Mapping Attribute Declaration (Attribute)

This section provides information on mapping attribute declarations.

IDL Mapping

When an attribute declaration (*attribute*) is defined in IDL, it is mapped to the object data *set* or *get* function. (The function name is the variable name defined in IDL.)

This is explained using the following IDL definition example.

IDL

```
module ODsample{
     interface     attrtest{
                attribute long  para1;
                attribute string para2;
                readonly attribute long para3;
        };
};
```

C++

```
       CORBA::Long para1;
     para1 = av->para1(*env);
       av->para1(100,*env);

      CORBA::Char* p;
     p = CORBA::string_alloc(7);
       av->para2( p,*env );
       p = av->para2( *env );

       CORBA::Long para3;
       para3 = av->para3(*env);
```

Parameters Handled by Client Applications

When you no longer require a string location, which you have allocated using the get function named with the variable name in IDL, a client must use the *CORBA::string_free()* function to release it.

```
      ODsample::attrtest_var  av
      env = new CORBA::Environment;

    CORBA::Long para1;
      para1 = av->para1(*env);
      av->para1(100,*env);
      para1 = av->para1(*env);
      cout << para1 << endl;

      CORBA::Char* p;
      p = CORBA::string_alloc(7);
      strcpy( p, "STRING" );
      av->para2( p,*env );
      CORBA::string_free( p );
```

```
       p = av->para2( *env );
             cout << p << endl;
             CORBA::string_free( p );

             para3 = av->para3(*env);
             cout << lpara << endl;
```

Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

When you no longer require a string location, which you have allocated using the get function named with the variable name in IDL, a client must use the *CORBA::string_free()* function to release it.

When a server uses attributes, for *in* parameters of primitive type (long etc), the server does not allocate/release locations. The server does allocate a location for a string *out* parameter. For a string *inout* parameter, the server allocates/releases a location and assigns new data. The allocated location for a string *inout* parameter is released within a skeleton.

```
CORBA::Long
ODsample_attrtest_impl::para1(
        CORBA::Environment      & )
        throw( CORBA::Exception )
{
        CORBA::Long   para1;
        return( para1 );
}

void
ODsample_attrtest_impl::para1(
        CORBA::Long             data,
        CORBA::Environment      & )
        throw( CORBA::Exception )
{
        CORBA::Long   para1;
        para1 = data;
}

CORBA::Char*
ODsample_attrtest_impl::para2(
        CORBA::Environment      & )
        throw( CORBA::Exception )
{
        static CORBA::Char* p;
        p = "test";
        return( p );
}

void
ODsample_attrtest_impl::para2(
        const CORBA::Char* data,
        CORBA::Environment      & )
        throw( CORBA::Exception )
{
        CORBA::String_var  para2("");
        para2 = data;
        return;
}

CORBA::Long
ODsample_attrtest_impl::para3(
        CORBA::Environment      & )
        throw( CORBA::Exception )
{
        CORBA::Long        para3;
```

```
        return( para3 );
}
```

5.7.11 Allocating and Releasing Parameter Area Using Dynamic Interface

This section describes how to create parameters using Dynamic Invocation Interface (DII). This section also explains how to fetch values from the created parameters.

Set parameters using the *CORBA::NVList::add_value()* function. To fetch values, use CORBA::NVList::item().

Examples are provided below for each data type. The examples use the following IDL definition:

Basic Data Type

```
module ODsample{
  interface  dyn1{
    long  add( in    long  a,    // 0-th parameter a
               out   long  b,    // 1st parameter b
               inout long  c );  // 2nd parameter c
  };
};
```

String and Wide String Types

```
module ODsample{
  interface  diistring{
    string  stringtest( in    string  a,    // 0-th parameter a
                        out   string  b,    // 1st parameter b
                        inout string  c );  // 2nd parameter c
  };
};
```

Structure

```
module ODsample{
  interface  diistruct{
    struct  data{
          long  para1;
          char  para2;
    };
    data  structtest( in    data  a,    // 0-th parameter a
                      out   data  b,    // 1st parameter b
                      inout data  c );  // 2nd parameter c
  };
};
```

Sequence Type

```
module ODsample{
  interface  diiseq{
    typedef sequence<long>  sampleseq;
    sampleseq  seqtest( in    sampleseq  a,    // 0-th parameter a
                        out   sampleseq  b,    // 1st parameter b
                        inout sampleseq  c );  // 2nd parameter c
  };
};
```

IN Mode

To pass in parameters to a server application, allocate parameter area in a client application and set the pointer to the 2nd parameter (*Any* type) of add_value() function. For details on how to assign a value to an Any type, refer to "5.7.4 Any Type" in "Data Type Mapping".

Release the allocated area, after the request is sent, when it is no longer used.

Examples of creating an "in" parameter are provided below.

Basic Data Type

```
CORBA::NVList_ptr       arg_list;
CORBA::Environment_ptr  env = new CORBA::Environment;

// List object creation
orb->create_list( 3, arg_list, *env );

// Parameter area allocation
CORBA::Long         x = 100;
// Setting of a parameter area for the Any type
CORBA::Any          p0;
p0 <<= x;

// in parameter creation
arg_list->add_value(
      "a",                 /* Specifies the same parameter name as specified in the IDL. */
      p0,                  /* Specifies an Any type for which the value to be used for in is
specified. */
      CORBA::ARG_IN,       /* Specifies CORBA::ARG_IN. */
      *env );

// Parameter area freeing
//  Because p0 release flag is set to CORBA_TRUE, the parameter area is automatically freed
//  as the result of destructor processing.
```

String and Wide String Types

```
CORBA::NVList_ptr       arg_list;
CORBA::Environment_ptr  env = new CORBA::Environment;

// List object creation
orb->create_list( 3, arg_list, *env );

// Parameter area allocation
CORBA::Char         *str0 = CORBA::string_alloc(3);
strcpy( str0, "IN" );
// Setting of a parameter area for the Any type
CORBA::Any          p0( _tc_string, str0 );

// in parameter creation
arg_list->add_value(
      "a",                 /* Specifies the same parameter name as specified in the IDL. */
      p0,                  /* Specifies an Any type for which the value to be used for in is
specified. */
      CORBA::ARG_IN,       /* Specifies CORBA::ARG_IN. */
      *env );

// Parameter area freeing
//  After request issuance, str0 must be freed when it is no longer needed.
CORBA::string_free( str0 );
```

Structure

```
CORBA::NVList_ptr       arg_list;
CORBA::Environment_ptr  env = new CORBA::Environment;

// List object creation
orb->create_list( 3, arg_list, *env );

// Parameter area allocation
ODsample::diistruct::data  *data0 = new ODsample::diistruct::data;
data0->para1 = 10;
data0->para2 = 'x';
// Setting of a parameter area for the Any type
CORBA::Any          p0( _tc_ODsample_diistruct_data, data0 );
                           /* _tc_ODsample_diistruct_data is defined */
                           /* in the header file created by IDLc. */

// in parameter creation
arg_list->add_value(
      "a",                 /* Specifies the same parameter name as specified in the IDL. */
      p0,                  /* Specifies an Any type for which the value to be used for in is
specified. */
      CORBA::ARG_IN,       /* Specifies CORBA::ARG_IN. */
      *env );

// Parameter area freeing
//  After request issuance, data0 must be freed when it is no longer needed.
delete data0;
```

Sequence Type

```
CORBA::NVList_ptr       arg_list;
CORBA::Environment_ptr  env = new CORBA::Environment;

// List object creation
orb->create_list( 3, arg_list, *env );

// Parameter area allocation
CORBA::Long         *p = ODsample::diiseq::sampleseq::allocbuf(3);
for( int i = 0; i < 3; i++ )
    p[i] = i*10;
ODsample::diiseq::sampleseq  *seq0 =
                    new ODsample::diiseq::sampleseq( 3, 3, p, CORBA_TRUE );
// Setting of a parameter area for the Any type
CORBA::Any          p0( _tc_ODsample_diiseq_sampleseq, seq0 );
                           /* _tc_ODsample_diiseq_sampleseq is defined */
                           /* in the header file created by IDLc. */

// in parameter creation
arg_list->add_value(
      "a",                 /* Specifies the same parameter name as specified in the IDL. */
      p0,                  /* Specifies an Any type for which the value to be used for in is
specified. */
      CORBA::ARG_IN,       /* Specifies CORBA::ARG_IN. */
      *env );

// Parameter area freeing
//  After request issuance, seq0 must be freed when it is no longer needed.
delete seq0;               /* Because the release flag is set to CORBA_TRUE, */
                           /* the area (p) allocated by allocbuf is also freed.*/
```

OUT Mode

To receive the server application processing from an out parameter, the client application does not need to allocate an area. For the second parameter of add_value(), specify an Any type with the appropriateTypeCode..

When an NVList object is deleted, the out area allocated on the server application side is automatically made free. Therefore, the out parameter area passed from the server application cannot be referenced after the area for the NVList class is freed. For further information on NVList object releasing, refer to "5.2.7 Deleting a Request".

Examples of creating an out parameter and examples of fetching the processing result are provided below.

Basic Data Type

```
// Setting of TypeCode for the Any type
CORBA::Any          p1( _tc_long, NULL );

// out parameter creation
arg_list->add_value(
      "b",                 /* Specifies the same parameter name as specified in the IDL. */
      p1,                  /* Specifies an Any type for which the TypeCode is specified. */
      CORBA::ARG_OUT,      /* Specifies CORBA::ARG_OUT. */
      *env );

// Processing result fetching
CORBA::NamedValue_ptr  nv1 = arg_list->item(1, *env);      /* Fetches the 1st parameter. */
CORBA::Any             *ap1 = nv1->value( *env );          /* Fetches Any from the parameter. */
CORBA::Long            *lp1 = (CORBA::Long *)ap1->value(); /*Fetches the value from Any. */

// Parameter area freeing
//  The area for lp1 is freed by CORBA::release(arg_list).
```

String and Wide String Types

```
// Setting of TypeCode for the Any type
CORBA::Any          p1( _tc_string, NULL );

// out parameter creation
arg_list->add_value(
      "b",                 /* Specifies the same parameter name as specified in the IDL. */
      p1,                  /* Specifies an Any type for which the TypeCode is specified. */
      CORBA::ARG_OUT,      /* Specifies CORBA::ARG_OUT. */
      *env );

// Processing result fetching
CORBA::NamedValue_ptr  nv1 = arg_list->item(1, *env);      /* Fetches the 1st parameter. */
CORBA::Any             *ap1 = nv1->value( *env );          /* Fetches Any from the parameter. */
CORBA::Char            *cp1 = (CORBA::Char *)ap1->value(); /* Fetches the value from Any. */

// Parameter area freeing
//  The area for cp1 is freed by CORBA::release(arg_list).
```

Structure

```
// Setting of TypeCode for the Any type
CORBA::Any          p1( _tc_ODsample_diistruct_data, NULL );
                        /* _tc_ODsample_diistruct_data is defined */
                        /* in the header file created by IDLc. */

// out parameter creation
arg_list->add_value(
      "b",                 /* Specifies the same parameter name as specified in the IDL. */
```

```
      p1,                 /* Specifies an Any type for which the TypeCode is specified. */
      CORBA::ARG_OUT,     /* Specifies CORBA::ARG_OUT. */
      *env );

// Processing result fetching
CORBA::NamedValue_ptr  nv1 = arg_list->item(1, *env);      /* Fetches the 1st parameter. */
CORBA::Any             *ap1 = nv1->value( *env );          /* Fetches Any from the parameter. */
ODsample::diistruct::data  *dp1 =
           (ODsample::diistruct::data *)(ap1->value());    /* Fetches the value from Any. */

// Parameter area freeing
//  The area for dp1 is freed by CORBA::release(arg_list).
```

Sequence Type

```
// Setting of TypeCode for the Any type
CORBA::Any          p1( _tc_ODsample_diiseq_sampleseq, NULL );
                        /* _tc_ODsample_diiseq_sampleseq is defined */
                        /* in the header file created by IDLc. */

// out parameter creation
arg_list->add_value(
      "b",              /* Specifies the same parameter name as specified in the IDL. */
      p1,               /* Specifies an Any type for which the TypeCode is specified. */
      CORBA::ARG_OUT,   /* Specifies CORBA::ARG_OUT. */
      *env );

// Processing result fetching
CORBA::NamedValue_ptr  nv1 = arg_list->item(1, *env);     /* Fetches the 1st parameter. */
CORBA::Any             *ap1 = nv1->value( *env );         /* Fetches Any from the parameter. */
ODsample::diiseq::sampleseq  *sp1 =
           (ODsample::diiseq::sampleseq *)(ap1->value()); /* Fetches the value from Any. */

// Parameter area freeing
//  The area for sp1 is freed by CORBA::release(arg_list).
//  Because the sp1 release flag is set to CORBA_TRUE, the area for _buffer is also freed.
```

INOUT Mode

To pass *inout* parameters to a server application, allocate parameter area in a client application. For information on how to assign a value to an Any type, refer to "5.7.4 Any Type".

After the request is sent to the server, the client application releases allocated area as needed. Use *CORBA_NVList_free()* to release passed area from the server. When an NVList object is deleted, the out area allocated on the server application side is automatically made free. Therefore, the inout parameter area passed from the server application cannot be referenced after the area for the NVList class is made free. For further information on NVList object releasing, refer to "5.2.7 Deleting a Request".

Examples of creating an inout parameter and examples of fetching the processing result are provided below.

Basic Data Type

```
// Parameter area allocation
CORBA::Long             z = 200;
// Setting of a parameter area for the Any type
CORBA::Any              p2;
p2 <<= z;

// inout parameter creation
arg_list->add_value(
      "c",              /* Specifies the same parameter name as specified in the IDL. */
      p2,               /* Specifies an Any type for which the value to be used for inout is
specified. */
      CORBA::ARG_INOUT,    /* Specifies CORBA::ARG_INOUT. */
```

```
      *env );

// Processing result fetching
CORBA::NamedValue_ptr  nv2 = arg_list->item(2, *env);      /* Fetches the 2nd parameter. */
CORBA::Any             *ap2 = nv2->value( *env );          /* Fetches Any from the parameter. */
CORBA::Long            *lp2 = (CORBA::Long *)ap2->value(); /* Fetches the value from Any. */

// Parameter area freeing
//  Because p2 release flag is set to CORBA_TRUE, the parameter area is automatically freed
//  as the result of destructor processing.
//  The area for lp2 is freed by CORBA::release(arg_list).
```

String and Wide String Types

```
// Parameter area allocation
CORBA::Char         *str2 = CORBA::string_alloc(6);
strcpy( str2, "INOUT" );
// Setting of a parameter area for the Any type
CORBA::Any          p2( _tc_string, str2 );

// inout parameter creation
arg_list->add_value(
      "c",                 /* Specifies the same parameter name as specified in the IDL. */
      p2,                  /* Specifies an Any type for which the value to be used for inout is
specified. */
      CORBA::ARG_INOUT,    /* Specifies CORBA::ARG_INOUT. */
      *env );

// Processing result fetching
CORBA::NamedValue_ptr  nv2 = arg_list->item(2, *env);      /* Fetches the 2nd parameter. */
CORBA::Any             *ap2 = nv2->value( *env );          /* Fetches Any from the parameter. */
CORBA::Char            *cp2 = (CORBA::Char *)ap2->value(); /* Fetches the value from Any. */

// Parameter area freeing
//  After request issuance, str2 must be freed when it is no longer needed.
//  The area for cp2 is freed by CORBA::release(arg_list).
CORBA::string_free( str2 );
```

Structure

```
// Parameter area allocation
ODsample::diistruct::data  *data2 = new ODsample::diistruct::data;
data2->para1 = 20;
data2->para2 = 'y';
// Setting of a parameter area for the Any type
CORBA::Any          p2( _tc_ODsample_diistruct_data, data2 );
                           /* _tc_ODsample_diistruct_data is defined */
                           /* in the header file created by IDLc. */

// inout parameter creation
arg_list->add_value(
      "c",                 /* Specifies the same parameter name as specified in the IDL. */
      p2,                  /* Specifies an Any type for which the value to be used for inout is
specified. */
      CORBA::ARG_INOUT,    /* Specifies CORBA::ARG_INOUT. */
      *env );

// Processing result fetching
CORBA::NamedValue_ptr  nv2 = arg_list->item(2, *env);     /* Fetches the 2nd parameter. */
CORBA::Any             *ap2 = nv2->value( *env );         /* Fetches Any from the parameter. */
ODsample::diistruct::data  *dp2 =
```

```
          (ODsample::diistruct::data *)(ap2->value());   /* Fetches the value from Any. */

// Parameter area freeing
//  After request issuance, data2 must be freed when it is no longer needed.
//  The area for dp2 is freed by CORBA::release(arg_list).
delete data2;
```

Sequence Type

```
// Parameter area allocation
CORBA::Long         *q = ODsample::diiseq::sampleseq::allocbuf(5);
for( int i = 0; i < 5; i++ )
    q[i] = i*100;
ODsample::diiseq::sampleseq  *seq2 =
                    new ODsample::diiseq::sampleseq( 5, 5, q, CORBA_TRUE );
// Setting of a parameter area for the Any type
CORBA::Any          p2( _tc_ODsample_diiseq_sampleseq, seq2 );
                        /* _tc_ODsample_diiseq_sampleseq is defined */
                        /* in the header file created by IDLc. */

// inout parameter creation
arg_list->add_value(
      "c",                /* Specifies the same parameter name as specified in the IDL. */
      p2,                 /* Specifies an Any type for which the value to be used for inout is
specified. */
      CORBA::ARG_INOUT,   /* Specifies CORBA::ARG_INOUT. */
      *env );

// Processing result fetching
CORBA::NamedValue_ptr  nv2 = arg_list->item(2, *env);     /* Fetches the 2nd parameter. */
CORBA::Any             *ap2 = nv2->value( *env );         /* Fetches Any from the parameter. */
ODsample::diiseq::sampleseq  *sp2 =
          (ODsample::diiseq::sampleseq *)(ap2->value()); /* Fetches the value from Any. */

// Parameter area freeing
//  After request issuance, seq2 must be freed when it is no longer needed.
//  The area for sp2 is freed by CORBA::release(arg_list).
//  Because the sp2 release flag is set to CORBA_TRUE, the area for _buffer is also freed.
delete seq2;              /* Because the release flag is set to CORBA_TRUE, */
                          /* the area (q) allocated by allocbuf is also freed. */
```

RETURN

To receive the server application processing result from a return parameter, the client application does not need to allocate an area. For the second parameter of add_value(), specify an Any type with the appropriate TypeCode.

Areas allocated for data passed from the server application are automatically freed when the area for the NVList class is freed. Therefore, the return parameter area passed from the server application cannot be referenced after the area for the NVList class is made free. For further information on NVList object releasing, refer to "5.2.7 Deleting a Request".

Examples of creating a return parameter and examples of fetching the processing result are provided below.

Basic Data Type

```
CORBA::NVList_ptr   arg_tmp;

// List object creation
orb->create_list( 1, arg_tmp, *env );

// Setting of TypeCode for the Any type
CORBA::Any          tmp( _tc_long, NULL );
```

```
// Return parameter creation
CORBA::NamedValue_ptr  result =
      arg_tmp->add_value(
             NULL,               /* Specifies NULL. */
             tmp,                /* Specifies an Any type for which the TypeCode is specified. */
             (CORBA::Flags)0,    /* Specifies 0. */
             *env);

// Processing result fetching
CORBA::Any             *ap3 = result->value( *env );       /* Fetches Any from the parameter. */
CORBA::Long            *lp3 = (CORBA::Long *)ap3->value(); /* Fetches the value from Any. */

// Parameter area freeing
//  The area for lp3 is freed by CORBA::release(arg_tmp).
```

String and Wide String Types

```
CORBA::NVList_ptr   arg_tmp;

// List object creation
orb->create_list( 1, arg_tmp, *env );

// Setting of TypeCode for the Any type
CORBA::Any          tmp( _tc_string, NULL );

// Return parameter creation
CORBA::NamedValue_ptr  result =
      arg_tmp->add_value(
             NULL,               /* Specifies NULL. */
             tmp,                /* Specifies an Any type for which the TypeCode is specified. */
             (CORBA::Flags)0,    /* Specifies 0. */
             *env);

// Processing result fetching
CORBA::Any             *ap3 = result->value( *env );       /* Fetches Any from the parameter. */
CORBA::Char            *cp3 = (CORBA::Char *)ap3->value(); /* Fetches the value from Any. */

// Parameter area freeing
//  The area for cp3 is freed by CORBA::release(arg_tmp).
```

Structure

```
CORBA::NVList_ptr   arg_tmp;

// List object creation
orb->create_list( 1, arg_tmp, *env );

// Setting of TypeCode for the Any type
CORBA::Any          tmp( _tc_ODsample_diistruct_data, NULL );
                                /* _tc_ODsample_diistruct_data is defined */
                                /* in the header file created by IDLc. */

// Return parameter creation
CORBA::NamedValue_ptr  result =
      arg_tmp->add_value(
             NULL,               /* Specifies NULL. */
             tmp,                /* Specifies an Any type for which the TypeCode is specified. */
             (CORBA::Flags)0,    /* Specifies 0. */
             *env);
```

```
// Processing result fetching
CORBA::Any            *ap3 = result->value( *env );       /* Fetches Any from the parameter. */
ODsample::diistruct::data  *dp3 =
          (ODsample::diistruct::data *)(ap3->value());    /* Fetches the value from Any. */

// Parameter area freeing
//  The area for dp3 is freed by CORBA::release(arg_tmp).
```

Sequence Type

```
CORBA::NVList_ptr   arg_tmp;

// List object creation
orb->create_list( 1, arg_tmp, *env );

// Setting of TypeCode for the Any type
CORBA::Any          tmp( _tc_ODsample_diiseq_sampleseq, NULL );
                                 /* _tc_ODsample_diiseq_sampleseq is defined */
                                 /* in the header file created by IDLc. */

// Return parameter creation
CORBA::NamedValue_ptr  result =
      arg_tmp->add_value(
             NULL,               /* Specifies NULL. */
             tmp,                /* Specifies an Any type for which the TypeCode is specified. */
             (CORBA::Flags)0,    /* Specifies 0. */
             *env);

// Processing result fetching
CORBA::Any            *ap3 = result->value( *env );      /* Fetches Any from the parameter. */
ODsample::diiseq::sampleseq  *sp3 =
          (ODsample::diiseq::sampleseq *)(ap3->value()); /* Fetches the value from Any. */

// Parameter area freeing
//  The area for sp3 is freed by CORBA::release(arg_tmp).
//  Because the sp3 release flag is set to CORBA_TRUE, the area for _buffer is also freed.
```

5.7.12 Passing Parameters to a Server Application

The following table lists the valid data types for passing parameters to a server application.

Table 5.22 Data Types Valid for Passing Parameters (C++)

CORBA data type	In	Out	Inout	Return
long	Long	Long&	Long&	Long
unsigned long	Ulong	ULong&	ULong&	ULong
short	Short	Short&	Short&	Short
unsigned short	UShort	UShort&	UShort&	UShort
long long	LongLong	LongLong&	LongLong&	LongLong
float	Float	Float&	Float&	Float
double	Double	Double&	Double&	Double
long double Windows32/64 Solaris32/64	LongDouble	LongDouble&	LongDouble&	LongDouble
char	Char	Char&	Char&	Char

CORBA data type	In	Out	Inout	Return
wchar	WChar	WChar&	WChar&	WChar
octet	Octet	Octet&	Octet&	Octet
boolean	Boolean	Boolean&	Boolean&	Boolean
enum	enum	enum&	enum&	enum
string	const Char*	Char*&	Char*&	Char*
wstring	const WChar*	WChar*&	WChar*&	WChar*
any	const any&	any*&	any&	any*
sequence	const sequence&	sequence*&	sequence&	sequence*
struct, fixed	const struct&	struct&	struct&	struct
struct, variable	const struct&	struct*&	struct&	struct*
union, fixed	const union&	union&	union&	union
union, variable	const union&	union*&	union&	union*
array, fixed	const array	array	array	array slice*
array, variable	const array	array slice*&	array	array slice*
Object	CORBA::Object_ptr	CORBA::Object_ptr&	CORBA::Object_ptr&	CORBA::Object_ptr
TypeCode	CORBA::TypeCode_ptr	CORBA::TypeCode_ptr&	CORBA::TypeCode_ptr &	CORBA::TypeCode_ptr

Assigning NULL pointer

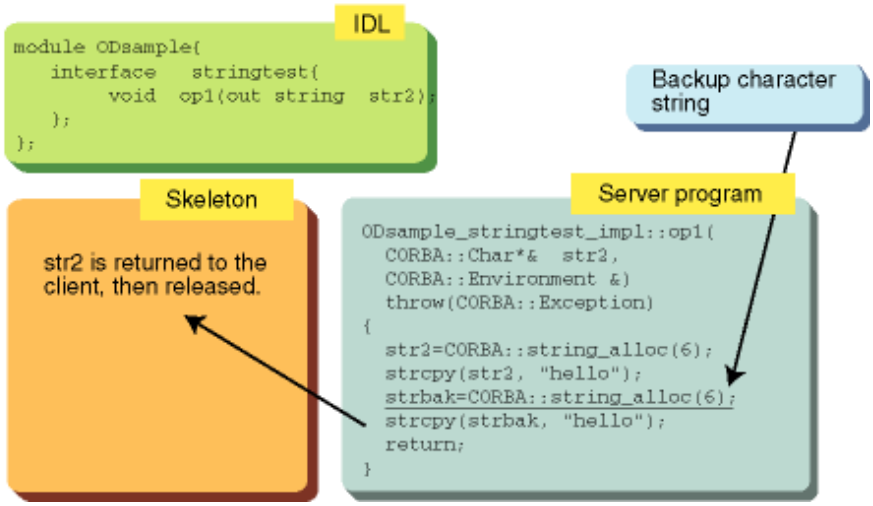
NULL pointer cannot be assigned to *out, inout* and *return* parameters of server application, and to *in* and *inout* parameters of client application, for string, sequence, struct, union, and array types.

5.8 Any Type and Sequence Type Release Flags

Variable-length data is returned to a client, and then automatically released in a skeleton. Data that is not for release must be backed up to another safe location by the application.

The following figure contains an example of releasing and backup copying of "string" type data.

Figure 5.5 Release of "string" Type Data Area



Copying *any*/*sequence* type data is complicated. To resolve this, the Release flag is assigned to release or non-release. The information in the following table shows you how to set the Release flag.

Table 5.23 Any and Sequence Type Release Flags

Variable type	Setting
Any	Use the constructor or "replace" method. The default value is CORBA_FALSE. (Refer to Example 1.)
Sequence	Use the constructor. The default value is CORBA_FALSE. (Refer to Example 2.)

Example 1

```
CORBA::Any_ptr p
   =  new CORBA::Any(_typecode, value, CORBA_TRUE);
     Note:  The default value is FALSE.
    CORBA::Any_ptr  p =  new CORBA::Any;
p->replace( _typecode, value, CORBA_TRUE );
```

Example 2

```
    {
     string  para1;
     long    para2;
    };
    typedef sequence<sample> data;
     ->
    sample_var *p = sample::allocbuf(2);
    data *q = new data( 2, 2, p, CORBA_TRUE );

    // "2,2" indicate maximum and length respectively.)
```

Note

When CORBA_TRUE is used: Release.

When CORBA_FALSE is used: Do not release.

Windows32/64

If the "any" type holds data areas for the basic data types (such as CORBA::Octet, CORBA::Char, and CORBA::Long), the data areas will not be made free regardless of the release flag setting. As indicated in the example below, the data areas must be explicitly released when they are no longer needed.

```
CORBA::Long   *ptr = new CORBA::Long[10]; // Allocates a data area.
for( int i = 0;  i < 10; i++ )
         ptr[i] = i;                      // Sets a value.

CORBA::Any    *p = new CORBA::Any(_tc_long, ptr, CORBA_FALSE);
                                          // Sets the release flag to CORBA_FALSE.

// Frees the area.
delete  p;                                // Frees the any type.
delete  ptr;                              // Frees the data area.
```

5.9 VAR Classes

A client should follow parameter passing rules in CORBA for variable length data:

- For *in* and *inout* parameters, an application must explicitly allocate a location using new. When the location is no longer required the application must then release it using delete.
- For *out* parameter and *return* values, a stub allocates a location. An application must then explicitly release it when it is no longer required.

A memory leak may occur if the above convention is not followed. To simplify the above memory management convention, CORBA also offers var class. Var class enables an application to use variable length data as an automatic variable. Var class is used to:

- Store a pointer to variable length data and release it within destructor.
- Retrieve a data pointer and change it.

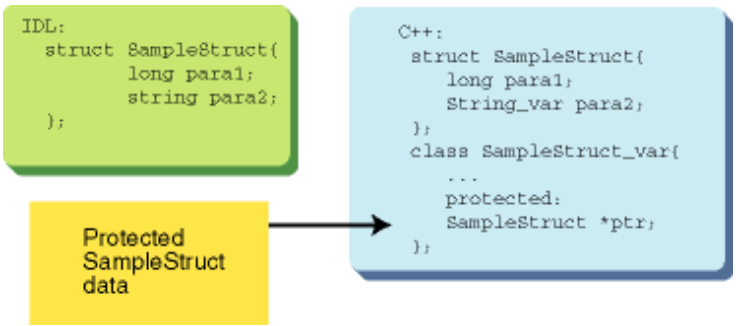
The *var* classes available are listed in the following table.

Table 5.24 Var Classes Available

Class	Description
String_var	var class for strings.
Any_var	var class for Any type.
User defined var class	var class for user defined class. An Object type composed of sequence, struct, union, array, and individual interfaces.

A *var* class example follows.

Figure 5.6 Var Class Example




```
func()
{
     SampleStruct *data = new SampleStruct;
```

```
    ...                          // set data
    SampleStruct_var  var_data
    var_data = data;             // set data pointer to var_data
    ...
    var_data->para1 = 3;         // extract from var_data and overwrite
                                 // struct element para1
    ...
}                                // var_data should be released. Destructor release
                                 // SampleStruct area
```

5.9.1 String_var Class

String_var class provides member functions to simplify string manipulation. It differs from other *var* classes in that two member functions are defined: one with *const* and one without *const*. The *String_var* class definition is shown below. Table 5.25 Method Descriptions in String_var Class gives details of method descriptions in *String_var* class.

```
class String_var {
    public :
        String_var();                                   //default constructor
        String_var( char * );                           // char * constructor
        String_var( const char * );                     // char * constructor
        String_var( const String_var & );               // copy constructor
        ~String_var(); // destructor

        String_var    &operator=( char * );             // char *assign operator
        String_var    &operator=( const char * );       // char *assign operator
        String_var    &operator=( const String_var & ); // assign operator

        operator    char*();                            //  exchange operator
        operator    const char*()       const;          // exchange operator

        char    &operator[]( ULong );                   // [] operator
        char    operator[]( ULong )     const;          // [] operator
    private:
        char                            *_ptr;
};
```

Table 5.25 Method Descriptions in String_var Class

String_var member	Description
default constructor	Creates an instance and initializes _ptr as NULL.
char * constructor	Without const, this creates an instance and assigns specified parameter pointer to _ptr. With const this creates an instance, copies the specified parameter and sets it to _ptr.
copy constructor	Creates an instance, copies the specified data and sets it to _ptr.
Destructor	Releases _ptr using CORBA::string_free.
char *assign operator	Without const: assigns char* pointer to _ptr. With const: copies char* and assigns it to _ptr.
assign operator	Copies _ptr of String_var _ptr and assigns it to _ptr.
exchange operator	Extracts _ptr pointer.
[]operator	Extract index-reference value element by _ptr number.

Default Constructor

Default constructor initializes *_ptr* as NULL. An example of default constructor is shown below.

```
String_var  data;

CORBA::String_var::String_var()
{
     _ptr = NULL;
}
```

Char * Constructor

*Char **constructor with *const* creates an instance and assigns the specified address to *_ptr*. *Char **constructor used without *const* will create an instance and copy the specified parameter to *_ptr*. An example of the *char **constructor is shown below.

```
CORBA::Char * str = COPBA::String_alloc(4);
strcpy( str, "test" );

CORBA::String_var data1( str );
// char* constructor (without const)

CORBA::String_var data2( (const CORBA::Char *)str );
// char *constructor (with const)

CORBA::String_var::String_var( char *p )
{
     _ptr = p;
}

CORBA::String_var::String_var( const char *p )
{
     ...           //copy p to _ptr
}
```

Copy Constructor

Copy constructor creates an instance and copies data pointed from the specified parameter to *_ptr*. An example of copy constructor is shown below.

```
CORBA::Char * str = COPBA::String_alloc(4);
strcpy( str, "test" );

CORBA::String_var data1( str );
             // char* constructor (without const)
CORBA::String_var data2( data1 );
             // copy constructor

CORBA::String_var::String_var(
const CORBA::String_var &r )
{
     ...     //extract _ptr of r and assign it to _ptr.
}
```

Destructor

Destructor releases an instance and location pointed from *_ptr*. An example of destructor is shown below.

```
CORBA::Char * str = COPBA::String_alloc(4);
strcpy( str, "test" );

CORBA::String_var *data = new CORBA::String_var( str );
// char* construct (without const)
```

```
delete data;          //destructor releases _ptr includes str.

CORBA::String_var::~String_var()
{
     CORBA::string_free( _ptr );
}
```

Char * Assign Operator

The *char **assign operator with *const* assigns the specified *char **pointer to *_ptr*. *Char **assign operator without *const* creates a copy of the specified *char **pointer data, and assigns it to *_ptr.* When an instance has *_ptr* data, for operators with or without *const*, it releases *_ptr* data and assigns new data. An example of the *char **assign operator is shown below.

```
CORBA::Char *str = COPBA::String_alloc(4);
strcpy( str, "test" );

CORBA::String_var data;

data = test;          // char *assign operator(without const)

data = (const CORBA::Char *)"new data";
                      // char *assign operator (with const)

CORBA::String_var  &
CORBA::String_var::operator=( char *p )
{
     if( _ptr )
          CORBA::string_free( _ptr );
     _ptr = p;
     return *this;
}

CORBA::String_var  &
CORBA::String_var::operator=( const char *p )
{
     if( _ptr )
          CORBA::string_free( _ptr );
     ...             // make a copy p and assign it to _ptr
     return *this;
}
```

Assign Operator

The assign operator makes a copy of the *_ptr* data of the specified *String_var* and assigns it to *_ptr*. When an instance has *_ptr* data, it releases the *_ptr* data and assigns new data. An example of the assign operator is shown below.

```
CORBA::Char *str = COPBA::String_alloc(4);
strcpy( str, "test" );

CORBA::String_var data;

data = test;          // char *assign operator(without const)

CORBA::String_var  data2;

data2 = data1;        // assign operator

CORBA::String_var  &
CORBA::String_var::operator=( const CORBA::String_var &r )
{
     if( _ptr )
```

```
        CORBA::string_free( _ptr );
        ...        // make a copy of _ptr or r and assign it to _ptr
        return *this;
}
```

[ ] Operator

The [ ] operator returns the data specified by the parameter set in *_ptr* plus the reference value of the first data item. An example and further details are given below.

```
CORBA::Char *str = COPBA::String_alloc(4);
strcpy( str, "test" );

CORBA::String_var data;

data = test;                // char *assign operator (without const)

CORBA::Char x = data[2];    // the 3rd element 's' is returned

char   &
CORBA::String_var::operator[]( ULong index )
{
     return _ptr[index];
}
```

5.9.2 WString_var Class

WString_var class provides member functions to simplify string manipulation. It differs from other *var* classes in that two member functions are defined: one with *const* and one without *const*. The *String_var* class definition is shown below. Table 5.26 Method Descriptions in WString_var Class gives details of method descriptions in W*String_var* class.

```
class WString_var {
     public :
          WString_var();                                    // default constructor
          WString_var( WChar * );                           // char * constructor
          WString_var( const WChar * );                     // char * constructor
          WString_var( const WString_var & );               // copy constructor
          ~WString_var(); // destructor

          WString_var     &operator=( char * );             // char *assign operator
          WString_var     &operator=( const WChar * );      // char *assign operator
          WString_var     &operator=( const WString_var & ); // assign operator

          operator     WChar*();                            // exchange operator
          operator     const WChar*()       const;          // exchange operator

          WChar     &operator[]( ULong );                   // [] operator
          WChar     operator[]( ULong )     const;          // [] operator
     private:
          WChar                             *_ptr;
};
```

Table 5.26 Method Descriptions in WString_var Class

WString_var member	Description
default constructor	Creates an instance and initializes _ptr as NULL.
WChar * constructor	Without const, this creates an instance and assigns specified parameter pointer to _ptr. With const, this creates an instance, copies the specified parameter and sets it to _ptr.

WString_var member	Description
copy constructor	Creates an instance, copies the specified data and sets it to _ptr.
Destructor	Releases _ptr using CORBA::string_free.
WChar *assign operator	Without const: assigns WChar* pointer to _ptr. With const: copies WChar* and assigns it to _ptr.
assign operator	Copies _ptr of WString_var _ptr and assigns it to _ptr.
exchange operator	Extracts _ptr pointer.
[]operator	Extract index-reference value element by _ptr number.

Default Constructor

Default constructor initializes *_ptr* as NULL. An example of default constructor is shown below.

```
WString_var  data;

CORBA::WString_var::WString_var()
{
     _ptr = NULL;
}
```

*WChar ** Constructor

*WChar ** constructor with *const* creates an instance and assigns the specified address to *_ptr*. *WChar ** constructor used without *const* will create an instance and copy the specified parameter to *_ptr*. An example of the *WChar ** constructor is shown below.

```
CORBA::WChar * str = COPBA::wstring_alloc(3);
(*str)[0] = 't';
(*str)[1] = 'e';
(*str)[2] = 's';
(*str)[3] = 't';
(*str)[4] = '\0';

CORBA::WsString_var data1( str );
// WChar* constructor (without const)

CORBA::WString_var data2( (const CORBA::WChar *)str );
// WChar *constructor (with const)

CORBA::WString_var::WString_var( char *p )
{
     _ptr = p;
}

CORBA::WString_var::WString_var( const WChar *p )
{
     ...           //copy p to _ptr
}
```

Copy Constructor

Copy constructor creates an instance and copies data pointed from the specified parameter to *_ptr*. An example of copy constructor is shown below.

```
CORBA::WChar * str = COPBA::wstring_alloc(3);
(*str)[0] = 't';
(*str)[1] = 'e';
```

```
(*str)[2] = 's';
(*str)[3] = 't';
(*str)[4] = '\0';

CORBA::WString_var data1( str );
             // WChar* constructor (without const)
CORBA::WString_var data2( data1 );
             // copy constructor

CORBA::WString_var::WString_var(
const CORBA::WString_var &r )
{
     ...     //extract _ptr of r and assign it to _ptr.
}
```

Destructor

Destructor releases an instance and location pointed from *_ptr*. An example of destructor is shown below.

```
CORBA::WChar * str = COPBA::wstring_alloc(3);
(*str)[0] = 't';
(*str)[1] = 'e';
(*str)[2] = 's';
(*str)[3] = 't';
(*str)[4] = '\0';

CORBA::WString_var *data = new CORBA::WString_var( str );
// WChar* construct (without const)

delete data;          //destructor releases _ptr includes str.

CORBA::WString_var::~WString_var()
{
     CORBA::wstring_free( _ptr );
}
```

*WChar ** Assign Operator

The *WChar ** assign operator with *const* assigns the specified *WChar ** pointer to *_ptr*. *WChar ** assign operator without *const* creates a copy of the specified *char ** pointer data, and assigns it to *_ptr*. When an instance has *_ptr* data, for operators with or without *const*, it releases *_ptr* data and assigns new data. An example of the *WChar ** assign operator is shown below.

```
CORBA::WChar *str = COPBA::wstring_alloc(3);
(*str)[0] = 't';
(*str)[1] = 'e';
(*str)[2] = 's';
(*str)[3] = 't';
(*str)[4] = '\0';

CORBA::WString_var data;

data = (CORBA::WChar *)str;  // WChar *assign operator(without const)

data = (const CORBA::WChar *)str;
                      // WChar *assign operator (with const)

CORBA::WString_var  &
CORBA::WString_var::operator=( WChar *p )
{
     if( _ptr )
          CORBA::wstring_free( _ptr );
```

```
     _ptr = p;
     return *this;
}

CORBA::WString_var  &
CORBA::WString_var::operator=( const WChar *p )
{
     if( _ptr )
          CORBA::wstring_free( _ptr );
     ...             // make a copy p and assign it to _ptr
     return *this;
}
```

Assign Operator

The assign operator makes a copy of the *_ptr* data of the specified *WString_var* and assigns it to *_ptr*. When an instance has *_ptr* data, it releases the *_ptr* data and assigns new data. An example of the assign operator is shown below.

```
CORBA::Char *str = COPBA::wstring_alloc(3);
(*str)[0] = 't';
(*str)[1] = 'e';
(*str)[2] = 's';
(*str)[3] = 't';
(*str)[4] = '\0';

CORBA::WString_var data;

data = test;          // WChar *assign operator(without const)

CORBA::WString_var  data2;

data2 = data1;        // assign operator

CORBA::WString_var  &
CORBA::WString_var::operator=( const CORBA::WString_var &r )
{
     if( _ptr )
          CORBA::wstring_free( _ptr );
          ...        // make a copy of _ptr or r and assign it to _ptr
          return *this;
}
```

[ ] Operator

The [ ] operator returns the data specified by the parameter set in *_ptr* plus the reference value of the first data item. An example and further details are given below.

```
CORBA::WChar *str = COPBA::wtring_alloc(3);
(*str)[0] = 't';
(*str)[1] = 'e';
(*str)[2] = 's';
(*str)[3] = 't';
(*str)[4] = '\0';

CORBA::WString_var data;

data = test;                // WChar *assign operator (without const)

CORBA::WChar x = data[2];     // the 3rd element 's' is returned

WChar   &
```

```
CORBA::WString_var::operator[]( ULong index )
{
     return _ptr[index];
}
```

5.9.3 Any_var Class

The *Any_var* class provides member functions to manipulate *Any* type data. Table 5.27 Method Descriptions in Any_var Class gives details of method descriptions in *Any_var* class.

```
class Any_var {
     public:
          Any_var();                                  // default constructor
          Any_var( Any*);                             // Any* constructor
          Any_var( const Any_var & );                 // copy constructor
          ~Any_var();                                 // destructor

          Any_var     &operator=( Any* );             // Any* assign operator
          Any_var     &operator=( const Any_var & );  // assign operator

          Any          *operator->();                 // pointer operator
          operator     Any*()     const;              // exchange operator

     private:
          Any     *_ptr;
};
```

Table 5.27 Method Descriptions in Any_var Class

Any_var member	Description
default constructor	Creates an instance and initializes _ptr as 0.
Any* constructor	Creates an instance and directs the Any pointer specified in the parameter to _ptr.
copy constructor	Creates an instance, copies __ptr data for the specified parameter and sets it in _ptr.
destructor	Releases _ptr.
Any* assign operator	Assigns Any* pointer to _ptr.
assign operator	Copies Any_var _ptr, and set it to _ptr.
pointer operator	Extracts _ptr.
exchange operator	Extracts _ptr.

Default Constructor

The default constructor creates an instance and initializes *_ptr* as 0. An example of default constructor is shown below.

```
CORBA::Any_var any;              // invoke default constructor

CORBA::Any_var::CORBA::Any_var()
{
     _ptr = 0;
}
```

Any * Constructor

The *Any **constructor creates an instance and assigns Any pointer to *_ptr*. An example of Any *constructor is shown below.

```
CORBA::Any *data = new CORBA::Any();
CORBA::Long x = 3;
```

```
(*data) <<= x;

CORBA::Any_var  any(data);       //invoke Any* constructor

CORBA::Any_var::Any_var( Any *p )
{
     _ptr = p;
}
```

Copy Constructor

The copy constructor creates an instance , makes a copy of *_ptr* specified as Any_var, and assigns it to *_ptr*. An example of copy constructor is shown below.

```
CORBA::Any *data = new CORBA::Any();
CORBA::Long x = 3;
(*data) <<= x;

CORBA::Any_var  any(data);       //invoke Any* constructor

CORBA::Any_var  any2( any );     // invoke copy constructor

CORBA::Any_var::Any_var( const CORBA::Any_var &r )
{
     ...                         // make a copy of _ptr of r and assign it to _ptr
}
```

Destructor

The destructor releases the *_ptr* area. An example of destructor is shown below.

```
func()
{
     CORBA::Any *data = new CORBA::Any();
     CORBA::Long x = 3;
     (*data) <<= x;

     CORBA::Any_var     any(data);     //invoke Any* constructor
     CORBA::Any_var     *any2 = new Any_var( any );

                                       //invoke copy constructor

     delete any2;                      // invoke destructor
}     // when an automatic variable any is released, destructor is invoked.

CORBA::Any_var::~Any_var()
{
     delete _ptr;
}
```

Any * Assign Operator

The *Any **assign operator assigns an *Any **pointer to *_ptr*. The *_ptr* is released when it has a value. An example of the Any * assign operator is shown below.

```
CORBA::Any_var  var_data;
     CORBA::Any *data = new CORBA::Any();
     CORBA::Long x = 3;

     (*data) <<= x;
```

```
    var_data = data;     // invoke Any *assign operator

    CORBA::Any_var::operator=( Any *p )
    {
         if( _ptr )
              delete _ptr;
         _ptr = p;
    }

    CORBA::Any_var::operator=( const Any_var &r )
    {
         if( _ptr )
              delete _ptr;
         ...             // copy _ptr of r to _ptr.
    }
```

Assign Operator

The assign operator copies the *_ptr* data of the specified Any _var and sets it to *_ptr*. When an instance has *_ptr* data, it releases the *_ptr* data and assigns new data. An example of assign operator is shown below.

```
CORBA::Any_var  var_data;
CORBA::Any *data = new CORBA::Any();
CORBA::Long x = 3;

(*data) <<= x;

var_data = data;     // invoke Any *assign operator

CORBA::Any_var var_data2 = var_data;
                     // invoke assign operator
```

Pointer Operator

The pointer operator returns any pointer set to *_ptr*. An example of pointer operator is shown below.

```
CORBA::Any_var  var_data;
CORBA::Any *data = new CORBA::Any();
CORBA::Long x = 3;

(*data) <<= x;

CORBA::Long  *p = var_data->value();
// pointer operator extracts CORBA::Any *
// invoke value() for CORBA::Any *value()

CORBA::Any_var::operator->()
{
     return _ptr;
}
```

Exchange Operator

The exchange operator returns any pointer set to *_ptr*. An example of exchange operator is shown below.

```
CORBA::Any_var  var_data;
CORBA::Any *data = new CORBA::Any();
CORBA::Long x = 3;

(*data) <<= x;
```

```
CORBA::Any *p = (CORBA:;Any *)var_data;
// exchange operator extracts CORBA::Any *

CORBA::Any_var::operator CORBA::Any *()
{
     return _ptr;
}
```

5.9.4 User Defined var Class

The IDL compiler generates corresponding classes and var classes for user defined data, with the type name and suffix of _var. Types of user defined data are sequence, struct, union, array, and Object classes of individual IDL interfaces. Assume that T is a user defined type name. The generated var class has the following member functions. Table 5.28 T_var Class Methods gives details of method descriptions in *T_var* class.

```
class T_var {
     public:
          T_var();                          // default constructor
          T_var( T* );                      // T* constructor
          T_var( const T_var & );           // copy constructor
          ~T_var();                         // destructor

          T_var &operator=( T* );           // T* assign operator
          T_var &operator=( const T_var & );  // assign operator

          operator T*()const;               // exchange operator
          T* operator->() const;            // pointer operator

          //the following is only for sequence
          X_var &operator[]( CORBA::ULong );

          //extract n+1 th element of X(variable length) sequence
          const X_var &operator[]( CORBA::ULong );
          // extract n+1 th element of X(variable length)sequence
          X     &operator[]( CORBA::ULong );
          //[] operator
          const X     &operator[]( CORBA::ULong );
          //[] operator

     protected:
          X *ptr;
}
```

Table 5.28 T_var Class Methods

T_var member	Description
default constructor	Creates an instance of T*.
T* constructor	Creates an instance and initializes _ptr with the specified value.
copy constructor	Creates an instance, copies the specified _ptr data and assigns it to _ptr.
destructor	Releases _ptr.
T* assign operator	Assigns T* pointer to _ptr.
assign operator	Copies T_var _ptr and assigns it to _ptr.
exchange operator	Extracts _ptr data.
pointer operator	Extracts _ptr data.
[] operator	Extracts n+1 th element of _ptr.

Default Constructor

The default constructor creates an instance and initializes *_ptr* as 0. An example of default constructor is shown below.

IDL

```
struct X {
     long x;
     string y;
};
```

C++

```
X_var x;

X_var::X_var()
{
     _ptr = new X;
}
```

T * Constructor

The *T** constructor creates an instance and assigns a *T* pointer to *_ptr*. An example of the *T** constructor is shown below.

IDL

```
struct X {
     long x;
     string y;
};
typedef sequence<long>     LongSeq;
```

C++

```
X *data = new X();
data->x = 3;
data->y = (const CORBA::Char *)"data";

X_var x( data );                  // T* constructor

CORBA::Long *buffer = LongSeq::allocbuf(2);
for( int i = 0; i < 2; i++ )
     buffer[i] = i;
LongSeq *seq = new LongSeq(2,2,buffer,CORBA_TRUE);

LongSeq_var seq_var(seq);          // T* constructor

X_var::X_var( X *p )
{
     _ptr = p;
}

LongSeq_var::Long_var( LongSeq *p )
{
     _ptr = p;
}
```

Copy Constructor

Copy constructor creates an instance, copies a *_ptr* specified as *T_var*, and assigns it to *_ptr*. An example of copy constructor is shown below.

IDL

```
struct X {
     long x;
     string y;
};
typedef sequence<long>     LongSeq;
```

C++

```
X *data = new X();
data->x = 3;
data->y = (const CORBA::Char *)"data";

X_var x_data1( data );               // invoke T* constructor
X_var x_data2( x_data1 );            // invoke copy constructor

CORBA::Long *buffer = LongSeq::allocbuf(2);
for( int i = 0; i < 2; i++ )
     buffer[i] = i;
LongSeq *seq = new LongSeq(2,2,buffer,CORBA_TRUE);

LongSeq_var seq_var1(seq);          // invoke T* constructor
LongSeq_var seq_var2( seq_var1 );   // invoke copy constructor

X_var::X_var( const X_var &r )
{
     ...     // make a copy of ptr of r and set it to _ptr.
}

LongSeq_var::LongSeq_var( const LongSeq_var &r )
{
     ...     // make a copy of _ptr of r and set it to _ptr.
}
```

Destructor

Destructor releases a *_ptr* location. An example of destructor is shown below.

IDL

```
struct X {
     long x;
     string y;
};
```

C++

```
X *data = new X();
data->x = 3;
data->y = (const CORBA::Char *)"data";
```

```
X_var *var_data = new X_var(data);

delete var_data;

X_var::~X_var()
{
     delete _ptr;
}
```

T* Assign Operator

T* assign operator assigns a T pointer to *_ptr*. T* assign operator is shown below.

IDL

```
struct X {
     long x;
     string y;
};
```

C++

```
X *data = new X();
data->x = 3;
data->y = (const CORBA::Char *)"data";
X_var x_var;

x_var = data;          // invoke assign operator

X_var::X_var::operator=( T* p)
{
     _ptr = p;
}
```

Assign Operator

Assign operator copies the *_ptr* data of the specified *T_var*, and sets it to *_ptr*. An example of assign operator is shown below.

IDL

```
struct X {
     long x;
     string y;
};
```

C++

```
X *data = new X();
data->x = 3;
data->y = (const CORBA::Char *)"data";
X_var x_var;

x_var = data;             // invoke T* assign operator

X_var x_var2 = x_var;     // invoke assign operator

X_var::X_var::operator=( const X_var &r )
```

```
{
     ...     // make a copy of _ptr of r and set it to _ptr.
}
```

Exchange Operator

IDL

```
struct X {
     long x;
     string y;
};
```

C++

```
X *data = new X();
data->x = 3;
data->y = (const CORBA::Char *)"data";
X_var x_var;

x_var = data;               // invoke T* assign operator

X *p = (X *)x_var;          // invoke exchange operator

X * X_var::operator X*()
{
     return _ptr;
}
```

Pointer Operator

Pointer operator returns any pointer set to *_ptr*. An example of pointer operator is shown below.

IDL

```
struct X {
     long x;
     string y;
};
```

C++

```
X *data = new X();
data->x = 3;
data->y = (const CORBA::Char *)"data";
X_var x_var;

x_var = data;                 // invoke T* assign operator

CORBA::Long z = x_var->x;     // invoke exchange operator. 2 is 3.
x_var->x = 5;     // invoke exchange operator. 5 is assigned to element x of _ptr.
x_var->y = (const CORBA::Char *)"test";
           // invoke exchange operaotr. "test" is assigned to the element of y of _ptr

X * X_var::operator->()
{
```

```
     return _ptr;
}
```

[] Operator

[] operator returns any pointer set to *_ptr*. An example of [] operator is shown below.

IDL

```
typedef sequence<long>     LongSeq;
typedef sequence<string>     StringSeq;
```

C++

```
CORBA::Long *p = LongSeq::allocbuf(2);
p[0] = 2; p[1] = 3;

LongSeq *long_seq = new LongSeq(2,2,p,CORBA_TRUE);

LongSeq_var long_seq_var = long_seq;     // assign operator

CORBA::Long y = long_seq_var[1];         // [] operator. Y is 3

CORBA::String_var *q = StringSeq::allocbuf(2);
q[0] = (const CORBA::Char *)"test";
q[1] = (const CORBA::Char *)"data";

StringSeq *str_seq = new StringSeq(2,2,q,CORBA_TRUE);

StringSeq_var str_seq_var = str_seq;     // assign operator

CORBA::String_var str = str_seq_var[1];
          // str_ptr is "data"

CORBA::Long &
LongSeq_var::operator[](CORBA::ULong index)
{
     return _ptr->operator[]( index );
}

CORBA::String_var &
StringSeq_var::operator[](CORBA::ULong index)
{
     return (const CORBA::String_var &)
_ptr->operator[](index);
}
```

5.10 Notes on Application Development

Refer to the "Notes on Developing CORBA Applications" chapter.

Chapter 6 Java Programming Guide

This chapter explains how to develop CORBA applications in Java.

Note

This manual explains how to develop CORBA applications using multiple development languages. For details on how to create Java EE applications, refer to the Java EE Operator's Guide.

Point

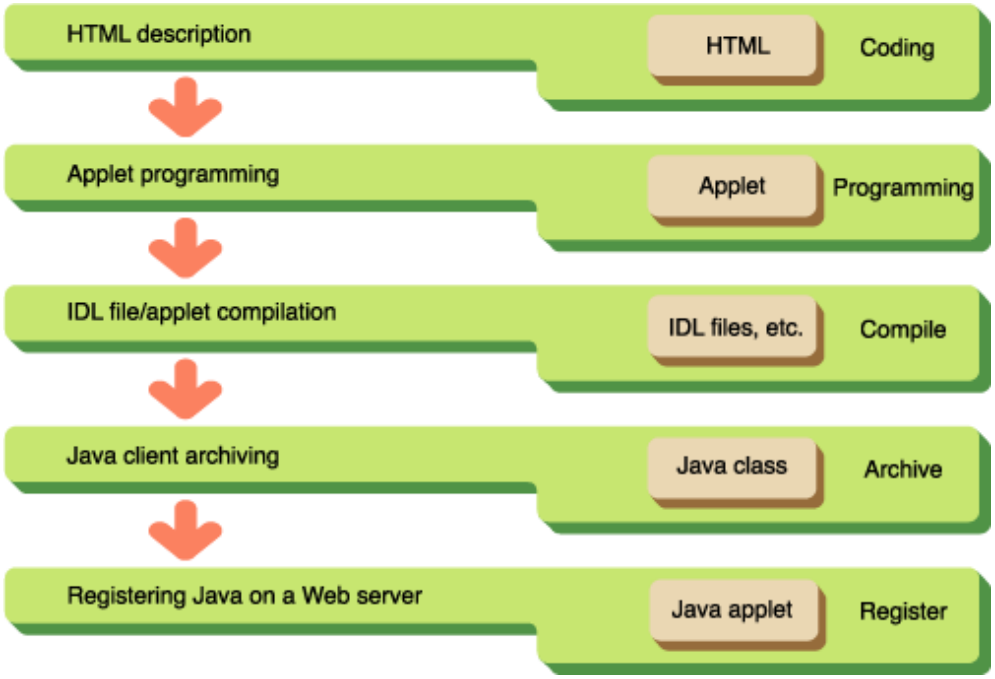
- We recommend using Java EE to develop Java applications to enable IJServer Cluster operations. Compared with CORBA WorkUnit operations, IJServer Cluster operations have the following advantages:
  - Components that conform to the Java EE 5 terms can be used
  - The following Java VM information can be used from the Interstage Management Console:
    - heap information
    - garbage collection information
  - Uptime executable classes/downtime executable classes can be specified

  When the Java CORBA application WorkUnit operation is executed, the heap information cannot be referenced in the Interstage Management Console, however it can be referenced using the jheap command.
- In this manual, "Interstage Studio" is referred to as "Studio".
- The JBK plug-in referred to in this manual is the JBK plug-in included in the client package of Interstage Application Server.

6.1 Java Applet Development Procedure

This section explains the development procedure for Java applets as CORBA clients. That procedure is shown below.

Figure 6.1 Java Applet Development Procedure



6.1.1 Development Procedure (Pre-installed Library)

This section describes development procedures for the pre-installed Java library.

6.1.1.1 Descriptions of HTML Files

To run an applet, specify the applet in a HTML file using the <applet> tag.

This is an operation that exchanges the browser's Java VM for a JDK by plugging it into Internet Explorer.

Refer to "6.20.2 Notes on Applet Operation in the Client Environment" for information on applet operation.

Examples of HTML file descriptions when using the pre-installed Java Library are provided below.

Using the JBK Plug-in

When using the JBK plug-in, create the HTML file as shown below. Refer to the J Business Kit Online Manual of Interstage Studio for more information.

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67" WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="CODE" VALUE="Sample.class">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin" NAME="Sample" CODE="Sample.class" WIDTH=300 HEIGHT=250>
</EMBED>
</COMMENT>
</OBJECT>
</BODY>
</HTML>
```

Using the JBK Plug-in (with a jar-format archive file)

Download the Sample.jar file using the ARCHIVE designation of the <PARAM> tag or the <EMBED> tag.

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67" WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="CODE" VALUE="Sample.class">
<PARAM NAME="ARCHIVE" VALUE="Sample.jar">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin" CODE="Sample.class" ARCHIVE="Sample.jar" WIDTH=300 HEIGHT=250>
</EMBED>
</COMMENT>
</OBJECT>
</BODY>
</HTML>
```

6.1.1.2 Applet Programming

In CORBA-based applications and their associated Java applets on a server, they should refer to org.omg.CORBA class using import text in the initial position. Also, the class declaration is made with the class name described by the HTML file's <applet> tag.

```
import org.omg.CORBA.*;                         // org.omg.CORBA class
import java.awt.*;                              // abstract Windows toolkit class
```

```
public class Sample extends java.applet.Applet  //applet class declaration
{
    ...                                         //process description by Java language
}
```

Note

You need to make the filename of the Java Applet the same as the class name declared within the applet (upper/lower case characters are discriminated).

6.1.1.3 IDL Creation Files and Applet Compilation

Create stubs for Java applets using the IDL compiler.

The stubs created and Java applets listed below should be compiled, with the *javac* command provided with Interstage Application Server:

1. Interface (xxx.java,xxxOperations.java)
2. Helper class (xxxHelper.java)
3. Holder class (xxxHolder.java)
4. Stub class (_xxxStub.java)
5. User-created Java applets

Note

You need to designate CLASSPATH as the current directory (.) and a Java library when running the compiler.

Some sample setups are described below.

Example

Windows32/64 **(OD_HOME:C:\Interstage\ODWIN)**

```
set CLASSPATH=.;%OD_HOME%\etc\class\ODjava4.jar;%CLASSPATH%
```

Solaris32/64 **(OD_HOME:/opt/FSUNod)** Linux32/64 **(OD_HOME:/opt/FJSVod)**

```
CLASSPATH=.:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH
export CLASSPATH
```

6.1.1.4 Java Class File Archives

When registering Java file on a Web server, create an archive file for the collected class files so you can download multiple files just once and shorten the download time. Create the archive file using the jar command (comes with Java Development Kit; shortened to JDK).

When executing the archive file (applet) of Java class files downloaded from a Web server, signatures must be done for the archive file. Refer to "6.7 Digital Signatures in Applets" for a detailed explanation of signatures.

Refer to the JDK documentation for more details about ways to use the jar command.

An example of using commands when creating an archive file from a class file that includes a sub directory is displayed below.

jar command Usage Examples

Designates the archive file name and the class file to be archived that you are creating. It is included in a file on the sub directory in the archive file you have created.

```
jar cvf Sample.jar *.class Samplemod\*.class
    adding: Samplemod/_SampleintfStub.class (in=1282) (out=704) (deflated 45%)
    adding: Samplemod/Sampleintf.class (in=302) (out=215) (deflated 28%)
    adding: Samplemod/SampleintfHelper.class (in=2175) (out=994) (deflated 54%)
    adding: Samplemod/SampleintfHolder.class (in=907) (out=461) (deflated 49%)
```

When recreating the jar archived file to cab archived file, extract the files for a while and re-archive with the cab form. Then, be careful that all the extracted files are contained in the cab archived file.

Sample HTML File Descriptions

A sample HTML file description for when a Java applet has archived the "Sample" and IDL creation file "Sample.jar" is displayed below.

Using the JBK Plug-in (with a jar-format archive file)

When using the JBK Plug-in (with the jar-format archive file), specify the Sample.jar file in the ARCHIVE designations of the <PARAM> or <EMBED> tag.

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67" WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="NAME" VALUE="Sample">
<PARAM NAME="CODE" VALUE="Sample.class">
<PARAM NAME="ARCHIVE" VALUE="Sample.jar">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin" NAME="Sample" CODE="Sample.class" ARCHIVE="Sample.jar"
WIDTH=300 HEIGHT=250>
</EMBED>
</COMMENT>
</OBJECT>
</BODY>
</HTML>
```

6.1.2 Development Procedure (Portable-ORB)

This section contains information about development procedures using the Portable-ORB.

6.1.2.1 Descriptions of HTML Files

To run an applet, specify the applet in an HTML file using the <applet> tag.

This is an operation that exchanges the browser's Java VM for a JDK by plugging it into Internet Explorer.

Refer to "6.20.2 Notes on Applet Operation in the Client Environment" for information on applet operation.

The follow files are described using the <APPLET ARCHIVE> tag or <PARAM> tag (cabbase) in a HTML file running a Java applet in a Portable-ORB download operation.

Table 6.1 Portable-ORB File Names Using Java VM

ORB core	NamingService (see Note below)	Interface Repository (see Note below)
ODporb4_plugin.jar	CosNaming4_plugin.jar	InterfaceRep4_plugin.jar

Note

1. When NamingService or Interface Repository are needed, describe them in an HTML file.

The Portable-ORB file names used differ according to the execution environment of the CORBA application.

Sample HTML file descriptions are provided below.

Using the JBK Plug-in

To use the JBK plug-in, write the HTML file as shown below. Refer to the J Business Kit Online Manual of Interstage Studio for more details.

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67" WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="CODE" VALUE="Sample.class">
<PARAM NAME=" ARCHIVE" VALUE="ODporb4_plugin.jar,CosNaming4_plugin.jar,InterfaceRep4_plugin.jar">
<PARAM NAME="PORB_HOME" VALUE="PORBDIR">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin"
    CODE="Sample.class" WIDTH=300 HEIGHT=250
    ARCHIVE="ODporb4_plugin.jar,CosNaming4_plugin.jar,InterfaceRep4_plugin.jar"
    PORB_HOME="PORBDIR">
</EMBED>
</COMMENT>
</OBJECT>
</BODY>
</HTML>
```

Using the JBK Plug-in (with a jar-format archive file)

Download the Sample.jar file using the ARCHIVE designation of the <PARAM> tag or the <EMBED> tag.

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67" WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="CODE" VALUE="Sample.class">
<PARAM NAME=" ARCHIVE"
VALUE="Sample.jar,ODporb4_plugin.jar,CosNaming4_plugin.jar,InterfaceRep4_plugin.jar">
<PARAM NAME="PORB_HOME" VALUE="PORBDIR">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin"
    CODE="Sample.class" WIDTH=300 HEIGHT=250
    ARCHIVE="Sample.jar,ODporb4_plugin.jar,CosNaming4_plugin.jar, InterfaceRep4_plugin.jar"
    PORB_HOME="PORBDIR">
</EMBED>
</COMMENT>
```

```
</OBJECT>
</BODY>
</HTML>
```

Portable-ORB Not Downloaded (when using the JBK plug-in)

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67" WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="CODE" VALUE="Sample.class">
<PARAM NAME="ARCHIVE" VALUE="Sample.jar">
<PARAM NAME="PORB_HOME" VALUE="PORBDIR">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin"
    CODE="Sample.class" WIDTH=300 HEIGHT=250
    ARCHIVE="Sample.jar" PORB_HOME="PORBDIR">
</EMBED>
</COMMENT>
</OBJECT>
</BODY>
</HTML>
```

Point

- In the example of the Portable-ORB download above, it is assumed that the Portable-ORB file is in the same directory as the HTML file. When they are not in the same directory, assume the path ARCHIVE/VALUE. When using a Solaris/Linux system as a Web server, instead of setting the path, you can substitute a link file.
- PORB HOME is specified by the <PARAM NAME> tag. Designate it as the subdirectory when searching for an operating environment file.

Refer to "6.5.5 Portable-ORB Operation Environment File Settings" for information on specifying the Web server's document root directory/PORBDIR/etc path and storage directory.

6.1.2.2 Applet Programming

CORBA-based applications and their associated Java applets should refer to org.omg.CORBA class using import text in the initial position. Also, the class declaration is made with the class name described by the HTML file's <applet> tag.

```
import org.omg.CORBA.*;                          // org.omg.CORBA class
import java.awt.*;                               // abstract Windows toolkit class

public class Sample extends java.applet.Applet  //applet class declaration
{
     ...                                         //process description by Java language
}
```

Note

You need to make the filename of the Java Applet the same as the class name declared within the applet (upper/lower case characters are discriminated).

6.1.2.3 IDL Creation Files and Applet Compilation

Create stubs for Java applets using the IDL compiler.

The stubs created and Java applets listed below should be compiled, with the *javac* command provided by Interstage Application Server:

1. Interface (xxx.java,xxxOperations.java)
2. Helper class (xxxHelper.java)
3. Holder class (xxxHolder.java)
4. Stub class (_xxxStub.java)
5. User-created Java applets

Note

You need to designate CLASSPATH as the current directory (.) and a Java library when running the compiler. Some sample setups are described below. The JAVA library to be set in CLASSPATH differs according to the execution environment of the CORBA application.

Example

Windows32/64 **(PORB_HOME: C:\Interstage\PORB)**

```
set CLASSPATH=.;%PORB_HOME%\lib\ODporb4.jar;%PORB_HOME%\lib\CosNaming4.jar;
    %PORB_HOME%\lib\InterfaceRep4.jar;%CLASSPATH%
```

Solaris32/64 Linux32/64 **(PORB_HOME: /opt/FJSVporb)**

```
CLASSPATH=.:$PORB_HOME/lib/ODporb4.jar:$PORB_HOME/lib/CosNaming4.jar:
    $PORB_HOME/lib/InterfaceRep4.jar:$CLASSPATH
export CLASSPATH
```

6.1.2.4 Java Class File Archives

When registering Java files on a Web server, create an archive file for the collected class files so you can download all of the files simultaneously; shortening download time. Create the archive file using the jar command (included in the Java Development Kit and commonly shortened to JDK).

The jar archive created with the jar command can be used with the JBK or the Java Plug-ins.

When executing the archive file (applet) of Java class files downloaded from a Web server, signatures must be done for the archive file. Refer to "6.7 Digital Signatures in Applets" for detail explanation about signature.

See

Refer to the JDK documentation for more details about how to use the jar command.

An example of using commands when creating an archive file from a class file that includes a sub directory is displayed below.

jar command Usage Examples

Designates the archive file name and the class file to be archived that you are creating. It is included in a file on the sub directory in the archive file you have created.

```
jar cvf Sample.jar *.class Samplemod\*.class
    adding: Samplemod/_SampleintfStub.class (in=1282) (out=704) (deflated 45%)
    adding: Samplemod/Sampleintf.class (in=302) (out=215) (deflated 28%)
    adding: Samplemod/SampleintfHelper.class (in=2175) (out=994) (deflated 54%)
    adding: Samplemod/SampleintfHolder.class (in=907) (out=461) (deflated 49%)
```

Note

- The class file and its path are stored in an archive. When a set of classes are in a directory structure, an archive containing all the classes and their paths is created.
- When "cabbase" is set to <PARAM> tag in HTML, setting the "ARCHIVE" to the <APPLET> tag will not work.

Sample HTML File Descriptions

A sample HTML file description for a Java applet which archives the "Sample" and IDL creation file as "Sample.jar" is shown below.

Using the JBK Plug-in (with a jar-format archive file)

When using the JBK Plug-in (with the jar-format archive file), specify the Sample.jar file in the ARCHIVE designations of the <PARAM> or <EMBED> tag.

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67"
 WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="NAME" VALUE="Sample">
<PARAM NAME="CODE" VALUE="Sample.class">
<PARAM NAME=" ARCHIVE"
VALUE="Sample.jar,ODporb4_plugin.jar,CosNaming4_plugin.jar,InterfaceRep4_plugin.jar">
<PARAM NAME="PORB_HOME" VALUE="PORBDIR">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin"
       NAME="Sample" CODE="Sample.class" WIDTH=300 HEIGHT=250
       ARCHIVE="Sample.jar,ODporb4_plugin.jar,CosNaming4_plugin.jar,InterfaceRep4_plugin.jar"
       PORB_HOME="PORBDIR">
</EMBED>
</COMMENT>
</OBJECT>
</BODY>
</HTML>
```

Portable-ORB Not Downloaded

This HTML file description deletes the Portable-ORB file description from the HTML file description of "Using the JBK Plug-in (with a jar-format archive file)".

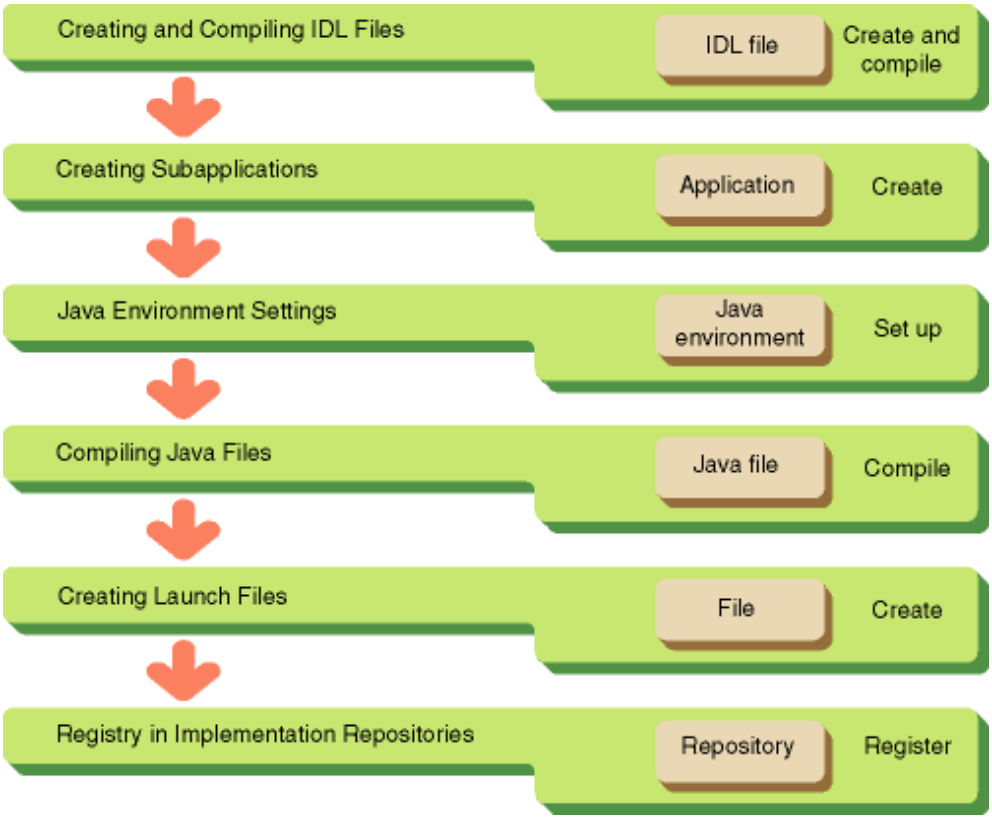
6.1.3 Registering Java Applets with a Web Server

HTML files, Java applets, and IDL creation file classes are stored in the same directory on a Web server.

6.2 Server Application Development Procedures and Environment Settings

In this section we will give an overview of the procedures for developing server applications, using the Static Skeleton Interface as an example. The environment settings needed to develop and run server applications will also be explained. The procedure is outlined in the figure below.

Figure 6.2 Overview of Server Application Development Procedures and Environment Settings



6.2.1 Creating and Compiling IDL Files

1. Create an IDL file in order to define a server application-supported interface.
2. Compile the IDL file with the IDLc command. With IDLc, the following files are generated:

<IDL-defined interface name>Operations.java <IDL-defined interface name>.java	Interface
<IDL-defined interface name>POA.java	Skeleton class
<IDL-defined interface name>POATie.java	Tie class using substitute format (*1)
xxxHolder.java	Holder class
xxxHelper.java	Helper class
<IDL-defined interface name>Stub.java	Stub class (*2)

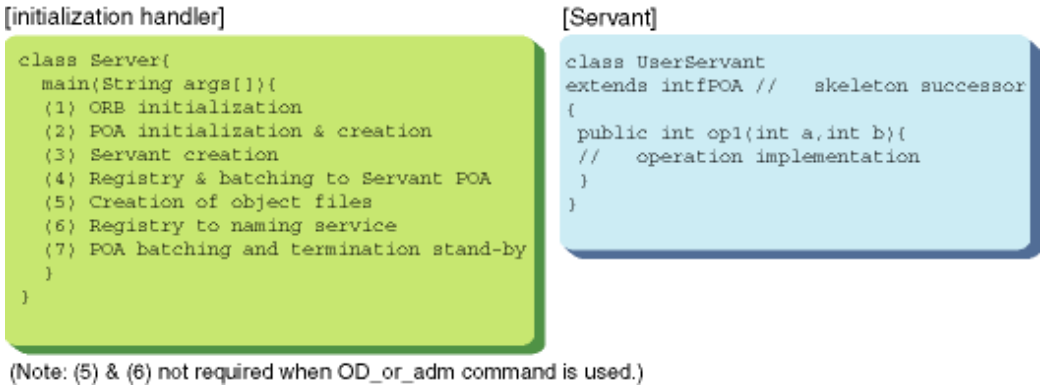
*1 Refer to "6.16 Server Application Implementation Approaches".

*2 It is used as for server applications.

6.2.2 Creating Server Applications

A server application is divided into an "initialization handler" and an "interface implementer". The respective portions are implemented along the lines of the functions displayed in the following figure. Refer to "6.14 Programming Server Applications (Static Skeleton Interface)" for details.

Figure 6.3 Server Application Structure



(Note: (5) & (6) not required when OD_or_adm command is used.)

6.2.3 Compiling Java Files

Compile with the javac command, which combines the *. java files created by the IDL compiler and the server applications created by the user.

Note

Also, you will need to set the Java library as the CLASSPATH when compiling Java files.

Refer to "6.1.1.3 IDL Creation Files and Applet Compilation" for the setting example.

6.2.4 Creating Launch Files

Prepare an ordinary batch file or shell script for launching server applications. At the time of launch, you will need to set an Implementation Repository ID to the environment variable OD_IMPLID. This setting can be run inside the launch file as well. You will not need the OD_IMPLID setting when you specify the Implementation Repository ID to a POA object's adapter name. For details, refer to "6.13 Relating Server Applications and Environment Settings".

Also, you will need to set the java library as the environment variable CLASSPATH when running Java server applications.

Example

Windows32/64(OD_HOME:C:\Interstage\ODWIN)

```
set CLASSPATH=.;%OD_HOME%\etc\class\ODjava4.jar;%CLASSPATH%
```

Launch Batch File Example

```
set OD_IMPLID=IDL:ODdemo/calculator:1.0
                Set Implementation Repository ID.
set CLASSPATH=<path of Server.class>;%CLASSPATH%;
                Enter environment variables and other settings, as required.
java Server     Server class in argument starts Java VM
```

Solaris32/64 (OD_HOME:/opt/FSUNod) Linux32/64 (OD_HOME: /opt/FJSVod)

$OD_HOME/lib must be set in both the CLASSPATH variable and the LD_LIBRARY_PATH environment variable.

```
CLASSPATH=.:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH
LD_LIBRARY_PATH-$OD_HOME/lib
export CLASSPATH
export LD_LIBRARY_PATH
```

Launch Shell Script Example Solaris32/64 Linux32/64

```
OD_IMPLID=IDL:ODdemo/calculator:1.0
                Set Implementation Repository ID.
export OD_IMPLID
                Enter environment variables and other settings, as required.
CLASSPATH=<Server.class path>:$OD_HOME/etc/class/ODjava.jar:$CLASSPATH:
LD_LIBRARY_PATH=$OD_HOME/lib
export CLASSPATH
export LD_LIBRARY_PATH
java Server      Server class in argument starts Java VM
```

6.2.5 Registration in Implementation Repositories

Use the OD_impl_inst command in order to register server application information in the Implementation Repository. An example of registry handling with the OD_impl_inst command and the information to specify is displayed below.

Example

```
OD_impl_inst -ax def
```

-ax def

Shows the registration of server application information with information specified by the definitions file.

Sample def File Contents

```
rep_id        = IDL:ODdemo/calculator:1.0
type          = persistent
mode          = SYNC_END
iswitch       = ON
proc_conc_max = 1
thr_conc_init = 1
ior           = 1.1
```

rep_id = IDL:ODdemo/calculator:1.0

Specifies the Implementation Repository ID.

type = persistent

Specifies subtypes. Specifies persistent for a Java application.

mode = *SYNC_END*

Is a mode for receipt preparation completion notification to CORBA Service. Specify SYNC_END for Java server applications.

iswitch = ON

Specifies whether the server application will preserve (ON/OFF)interface information to each client application. Be sure to specify ON for selecting the Factory mode or user interface control mode as the application configuration. Refer to "6.11.1 Types of Application Configurations" for information about application configurations.

proc_conc_max = 1

Specifies process maximum multiplicity.

thr_conc_init = 1

Specifies thread initial multiplicity. Default is 16.

ior = 1.1

Specifies cooperative Interoperable Object Reference (IOR) versions. Be sure to specify 1.1 when locale = UNICODE. (Default 1.0).

6.2.6 Security Manager Settings Solaris64

To use Security Manager, set one of the following:

System property settings

Set the following system property:

```
jdk.corba.allowOutputStreamSubclass=true
```

Java library permission settings

Set the SerializablePermission enableSubclassImplementation permission in the following Java libraries:

- /opt/FSUNod/etc/class/ODjava4.jar
- User application classes and libraries

6.3 Execution of CORBA Applications

This section explains the Java execution environment provided as the execution environment for a CORBA application.

The following table shows the operable combinations of JDK/JRE provided by JBK plug-in and Interstage Studio.

Table 6.2 Operable JDK/JRE Combinations

Version of JDK/JRE	JBK plug-in	Interstage Studio		
		V9.2	V10	V11 or later
JDK/JRE 7	Supported	Not supported	Not supported	Supported
JDK/JRE 6	Supported	Supported	Supported	Supported

6.3.1 ORB (Object Request Broker) Setup

The environment variables to launch applications or applets need to select the ORB to be used.

The procedure for selecting an ORB in the system is established by CORBA.

This section explains how to specify a CORBA service (ObjectDirector) as the ORB to be used based on the specification.

Note

ORB must be specified. For Portable-ORB applet operations, do not specify org.omg.CORBA.ORBSingletonClass as the property name.

Ways of Specifying ORB

There are two ways of specifying the ORB to be used:

- Specifying when launching application

- Preparing a setup file

In both methods, specifying the properties shown below in Java runtime makes that CORBA service's ORB usable.

Property name	Value
org.omg.CORBA.ORBClass	com.fujitsu.ObjectDirector.CORBA.ORB
org.omg.CORBA.ORBSingletonClass	com.fujitsu.ObjectDirector.CORBA.SingletonORB

The following describes each way of specifying ORB.

Specifying when Launching Application

When executing a Java application, set property information as parameters of the java command. As shown below, enter the required information following the -D option (if an application is activated using a batch file or shell script, specify the following in these files).

Property Setup

```
java -Dorg.omg.CORBA.ORBClass=com.fujitsu.ObjectDirector.CORBA.ORB
     -Dorg.omg.CORBA.ORBSingletonClass=com.fujitsu.ObjectDirector.CORBA.SingletonORB
     <application class name>
```

Notes

A space is not put after "-D".

Preparing a Setup File

Create a text file to describe the property information (file name: orb.properties) and store it in the directory <directory name to be set to property name "java-home">\lib.

Store it in the following directories according to the version of JDK/JRE used:

For a Java applet, set it in the executing machine environment.

For Interstage Studio package

```
For JRE 7 (see Note below):
<Java Execution Environment installation directory>\jre7\lib
For JRE 6:
<Java Execution Environment installation directory>\jre6\lib
```

Note: Interstage Studio client application package V11.0 or later

For JDK/JRE in the server package

```
For JRE 7:
C:\Interstage\jre7\lib
For JRE 6:
C:\Interstage\jre6\lib
For JDK 7:
C:\Interstage\jdk7\jre\lib
```

```
For JDK 6:
C:\Interstage\jdk6\jre\lib
```

For JBK Plug-in in the client package

```
For JRE 7:
C:\Interstage\JBKDI\jre7\lib
For JRE 6:
C:\Interstage\JBKDI\jre6\lib
For JDK 7:
C:\Interstage\JBKDI\jdk7\jre\lib
For JDK 6:
C:\Interstage\JBKDI\jdk6\jre\lib
```

Solaris32/64 Linux32/64

```
For JRE 7:
<Java Execution Environment installation directory>\jre7/lib
For JRE 6:
<Java Execution Environment installation directory>\jre6/lib
For JDK 7:
<Java Execution Environment installation directory>\jdk7/jre/lib
For JDK 6:
<Java Execution Environment installation directory>\jdk6/jre/lib
```

Example File Contents (orb.properties)

```
org.omg.CORBA.ORBClass=com.fujitsu.ObjectDirector.CORBA.ORB
org.omg.CORBA.ORBSingletonClass=com.fujitsu.ObjectDirector.CORBA.SingletonORB
```

Example File Contents at Portable-ORB Applet Operation (orb.properties)

```
org.omg.CORBA.ORBClass=com.fujitsu.ObjectDirector.CORBA.ORB
```

6.4 Client Setup (Pre-installed Java Clients)

This section describes the client environment settings for use of Pre-installed Java libraries.

In the client environment settings, it is necessary to set the applet compile environment and describe HTML according to the CORBA application execution environment. Details of the required libraries and how to set and describe each of them are provided below.

See

Solaris32/64

When operating a pre-installed type Java client, refer to "6.7 Digital Signatures in Applets".

6.4.1 Description of HTML Files

Use the applet @JBKPLGPROP parameter to replace Java VM in JBK plug-in units as the execution environment for a CORBA application.

For details of HTML coding, refer to "6.1.2.1 Descriptions of HTML Files".

For details on the relevant parameters, refer to the J Business Kit Online Manual of Interstage Studio.

6.4.2 Setting Permission for Java Libraries

When executing a Java applet, set permission for the Java library.

The method for setting permission for Java libraries using PolicyTool (attached to JDK) is described below.

Use the following procedure:

1. Start policytool.

```
policytool
```

2. Click **Add Policy Entry** on the [Policy Tool] screen that is displayed after startup.
3. Enter the following on the [Policy Entry] screen:

Item	Set Value
CodeBase	file:<CORBA Service client installation directory>/etc/class/ODjava4.jar (*1)
signed by	(None)

*1 Use "/" as a separator.

4. Click **Add Permission** on the [Policy Entry] screen.
5. Enter a value for each field on the [Permissions] screen, as shown in the following table.

   To set permission from the [Permissions] screen, enter values for the [Permission/Target Name/Actions] fields, and then click **OK**. At this point, control is returned to the [Policy Entry] screen.

   To set another permission, click the Add Permission button again from the [Policy Entry] screen. Repeat this process and enter the required information.

   Click the Done button on the [Policy Entry] screen after all of the values have been entered.

   Permission necessary for usual operation

   These permissions ensure security during usual operation.

Permission type	Setting Permission		
	Permission	Target Name	Actions
Run time permission	RuntimePermission (java.lang.RuntimePermission)	loadLibrary.DLL name (*2)	Setting not required
Property permission	PropertyPermission (java.util.PropertyPermission)	com.fujitsu.*	read

*2 The following dynamic link library (DLL) names are specified depending on the function to install when JDK/JRE is used. The extension need not be specified.

Function to Install	Dynamic Link Library
CORBA Service Client (Client Function)	ODjava4
CORBA Service (Server Function)	ODjavas4

Permission necessary to collect internal logs of CORBA service (*3)

Permission type	Setting Permission		
	Permission	Target Name	Actions
Property permission	PropertyPermission (java.util.PropertyPermission)	user.dir	read
		java.class.path	read
File permission	FilePermission (java.io.FilePermission)	${user.dir}*	read, write
		%OD_HOME%\etc\ config (*4)	read

*3 Refer to "config" in the "CORBA Service Environment Definition" appendix of the Tuning Guide for details of an internal log of the CORBA Service. Delete the added permission after collecting logs.

*4 %OD_HOME% specifies the installation directory of the CORBA Service or the CORBA Service client. Default is C:\Interstage \ODWIN.

6. Select File | Save from the menu bar.
7. Close PolicyTool by selecting File | Exit from the menu bar.

Note

At the initial execution of PolicyTool, select [File]->[Save As] from the [Policy Tool] menu bar before termination of PolicyTool, and specify the name and storage location of the policy file. Refer to "6.7.3 Digital Signature Procedure" for details of specifying policy files.

6.4.3 Library Settings Solaris32/64 Linux32/64

When the pre-installed type Java library is run under Solaris, set the CORBA Service installation directory/lib as the environment variable LD_LIBRARY_PATH.

6.4.4 Security Manager Settings Solaris64

To use Security Manager, set one of the following:

System property settings

Set the following system property:

```
jdk.corba.allowOutputStreamSubclass=true
```

Java library permission settings

Set the SerializablePermission enableSubclassImplementation permission in the following Java libraries:

- /opt/FSUNod/etc/class/ODjava4.jar
- User application classes and libraries

6.5 Client Setup (Portable-ORB)

This section describes the client environment settings for the Portable-ORB.

In the client environment settings, it is necessary to set the applet compile environment and describe the HTML according to the execution environment of CORBA applications. Information on the required libraries and how to set and describe each of them is shown below.

6.5.1 Description of HTML Files

Use the applet @JBKPLGPROP parameter to replace Java VM in JBK plug-in units as the execution environment for a CORBA application.

For details of HTML coding, refer to "6.1.2.1 Descriptions of HTML Files".

For details on the relevant parameters, refer to the J Business Kit Online Manual of Interstage Studio.

Digital signatures are required for applets. For details, refer to "6.7 Digital Signatures in Applets".

6.5.2 Setting Permission for Java Libraries

When executing a Java applet without downloading the Portable-ORB (by assigning the client in advance), set permission for the Java library.

To set permission for Java libraries using PolicyTool (attached to JDK), use the following procedure:

1. Start policytool.

```
policytool
```

2. After starting the tool, click **Add Policy Entry** on the [Policy Tool] screen that is displayed after startup.
3. Enter a value for each item on the [Policy Entry] screen, as shown in the table below.

Item	Value
CodeBase	file:<Portable-ORB Jar file storage directory>/* (*1)
signed by	(None)

*1 Use "/" as a separator.

4. Click **Add Permission** on the [Policy Entry] screen.
5. Enter a value for each field on the [Permissions] screen, as shown in the following table.

   To set permission from the [Permissions] screen, enter values for the [Permission/Target Name/Actions] fields, and then click **OK**. At this point, control is returned to the [Policy Entry] screen.

   To set another permission, click **Add Permission** again from the [Policy Entry] screen. Repeat this process and enter the required information.

   Click **Done** on the [Policy Entry] screen after all the values have been entered.

   Permission necessary for usual operation

   This permission ensures security during usual operation.

Permission type	Setting permission		
	Permission	Target Name	Actions
Communication permission	SocketPermission (java.net.SocketPermission)	Communicated server name (*2)	connect

*2 The communicated server name is set when communicating with server machines other than Web Server which downloads the Java applet. The following host names are specified for the communicated server name.

- The host name set by "Host" of porbeditenv command
- The host name set for object reference of communicated server application
  ("Object host name" displayed by -l option of the *odlistns* command.)

- The host name specified for URL schema
  (Refer to "8.5 URL Schemas" in the "Chapter 8 Naming Service Programming" chapter and "12.4 corbaloc URL Schemas" in the "Chapter 12 Obtaining Naming Service Initial References" chapter for details.)

For example, if the communicated server is serverA.interstage.co.jp and serverB.interstage.co.jp, two communication permissions for serverA.interstage.co.jp and serverB.interstage.co.jp are set to the communicated server name. Or, after verification, it can be set using a wild-card (*) as *.interstage.co.jp. However, do not only specify a wild-card (*) as this may compromise security.

Permission necessary for using EJB application (*3)

Permission type	Setting Permission		
	Permission	Target Name	Actions
Property permission	PropertyPermission (java.util.PropertyPermission)	com.fujitsu.*	read
		java.class.path	read
Runtime permission	RuntimePermission (java.lang.RuntimePermission)	getClassLoader	(Specification not required)

*3 Refer to "EJB Edition" in the J2EE User's Guide for details of the EJB application.

Permission necessary to collect logs of Portable-ORB (*4)

Permission type	Setting Permission		
	Permission	Target Name	Actions
File permission	FilePermission (java.io.FilePermission)	Log collection file (*5)	read, write, delete
		Log collection directory (*6)	read

*4 Refer to "porbeditenv" in the Reference Manual (Command Edition) for details of the log information of the Portable-ORB. It is necessary to create the directory specified by the "Logging Directory" option of the porbeditenv command beforehand when logs are collected. Also, do not store the user resources etc. other than the log collection file in "Logging Directory". Delete the added permission after collecting logs.

*5 This adds "\ *" to the directory name specified by "Logging Directory" of the porbeditenv command. When "Logging Directory" is "C:\log\porb", it is "C:\log\porb\ *".

*6 The directory name specified by "Logging Directory" of the porbeditenv command is specified.

Permission necessary for using SSL

Permission type	Setting Permission		
	Permission	Target Name	Actions
File permission	FilePermission (java.io.FilePermission)	keystore directory	read

Permission necessary for using the user identifier acquisition API by server application (*7)

Permission type	Setting Permission		
	Permission	Target Name	Actions
Property permission	PropertyPermission (java.util.PropertyPermission)	user.name	read

*7 Refer to "TD_get_user_information", "TD::get_user_information", and "TDGETUSERINFORMATION" in the Reference Manual (API Edition) for details of the user identifier acquisition API.

6. Select File | Save from the menu bar.
7. Close PolicyTool by selecting File | Exit from the menu bar.

Note

At the initial execution of PolicyTool, select [File]->[Save As] from the [Policy Tool] menu bar before closing the PolicyTool, and specify the name and storage location of the policy file. Refer to "6.7.3 Digital Signature Procedure" for details of specifying policy files.

6.5.3 Setting Signature for Java Libraries

To execute a Java applet by downloading the Portable-ORB, it is necessary to set the digital signature to the Java libraries of Portable-ORB and set authorization to the signed Java libraries on the client machine that executes the Java applet.

Set the signature and rights, referring to Section 6.7 Digital Signatures in Applets.

At this point, affix your digital signature to the Java libraries of Portable-ORB, using the certificate used for the signature of the applet to be executed after downloading it.

6.5.4 Library Settings

(1) Operation Form in which Portable-ORB is Downloaded

In an operation form in which the Portable-ORB is downloaded, store signed Java libraries on the Web server that downloads Portable-ORB.

An example of what to include in the HTML file (when the Java VM is used) is shown below (TITLE: Java sample Applet, Java applet NAME: Sample)

In the following HTML document example, store them in the same directory as this document.

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67" WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="NAME" VALUE="Sample">
<PARAM NAME="CODE" VALUE="Sample.class">
<PARAM NAME=" ARCHIVE" VALUE="Sample.jar,ODporb4_plugin.jar,
CosNaming4_plugin.jar,InterfaceRep4_plugin.jar">
<PARAM NAME="PORB_HOME" VALUE="PORBDIR">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin"
       NAME="Sample" CODE="Sample.class" WIDTH=300 HEIGHT=250
       ARCHIVE="Sample.jar,ODporb4_plugin.jar,CosNaming4_plugin.jar, InterfaceRep4_plugin.jar"
       PORB_HOME="PORBDIR">
</EMBED>
</COMMENT>
</OBJECT>
</BODY>
</HTML>
```

(2) Operation Form in which Portable-ORB is not Downloaded

In an operation form in which Portable-ORB is not downloaded, the Java libraries must be set to the CLASSPATH variable on the client on which Portable-ORB is run (Portable-ORB is installed).

In an operation form in which pre-installed Java libraries and Portable-ORB are downloaded, no setting/operation is required. Depending on the operation form, the Java library to be set to the CLASSPATH variable is different.

The following is a setting example.

Example

Windows32/64 (PORB_HOME:C:\Interstage\PORB, The installation path is the default path.)

```
set CLASSPATH=.;%PORB_HOME%\lib\ODporb4.jar;%PORB_HOME%\lib\CosNaming4.jar;
    %PORB_HOME%\lib\InterfaceRep4.jar;%CLASSPATH%
```

Solaris32/64 (PORB_HOME:/opt/FJSVporb, The installation path is the default path.)

```
CLASSPATH=.:$PORB_HOME/lib/ODporb4.jar:$PORB_HOME/lib/CosNaming4.jar:
    $PORB_HOME/lib/InterfaceRep4.jar:$CLASSPATH
export CLASSPATH
```

Linux32/64 (PORB_HOME:/opt/FJSVporb)

```
CLASSPATH=.:$PORB_HOME/lib/ODporb4.jar:$PORB_HOME/lib/CosNaming4.jar:
    $PORB_HOME/lib/InterfaceRep4.jar:$CLASSPATH
export CLASSPATH
```

Note

If a Java library (including any pre-installed Java library) of any other ORB is set to CLASSPATH, that other ORB may operate. Do not set Java libraries of any ORB other than Portable-ORB to CLASSPATH.

6.5.5 Portable-ORB Operation Environment File Settings

When using Portable-ORB, you need to set the PORB_HOME parameters with an HTML file in order to specify the storage position for the operation environment files. The operation environment files shown in the following table exist in Portable-ORB.

Table 6.3 Operation Environment Files

Operation Environment Files	File Name
Environment Definition File	config
Object References Information Storage File	initial_services
Object References Search Information File	initial_hosts

You need to store these operation environment files in the Web server's document root sub directory.

Note

Do not store the operation environment files under the user authentication directory when authentication is to be performed by the Web Server based on the user name and password.

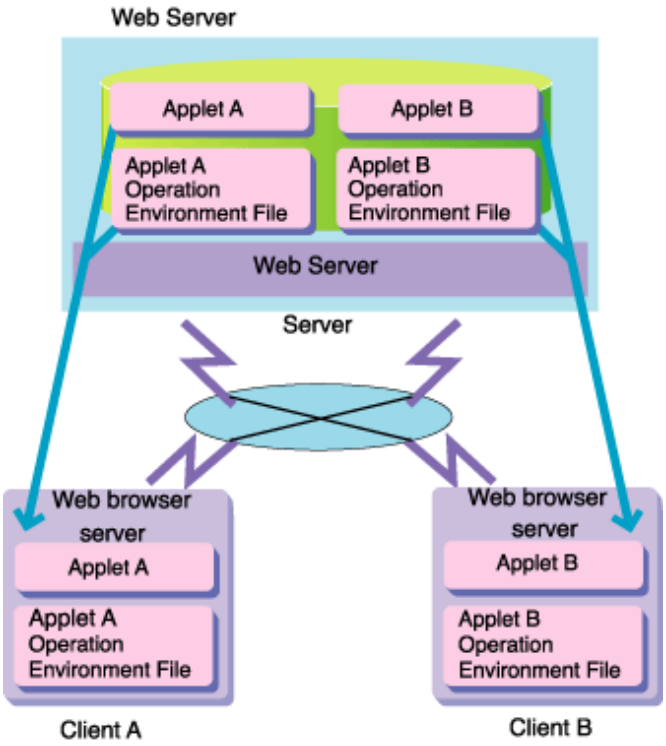
There is a way to specify a different operation environment file for each applet and a way to specify the same operation environment file for multiple applets.

These methods are described below:

Using a Different Operation Environment in each Applet

Using a different operation environment file for each applet is depicted in the following figure.

Figure 6.4 Using Different Operation Environments with Different Applets



Applet A that executes on Client A uses the Applet A operation environment file, and Applet B that executes on Client B uses the Applet B operation environment file

There are two procedures for specifying operation environment files in applet units: one that specifies PORB_HOME and another that does not specify PORB_HOME.

Specifying PORB_HOME

When specifying PORB_HOME, create an etc directory for the operation environment file of each applet and store it in its sub directory. Specify the relative path from the Web server's document root as the storage location for the operation environment files with an HTML file's <PARAM> tag in PORB_HOME parameters. The etc directory is not included in this designation.

Sample directory structures are displayed below. The operation environment file for Applet A is stored in the ap1Aenv/etc sub directory and the operation environment file for Applet B is stored in the ap1Benv/etc sub directory on the Web server's document root directory on the World Wide Web

```
/-
 - [WWW] - [envfile] - [aplAenv] - [etc] - appletA operation environment file
                     - [aplBenv] - [etc] - appletB operation environment file
         - [applet ] - [appletA] - appletA.html
                                 - appletA.class
```

```
                        [appletB] - appletB.html
                                  - appletB.class
```

In the above example, the PORB_HOME parameters for each applet are set with a <PARAM> tag, shown in the following examples.

Applet A <PARAM> Tag Example

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<H1>Java sample Applet</H1>
<applet code="applet.class" width=300 height=250>
<PARAM NAME=PORB_HOME VALUE=envfile/aplAenv>
</applet><BR>
</BODY>
</HTML>
```

Applet B <PARAM> Tag Example

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<H1>Java sample Applet</H1>
<applet code="applet.class" width=300 height=250>
<PARAM NAME=PORB_HOME VALUE=envfile/aplBenv>
</applet><BR>
</BODY>
</HTML>
```

PORB_HOME Not Specified

When PORB_HOME is not specified, create the etc directory in the directory where each applet is stored and store the operation environment file there. Sample directory structures are displayed below.

```
/-
 - [WWW] - [applet  ] - [appletA] - appletA.html
                                  - appletA.class
                                  - [etc] - appletA operation environment file
                        [appletB] - appletB.html
                                  - appletB.class
                                  - [etc] - appletB operation environment file
```

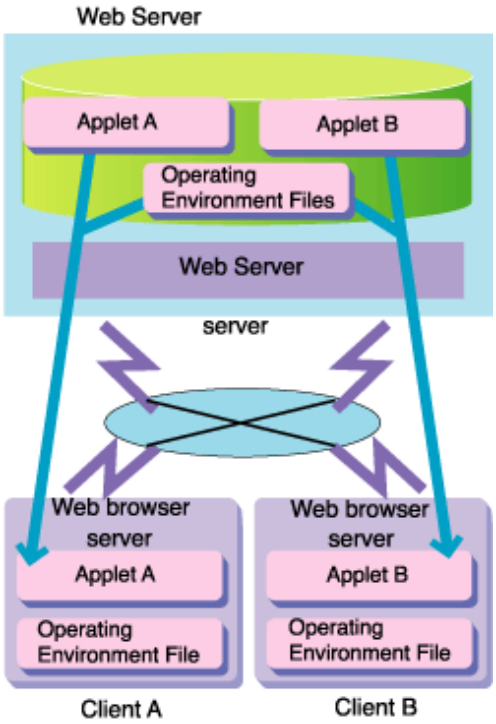
Point

Use the "PORB_HOME not specified" method when using the file protocol. The file protocol is used when HTML files stored on a local disk are directly specified using applet viewer (a browser and JDK tool) instead of the Web Server.

Using a Uniform Operation Environment with Multiple Applets

This is depicted below.

Figure 6.5 Operation Environments under Multiple Applets



Use a common operation environment file with Applet A executing on Client A and Applet B executing on Client B. For the operation environment files' storage location, specify the relative path from the Web server's document root with a <PARAM> tag for Applet A's HTML file and Applet B's HTML file in PORB_HOME parameters. The etc directory is not included in this designation.

Sample directory structures are shown below. Portable-ORB is installed on the /WWW (The Web server's document root directory) sub directory's porb directory, and the operation environment file is stored in its sub directory's etc directory.

```
/-
 - [WWW] - [porb   ] - [lib] - ODporb4.jar...
                     - [etc] - operation environment file
         - [applet ] - [appletA] - appletA.html
                                 - appletA.class
                       [appletB] - appletB.html
                                 - appletB.class
```

Specify a path to /WWW/porb to be able to utilize the operation environment files on the /WWW/porb/etc subdirectory in the HTML files for Applet A and B. The following examples show the <PARAM> tag descriptions for each applet.

Applet A <PARAM> Description Example

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<H1>Java sample Applet</H1>
<applet code="applet A.class" width=300 height=250>
<PARAM NAME=PORB_HOME VALUE=porb>
</applet><BR>
```

```
</BODY>
</HTML>
```

Applet B <PARAM> Description Example

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<H1>Java sample Applet</H1>
<applet code="appletB.class" width=300 height=250>
<PARAM NAME=PORB_HOME VALUE=porb>
</applet><BR>
</BODY>
</HTML>
```

Note

- Use the environment settings command porbeditenv for creating and/or editing operation environment files.
- When creating an etc directory to store the operating environment file, make sure you create it using lower-case alphabetical characters.

6.6 Server Setup

This section describes the server environment settings.

Example

Environment Variable Setting Example Windows32/64

If CLASSPATH is set in a batch file for activation, modify the description in the batch file as follows:

```
set CLASSPATH=.;%OD_HOME%\etc\class\ODjava4.jar;%CLASSPATH%;
```

Environment Variable Setting Example Solaris32/64 Linux32/64

If CLASSPATH is specified in the launch shell script, change the shell script contents as follows:

In addition to CLASSPATH variables, set the <CORBA Service installation directory>/lib as the environment variable LD_LIBRARY_PATH (the OD_HOME:CORBA Service installation directory).

```
CLASSPATH=.:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH:
LD_LIBRARY_PATH=$OD_HOME/lib
export CLASSPATH
export LD_LIBRARY_PATH
```

6.7 Digital Signatures in Applets

You need to execute a digital signature in the primary applet or in the Portable-ORB when downloading and running Java applets. Java applets as CORBA clients are downloaded from the Web server and accessed over a network to a remote machine in which a CORBA server application is running. Therefore, before downloading and operating Java applets, you must perform a digital signature if either a pre-installed Java library or Portable-ORB is to be used.

6.7.1 Digital Signature Overview

Java classes downloaded from the Web server are placed under the control of a security feature called the sand box and access to local computer resources such as files is restricted.

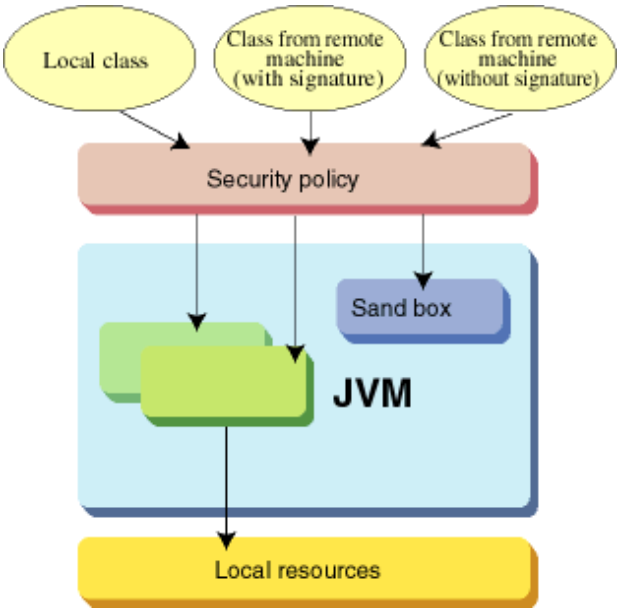
However, since signed (authorized) Java classes are not placed under the control of this sand box, local computer resources become accessible.

Since pre-installed Java libraries access local resources internally (such as access to the dynamic library's load and file), it is necessary to affix your signature to the class that invokes the API of the pre-installed Java libraries.

Security policies define whether code to be executed is executable, or restrictions on the path to accessible files. Java classes check how to execute or to what extent local resources are accessible based on the security policies, prior to execution.

The figure below shows the security features.

Figure 6.6 Security Features



6.7.2 Download Objects and Signature Objects

The relationship between the download objects and digital-signature-executing objects is displayed below.

Table 6.4 Relationship between Download Objects and Signature Objects

Download Object (see Note below)	Signature Object (see Note below)	Signature Format
Applet only	Applet	X.509
Applet/Portable-ORB	Applet/Portable-ORB	

Note: "Applets" include those produced from user create class and IDL files.

See

Refer to "6.1.2.1 Descriptions of HTML Files" regarding the Portable-ORB files.

6.7.3 Digital Signature Procedure

This section describes the procedures for using digital signatures. JDK signature tools such as keytool, jarsigner or policytool are used as the signature tool in this case.

Note

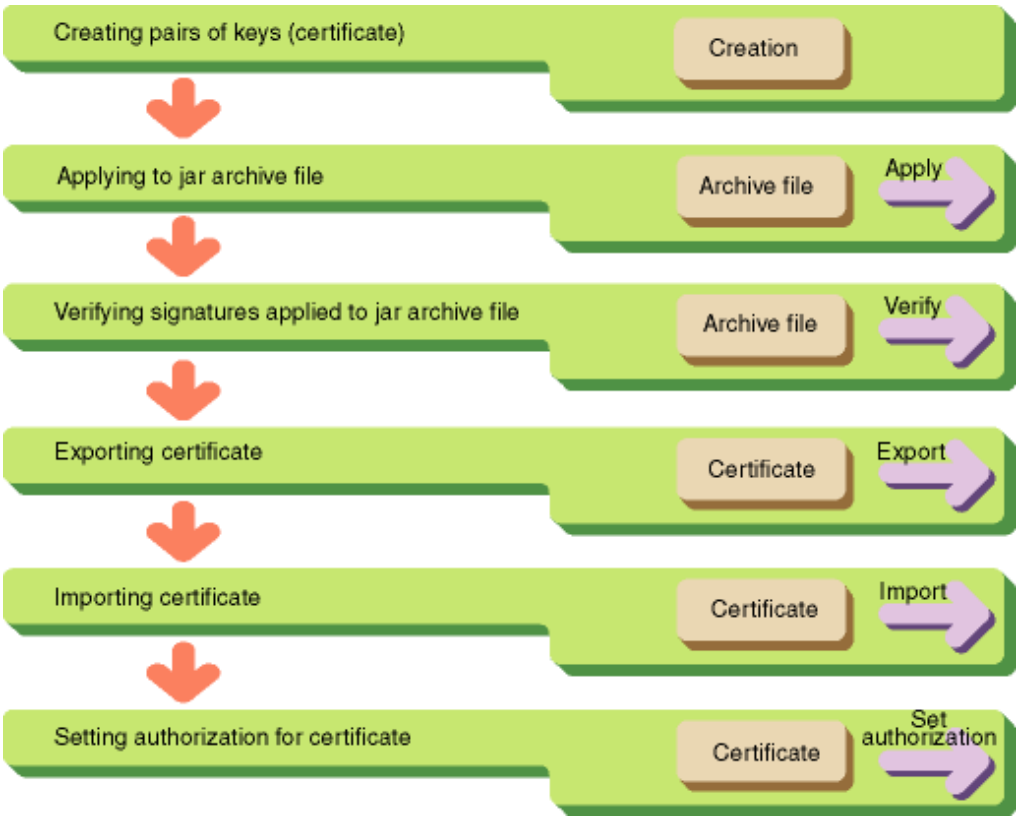
If operation is carried out without downloading Portable-ORB or if pre-installed Java clients are used, authorization needs to be set for the Java libraries in each environment, using policytool. For information about authorization setting for Java libraries, refer to "6.4.2 Setting Permission for Java Libraries".

See

Refer to JDK documentation for details about the signature tools. When using the J Business Kit, refer to the J Business Kit Online Manual of Interstage Studio.

Perform the digital signing procedure shown below.

Figure 6.7 Digital Signing Procedures



(1) Creating a Certificate Key Pair

Create a certificate key pair to specify the additional information of the certificate and the alias name to access it. The example when setting an alias name of 'samplesigner' and 365-day certificate validity is shown below.

```
keytool - genkeypair -alias samplesigner -dname "cn=samplesigner,
  ou=JAVA PROJECT, o=FUJITSU, c=JA" -validity 365
```

-genkeypair

To create a certificate key pair

-alias

Alias name to access the certificate

-dname

Signer, organization, company or country

-validity

Valid term of certificate

When executing, the passwords for the key store and the certificate key pair to be created are required. These passwords are necessary to access the key store and the certificate key pair.

The key store is a database which manages information of the certificate key pair, and it doesn't exist when JDK/JRE is installed but it is created at the first execution of keytool.

Note

The certificates created here and the certificates to be imported by each client machine must be created at the same time. Even if the same content is specified in the -dname option, certificates created at different times will be identified as different certificates.

(2) Application to jar Archive File

Apply digital signature to jar archive file with the created certificate.

In the following example, it is signed for Sample.jar with the certificate created in step (1) above.

```
jarsigner -signedjar Sample.jar.sig Sample.jar samplesigner
```

-signedjar

Specifies the name of the jar file to which a signature is applied (the default value is the name of the signature source jar file).

Sample.jar

Specifies the name of the certificate source jar file.

samplesigner

Specifies the alias of the certificate for which a signature is executed.

When executing, the passwords for the key store and the certificate key pair are required. Input the passwords that you specified in (1) above.

When multiple jar archive files are to be created by the same author, repeat the processing in (2).

(3) Verifying Application of a Signature in a jar Archive File

Use the following commands to verify that the signature is correctly implemented in the jar archive file that applies a signature.

```
jarsigner  -verify Sample.jar.sig
```

-verify

Specifies verification of the digital signature of the jar archive file.

Sample.jar.sig

Specifies the jar archive file whose digital signature is to be verified.

Normally, when a digital signature is implemented, message "jar verified" is displayed.

When a digital signature is not implemented, message "jar is unsigned. (signatures missing or not parsable)" is displayed.

To display the detailed digital signature information, execute the command by specifying the -verbose/-certs option.

(4) Changing jar archive file name

When you specify the jar file to which a signature is applied by the alias of the signature source jar file, change the extension to *.jar to use the file as a jar file (example: When you specify "Sample.jar.sig" for the alias by (2), delete sample.jar used before applying the signature and change "Sample.jar.sig" to "Sample.jar").

(5) Exporting a Certificate

Export the certificate to be used at the client system whom an applet is downloaded into. Specify the alias name of the certificate created in (1) above.

```
keytool -exportcert -alias samplesigner -file samplesign.cer
```

-exportcert

To acquire the certificate

-alias

Alias name of the certificate to acquire

-file

File name to store the certificate to acquire

When executing, the password for the key store is required. Input the password that you specified in (1) above.

(6) Importing a Certificate

Import the certificate to the client system whom an applet is downloaded into. This operation needs to run at the client system. Copy the certificate, samplesign.cer to the client machine in advance.

```
keytool -importcert -alias sampleuser -file samplesign.cer
```

-importcert

To import the certificate

-alias

Alias name of the certificate to import

-file

File name to store the certificate to import

When executing, the password for the key store is required. This is required to access to the key store. When prompted to accept the certificate imported, enter "yes".

The alias (specified in -alias) option is required to specify the certificate (specified in -file) for subsequent operations. Specify an alias of the certificate for which authorization is to be set when setting policy in "(7), Setting a Permission for the Certificate."

(7) Setting a Permission for the Certificate

Use policytool to set permission for the certificate which is imported to the client machine. The permission is set on the client machine using the policytool command (a GUI tool that defines the security policy).

This command is used to set or change the authorization of Java classes signed using this command or stored in any location. For information on the settings of the policytool command, refer to "6.7.4 policytool Command Setting (Supplements)".

6.7.4 policytool Command Setting (Supplements)

This section describes how to set the policytool command using concrete examples. The screen examples used here are based on the following environment:

Library used

Pre-installed Java library

OS

Windows(R) 2003

Java version

JDK/JRE 6

CORBA client installation directory

C:\Interstage\ODWIN

User name

guest

Alias of the imported certificate

sampleuser

(1) Starting policytool

Start policytool as follows.

```
policytool
```

The following [Policy Tool] screen is displayed.

Figure 6.8 Policy Tool Screen

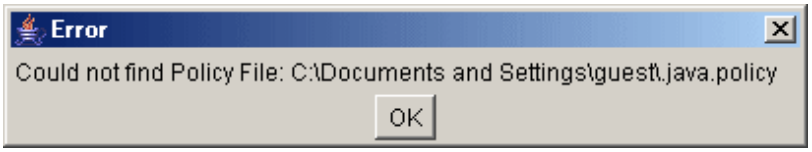
Note

When the policy tool is first executed, warning message is displayed indicating that there are no files to load. This warning can be ignored.

In the second and subsequent activation of the policy tool, the key store and policy tool files that are stored in the default directory are loaded. Therefore, when the files are created in a directory other than the default directory, the files to be used must be specified.

When the files are stored in the default directory, leave the policy settings alone.

Figure 6.9 Policy File Error

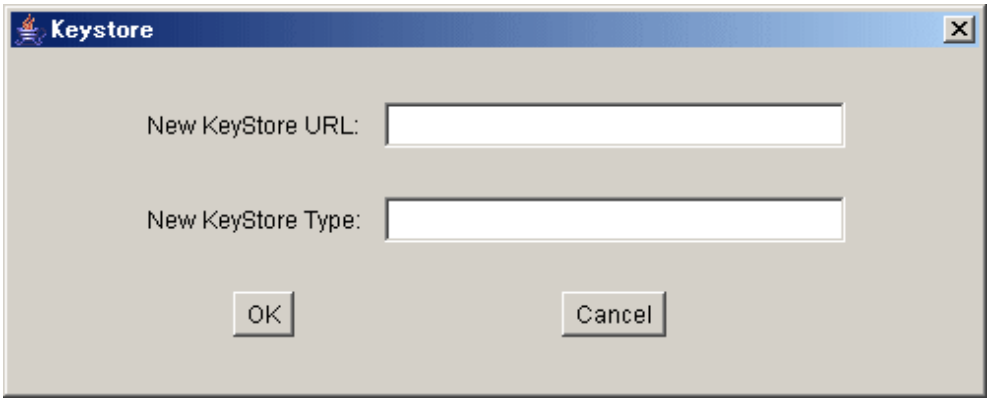


(2) Setting Key Store File

Set the location of key store file and the type of the key store.

Select Edit | Change Key Store on the [Policy Tool] screen that is displayed when the policy tool is invoked. The following [Keystore] screen is displayed.

Figure 6.10 KeyStore Dialog



Specify the key store storage location to be used in the "New KeyStore URL" field and specify the key store type in the "New KeyStore Type" field. The contents to be specified in each file are shown below.

After entering the appropriate values, click the OK button on the [KeyStore] screen.

"New KeyStore URL"

Specify the key store file name created in step (6) including the location in URL form.

The default directory varies in each OS. The storage location in each OS is shown in the following table.

Table 6.5 Default Directory of Key Store File and Policy File

OS	Directory
Windows Server(R) 2003 Windows(R) XP	Profile directory of login user (see Example A)
Windows Server(R) 2008 Windows Server(R) 2012 Windows Vista(R) Windows(R) 7 Windows(R) 8	Profile directory of login user (see Example B)
Solaris Linux	Home directory of login user

Example A

Specify the location of key store as follows, where the user name is guest:

```
"file:c:/Documents and Settings/guest/.keystore"
```

Specify the policy file directory as follows, where the user name is guest:

```
"c:\Documents and Settings\guest\.java.policy"
```

Example B

Specify the location of key store as follows, where the user name is guest:

```
"file:c:/Users/guest/.keystore"
```

Specify the policy file directory as follows, where the user name is guest:

```
"c:\Users\guest\.java.policy"
```

Point

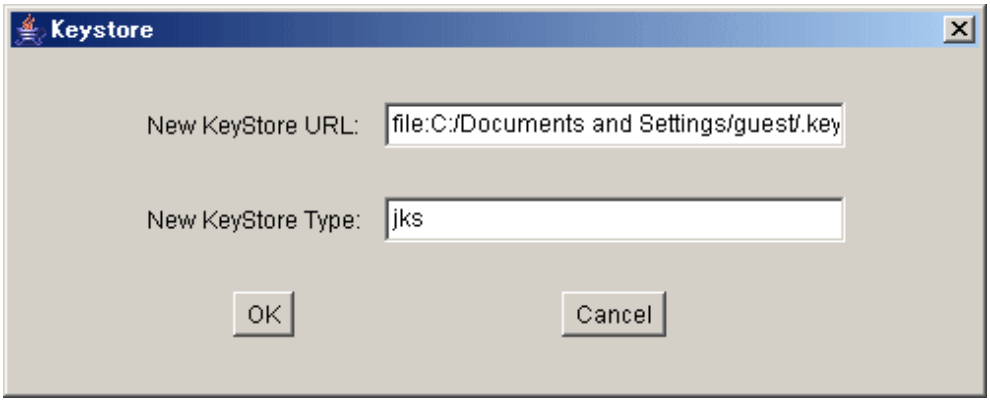
- It is possible to specify the key store in the URL as the location of key store. In this case, the procedures of exporting and importing the certificate shown in step (5) and (6) are not needed.
- The default name of key store file is ".keystore".
- The default name of policy file is ".java.policy".

"New KeyStore Type"

Specify "jks".

An example of the setting screen is shown below.

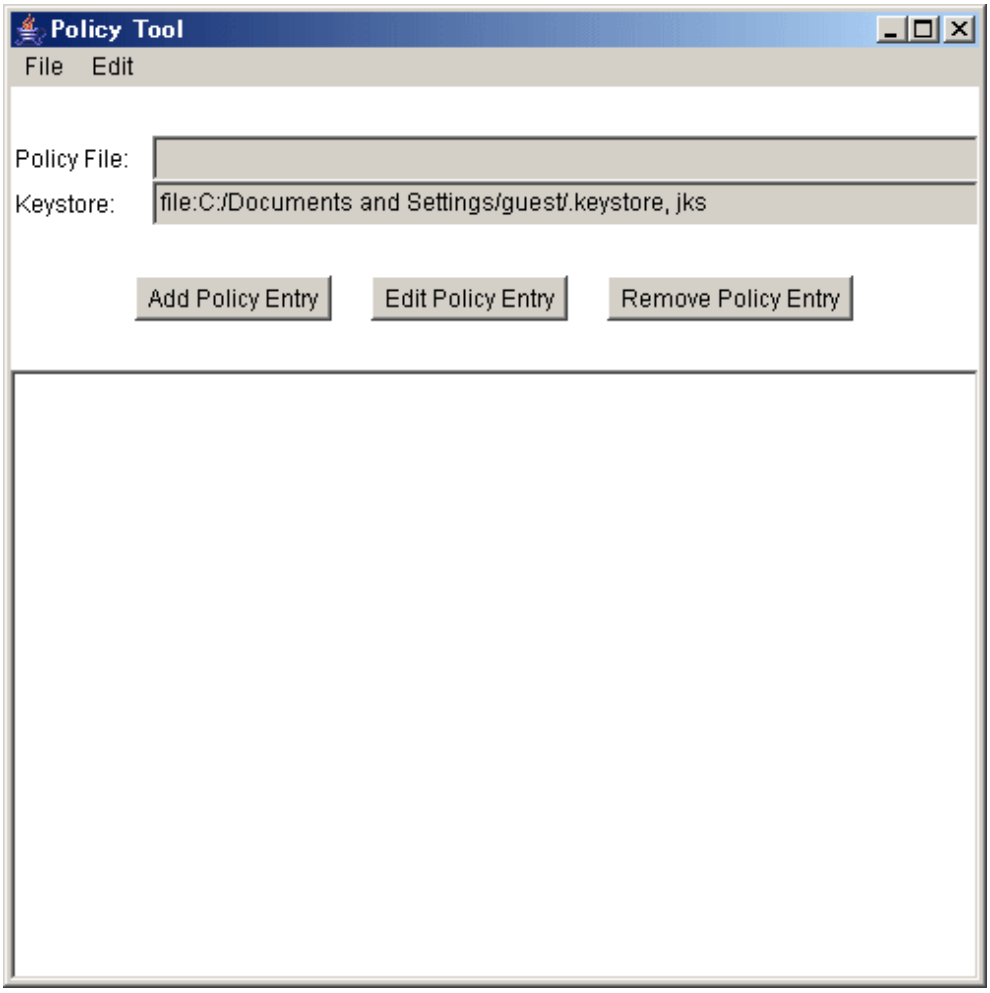
Figure 6.11 Setting New KeyStoreType



(3) Creating an Entry

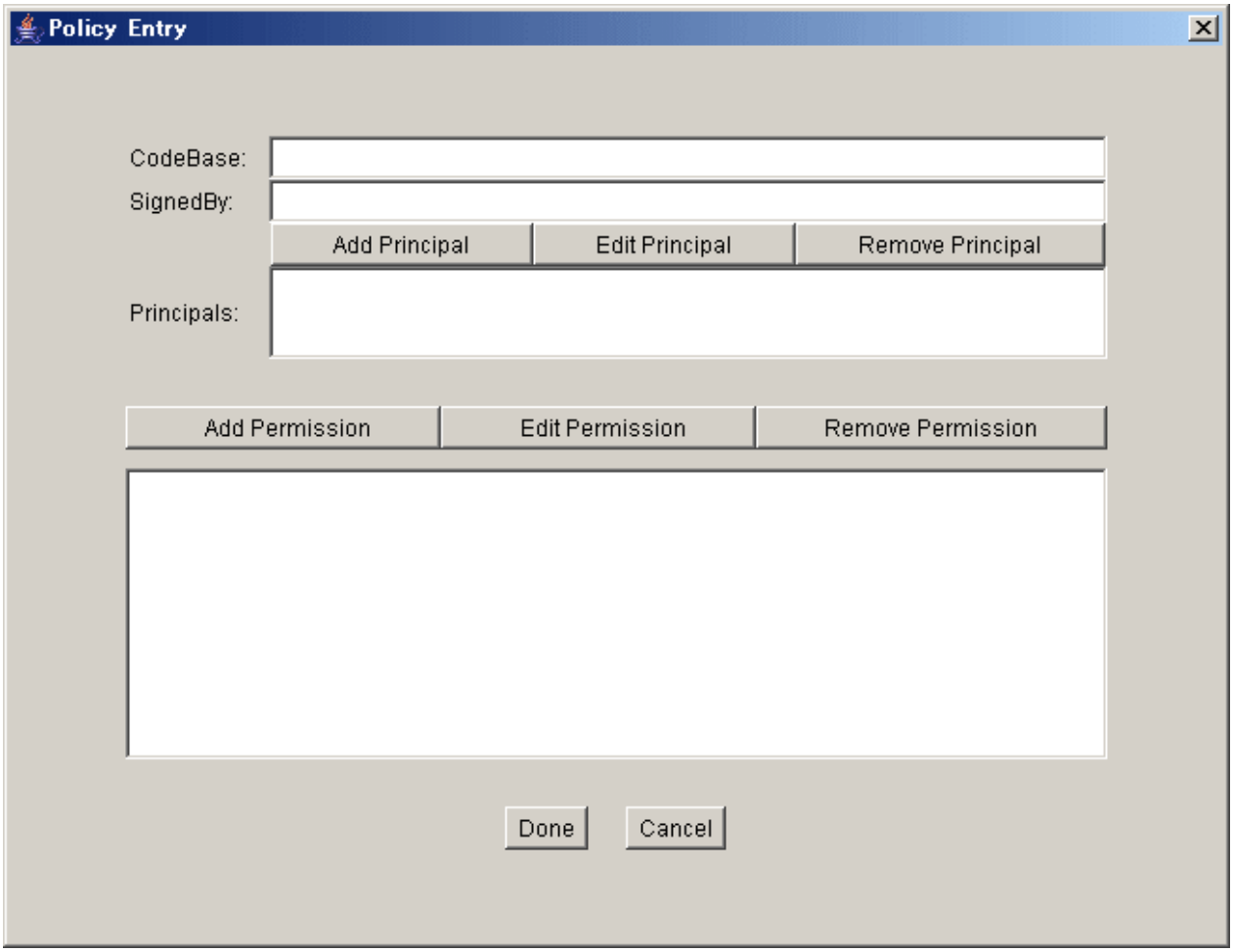
If the storage location of key store is specified, set contents are reflected in the Keystore field.

Figure 6.12 Creating a Policy Entry



Click the Add Policy Entry button on the [Policy Tool] screen. The following [Policy Entry] screen is displayed.

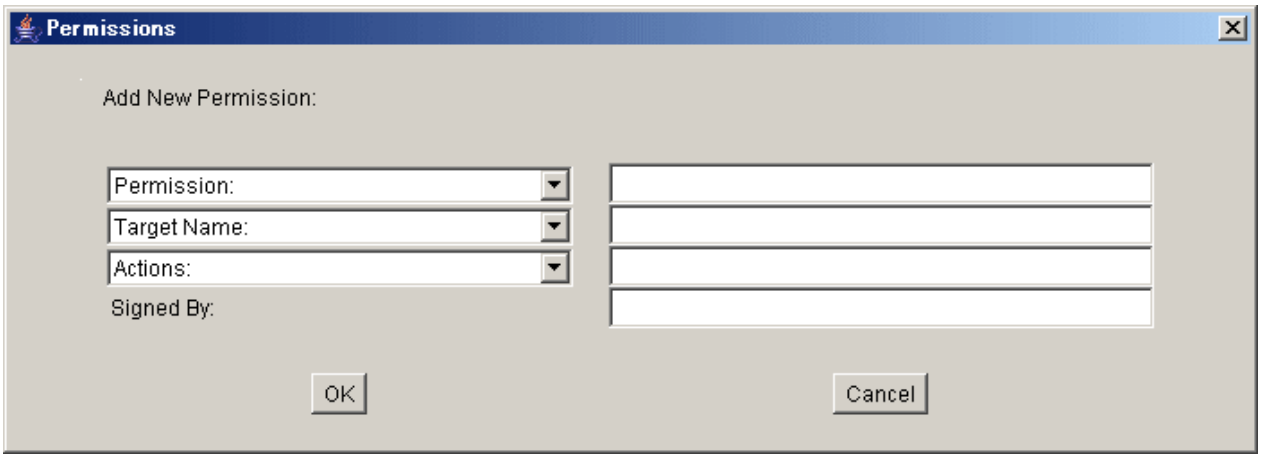
Figure 6.13 Policy Entry Screen



(4) Setting Authorization

When the Add Permission button is clicked on the [Policy Entry] screen, the following [Permissions] screen is displayed. Set the appropriate permissions on this screen.

Figure 6.14 Add New Permission Dialog



To set a permission on the [Permissions] screen, set values in the Permission, Target Name and Actions fields, and then click **OK**. As a result, control is returned to the [Policy Entry] screen. (To set another set of values, click the Add Permission button again. Repeat this procedure as required.)

The permissions to be set differ depending on whether you are using a "pre-installed Java library" or the "Portable-ORB." The permissions to be set are shown below.

For Pre-installed Java Library

Permission necessary for usual operation

Use these permissions to ensure security during usual operation.

Permission type	Setting Permission		
	Permission	Target Name	Actions
Run time permission	RuntimePermission (java.lang.RuntimePermission)	loadLibrary.DLL name (*1)	Setting not required
Property permission	PropertyPermission (java.util.PropertyPermission)	com.fujitsu.*	read

*1 The following dynamic link library (DLL) names are specified by the function to install when JDK/JRE is used. The extension need not be specified.

Function to Install	Dynamic Link Library
CORBA Service Client (Client Function)	ODjava4
CORBA Service (Server Function)	ODjavas4

Permission necessary to collect internal logs of CORBA service (*2)

Permission type	Setting Permission		
	Permission	Target Name	Actions
Property permission	PropertyPermission (java.util.PropertyPermission)	user.dir	read
		java.class.path	read
File permission	FilePermission (java.io.FilePermission)	${user.dir}*	read, write
		%OD_HOME%\etc\config (*3)	read

*2 Refer to "config" in the "CORBA Service Environment Definition" appendix of the Tuning Guide for details of an internal log of the CORBA Service. Delete the added permission after collecting logs.

*3 %OD_HOME% specifies the installation directory of the CORBA Service or the CORBA Service client. Default is C:\Interstage \ODWIN.

For Portable-ORB

Permission necessary for usual operation

Use this permission to ensure security during usual operation.

Permission type	**Setting permission**		
	Permission	**Target Name**	**Actions**
Communication permission	SocketPermission	Communicated server name (*1)	connect

Permission type	Setting permission		
	Permission	Target Name	Actions
	(java.net.SocketPermission)		

*1 The communicated server name is set when communicating with the server machines other than Web Server which downloads the Java applet. It is not necessary to set it if the communicated server machine is the Web Server which downloads the Java applet. The following host names are specified for the communicated server name.

- The host name set by "Host" of porbeditenv command
- The host name set for object reference of communicated server application
- (It is "Object host name" displayed by -l option of the odlistns command.)

  The host name specified for URL schema

  (Refer to "8.5 URL Schemas" in the "Chapter 8 Naming Service Programming" chapter and "12.4 corbaloc URL Schemas" in the "Chapter 12 Obtaining Naming Service Initial References" chapter for details.)

For example, if the communicated server is serverA.interstage.co.jp and serverB.interstage.co.jp, two communication permissions of serverA.interstage.co.jp and serverB.interstage.co.jp are set to the communicated server name. Or, after verification, it can be set with a wild-card (*) as *.interstage.co.jp. However, do not only specify a wild-card (*) as this may compromise security.

Permission necessary for using EJB application (*2)

Permission type	Setting Permission		
	Permission	Target Name	Actions
Property permission	PropertyPermission (java.util.PropertyPermission)	com.fujitsu.*	read
		java.class.path	read
Runtime permission	RuntimePermission (java.lang.RuntimePermission)	getClassLoader	(Specification not required)

*2 Refer to "EJB Edition" in the J2EE User's Guide for details of the EJB application.

Permission necessary to collect logs of Portable-ORB (*3)

Permission type	Setting Permission		
	Permission	Target Name	Actions
File permission	FilePermission (java.io.FilePermission)	Log collection file (*4)	read, write, delete
		Log collection directory (*5)	read

*3 Refer to "porbeditenv" in the Reference Manual (Command Edition) for details of the log information of the Portable-ORB. It is necessary to create the directory specified by the "Logging Directory" option of the porbeditenv command beforehand when logs are collected. Also, do not store the user resources etc. other than the log collection file in "Logging Directory". Delete the added permission after collecting logs.

*4 "\ *" is added to the directory name specified by "Logging Directory" of the porbeditenv command. When "Logging Directory" is "C:\log\porb", it is "C:\log\porb\ *".

*5 The directory name specified by "Logging Directory" of the porbeditenv command is specified.

Permission necessary for using SSL

Permission type	Setting Permission		
	Permission	Target Name	Actions
File permission	FilePermission (java.io.FilePermission)	keystore directory	read

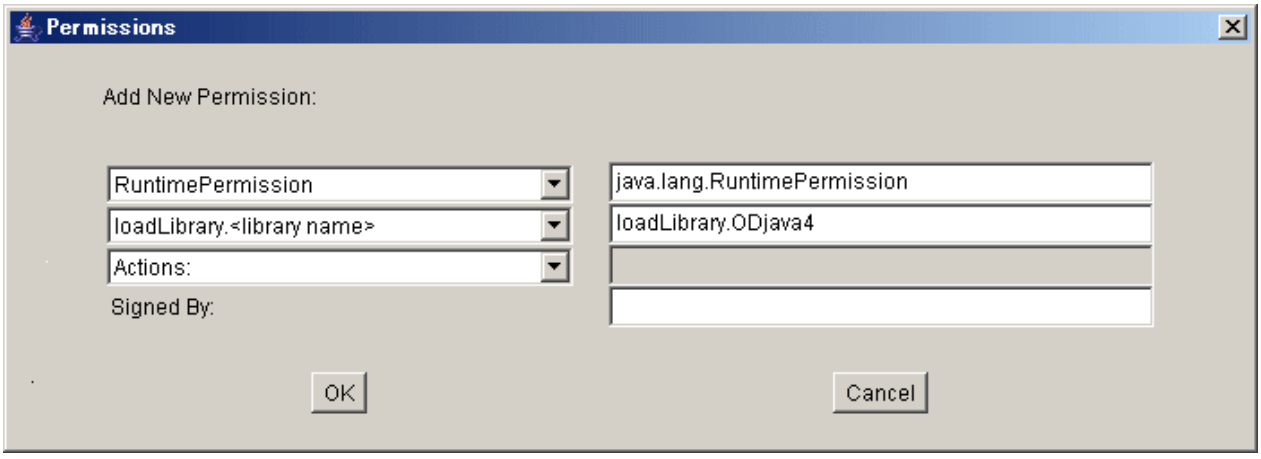
Permission necessary for using the user identifier acquisition API by server application (*6)

Permission type	Setting Permission		
	Permission	Target Name	Actions
Property permission	PropertyPermission (java.util.PropertyPermission)	user.name	read

*6 Refer to "TD_get_user_information", "TD::get_user_information", and "TDGETUSERINFORMATION" in the Reference Manual (API Edition) for details of the user identifier acquisition API

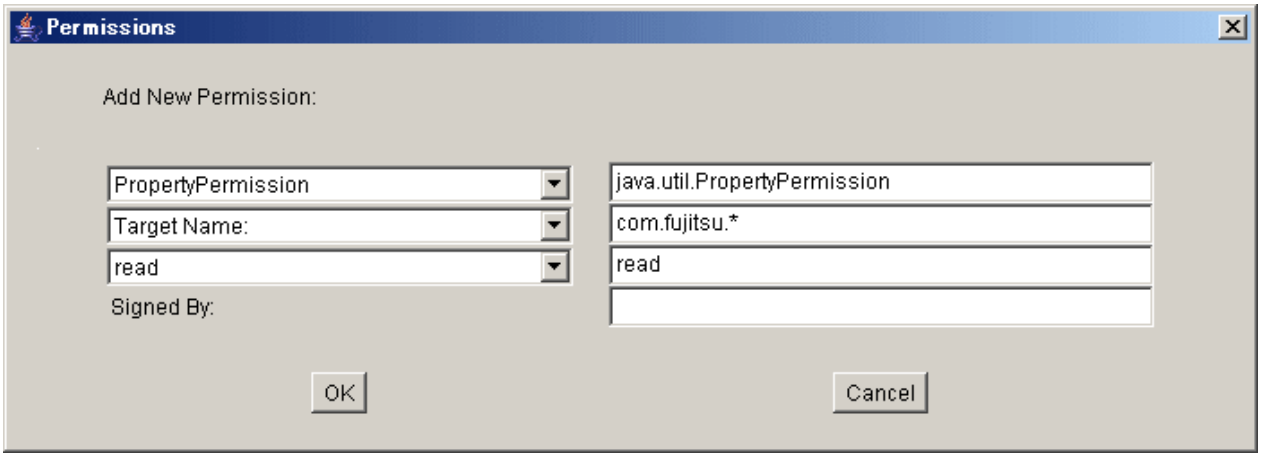
An example of the setting screen for run-time authorization is shown below.

Figure 6.15 Add Runtime Permission Settings



An example of the setting screen for property authorization is shown below.

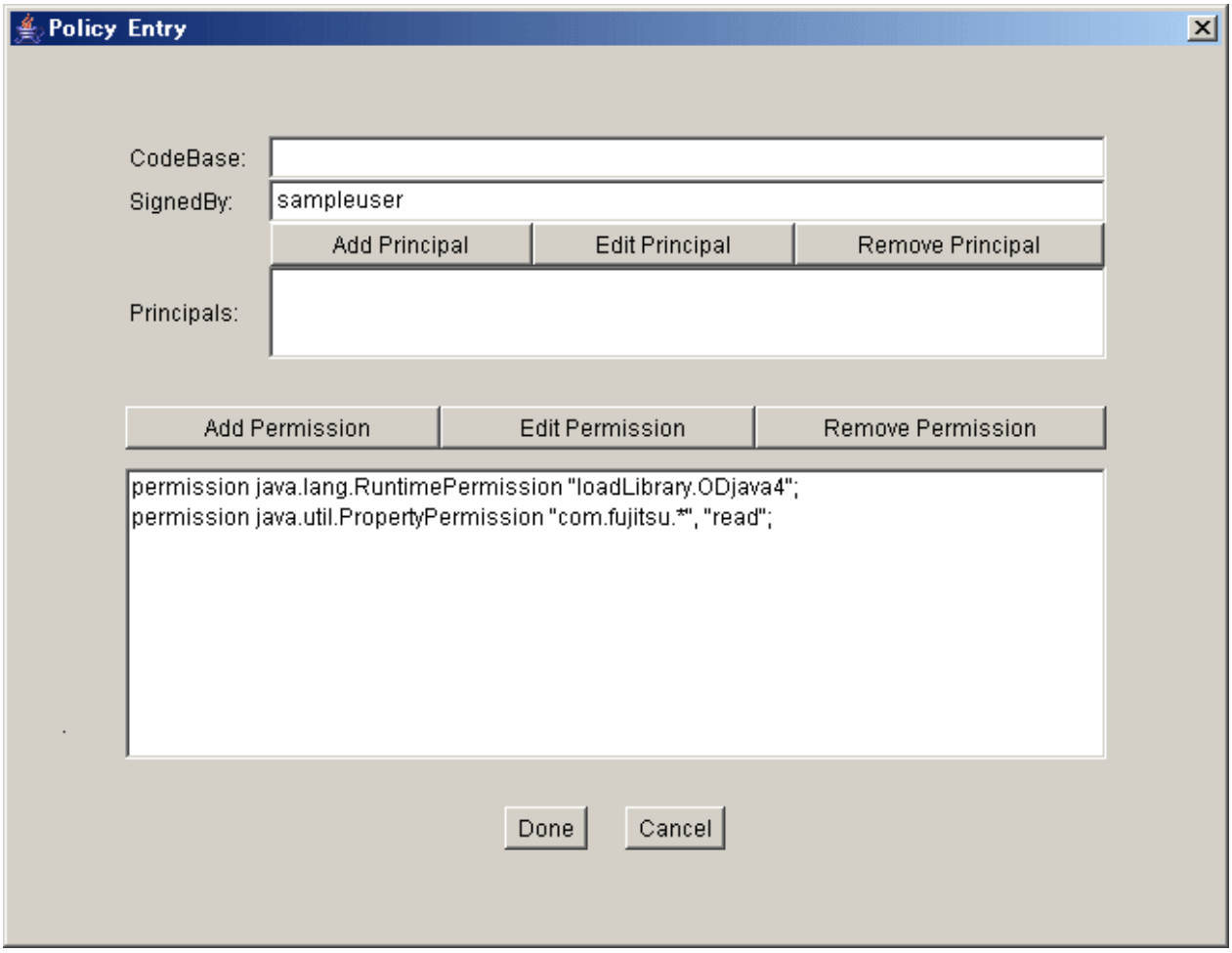
Figure 6.16 Property Authorization Permission



(5) Specifying a Certificate for the Signed Java Class

Specify the name (alias) imported for key store in the SignedBy field so that the certificate for which authorization is intended can be specified. Once the input is complete, click **Done**.

Figure 6.17 Specifying the Alias for Key Store



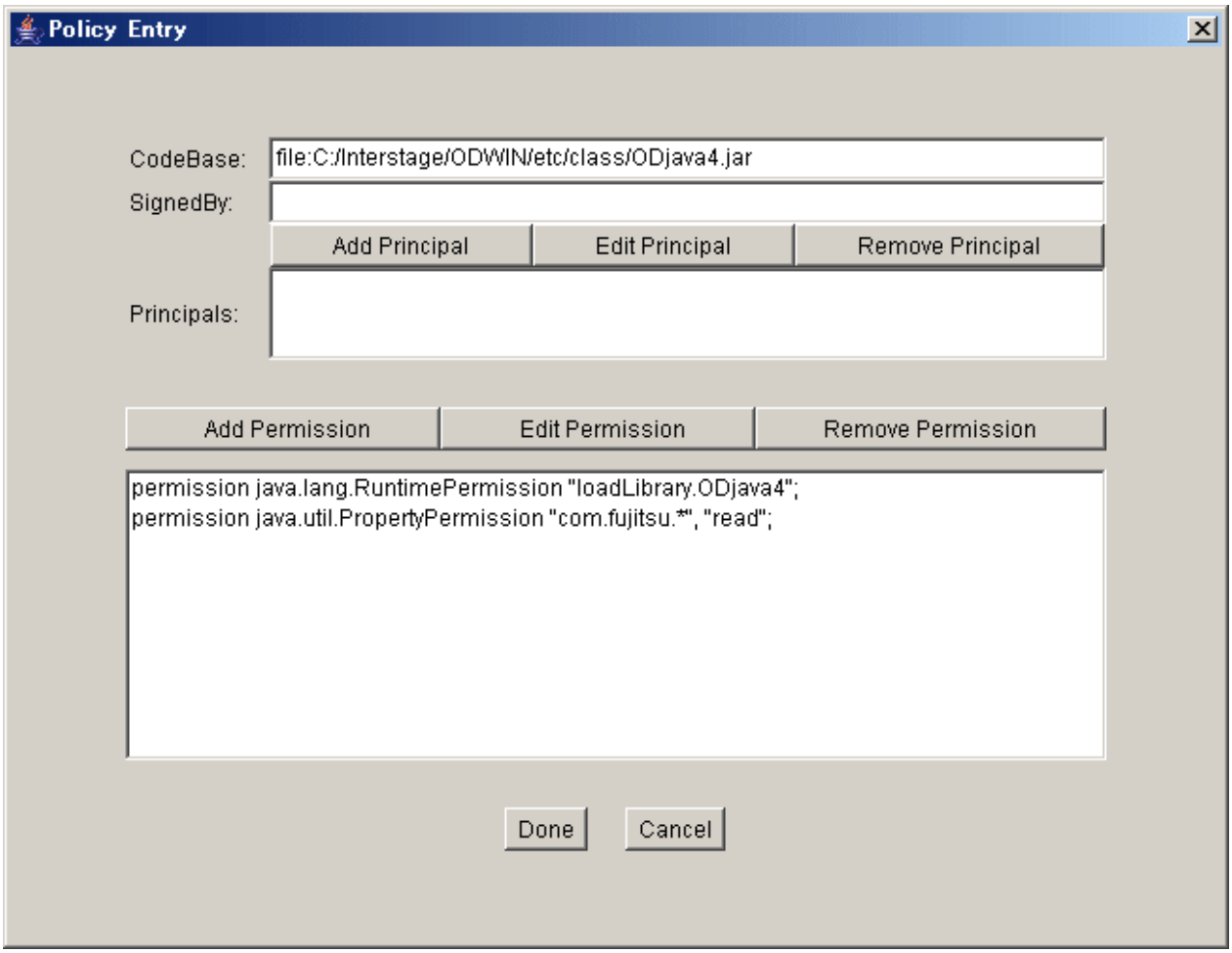
(6) Setting Authorization for Pre-installed Java Libraries

Set authorization for pre-installed Java libraries.

Click **Add Policy Entry** on the [Policy Tool] screen, and set the run-time authorization/property authorization/file authorization (refer to "(3) Creating an Entry" and "(4) Setting Authorization" above).

After each authorization setting is completed, specify pre-installed Java libraries in the CodeBase field on the [Policy Entry] screen. Once the input is complete, click **Done**.

Figure 6.18 Policy Entry Screen



(7) Saving

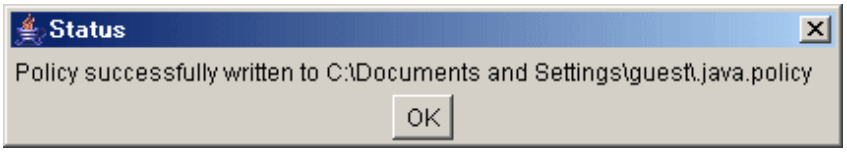
Save the set authorization as a security policy file.

When creating it for the first time, select File | Save As on the pull-down menu to specify the policy file name and storage location. When creating it for the second and subsequent times, specify the name and storage location of the policy file in the "Policy file" field on the [Policy Tool] screen.

The default directory varies in each OS. The default directory in each OS is the same as Table 6.5 Default Directory of Key Store File and Policy File.

Select File | Save As on the pull-down menu. When saving is completed, a dialog box as shown below is displayed from the policy file:

Figure 6.19 Policy Status Dialog

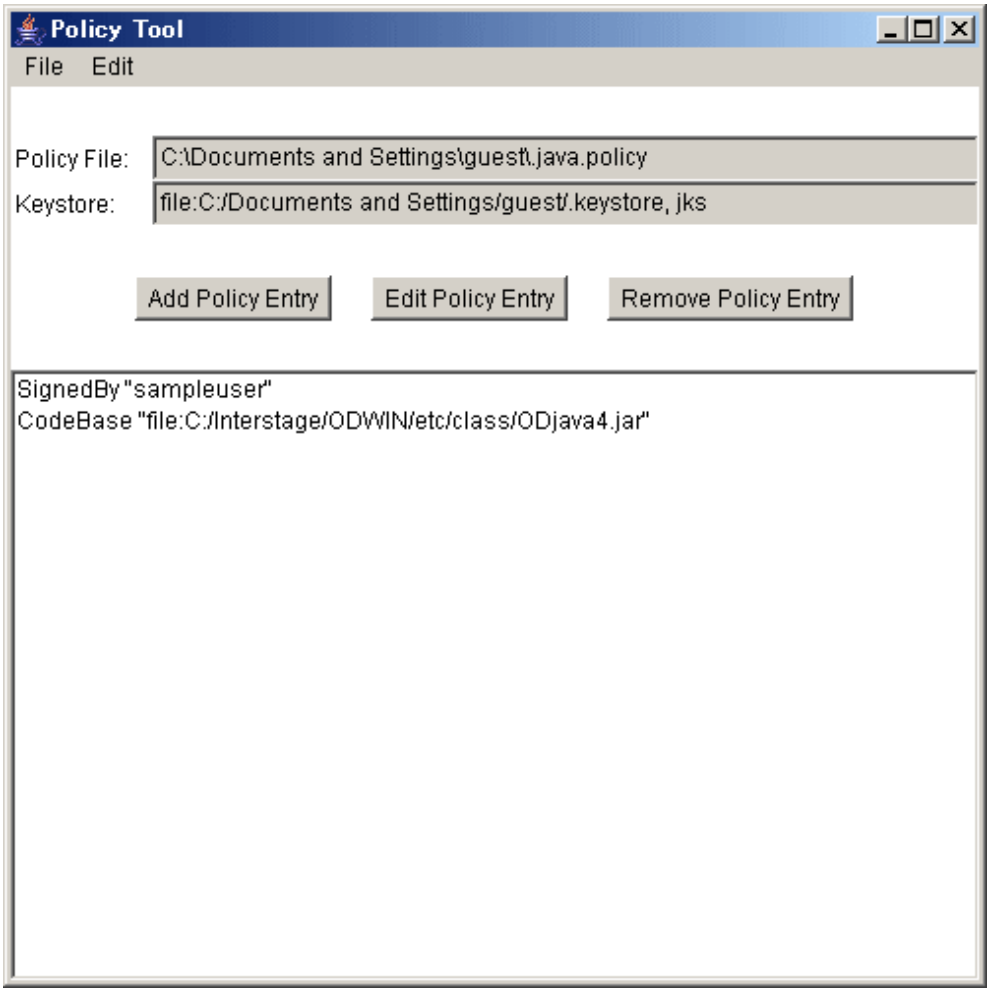


(8) End Policy Tool

Once saving is completed, you can check that the Policy File field contains the saved policy file.

Select File | Exit on the [Policy Tool] screen.

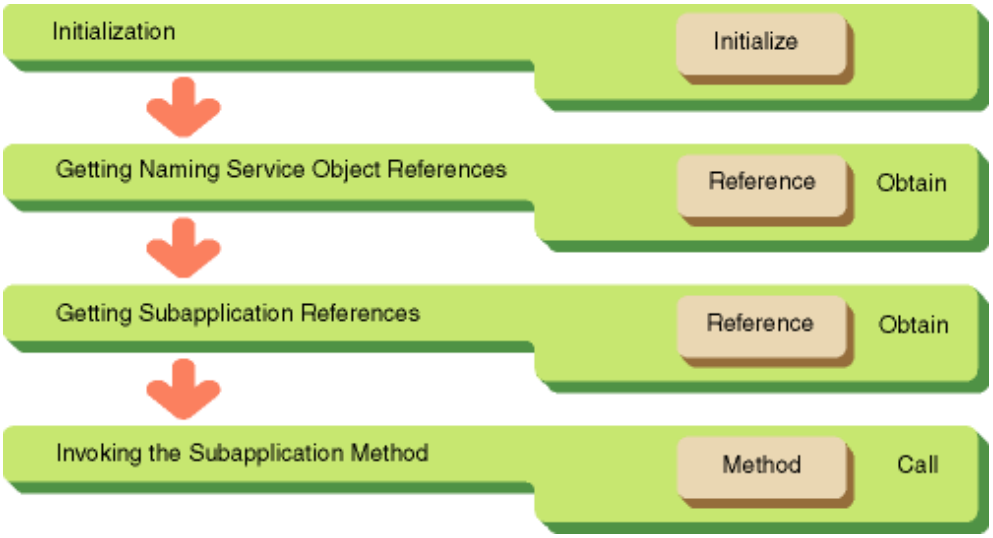
Figure 6.20 Policy Tool Screen



6.8 Client Application Programming (Static Invocation Interface)

The process flow for a client application when using a Static-Invocation Interface is shown below.

Figure 6.21 Client Application (STI) Process Flow



6.8.1 Initialization

Perform the initialization process by invoking the CORBA initialization method, org.omg.CORBA.ORB.init(). As a result of this method, ORB objects are returned. These objects will be specified later when we use ORB objects to be invoked.

Example

Application

```
public class Sample { //class declaration
    public static void main( String args[] ) {
        org.omg.CORBA.ORB   Orb;     // ORB object

        try {
            // create and initialize ORB
            Orb = org.omg.CORBA.ORB.init( args, null );
              ...
        }
        catch( java.lang.Exception e ) {
              ... //exception handling
        }
    }
}
```

Applet

```
org.omg.CORBA.ORB   Orb;    // ORB objects
public void init() {
    try {
        // create and initialize ORB
        Orb = org.omg.CORBA.ORB.init( this, null );
          ...
    }

    catch( java.lang.Exception e ) {
      ...//exception handling
    }
  ...//Screen display handling
}
```

```
public void start() {
    ...
}
```

See

Refer to "6.9 Exception Handling for Client Applications" regarding the try-catch exception handling mechanism.

6.8.2 Getting NamingService Object References

In order to search for an object to run from a Naming Service, you need the Naming Service's object references.

Retrieve the Naming Service's object references with org.omg.CORBA.ORB.resolve_initial_references().

```
// NamingService object references
org.omg.CosNaming.NamingContextExt  Cos;

// get NamingService object references
try {
    org.omg.CORBA.Object _tmpObj =
                        Orb.resolve_initial_references( "NameService" );
    Cos  = org.omg.CosNaming.NamingContextExtHelper.narrow( _tmpObj );
      ...
}
catch( java.lang.Exception e ) {
      ...//exception handling
}
```

6.8.3 Getting Server Application Object References

Retrieve the object references for the server applications you want to execute from now on with the Naming Service method, CosNaming.NamingContext.resolve(). Specify the object name you want to search for with the same method.

```
// run NamingService resolve method,
// get server application object references
String NCid   = new String( "ODdemo::calculator" );   // object name
String NCkind = new String( "" );                     // object type
try {
    org.omg.CosNaming.NameComponent nc =  new NameComponent( NCid, NCkind );
    org.omg.CosNaming.NameComponent NCo[] = { nc };
    //run NamingService resolve method
    org.omg.CORBA.Object  Obj = Cos.resolve( NCo );
      ...
}
catch( java.lang.Exception e ) {
      ... //exception handling
}
```

6.8.4 Invoking Methods

Invoke the subprogram method. A method name is designated according to the format in which a "." (dot) is attached to IDL-specified module, interface, or object names. In this example, we have in dots Oddemo.calculator.calculate. Further, designate the try-catch block when invoking methods in order to set up the server application's object references requested by the Naming Service and the exception information when exceptions are generated by server applications.

```
ODdemo.calculator target;    //application object references
try {
```

```
    //ODdemo.calculator class convert
    target = ODdemo.calculatorHelper.narrow(obj);
    ODdemo.calculatorPackage.result res = new ODdemo.calculatorPackage.result();
    res = target.calculate( a, b );
          ...
}
catch( NumberFormatException e ){
         ... //exception handling
}
catch( ODdemo.calculatorPackage.ZEROPARAM e ){
         ... //exception handling
}
catch( org.omg.CORBA.SystemException err ){
         ... //exception handling
}
```

6.9 Exception Handling for Client Applications

You can determine whether the server application processing client requests terminated normally or abnormally. Also, you can determine, in the event of an abnormal termination, whether it was the system or server application that ended abnormally. The former are system exceptions, while the latter are user exceptions. System exceptions are described in the following table.

Among the system exceptions, org.omg.CORBA.SystemException is defined as a new class, and successor classes are org.omg.CORBA.UNKNOWN and org.omg.CORBA.BAD_PARAM which contain detailed information. For user exceptions, org.omg.CORBA.UserException is defined as a new class.

Refer to the Reference Manual (API Edition) for the meaning of each exception.

Table 6.6 Exception Classes

Exception Information	Exception Class
BAD_CONTEXT	org.omg.CORBA.BAD_CONTEXT
BAD_INV_ORDER	org.omg.CORBA.BAD_INV_ORDER
BAD_OPERATION	org.omg.CORBA.BAD_OPERATION
BAD_PARAM	org.omg.CORBA.BAD_PARAM
BAD_QOS	org.omg.CORBA.BAD_QOS
BAD_TYPECODE	org.omg.CORBA.BAD_TYPECODE
CODESET_INCOMPATIBLE	org.omg.CORBA.CODESET_INCOMPATIBLE
COMM_FAILURE	org.omg.CORBA.COMM_FAILURE
DATA_CONVERSION	org.omg.CORBA.DATA_CONVERSION
FREE_MEM	org.omg.CORBA.FREE_MEM
IMP_LIMIT	org.omg.CORBA.IMP_LIMIT
INITIALIZE	org.omg.CORBA.INITIALIZE
INTERNAL	org.omg.CORBA.INTERNAL
INTF_REPOS	org.omg.CORBA.INTF_REPOS
INV_FLAG	org.omg.CORBA.INV_FLAG
INV_IDENT	org.omg.CORBA.INV_IDENT
INV_OBJREF	org.omg.CORBA.INV_OBJREF
INV_POLICY	org.omg.CORBA.INV_POLICY
MARSHAL	org.omg.CORBA.MARSHAL
NO_IMPLEMENT	org.omg.CORBA.NO_IMPLEMENT

Exception Information	Exception Class
NO_MEMORY	org.omg.CORBA.NO_MEMORY
NO_PERMISSION	org.omg.CORBA.NO_PERMISSION
NO_RESOURCES	org.omg.CORBA.NO_RESOURCES
NO_RESPONSE	org.omg.CORBA.NO_RESPONSE
OBJ_ADAPTER	org.omg.CORBA.OBJ_ADAPTER
PERSIST_STORE	org.omg.CORBA.PERSIST_STORE
REBIND	org.omg.CORBA.REBIND
TIMEOUT	org.omg.CORBA.TIMEOUT
TRANSIENT	org.omg.CORBA.TRANSIENT
UNKNOWN	org.omg.CORBA.UNKNOWN
INVALID_TRANSACTION	org.omg.CORBA.INVALID_TRANSACTION
TRANSACTION_MODE	org.omg.CORBA.TRANSACTION_MODE
TRANSACTION_REQUIRED	org.omg.CORBA.TRANSACTION_REQUIRED
TRANSACTION_ROLLEDBACK	org.omg.CORBA.TRANSACTION_ROLLEDBACK
TRANSACTION_UNAVAILABLE	org.omg.CORBA.TRANSACTION_UNAVAILABLE

Define the catch block that corresponds to the exception class in order to handle exceptions with a try block. For common handling of system exceptions you can specify org.omg.CORBA.SystemException class as a catch phrase. For common handling of system, user, and various exceptions, you can specify class java.lang.Exception as a catch phrase.

org.omg.CORBA.SystemException.minor can be used to obtain minor codes when a system exception occurs. Refer to "CORBA Service Minor Codes" in the Messages manual for details of minor code values.

```
try {
    fret =target.divide( inArg1,inArg2 );
    //run divide method
}
catch( org.omg.CORBA.UNKNOWN e ){
    // org.omg.CORBA.UNKNOWN handling
    System.out.println("ERROR : " + e.getClass().getName()
      + ": Minor = 0x" + java.lang.Integer.toHexString(e.minor)); // Obtain minor code
}
catch( org.omg.CORBA.BAD_PARAM e ){
    // org.omg.CORBA.BAD_PARAM handling
    System.out.println("ERROR : " + e.getClass().getName()
      + ": Minor = 0x" + java.lang.Integer.toHexString(e.minor)); // Obtain minor code
}
catch( org.omg.CORBA.SystemException e ){
    System.out.println("ERROR : " + e.getClass().getName()
      + ": Minor = 0x" + java.lang.Integer.toHexString(e.minor)); // Obtain minor code
    //except above system exceptions
}
catch( demo.calc.ZERODIVIDE e ){
    //handle user exceptions
}
```

or

```
try {
    fret =target.divide( inArg1, inArg2 );   //run divide method
}
```

```
catch( java.lang.Exception e ){
    if ( e instanceof demo.calc.ZERODIVIDE ){
        // handle user exceptions
    }
    if ( e instanceof org.omg.CORBA.SystemException ){
        // handle system exceptions
        System.out.println("ERROR : " + e.getClass().getName()
          + ": Minor = 0x"                    // Obtain minor code
          + java.lang.Integer.toHexString(((org.omg.CORBA.SystemException)e).minor));
              :
      }
      //handle all exceptions, etc.
}
```

The example given below is of a client application program in which user exceptions have been defined with IDL language.

IDL Mapping

IDL

```
module ODsample {
    interface  exptest{
        exception testException { string reason; };
        void  op1() raises( testException );
    };
};
```

Written in Java language, this comes out as follows.

Java Language

<Interface>

```
package ODsample;
public interface exptest  extends org.omg.CORBA.Object  {
    public void op1()  throws ODsample.exptestPackage.testException;
}
```

<User Exception Class>

```
package ODsample.exptestPackage;
public class testException extends org.omg.CORBA.UserException {
    public java.lang.String reason;
    public testException()  {
        reason = null;
    }
    public testException( java.lang.String _reason )  {
        reason = _reason;
    }
}
```

Client Application Handling

```
import org.omg.CORBA.*;
import ODsample.*;

public class expClient {
    public static void main( String args[] ) {
```

```
        // ORB previous handling
        // get object references

        try {
            // invoke server application method
            target.op1( );
        }

        catch ( ODsample.exptestPackage.testException e ) {
            System.out.println( "UserException : " );
            System.out.println( "   " + e  + " -> " + e.reason );
        }

        catch (java.lang.Exception e) {
            if ( e instanceof demo.calc.ZERODIVIDE ){
                // handle user exceptions
            }
            if ( e instanceof org.omg.CORBA.SystemException )
                // handle system exceptions
                System.out.println("ERROR : " + e.getClass().getName()
                  + ": Minor = 0x"        // Obtain minor code
                  + java.lang.Integer.toHexString(((org.omg.CORBA.SystemException)e).minor));
                    :
        }
    }
}
```

6.10 Examples of Simple Server Application Creation

In this section we will explain the flow of simple server applications from development through execution using specific examples based on the explanations we have provided so far.

As one example, we will explain the process up to the point of actually launching on a system, using the sample application described in 6.18.1 Default Servant (Default Instances Method).

Point

Examples in this section are based on the following assumptions. The client application is assumed to run on the same system as the server application.

Windows32/64

The current (work) directory is:

C:\home\example

The command input window is:

The DOS window

Solaris32/64 Linux32/64

The current (work) directory is:

/home/example

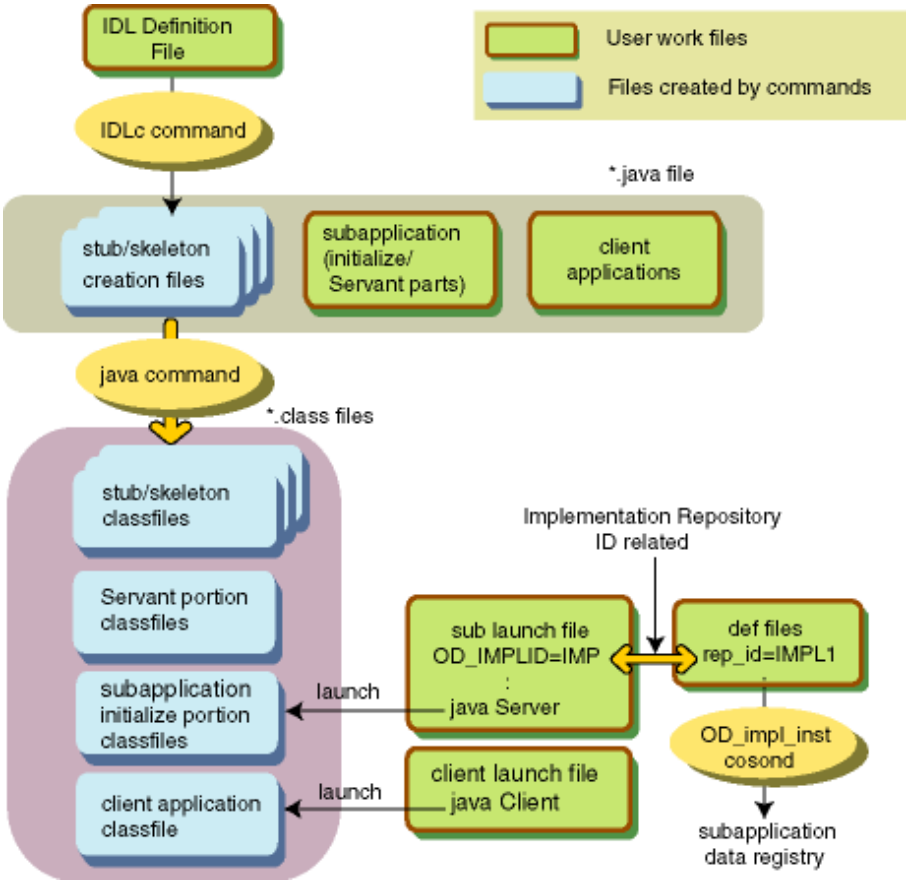
The command input window is:

The terminal window

6.10.1 Work Flow

The total workflow and the relationships among the files are shown below.

Figure 6.22 Application Work Flow



The following lists the files that have to be created by users:

1. IDL files [TEST.idl]
2. Server application (initialization handler, Servant portion) source file [Server.java]
3. Client application source file [Client.java]
4. def files [def]
5. Launch files (batch/shell-script) [exec-SV.bat/exec-SV, exec-CL.bat/exec-CL]

The file names listed in square brackets are those used in examples throughout this section.

6.10.2 Creating and Compiling IDL Files

Create IDL files with content similar to the example shown below.

IDL Definition File (TEST.idl)

```
module ODsample{
    interface  intf{
        long  add( in  long  a,  in  long  b );
```

```
    };
};
```

Store this file in the /home/example or C:\home\example under the file name TEST.idl. Next, compile the created file TEST.idl by using the IDLc command. Based on this, the stub/skeleton and other files that the Java application uses are created. Enter the following command:

```
IDLc -java TEST.id
```

From this, the ODsample directory in the /home/example or C:\home\example is created, and the stub and skeleton *.java files are created in its sub directory.

6.10.3 Creating Server Application Source Files

Server applications may be divided into "Servants," which implement the interfaces defined by IDL, and "Initialization Handlers," which perform the creation and registration of ORB initialization or Servant objects. Here, for simplicity's sake, we will presume that "Servant" and "Initialization Handler" are items that are described in the identical file (being the identical file is not necessary).

Create the source file described in the server application section noted in 6.18.1 Default Servant (Default Instances Method), and store it under /home/example or C:\home\example under the file name Server.java

6.10.4 Creating Client Application Source Files

Create the source file described in the server application segment noted in 6.18.1 Default Servant (Default Instances Method), and store it under /home/example or C:\home\example under the file name Server.java.

6.10.5 Compiling Java Files

Compile a Java files under /home/example or C:\home\example (the *.java files the IDLc command created Server.java, Client.java) with the javac command. Make /home/example or C:\home\example the current directory and input the following commands. Enter the following commands:

Example

Windows32/64 (OD_HOME: C:\Interstage\ODWIN)

```
set CLASSPATH=.;%OD_HOME%\etc\class\ODjava4.jar;%CLASSPATH%;
java -d .  *.java
```

Solaris32/64 (OD_HOME: /opt/FSUNod) Linux32/64 (OD_HOME: /opt/FJSVod)

```
CLASSPATH=./:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH:
export CLASSPATH
javac  -d  .  *.java
```

6.10.6 Creating and Registering Def Files

Create a file that described the information relating to the server application. Create a file with contents like the following and store it in /home/example or C:\home\example with the filename of def.

def File Example [def]

```
rep_id        = IMPL1
type          = persistent
mode          = SYNC_END
proc_conc_max = 1
thr_conc_init = 1
ior           = 1.1
```

Next, using the OD_impl_inst command, register the information described in the def file. Do this by inputting commands in the following manner:

```
OD_impl_inst -ax def
```

6.10.7 Creating Launch Files

Create a share script for launching applications. Create a file with contents like those below and store it under /home/example or C:\home \example as filename exec-CL.bat/exec-CL exec-SV.bat/exec-SV .

Note

Solaris32/64

Set exec-CL and exec-SV in executable files.

Server Launch File Example

exec-SV.bat Windows32/64

```
set OD_IMPLID=IMPL1
java Server
```

exec-SV Solaris32/64 Linux32/64

```
OD_IMPLID=IMPL1
export OD_IMPLID
java Server
```

Client Launch File Example

exec-CL.bat Windows32/64

```
java Client
```

exec-CL Solaris32/64 Linux32/64

```
java Client
```

6.10.8 Running Applications

Launch client or server terminals, and do the settings for the following environment variables at their respective terminals.

Setting Environment Variable

Example

Windows32/64 (OD_HOME: C:\Interstage\ODWIN)

```
set CLASSPATH=.\;%OD_HOME%\etc\class\ODjava4.jar;%CLASSPATH%;
```

Solaris32/64(OD_HOME: /opt/FSUNod) Linux32/64(OD_HOME: /opt/FJSVod)

$OD_HOME/lib must be set in both the CLASSPATH variable and the LD_LIBRARY_PATH environment variable.

```
CLASSPATH=./:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH:
export CLASSPATH
LD_LIBRARY_PATH=$OD_HOME/lib
export LD_LIBRARY_PATH
```

Server Application Launch Execution Example

```
exec-SV
```

Client Application Launch Execution Example

```
exec-CL
```

Note

- Set 16 for the thread multiplicity at execution of a server application.
- Set 100 for the total number of client connections at execution of a server application.

6.10.9 Application Execution Results

If the application runs successfully, execution results like those below will be obtained.

Execution Result Example (Client Indication)

```
  in1 => 888 input value
  in2 => 111 input value
  888 + 111 = 999  result indication
```

Press Ctrl+C to terminate an application.

6.10.10 Deleting Server Application Data

After terminating a server application, execute the following command.

Delete Server Application Data Execution Example

```
OD_impl_inst  -d  -r  IMPL1
```

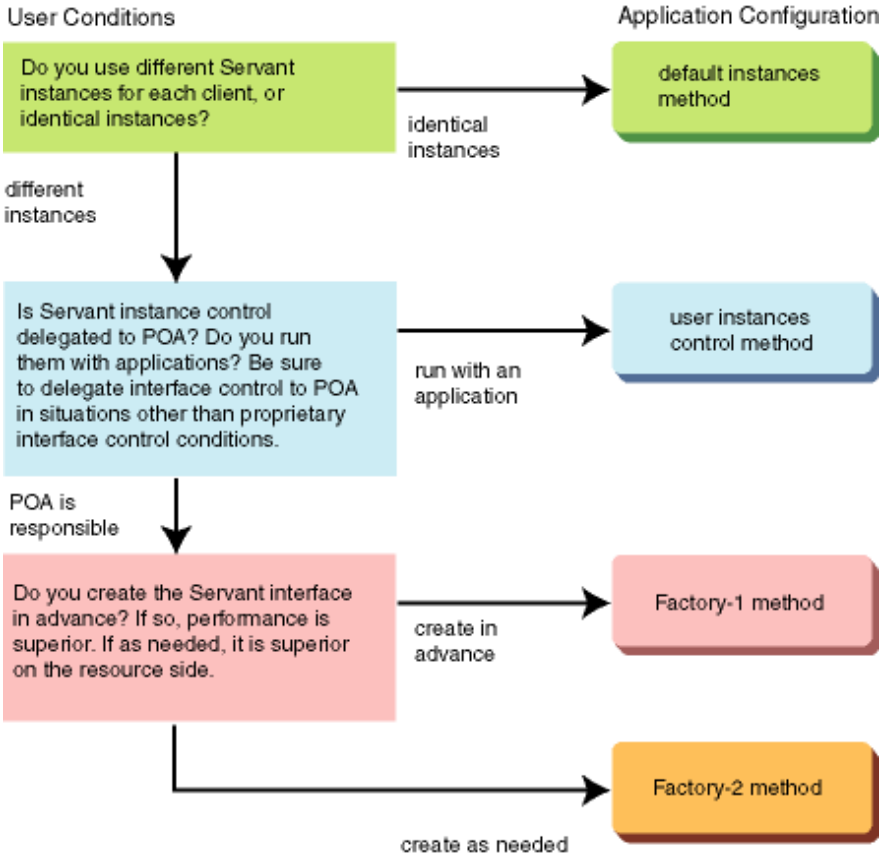
6.11 Instance Control and Application Configuration

This section provides information on instance control and application configuration.

6.11.1 Types of Application Configurations

A server application is capable of taking on the four configurations shown in the figure below, based on the Servant's (interface implementer) interface control procedures. Select your application configuration depending on the conditions.

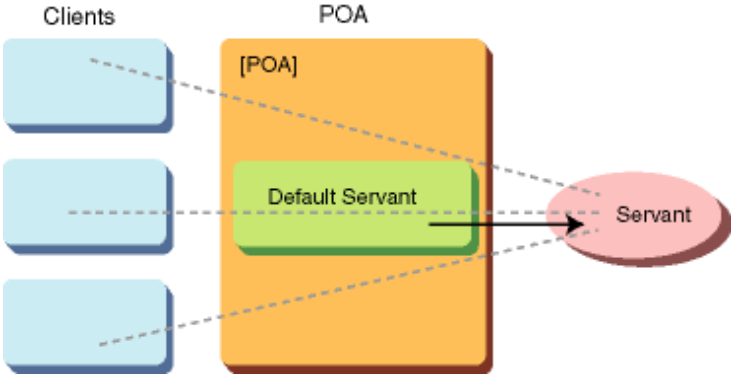
Figure 6.23 Types of Application Configurations



Default Instances Method

A method in which each client uses the identical Servant objects. In other words, it is a method that does not perform instance control on each client. Each client uses the identical Servant instances in the manner shown below.

Figure 6.24 Instance Control by the Default Instances Method



Factory-1 Method

A method for using different Servant objects on each client. A method for running instance control on each client by POA objects. Creation of object references and instances is done in Factory.

Factory-2 Method

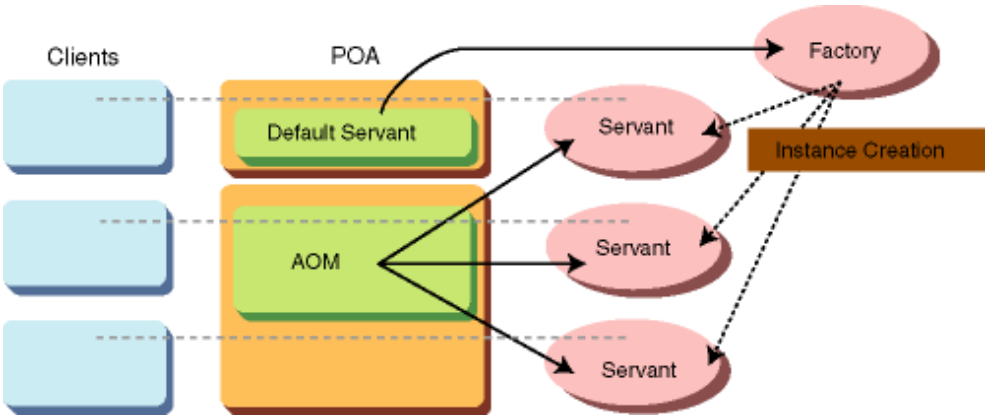
A method for using different Servant objects on each client. A method for running instance control on each client by POA objects. Creation of object references is done in Factory, and instances are created in ServantManager objects at the time of request reception.

Factory is an object for creating object references and instances for each request, and is required for running instance control.

AOM refers to Active Object Map, and has to do with instance control tables. Refer to "6.12 Server Application Programming POA Overview" for details on AOM, ServantManager, etc.

Each client in the Factory-1 and Factory-2 methods uses different Servant instances for each client in the manner shown in the figure below.

Figure 6.25 Factory-1/Factory-2 Instance Control Methods



Factory-1 Method & Factory-2 Method

Since instances in the Factory-1 method are created in Factory beforehand, they are somewhat superior performance-wise to the Factory-2 method in which instances are created at the moment a request is received.

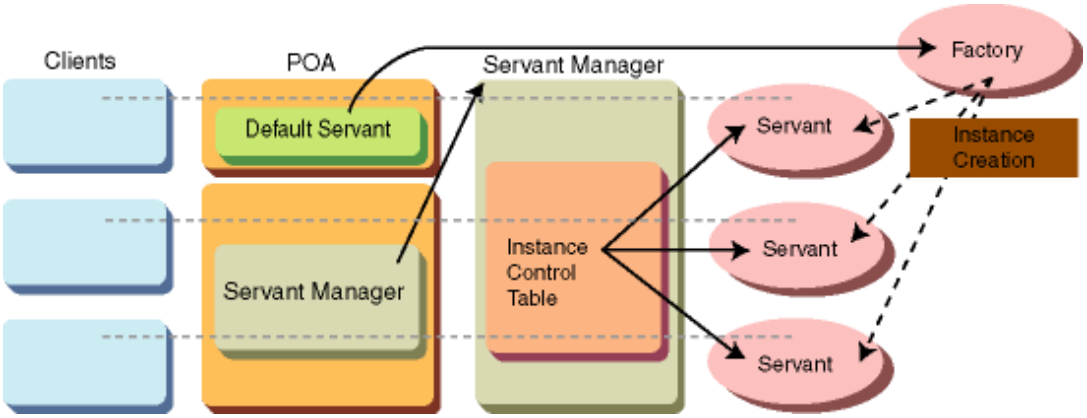
Because Factory-2 method creates instances at the time requests are received, they are somewhat superior in terms of resources compared to instances created beforehand in Factory under the Factory-1 method.

User Instance Control Method

A method that delegates instance control to a ServantManager object created by the user.

In the manner shown below, Al clients use different Servant instances on each client. These instances are run by a instance control table in the ServantManager objects created by the user.

Figure 6.26 Instance Control by the User Instance Control Method



Choose an application configuration based on the specifications in the Servant Retention and RequestProcessing policies of the POA objects that come under Servant objects (in the table below). Refer to "6.12.3 POA Objects" for details.

Table 6.7 Application Configurations & Combining POA Policies

Application Configurations	ServantRetention Policy	RequestProcessing Policy
Default Instance Method	NON_RETAIN	USE_DEFAULT_SERVANT
Factory-1 Method	RETAIN	optional
Factory-2 Method	RETAIN	USE_SERVANT_MANAGER
User Instance Control Method	NON_RETAIN	USE_SERVANT_MANAGER

About Factory

What we call the Factory is actually an interface whose function is to create instances for Servant objects and return those object references to the client.

Based on this, it becomes possible to have the server side wait for Servant object instances that its clients can have for dedicated use.

The Factory concept is displayed based on the IDL definition below. When the previously noted Factory-1 or Factory-2 methods are used, the Factory interface is defined, along with other interfaces, in the IDL definitions.

IDL Factory Definition Example (a)

```
//any interface
interface intf{
    void  op();
};

//Factory interface
interface Factory{
    intf   create();
};
```

As displayed in the above IDL definitions, it is possible for the client to obtain intf objects for its use by running the Factory object's operation create(). Then, the client performs its primary processes by running the op() operation on the obtained intf objects.

Conversely, on the server side you need to describe the following sequence of processes as the implementation of the create() operation.

1. Create (new) Servant object (intf interface implementation instance).
2. Register the relationship between the new instances and the object references in the AOM of POA.

3. Repeat the related object references

(For implementation of the create() operation with the Factory-2 method, create the object references in (3) and do a return only, and run (1) and (2) when needed in the ServantManager. But the function of the create() operation seen from the client side is the same.)

Factory is the concept for this kind of interface functionality. Consequently, in IDL definitions the categories for the Factory interface name (Factory in the above example) and the implementing operation name (create() in the above example) are independent. However, for the purposes of Factory, you would need to have as a minimum limit an operation with the function of "repeat object references as return value," as with the above-mentioned create().

In actual application development, the implementation of an operation for opening the created Servant object instances in Factory is recommended. With this, economical use of server resources is possible. In the IDL definition example below, destroy() is defined as an operation for opening Servant object instances.

IDL Factory Definition Example (b)

```
//any interface
interface intf{
    void  op();
};

//Factory interface
interface Factory{
    intf  create();
    void  destroy(in intf obj);
};
```

At the point when the intended process has terminated on the client side, invoke the destroy() operation using the object references obtained previously by the create() method as arguments. Meanwhile, run the process to erase Servant objects from AOM that correspond to object references released by the server side.

We will demonstrate below sample runs of the create() and destroy() operations for Factory objects that correspond to the IDL previously mentioned. Here, the intf interface implementation class is taken as UserServant class.

Refer to "6.12 Server Application Programming POA Overview " for details on the process.

Operation Implementation Example (Factory-1 Method)

```
public intf create() {
    intf       ior;     // object references for Servant object
    try {
        //  create Servant object
        org.omg.PortableServer.Servant svt = new UserServant();
        //  register with AOM
        poa.activate_object( svt );
        //  create object references
        org.omg.CORBA.Object Obj = poa.servant_to_reference( svt );
        // convert to intf type
        ior = intfHelper.narrow( Obj );
    }
    catch( Exception e ) {  /*exception handler*/  }
    return( ior );
}

public void destroy( intf obj ) {
    try {
        // determine Object ID from object references
        byte oid[] = poa.reference_to_id( obj );
```

```
        // deactivate Servant (erase from AOM
        poa.deactivate_object( oid );
    }
    catch( Exception e ) {  /*exception handler*/ }
}
```

Note

poa in source is POA object instance.

About Handling Factory Object Instances

You will need to register the instances of Factory objects in POA the same as the instances of conventional Servant objects.

In the examples shown in "6.18.2 Active Object Map (AOM) Sample Usage (Factory-1 Method)" and "6.18.3 Servant Activator Sample Usage (Factory-2 Method)", Factory objects are registered as Default Servant using the "POA for Factory Exclusive Use." Additionally, the Factory POA is considered a setting that does not utilize the AOM (NON_RETAIN policy). This is based on the following reasons:

- It is possible for Factory objects and the Servant objects their Factory generates to be controlled in the AOM of the same POA. In this case, however, it is not efficient because a search is run in AOM, even in response to Factory objects.
- You may register a Factory object as a POA Default Servant to control Servant objects. However, when using AOM and Default Servant simultaneously, requests to the Default Servant come after AOM searches. For this reason, requests to Factory objects are not efficient.

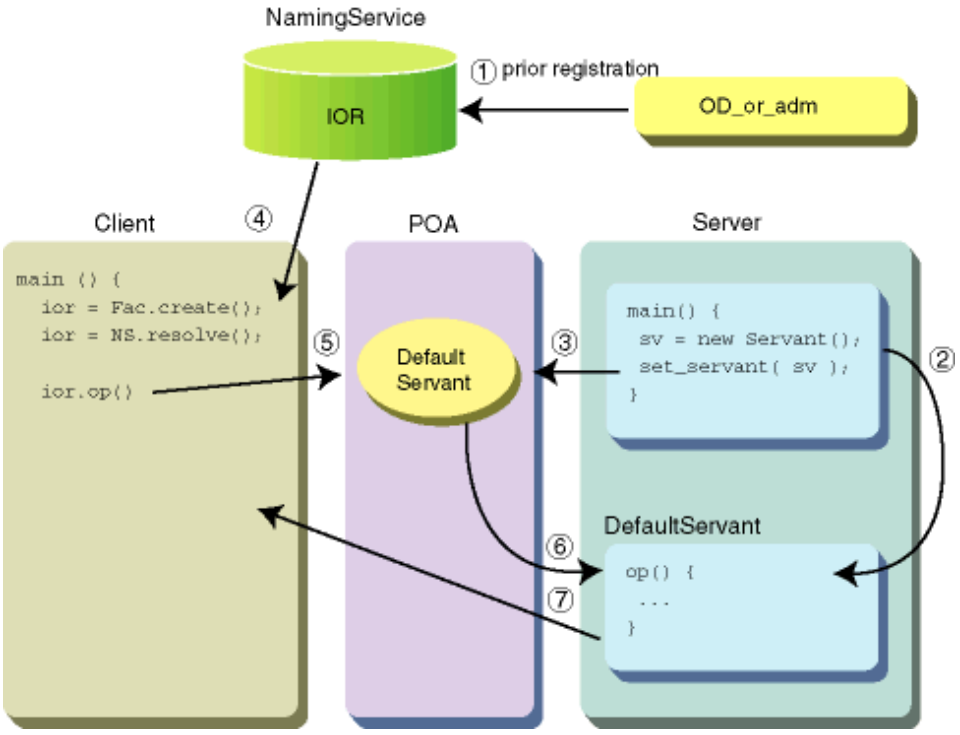
6.11.2 Configuring Each Application

This section provides information on configuring each application.

Default Instances Method

This method (shown in the following figure) does not control instances on each client.

Figure 6.27 Default Instances Method



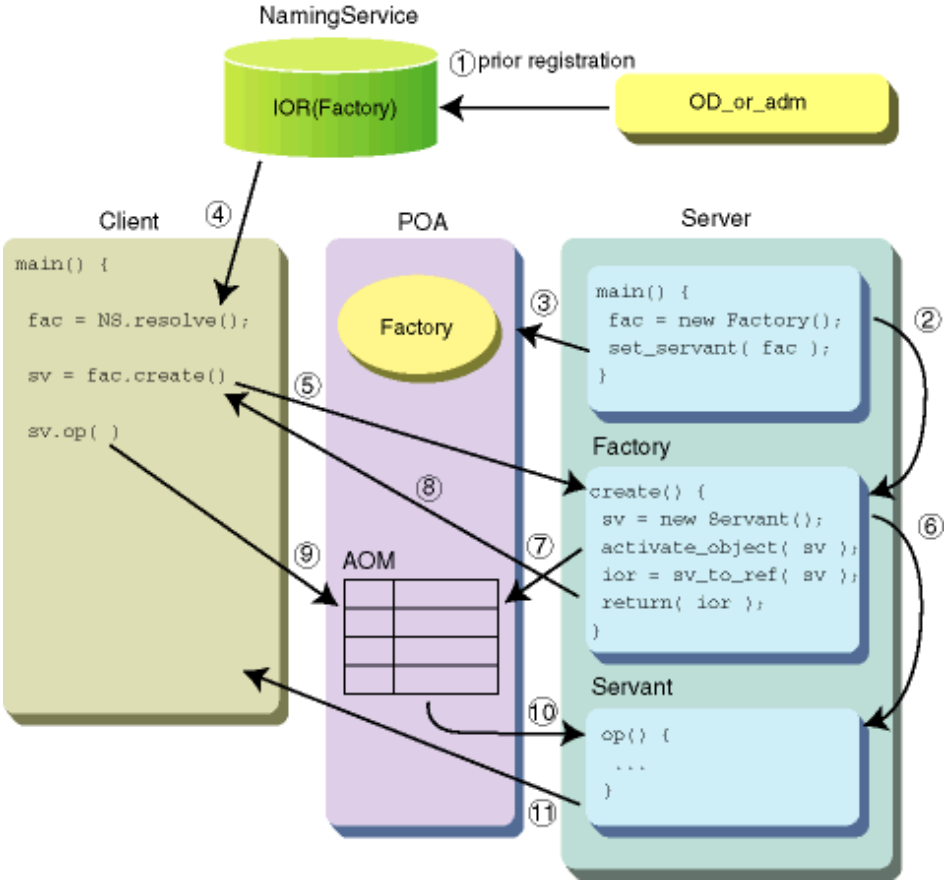
The following explanations correspond to the numbers in the figure above:

1. Register Servant's object references beforehand in NamingService with the OD_or_adm command.
2. Server application creates Servants.
3. Server application registers Servant in POA as default Servant.
4. Client application gets object references from NamingService.
5. Client application requests operation().
6. POA launches default server.
7. POA notifies results of operation op() to client.

Factory-1 Method

A method for running instance control on each client with POA objects. The creation of object references and instances is done in Factory. The following figure illustrates this method.

Figure 6.28 Factory-1 Method



The following explanations correspond to the numbers in the figure above.

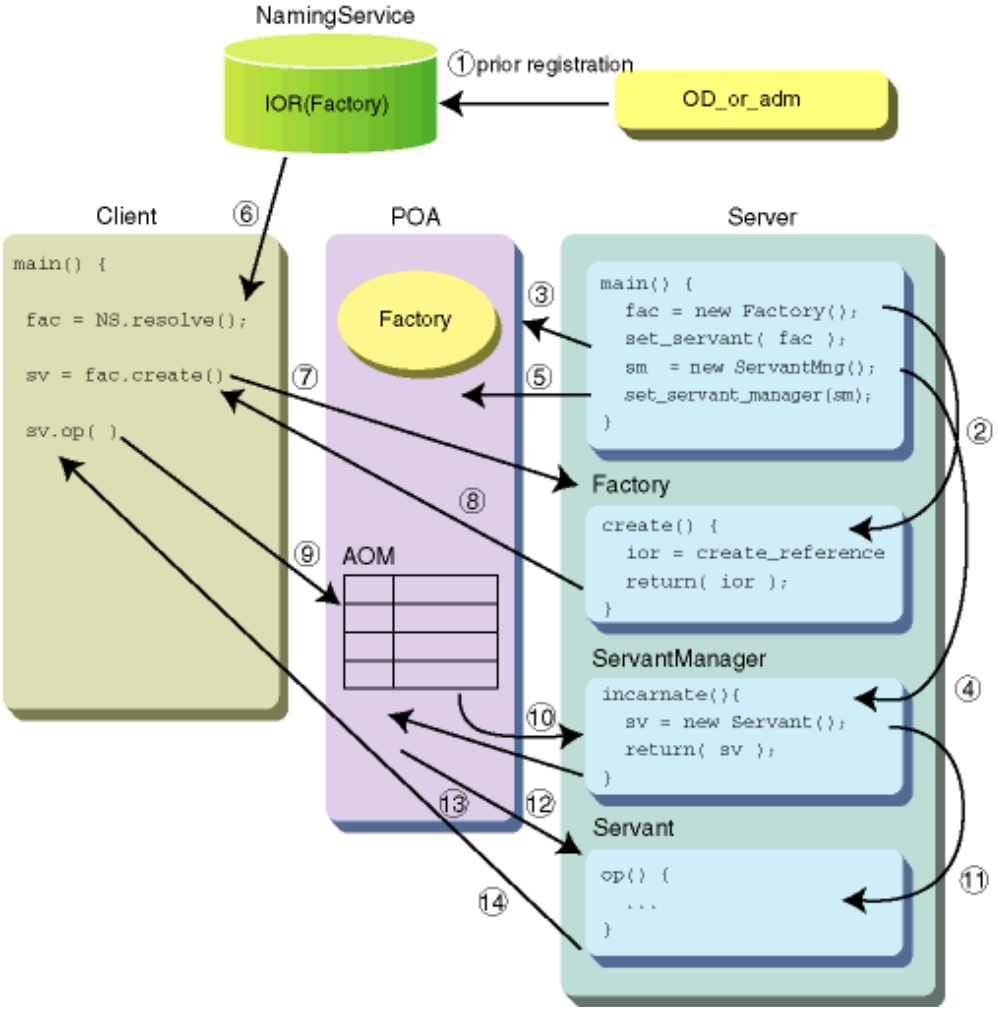
1. Register Factory object references beforehand in NamingService with the OD_or_adm command.
2. Server application creates Factory.
3. Server application registers Factory in POA as default Servant.
4. Client application gets Factory object references from NamingService.
5. Client application prompts Factory for operation create.
6. Factory constructs Servant instances.

7. Factory registers instances in AOM.
8. Factory constructs object references and informs client.
9. Client application requests operation op().
10. POA searches AOM and requests execution to Servant of operation op() with instances obtained.
11. POA notifies results of operation op() to client.

Factory-2 Method

A method for running instance control on each client with A POA object. Object references are done in Factory, and instances are created in ServantManager objects at the time of request reception. The following figure illustrates this method.

Figure 6.29 Factory-2 Method



The following explanations correspond to the numbers in the figure above:

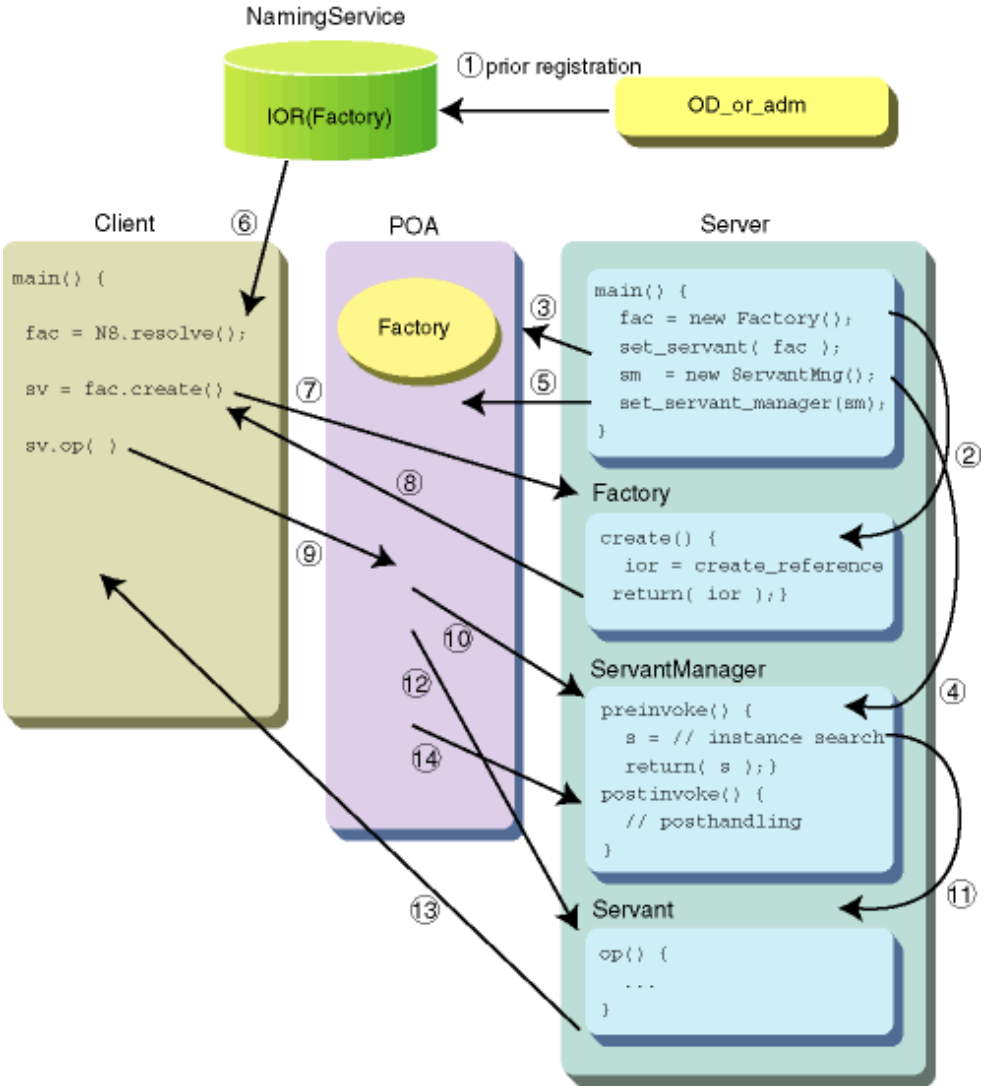
1. Register Factory object references beforehand in NamingService with the OD_or_adm command.
2. Server application creates Factory.
3. Server application registers Factory as Servant in POA.
4. Server application creates ServantManager.
5. Server application registers ServantManager in POA.

6. Client application gets Factory object references from NamingService.
7. Client application prompts Factory for operation create.
8. Factory constructs Servant object references and notifies client of object references.
9. Client application prompts for operation op().
10. POA searches AOM and invokes ServantManager with non-registered prompt.
11. ServantManager searches appropriate Servant and returns.
12. POA receives Servant from ServantManager and registers it in AOM.
13. POA requests execution of operation op() to Servant.
14. POA notifies results of operation op() to client.

User Instances Control Method

A method that delegates instance control to user-created ServantManager objects. The figure below illustrates this method.

Figure 6.30 User Instance Control Method



The following explanations correspond to the numbers in the above figure:

1. Register Factory object references beforehand in NamingService with the OD_or_adm command.
2. Server application creates Factory.
3. Server application registers Factory as Servant in POA.
4. Server application creates ServantManager.
5. Server application registers ServantManager in POA.
6. Client application gets Factory object references from NamingService.
7. Client application prompts Factory for operation create.
8. Factory constructs Servant object references and informs client of object references.
9. Client application prompts for operation op().
10. POA invokes ServantManager preinvoke().
11. ServantManager searches Servant instances and notifies POA.
12. POA requests execution of operation op() to Servant with instances obtained.
13. POA notifies client of results of operation op().
14. POA invokes ServantManager postinvoke().

6.11.3 A Comparison of Application Configurations

The following table compares application configurations.

Table 6.8 A Comparison of Application Configurations

	Default Instances Method	Factory-1 Method	Factory-2 Method	User Instances Control Method
Instance Control	No	Yes	Yes	Yes
Instance Controllers	--	POA objects	POA objects	User-created ServantManager objects
Performance	excellent	better than Factory-2 Method	less than Factory-1 Method	dependent on user ability
Resources	few	more than Factory-2 Method	fewer than Factory-1 Method	dependent on user ability
Ease of development	simply	normal	normal	difficult

6.12 Server Application Programming POA Overview

This section gives an overview of Portable Object Adapter (POA).

6.12.1 What is POA?

POA (Portable Object Adapter) is an interface newly adopted with CORBA2.2 as a standardized object adapter. With the clarification of the POA as an object adapter interface, the portability of server applications involving different ORB products is improved.

POA Features

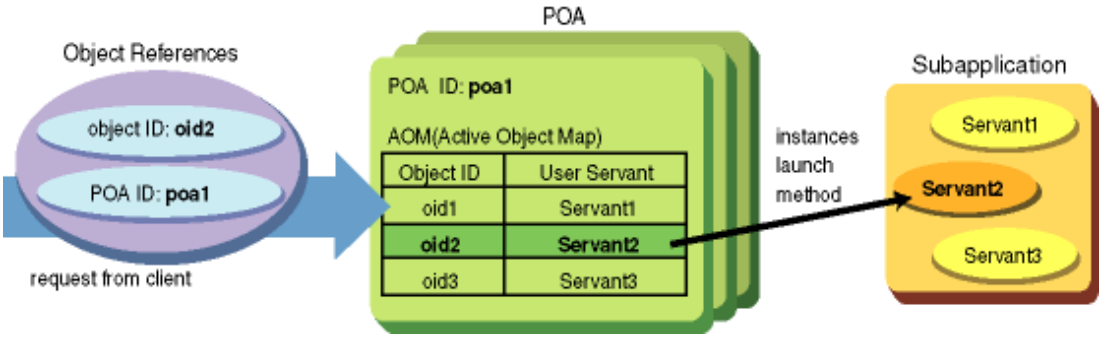
This section describes POA features.

Instance Control

It has AOM (Active Object Map: the instance control table) in a POA object and executes instance controls. Server applications can execute the two instance controls shown in the two figures below.

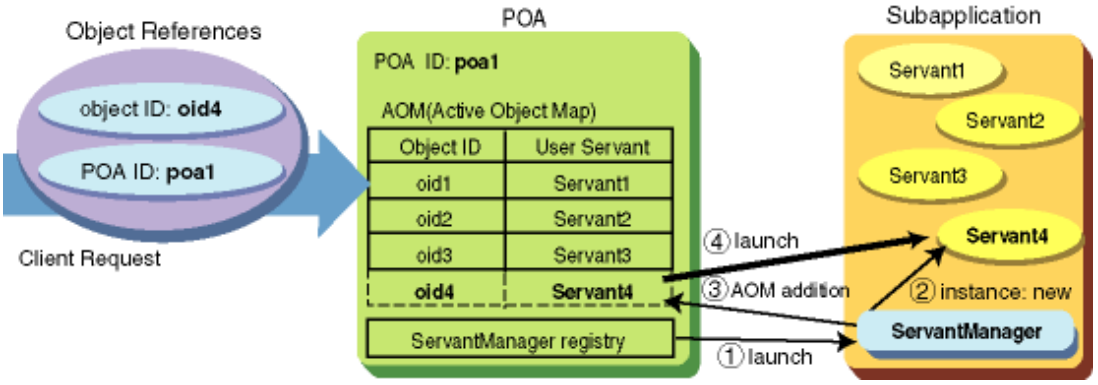
- Server applications created instances in advance and register them in AOM. When requests are received from a client, the Servant object that searches the AOM instances is run.

Figure 6.31 Instance Control (1)



- When a client request is received, the POA object invokes the user-implemented ServantManager. You create instances with ServantManager and register them in AOM. Then, Servant objects are run.

Figure 6.32 Instance Control (2)



POA Policies

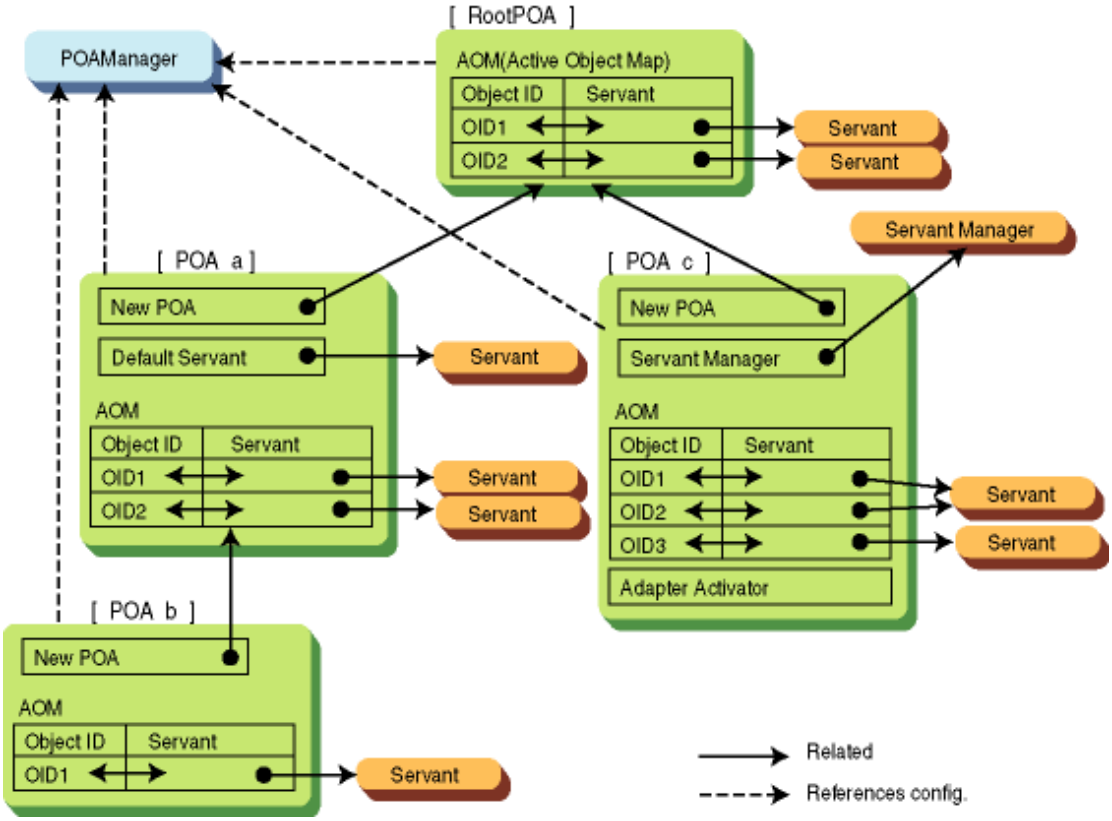
By setting POA policies, you are able to specify instance control methods, the life span of objects, and multiple action policies. As POA policies, the types below are available. For details, refer to "6.12.3 POA Objects".

- ID Assignment Policy: IdAssignmentPolicy
- Object ID Uniqueness Policy: IdUniquenessPolicy
- Implicit Activation Policy: ImplicitActivationPolicy
- Lifespan Policy: LifespanPolicy
- Request Processing Policy: RequestProcessingPolicy
- Servant Retention Policy: ServantRetentionPolicy
- Thread Policy: ThreadPolicy

6.12.2 POA Architecture

POA provides the structure shown below.

Figure 6.33 POA Architecture



POA Objects

POA objects control the relationships of object Ids, Servant objects (user-provided server implemented instances) in an AOM (Active Object Map), and run Servant object methods in accordance with the request messages from clients.

POAManager Objects

Run configuration controls relating to POA request message handling reception.

Servant Objects

Implement server application-provided interfaces, actually run request processing, and indicate user-created objects.

Default Servant Objects

USE_DEFAULT_SERVANT is set in the RequestProcessing policy for DefaultServant objects, and in cases where a Servant object corresponding to a request does not exist in the AOM (Active Object Map) of a POA, the POA will run the Default Servant object's method.

ServantManager Objects

ServantManager objects are invoked from A POA object, and run the creation and searches of Servant objects.

When USE_SERVANT_MANAGER is set in RequestProcessing policy for ServantManager objects, A POA object invoke them and we have the following two types.

- **ServantActivator** (when ServantRetention policy is RETAIN)

  When a Servant object corresponding to a request does not exist in the AOM (ActiveObjectMap) of the POA object, the POA object will request the creation of a Servant object to the ServantActivator object. ServantActivator objects will return Servant objects, and a POA object will run Servant object methods. The user will need to create these ServantActivator objects.

- **ServantLocator** (when ServantRetention policy is NON_RETAIN)

  A POA object will query the ServantLocator object for a corresponding Servant object upon request reception. The ServantLocator object will return a Servant object, and the POA object will run the Servant object methods. The user will need to create it, and it is necessary to control the corresponding Servant object methods.

Refer to "6.12.10 ServantManager Objects" for details.

AdapterActivator Objects

On receiving a request, if a corresponding POA object does not exist they will create A POA object. The user will need to create these AdapterActivator objects.

Object ID

The object ID is a value for the POA objects to assign to corresponding Servant objects. There are situations in which the object ID value is allocated on the basis of IdAssignment policy or A POA object, and there are cases in which it is allocated by the user application. Since the object ID is included in object references, it is unnecessary to get it from the client.

POA ID

POA ID is a value for assigning to A POA object at request reception. Since the POA ID is included in object references, it is unnecessary to get it from the client.

6.12.3 POA Objects

When creating a POA object, you will first need to create a POA policy list. Next, you will designate the policy list and create A POA object.

POA Policies

You may designate the policies summarized in the following table at the time of creating a POA object.

Table 6.9 POA Policies Summary

POA Policy	Significance	Value	Value explanation	Comment
ServantRetention:Servant Retention Policy	Specifies whether or not to retain an AOM table in a POA object. In brief, it specifies whether or not to run instance control in a POA object. Refer to "6.12.7 Request Processing" for details.	RETAIN (*1)	Remembers the active ServantObjects in ActiveObjectMap.	
		NON_RETAIN	Cannot use ActiveObjectMap	
Request Processing: Request Processing Policy	Specifies behavior in the event object instances are not retained in AOM or	USE_ACTIVE_ OBJECT_MAP_ONLY	Use ActiveObjectMap only in RequestProcessing	

POA Policy	Significance	Value	Value explanation	Comment
	even if AOM itself is not preserved. Refer to "6.12.7 Request Processing" for details.	(*1)		
		USE_DEFAULT_ SERVANT	Delegates RequestProcessing to Default Servant	
		USE_SERVANT_ MANAGER	Find the intended Servant objects with ServantManager objects. If needed, activate will also run.	
ImplicitActivation: Implicit Activation Policy	Servant activation mode can be designated as follows: Servant is activated implicitly. Servant is not activated implicitly. (applications explicitly register Servant in POA.) For details, refer to " 6.12.8 Implicit Activation".	IMPLICIT_ ACTIVATION (*1)	Servant objects implicitly activated.	
		NO_IMPLICIT_ ACTIVATION	Servant object not implicitly activated (applications explicitly create Servant object and register them in a POA object.)	
IdAssignment: ID Assignment Policy	Can designate whether system (POA object) auto-sets object ID or user sets it. USER ID is useful when you want to give some intrinsic Significant to the object ID with an application. For details, refer to " 6.12.6 Object Activation".	SYSTEM_ID (*1)	POA object assigns the object ID	
		USER_ID	User assigns object ID	
IdUniqueness: Object ID Uniqueness Policy	Can designate whether a unique ID is assigned or a multiple ID is allowed for the Servant. For details, refer to "6.12.5 Relating Object References, Object IDs, and Servant Objects".	UNIQUE_ID (*1)	Assigns a unique ID to Servant objects.	
		MULTIPLE_ID	Permits multiple ID for Servant objects.	
Lifespan: Lifespan Policy	Can designate whether to handle as transient type or persistent type.	TRANSIENT (*1)	Handles transient type objects (sub lifespan = object lifespan).	PERSIST-ENT cannot be specified
		PERSISTENT	Handles persistent type objects (sub lifespan < object lifespan).	
Thread: Thread Policy	Specifies whether thread processing is delegated to ORG or activated by single thread.	ORB_CTRL_MODEL (*1)	Thread processing delegated to ORB.	The setting value of this POA policy
		SINGLE_THREAD_	Activate by single thread.	

POA Policy	Significance	Value	Value explanation	Comment
		MODEL		is invalid. (*2)

1 Values indicated by an asterisk ('') in the above table denote default values.

*2 Exists in the setting contents for maximum process multiplicity (proc_conc_max) at the time of implementation policy registration and thread initiate multiplicity (thr_conc_init).

POA Policy Creation Example

```
// create POA policies
org.omg.CORBA.Policy policies[ = new org.omg.CORBA.Policy[4];
policies[0] =  Poa.create_servant_retention_policy(
    org.omg.PortableServer.ServantRetentionPolicyValue.NON_RETAIN );
policies[1] =  Poa.create_request_processing_policy(
    org.omg.PortableServer.RequestProcessingPolicyValue.USE_DEFAULT_SERVANT );
policies[2] =  Poa.create_id_assignment_policy(
    org.omg.PortableServer.IdAssignmentPolicyValue.SYSTEM_ID );
policies[3] =  Poa.create_id_uniqueness_policy(
    org.omg.PortableServer.IdUniquenessPolicyValue.MULTIPLE_ID );
```

Note

POAs in source are POA class instances.

POA Object Creation

This section provides information on POA object creation.

RootPOA Objects

You will need to create a RootPOA object first in order to use a POA object. POA objects are capable of being formed into tree structures by creating child or relative POA objects with RootPOA objects as the route. RootPOA objects are created at an ORB sub initialization. POA objects of RootPOA objects can be acquired in the manner shown in the example provided.

Example

```
// create and initialize ORB
org.omg.CORBA.ORB  Orb = org.omg.CORBA.ORB.init( args, null );

// get object references of RootPOA object
org.omg.CORBA.Object  _tmpObj = Orb.resolve_initial_references( "RootPOA" );
// get POA object of RootPOA object
org.omg.PortableServer.POA  rootPOA = org.omg.PortableServer.POAHelper.narrow( _tmpObj );
```

The following POA policies are set in a POA object.

- IdAssignmentPolicy SYSTEM_ID
- IdUniquenessPolicy UNIQUE_ID
- ImplicitActivationPolicy IMPLICIT_ACTIVATION
- LifespanPolicy TRANSIENT
- RequestProcessingPolicy USE_ACTIVE_OBJECT_MAP_ONLY

- ServantRetentionPolicy RETAIN
- ThreadPolicy ORB_CTRL_MODEL

It is possible to utilize RootPOA objects unchanged when running instance control by the POA policies above. But by creating new POA objects (descendant objects of RootPOA objects) and setting native POA policies, it becomes possible to run instance control with POA policies that are different from the RootPOA objects.

Descendant POA Objects

To create a new POA, invoke the create_POA() method in a previously existing POA. As a result of this, the POA that ran create_POA() and the newly created POA object become new descendants. In an application, since the first POA object to be created is the RootPOA object, all other POA objects become descendant POA objects of the RootPOA object.

By delivering the POA policy list (Policy class array) as the create_POA() attribute, setting the policies of the created child POA objects is possible.

 Example

```
// child POA object (Poa1) creation
orb.omg.PortableServer.POA    Poa1 = rootPOA.create_POA("poa_1", poamanager, policies1);
// grandchild POA object (Poa2) creation
orb.omg.PortableServer.POA    Poa2 = Poa1.create_POA("poa_2", poamanager, policies2);
```

Notes

- poa_1 and poa_2 in source are the POA name they create
- poamanager in source is POAManager instance.
- policies1and policies2 are Policyclass instances

In the above manner, it is possible for a POA object to hold a layered structure. Refer to "6.13 Relating Server Applications and Environment Settings" regarding the creation of objects and the relationship between objects and a POA object that controls them. In cases when certain POA objects are revoked by the destroy() method, their descendent POA objects are all revoked.

Combining POA Setting Values

Whether the combination of the various POA policy values set in the policy list is appropriate or not is checked when the create_POA method is run. If the combinations of setting values are somehow inappropriate, it generates an exception.

The combinations of settings values in the POA policy list is displayed below. We will demonstrate below how the org.omg.PortableSever.POAPackage.InvalidPolicy exception is generated when create_POA() method is run in a situation when the part with this value ("*") in the Suitability column has been set.

Furthermore, be sure to use TRANSIENT as a LifespanPolicy setting value. Also, the ThreadPolicy setting value is disregarded.

Table 6.10 Combining POA Policy Settings Values

Policy	Servant Retention	Request Processing	Implicit Activation	IdAssignment	IdUniqueness	Suitability
Settings Values	RETAIN	USE_ACTIVE_OBJECT_MAP_ONLY	IMPLICIT_ACTIVATION	SYSTEM_ID	UNIQUE_ID	
					MULTIPLE_ID	
				USER_ID	-	*
			NO_IMPLICIT_	SYSTEM_ID	UNIQUE_ID	
					MULTIPLE_ID	

Policy	Servant Retention	Request Processing	Implicit Activation	IdAssignment	IdUniqueness	Suitability
			ACTIVATION	USER_ID	UNIQUE_ID	
					MULTIPLE_ID	
		USE_ DEFAULT_ SERVANT	IMPLICIT_ ACTIVATION	SYSTEM_ID	UNIQUE_ID	*
					MULTIPLE_ID	
				USER_ID	-	*
			NO_ IMPLICIT_ ACTIVATION	SYSTEM_ID	UNIQUE_ID	*
					MULTIPLE_ID	
				USER_ID	UNIQUE_ID	*
					MULTIPLE_ID	
		USE_ SERVANT_ MANAGER	IMPLICIT_ ACTIVATION	SYSTEM_ID	UNIQUE_ID	
					MULTIPLE_ID	
				USER_ID	-	*
			NO_ IMPLICIT_ ACTIVATION	SYSTEM_ID	UNIQUE_ID	
					MULTIPLE_ID	
				USER_ID	UNIQUE_ID	
					MULTIPLE_ID	
	NON_ RETAIN	USE_ACTIVE_ OBJECT_MAP_ONLY	-	-	-	*
		USE_ DEFAULT_ SERVANT	- (*1)	SYSTEM_ID	UNIQUE_ID	*
					MULTIPLE_ID	
				USER_ID	UNIQUE_ID	*
					MULTIPLE_ID	
		USE_ SERVANT_ MANAGER	- (*1)	SYSTEM_ID	UNIQUE_ID	
					MULTIPLE_ID	
				USER_ID	UNIQUE_ID	
					MULTIPLE_ID	

*1 When NON_RETAIN is set in ServantRetentionPolicy, the setting value for ImplicitActivationPolicy is disregarded.

Default Setting Values of POA Policies

POA has the values given in the following table as default policy settings.

Table 6.11 Default Setting Values of POA Policies

Policy	Default Value
ServantRetention	RETAIN
RequestProcessing	USE_ACTIVE_OBJECT_MAP_ONLY
ImplicitActivation	NO_IMPLICIT_ACTIVATION
IdAssignment	SYSTEM_ID
IdUniqueness	UNIQUE_ID

Policy	Default Value
Lifespan	TRANSIENT
Thread	ORB_CTRL_MODEL

When you create a new POA using the create_POA () method, you can only run the policy setting for wanting to modify the default settings.

In the following example, a child POA is created in which RequestProcessingPolicy was set to USE_DEFAULT_SERVANT, and IdUniquenessPolicy was set to MULTIPLE_ID. In this situation, the settings values for other policies become the same as the default settings values above.

 Example

```
// create POA policy list
org.omg.CORBA.Policy policies[] = new org.omg.CORBA.Policy[2];
policies[0] =  Poa.create_request_processing_policy(
    org.omg.PortableServer.RequestProcessingPolicyValue.USE_DEFAULT_SERVANT);
policies[1] =  Poa.create_id_uniqueness_policy(
    org.omg.PortableServer.IdUniquenessPolicyValue.MULTIPLE_ID );
// create child POA object
org.omg.PortableServer.POA  newPOA = Poa.create_POA("poa_1", poamanager, policies);
```

Notes

- poa in source is POA class instance
- poamanager in source is POAManager class instance

Additionally, in a case where the create_POA() method's 3rd parameter is set to null, the child POA policy becomes the default setting value.

6.12.4 Creating Object References

You can create object references using the two methods covered in this section:

- Pre-Servant Object Activation Configuration Method (Direct Configuration)
- Post-Servant Object Activation Configuration Method

Pre-Servant Object Activation Configuration Method (Direct Configuration)

For server applications you may construct direct object references with the create_reference() or create_reference_with_id() methods of a POA object. These operations create object references only. There are also situations in which actual active Servant objects are created one after another by the ServantManager object.

 Example

```
// object reference creation
org.omg.CORBA.Object  Obj = Poa.create_reference( "IDL:Intfid11:1.0" );

// object reference creation by ID designation
String userid = "USERID";
org.omg.CORBA.Object Obj = Poa.create_reference_with_id( userid.getBytes(), "IDL:Intfid11:1.0" );
```

Note

The poa in source is a POA class instance.

IDL:Intfid11:1.0: Interface Repository ID

Post-Servant Object Activation Configuration Method

For server applications, you may activate Servant objects by the activate_object() or activate_object_with_id() methods. Once a Servant object is activated, it's possible for a server application to construct object references easily by designating a Servant object or object ID and then running the servant_to_reference() or id_to_reference() methods. Also, if POA objects has been constructed by a IMPLICIT_ACTIVATION policy, when servant_to_reference() is used to create references Servant objects are automatically activated and the references become valid.

Example

```
// Servant creation
org.omg.PortableServer.Servant svt = new UserServant();

// when IMPLICIT_ACTIVATION policy is specified
// activated automatically
org.omg.CORBA.Object Obj =  Poa.servant_to_reference( svt );
```

Note

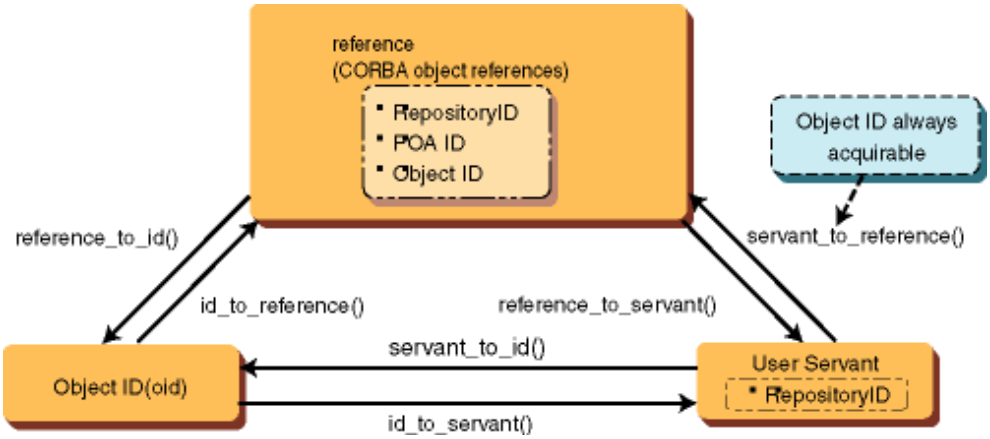
Poa in source is POAclass instance.

Once object references are acknowledged to the client, if seen from the client side, those object references take on the ID of the object. To the extent a client program uses those references, the requests that can be made with their object references are sent to the identical object instances.

6.12.5 Relating Object References, Object IDs, and Servant Objects

The following figure shows the relationship between object references, object IDs, and Servant objects.

Figure 6.34 The Reference/oid/Servant Object Relationship



You determine the relationship of object references, object ID (oid), and Servant objects by the IdUniquenessPolicy. Because object references encompass the oid, reference and oid take on a 1 to 1 relationship. As for the relationship of oid and Servant objects, in a UNIQUE_ID situation, a unique oid is assigned to each Servant object. In a MULTIPLE_ID situation, it is possible to assign multiple oid to the same Servant object. This has an impact on the operations of servant_to_id() or servant_to_reference() that create oid.

Figure 6.35 IdUniquenessPolicy

6.12.6 Object Activation

In regard to object activation states, they specify configurations in which certain Servant objects are related to an objectID and registered in the POA-controlling AOM (Active Object Map).

Not only have Servant objects been created as instances, but since they are not recognized in a POA object you will need to register their relationship with an object ID in a POA object using the activate_object() or activate_object_with_id() methods.

The two methods for activating Servant objects are:

- Explicit Servant Object Activation Method
- Implicit Servant Object Activation Method

Explicit Servant Object Activation Method

After the create(new) of a Servant object, you will be able to register it in AOM with the activate_object() or activate_with_id() methods.

Example

```
// create Servant
org.omg.PortableServer.Servant svt = new UserServant();

// activate Servant
Poa.activate_object ( svt );

// activate Servant by ID specification
String userid = "USERID";
Poa.activate_object_with_id( userid.getBytes(), svt );
```

Note

poa objects inside source are POA class instances.

The object activation needs an objectID, and it is assigned following IdAssignmentPolicy

- SYSTEM_ID

  POA objects assign an objectID automatically. Use activate_object() when activating a Servant object.

- USER_ID

  A user assigns an objectID. Use activate_object_with_id() when activating a Servant object.

Implicit Servant Object Activation Method

If you create references using the servant_to_reference() method when a POA object has an IMPLICIT_ACTIVATION policy, the Servant objects are activated automatically and object references are communicated. Refer to "6.12.8 Implicit Activation" for information about implicit activation.

Example

```
// create Servant
org.omg.PortableServer.Servant svt = new UserServant();

// when IMPLICIT_ACTIVATION policy is specified, activated automatically
org.omg.CORBA.Object Obj =  Poa.servant_to_reference( svt );
```

Note

poa objects in source are POA class instances.

Furthermore, an object ID is assigned automatically by POA object in the case of the implicit activation. Therefore, IdAssignmentPolicy must be set to SYSTEM_ID.

6.12.7 Request Processing

This section provides information on request processing.

Searching POA Objects

If a POA object corresponding to the Interface Repository ID does not exist in the server process at the time a request is received from a client, a new POA object will invoke an AdapterActivator object. You will create the requested POA object with an AdapterActivator object. The user will need to construct the AdapterActivator object, and it is necessary to register it in a POA object in advance. If the AdapterActivator object has not been registered, the client will get the OBJECT_NOT_EXIST exception.

Searching Servant Objects

At the time a request is received from a client, the POA object will search the Servant object that is registered in AOM. Also, whether or not AOM is used depends upon whether an object's POA objects have specified a RETAIN (use) or NON_RETAIN (not use) policy.

If RETAIN has been specified and the targeted Servant object is found in AOM, the search process stops there and the Servant object's method runs. In situations where Servant objects are not found or NON_RETAIN applies, the process is run according to one of the following RequestProcessing method.

- **USE_DEFAULT_SERVANT**

A method for delegating processing to the default Servant object when an object's Servant object was not found. In this situation, you will need to register the default Servant objects beforehand by the set_servant() method. If the default Servant objects are not registered, an OBJ_ADAPTER error is returned.

- **USE_SERVANT_MANAGER**

A method for delegating the process of searching for Servant objects to a ServantManager object when an object's Servant objects were not found.

In order to find a Servant object that can handle the request, the POA object invokes either incarnate() or preinvoke() (The method selection exists in the POA object's NON_RETAIN or RETAIN policies.). The ServantManager object announces the discovered Servant object to

the POA object. In this case you will need to register the ServantManager object beforehand by the set_servant_manager() method. If the ServantManager object has not been registered, an OBJ_ADAPTER error is returned.

- **USE_ACTIVE_OBJECT_MAP_ONLY**

An OBJECT_NOT_EXIST error is returned in this case.

6.12.8 Implicit Activation

There is an IMPLICIT_ACTIVATION policy that we can add to the other methods for activating Servant objects in which applications explicitly activate Servant objects. This one activates a Servant object implicitly by means of a POA object.

You will need to specify the SYSTEM_ID and RETAIN policies in order to use the IMPLICIT_ACTIVATION policy. By specifying the IMPLICIT_ACTIVATION policy, when methods such as servant_to_reference() or servant_to_id() for converting object references or object ID from a Servant object were used, the POA object can register the Servant object and object ID in AOM and activate those Servant objects. In situations when a UNIQUE_ID policy has been specified or, again, when servant_to_references() or servant_to_ID() have been used at this point, the same references and object ID as before are returned. When the MULTIPLE_ID policy has been specified, a different object reference and object ID is returned each time. This signifies that multiple object references and object ID's are allocated to the same Servant objects.

6.12.9 POAManager Objects

POAManager objects are related to a POA object by passing as a create_POA() method attribute at the time the POA objects are created. As shown in the following figure, the POAManager has the following four configurations:

holding configuration

Queues a received message.

active configuration

Processes a received message.

discarding configuration

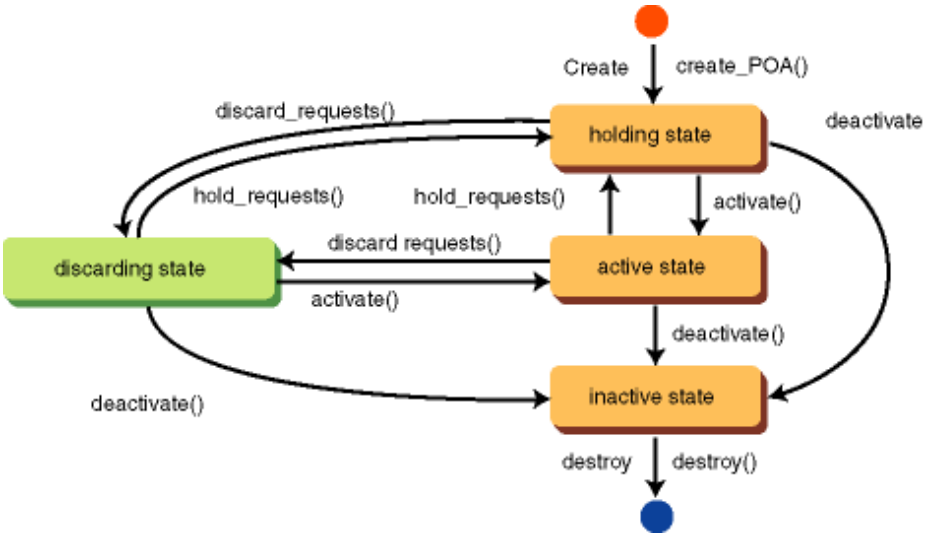
Discards a received message. In this case, an error is alerted to the client.

inactive configuration

Configuration immediately prior to shutdown.

A POA object determines its processes according to the POAManager's configuration. With these configurations, it is possible to make modifications with methods such as activate(), hold_requests(), discard_requests(), and deactivate().

Figure 6.36 Configuration Shift of POA Objects



6.12.10 ServantManager Objects

The ServantManager object is a function that enables the creation/deletion or search of Servant objects at the time a request is received. Implementation of the various methods of ServantManager objects is constructed by the user. Constructed ServantManager objects are to be registered in a POA object.

In situations where a Servant object that matches the object ID in AOM is not registered when a POA object receives a request, the POA object invokes the ServantManager object's method, and requests creation/deletion or search of the Servant object.

Two types of ServantManager objects exist:

- **ServantActivator Objects**

  These run the search/delete of Servant objects

- **ServantLocator Objects**

  These search Servant objects.

POA objects invoke ServantActivator objects when the ServerRetention policy is RETAIN, and invoke ServantLocator objects when it is NON_RETAIN.

ServantActivator Objects

It is required that the ServantActivator class you are constructing should inherit the org.omg.PortableServer.ServantActivator class. Also, you will need to have the incarnate_etherealize() method implemented.

Example ServantActivator Class Implementation Format

```
class UserServantActivator extends org.omg.CORBA.LocalObject
        implements org.omg.PortableServer.ServantActivator{
      public org.omg.PortableServer.Servant incarnate(...) {...}
      public void etherealize(...) {...}
}
```

When a POA object requests the creation of a Servant object it invokes the incarnate() method with the ServantActivator object, and when it requests the deletion of a Servant object it invokes the etherealize() method. Parameters that are passed from a POA object can be used, as required, by the implementation section of each method.

- **Incarnate (Create Servant Object)**

  POA invokes the incarnate() method in cases where an object's Servant object is not registered in AOM when a request is received. The incarnate() method creates and returns Servant objects to a POA object.

  ```
  org.omg.PortableServer.Servant  incarnate(
      byte[] oid,
      org.omg.PortableServer.POA adapter )
  ```

  oid : object ID of the request object

  adapter : POA object of invocation origin

- **Etherealize (Delete Servant Object)**

  POA invokes the etherealize() method when a deactivate_object() method has been invoked from an application. It specifies the Servant objects that POA objects delete by parameters. The etherealize() method will delete specified Servant objects.

  ```
  void etherealize(
      byte[] oid,
      org.omg.PortableServer.POA adapter,
      org.omg.PortableServer.Servant serv,
  ```

```
    boolean cleanup_in_progress,
    boolean remaining_activations );
```

oid : object ID corresponding to Servant object of deleted object

adapter : POA object of invocation source

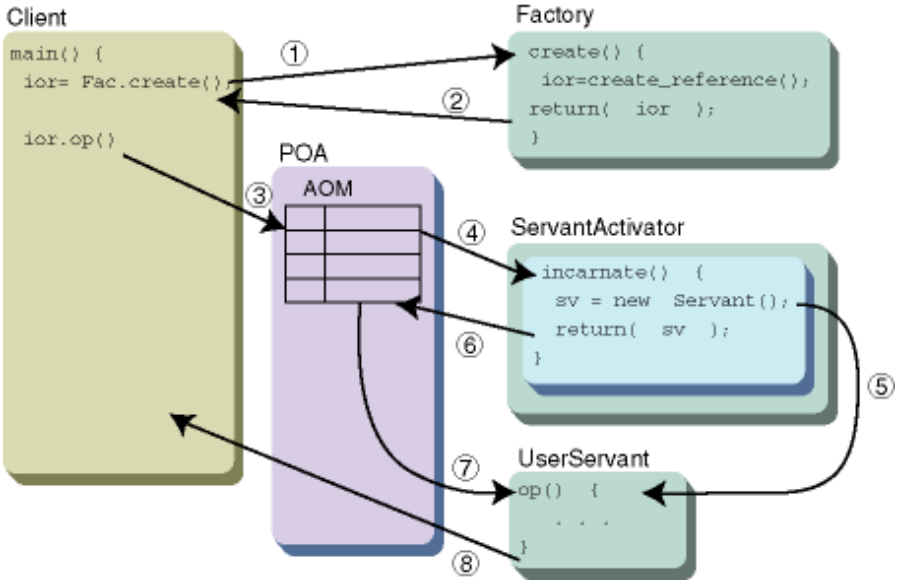
serv : Servant object of delete object

cleanup_in_progress : true when main method has been invoked from POA destroy()

remaining_activations : usually false when invoking main method

The flow of specific ServantActivator processes is displayed below.

Figure 6.37 Work Overview of ServantActivator Objects



The following explanations correspond to the numbers in the figure above:

1. Requests creation of Servant's object references to Factory.
2. Returns object references created by Factory to client.
3. Requests operation op() to Servant.
4. Invokes ServantActivator with POA for registration request to AOM.
5. Searches appropriate Servant with ServantActivator and returns.
6. Gets Servant from ServantActivator and registers it with AOM.
7. Requests running of operation op() on Servant.
8. Notifies client of operation op() results.

Note

Factory is an object that constructs object references or instances for each request, and is needed in situations where instance processing is run.

ServantLocator Objects

You will need to have the ServantLocator you construct inherit the org.omg.PortableServer.ServantLocator class in the manner below. Also, it will be necessary for the preinvoke() and postinvoke() methods to be implemented.

Example Implementation Formats of ServantLocator Classes

```
class UserServantLocator extends org.omg.CORBA.LocalObject
        implements org.omg.PortableServer.ServantLocator{
      public org.omg.PortableServer.Servant preinvoke(...) {...}
      public void postinvoke(...) {...}
}
```

POA objects invoke the preinvoke() method of the ServantLocator objects each time to process requests from a client, and inherit the Servant objects of the process object. POA objects request processing from their Servant objects, and when that processing finishes they invoke the postinvoke() method of the ServantLocator objects and request post-processing to the ServantLocator objects.

- preinvoke

  POA objects invoke the preinvoke() method of the ServantLocator objects in order to inherit the Servant objects whose processing they must request. The preinvoke() method simultaneously returns postinvoke() information (the cookie) and Servant objects to a POA object.

```
org.omg.PortableServer.Servant  preinvoke(
       byte[] oid,
       org.omg.PortableServer.POA adapter,
       String operation,
       org.omg.PortableServer.ServantLocatorPackage.CookieHolder the_cookie )
```

  oid : object ID of the request object

  adapter : POA object of invocation source

  operation : operation name of request object

  the_cookie : CookieHolder object for storing Cookie objects

- postinvoke

  POA objects invoke a postinvoke() for the ServantLocator object immediately after termination of the Servant object's request processing. At that time, it specifies a cookie returned by preinvoke() in order to obtain a correspondence with preinvoke().

```
void  postinvoke(
    byte[] oid,
    org.omg.PortableServer.POA adapter,
    String operation,
    java.lang.Object the_cookie,
    org.omg.PortableServer.Servant the_servant );
```

  oid : Object ID of request object

  adapter : POA object of invocation source

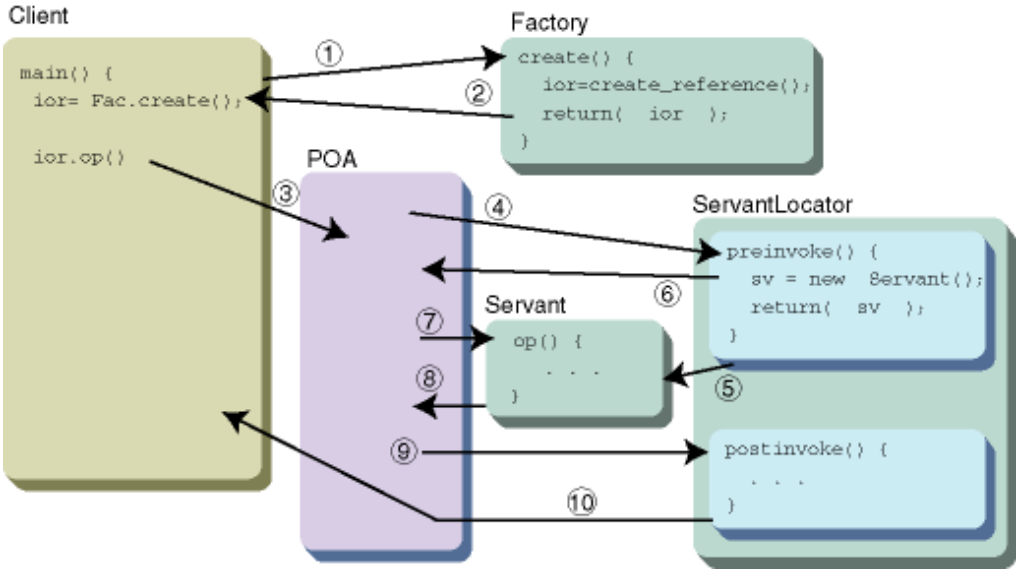
  operation : Operation name of request object

  the_cookie : Cookie object returned by preinvoke

  the_servant : Servant object that processed the request

The flow of specific ServantLocator processes is displayed below.

Figure 6.38 Operation Overview of ServantLocator Objects



The following explanations correspond to the numbers in the figure above:

1. Requests creation of Servant object references to Factory.
2. Returns object references created by Factory to clients.
3. Requests operation op() to Servant.
4. Invokes ServantLocator with POA.
5. Searches Servant with ServantLocator preinvoke().
6. Gets Servant.
7. Request running of operation op() on Servant.
8. Informs clients of operation op() results.
9. Invokes postinvoke() of ServantLocator.
10. Restores client applications.

6.12.11 AdapterActivator Objects

AdapterActivator objects are used to create a POA object at the time requests are received. They add an object ID to object references that POA objects have created, and a POAid is embedded which shows the creator POA object. When a request has been sent from a client, the POA objects that correspond to the POAid are searched and those POA objects process the request. In a situation where it could not be found, if an AdapterActivator object were to be specified to a parent POA object for the search object, it would request the creation of a child POA object to that AdapterActivator object. When the AdapterActivator object has not been set in the parent POA, the client is notified of an org.omg.CORBA.OBJ_ADAPTER exception.

AdapterActivator objects require registration in a user-created POA.

The AdapterActivator class you construct requires the inheritance of the org.omg.PortableServer.AdapterActivator class. Also, it is necessary for unknown_adapter() to be implemented as a method.

Implementation Formats for AdapterActivator Class Examples

```
class UserAdapterActivator extends org.omg.CORBA.LocalObject
        implements org.omg.PortableServer.AdapterActivator {
      public boolean unknown_adapter(...){...}
}
```

- unknown_adapter

  In the event the POA corresponding to POAid could not be searched, the AdapterActivator's unknown_adapter() method is invoked. In unknown_adapter(), POA objects and String objects are passed as parameters. In unknown_adapter(), child POA of passed POA objects are created using create_POA(). The passed String object is used as the adapter name at that time. It is possible to describe the policy settings for child POA and other necessary processing (create/registration of Servant or ServantManager) within this method. In cases where POA creation and other implemented processes have been successful, the return value is true. When that is not the case, it returns false. When true is returned, the Servant object from the created child POA is launched. If false was returned, the client is informed of a org.omg.CORBA.OBJECT_NOT_EXIST exception.

```
public boolean unknown_adapter( POA parent, String name )
```

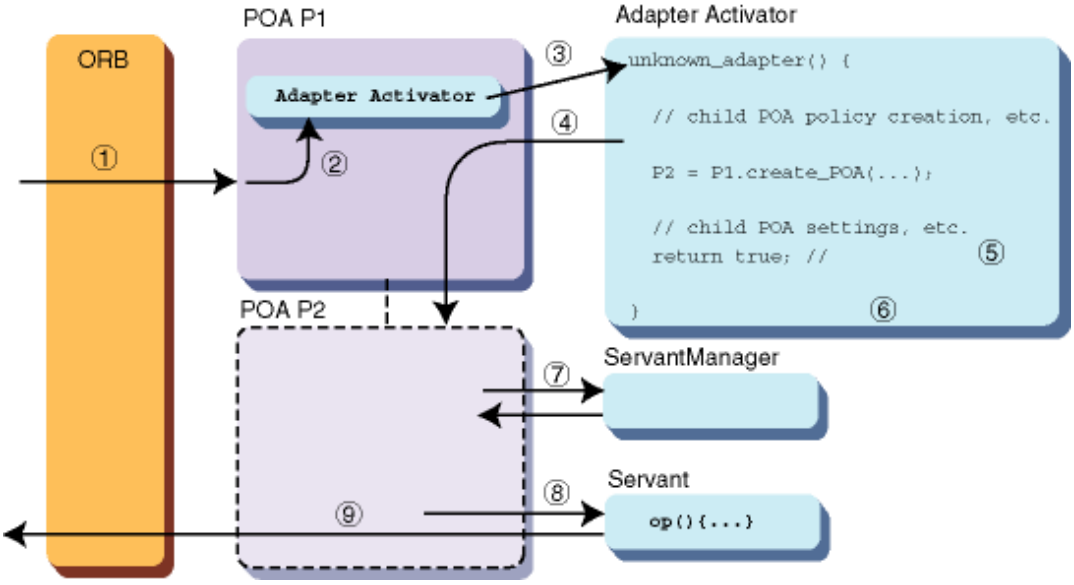
  parent : Parent POA of POA created

  name : Adapter name of POA created

Invocation of an AdapterActivator object's unknown_adapter method at the time the org.omg.PortableServer.POA.find_POA method is running is also possible.

The following figure displays the flow of the AdapterActivator process.

Figure 6.39 AdapterActivator Process Flow



The following explanations correspond to the numbers in the figure above:

1. Receives a request from client. The object references of the request object incorporate the POAid.
2. Searches for POA object corresponding to POAid. If nonexistent, searches the parent POA of the object POA from the existing POA.
3. The unknown_adapter() method is run on the AdapterActivator class instances set in the parent POA object.
4. Creates a child POA object of the (parent) POA passed by parameters in the implementation section of unknown_adapter(). Afterwards , processing of requests to the same object references is done with this POA.
5. Does the settings for the created child POA (here, it is possible to create a direct Servant and register it in the child POA, and possible to create and register ServantManager as well).
6. Returns true when child POA creation is successful.

7. Does creation, registration of Servants (when using ServantManager).
8. Launches Servant operations.
9. Returns operation run results.

6.13 Relating Server Applications and Environment Settings

The OD_impl_inst command, for registering implementation information, and OD_or_adm command, for registering with a NamingService, are both contained in CORBA Service.

We will explain in this section the relationship between server applications and the OD_impl_inst and OD_or_adm commands.

6.13.1 Relating Implementation Information

In order to get a server application that has used a POA object to function, you will need to relate it to the implementation repository ID specified in the OD_impl_inst command.

There are two ways to perform this process:

1. Specifying to Environment Variable OD_IMPLID
2. Specifying POA Objects to an Adapter Name

Specifying to Environment Variable OD_IMPLID

Specifying a implementation repository ID name to the environment variable OD_IMPLID becomes possible in the server application's operating environment. For example, in a situation in Windows where you launch a server application with a batch file, you will specify that as follows.

 Example

[xxx.bat]

```
set OD_IMPLID=Implid1 <-  set implementation repository to OD_IMPLI
     ...
java.exe  Server      <-  Launch Java VM in parameter to server application
```

Specifying POA Objects to an Adapter Name

You will be able to relate a POA object and implementation information explicitly in applications without setting the environment variable OD_IMPLID. With this method, it becomes possible to use the implementation policy repository ID in the child POA object's adapter name under the RootPOA object value.

6.13.2 Methods for Creating Object References

There are two methods for creating object references and registering them to a NamingService:

- Dynamic Creation
- Pre-Creation

Dynamic Creation

You will create them when running in POA Objects (inside a server application) by the create_reference() method for a POA object. The created object references are either registered to a NamingService inside the application or returned to the client application by notifying the client application as a parameter.

Since this method allows you to construct object references to each request from the client, it can run each client's instance controls dynamically. The object references for the "Factory-1 Method," "Factory-2 Method," and "User Instances Control Method" Servant objects demonstrated in 6.11.2 Configuring Each Application are constructed by dynamic creation. Additionally, refer to "6.12.4 Creating Object References" regarding the dynamic creation method for object references.

Pre-Creation

Using the CORBA Service control OD_or_adm command, it performs creation and registration with a NamingService in advance. Because this method registers object references in advance, it is superior in performance and resource maximization compared with dynamic creation.

The object references for the "Factory-1 Method," "Factory-2 Method," and "User Instance Control Method" Factory objects and the "Default Instances Method" Servant objects demonstrated in 6.11.2 Configuring Each Application are constructed by pre-creation.

6.13.3 Sample Methods for POA Uses

The POA object configuration depends on which combination of the following methods are used for the server application implementation:

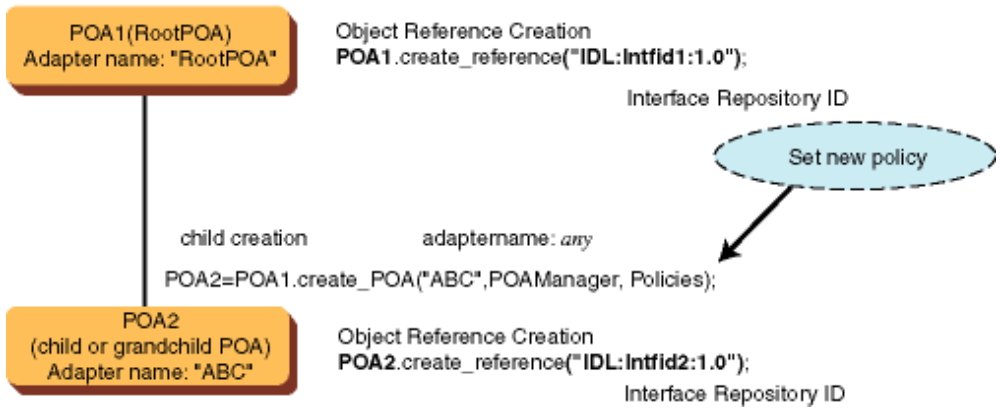
- Implementation repository ID associated method (for details, refer to "6.13.1 Relating Implementation Information")
  - The environment variable OD_IMPLID was used (Specifying to Environment Variable OD_IMPLID)
  - The environment variable OD_IMPLID was not used (Specifying POA Objects to an Adapter Name)
- Object reference creation method (for details, refer to "6.13.2 Methods for Creating Object References")
  - Dynamic Creation
  - Pre-Creation

The method to use the POA object and configuration is explained for each combination.

POA Object Use Method 1

This method (shown in the figure below) uses dynamic creation with the environment variable OB_IMPLID.

Figure 6.40 POA Object Configuration Method 1



This is a general POA object configuration:

1. Creates POA (any adapter name).
2. Creates object references to hold the specified interface with the create_reference method.
3. Creates Servant
4. Gets object ID for the created object references.
5. Registers the relation of Servant interface and object ID in POA's AOM(initialization).
6. Does registration of object references with a NamingService.

In the Figure above, the instances of an object holding the interface for Interface Repository ID "IDL:Intfid1:1.0" are controlled in a POA object (RootPOA), and the instances of an object holding the interface for Interface Repository ID "IDL:intfid2:1.0" are controlled in a POA2 object (child or grandchild POA object).

While it is possible to control multiple objects in a RootPOA object, it becomes possible to control each interface with a different POA policy by constructing them like the ones above.

It would be convenient when constructing applications if you were to use POA object methods in the same manner instead of create_reference():

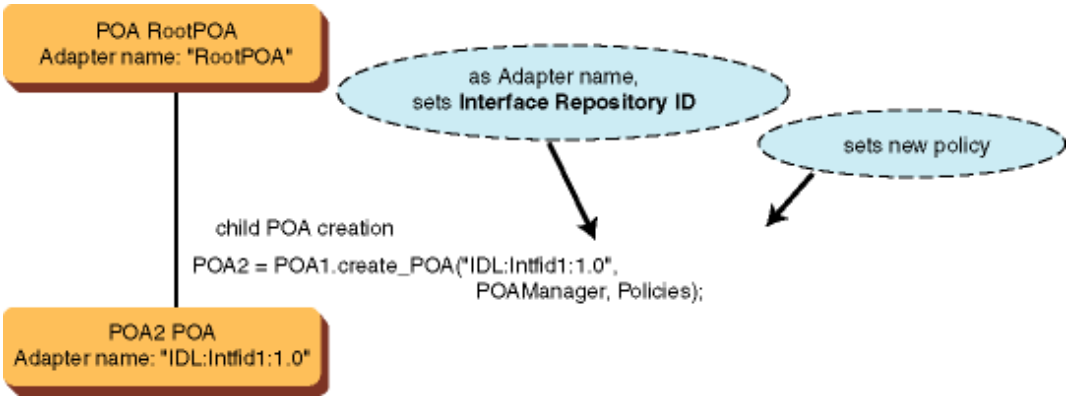
1. Creates POA (any adapter name).
2. Creates Servant.
3. Performs servant_to_reference(), and gets object references.
4. Performs registration (bind) of object references to NamingService.

As the result of servant_to_reference(), the registration (initialization) of the relationship of a Servant object and object ID is done and, in addition, object references including the object ID is returned. Since a Servant object includes an Interface Repository ID, it becomes unnecessary to specify it explicitly, as is the case when create_reference() was used. Additionally, since an object ID is automatically created at the time of initialization, there is no need to be aware of it (but you will need to specify IMPLICIT_ACTIVATION in POA policies).

POA Object Use Methods 2

This method (shown in the figure below) uses pre-creation with the environment variable OD_IMPLID.

Figure 6.41 POA Object Configuration Method 2



This method is employed in cases where the object reference creation and registration with a NamingService is done in advance using the OD_or_adm command. You will create a "child POA object" in order to control the object that holds the Interface Repository ID "IDL:Intfid1:1.0" interface. At this time, you will specify a row of characters (String object) the same as the Interface Repository ID.

Initialization of Servant objects is carried out as follows:

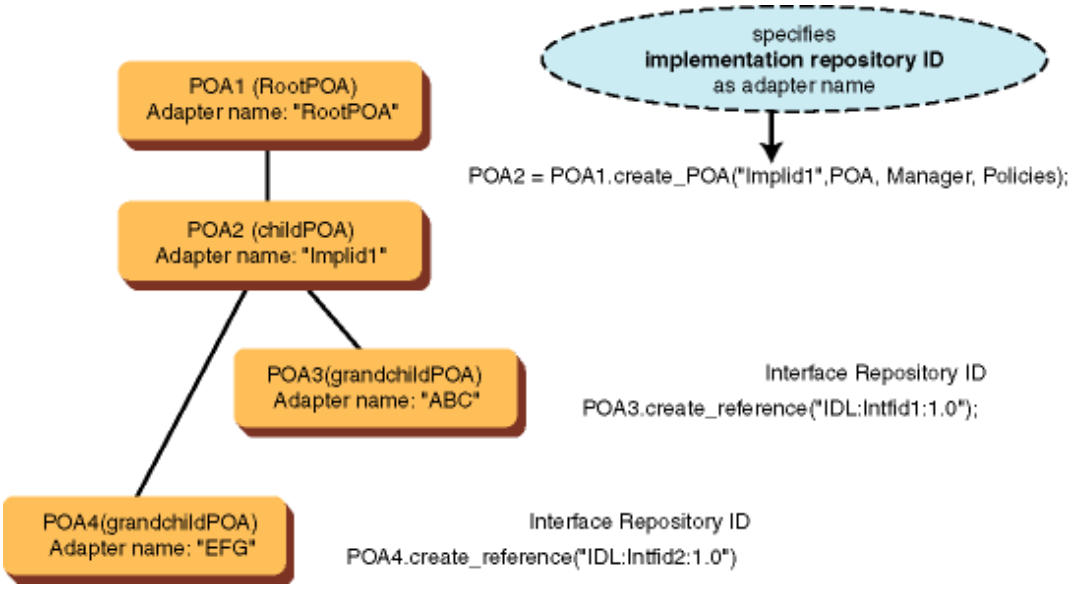
1. Create Servant object.
2. Run activate_object() on the POA with the adapter name for the Interface Repository ID corresponding to the Servant object.

Since the creation of object references and registration to a NamingService is done in advance with the OD_or_adm command, creation and registration in the application is not be necessary.

POA Object Use Method 3

This method (shown in the figure below) uses dynamic creation without the environment variable OD_IMPLID.

Figure 6.42 POA Object Configuration Method 3



In order to relate to implementation information, you will specify an implementation ID in Adapter Names and construct a child POA object that goes with the RootPOA object.

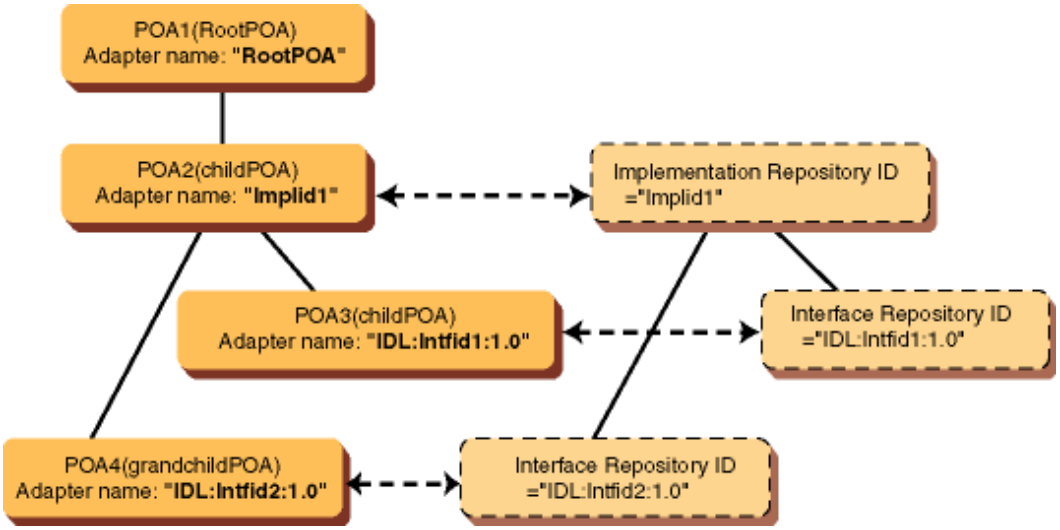
Next you will construct a child (or its descendant) POA object and then, using the create_reference() or servant_to_reference() methods in its POA object, construct object references.

After Servant object initialization, the object's instance control is done in these grandchild (or child) POA objects. When this method is used, you cannot use instance control in a RootPOA object.

POA Object Use Method 4

This method (shown in the figure below) uses pre-creation without the environment variable OD_IMPLID

Figure 6.43 POA Object Configuration Method 4



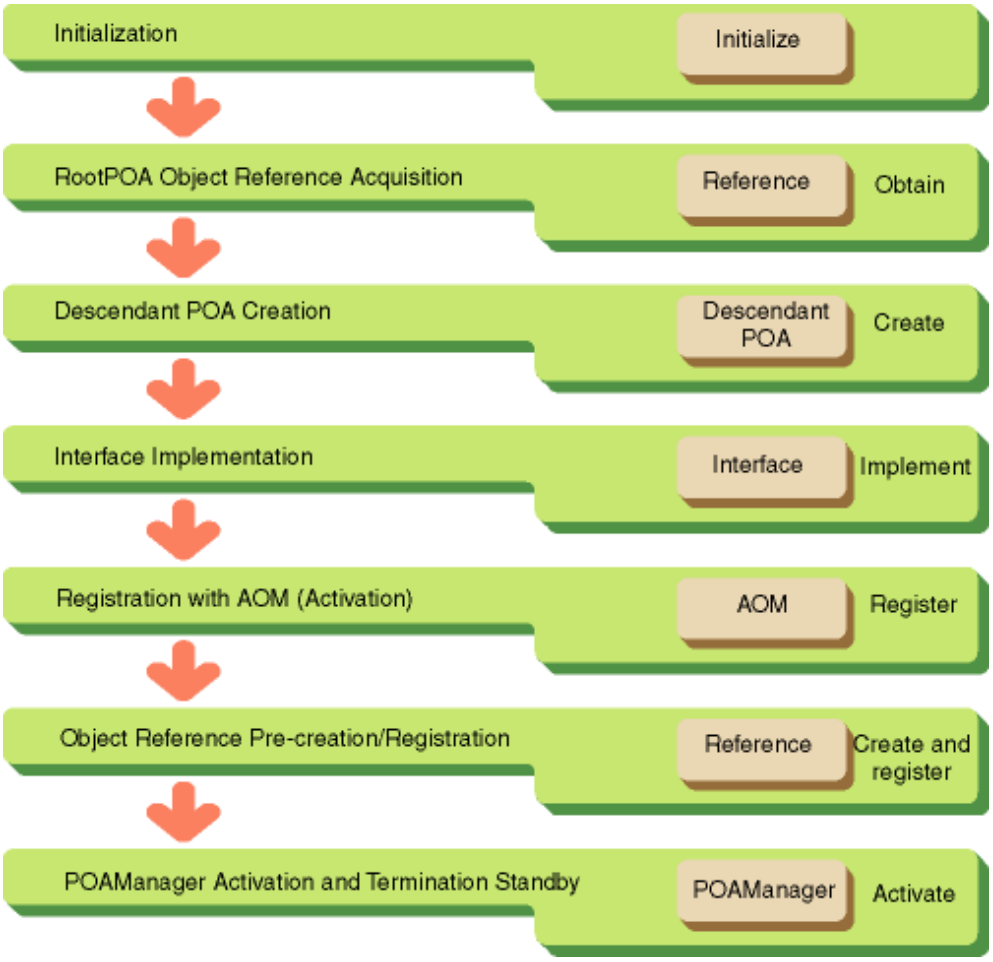
To relate to implementation information, you will create a POA object with 3 layers as shown on the left side of the Figure above. At this time the child POA object under the RootPOA object value (POA2) will specify and create an implementation policy ID in Adapter Name.

Then, the additional child POA object (grandchild POA: POA3, POA4) will specify and create an Interface Repository ID in Adapter Name. The result of this is that implementation information and interface information are related to a POA as in the above diagram. Instance control for the object holding the interface with repository ID "IDL:Intfid2:1.0" is performed by the POA4 object.

6.14 Programming Server Applications (Static Skeleton Interface)

The process flow for a server application in a case when a static launch interface is used is displayed below.

Figure 6.44 Server Application Process Flow



6.14.1 Initialization

Invoke the CORBA initialization method org.omg.CORBA.ORB.init() and run the initialization process. As the result of this method, object references for ORB are acknowledged. You will specify these object references in situations as below when you use the ORB interface for invoking.

```
public class Server { //class declaration
    public static void main( String args[] ) {
        org.omg.CORBA.ORB   Orb;     // ORB object references

        try {
            // Create and initialize ORB
            Orb = org.omg.CORBA.ORB.init( args, null );
              ...
        }
        catch( java.lang.Exception e ) {
          ...//exception handling
        }
    }
}
```

Refer to "6.15 Server Application Exception Handling" in regard to try-catch exception handling.

6.14.2 RootPOA Object Reference Acquisition

In order to utilize a POA object in server applications, you will request the RootPOA object references by the resolve_initial_references() method which fetches the object references for the initial service.

```
try {
    // RootPOA object reference acquisition
    org.omg.CORBA.Object  _tmpObj = Orb.resolve_initial_references( "RootPOA" );
    // get RootPOA POA objects
    org.omg.PortableServer.POA  rootPOA  = org.omg.PortableServer.POAHelper.narrow( _tmpObj );
}
catch( java.lang.Exception e ) {
      ... // exception handling
}
```

6.14.3 Descendant POA Creation

You create a parent POA in cases when you'd like to run instance control with a policy that's different from RootPOA or when you'd rather have instance control distributed over multiple POAs in each interface.

In the following example, a child POA is constructed using the acquired RootPOA method. You will specify the Adapter Name (the name of the POA you are creating) in the first parameter of create_POA(). The Adapter Name will specify any row of characters (String object). (As shown in "6.13 Relating Server Applications and Environment Settings", you will be able to specify an implementation repository ID and an Interface Repository ID.) You will specify the policy list that was in the application configuration to the Third parameter policy for create_POA(). In this example, a POA is constructed to utilize USE_DEFAULT_SERVANT.

```
try {
    // policy list creation
    org.omg.CORBA.Policy policies[] = new org.omg.CORBA.Policy[4];
    policies[0] = rootPOA.create_servant_retention_policy(
        org.omg.PortableServer.ServantRetentionPolicyValue.NON_RETAIN );
    policies[1] = rootPOA.create_request_processing_policy(
        org.omg.PortableServer.RequestProcessingPolicyValue.USE_DEFAULT_SERVANT );
    policies[2] = rootPOA.create_id_assignment_policy(
        org.omg.PortableServer.IdAssignmentPolicyValue.SYSTEM_ID );
    policies[3] = rootPOA.create_id_uniqueness_policy(
        org.omg.PortableServer.IdUniquenessPolicyValue.MULTIPLE_ID );

    // get POA object for corresponding interface policy ID
    org.omg.PortableServer.POA  POA1 =
        rootPOA.create_POA( "childPOA", RootPOA.the_POAManager(), policies );
}
catch( java.lang.Exception e ) {
      ... //exception handling
}
```

The procedure for creating a POA under descendants is the same. You can create a child POA by running the create_POA() method in the instances of the child POA (POA1 in the above example).

Multi-byte character languages such as Japanese cannot be used in the POA Adapter Name ("childPOA" in the above example).

6.14.4 Interface Implementation

The skeleton created by the IDL compiler (xxxPOA.java:xxx being the interface name) will construct the inherited Servant class. You will describe the implementation of operations in this class. You know that you can refer to the above description in the xxxOperations.java file (xxx being the interface name) for the post-Java mapping operation signature.

Note

This implementation method is based on Inheritance-Based Implementation. In a situation when you use Delegation-Based Implementation, refer to "6.16 Server Application Implementation Approaches".

IDL Definition

```
module ODsample{
    interface intf{
        long add( in long a, in long b );
    };
};
```

Servant Class

```
class UserServant extends intfPOA{
    public int add( int a, int b ) {
        return( a + b );
    }
}
```

6.14.5 Registration with AOM (Activation)

You create Servant objects and register them in the AOM of the POA. In the examples below, Servant objects are registered in AOM of the RootPOA. By using the servant_to_reference() method, you will register Servant objects while at the same time creating and acquiring the corresponding object references (when POA has a IMPLICIT_ACTIVATION policy).

Example

This is an example of servant object instance creation and registration to AOM.

```
try {

    // Servant creation
      Servant svt = new UserServant();

    // register Servant to AOM (activate), get object references
      org.omg.CORBA.Object c_obj = rootPOA.servant_to_reference( svt );
}
catch( java.lang.Exception e ) {
      ...  //exception handling
}
```

6.14.6 Registering with NamingService

You register the object references you created with a NamingService. This is unnecessary in a situation when object references are pre-created and registered with the OD_or_adm command.

Registration with a NamingService Example

```
import org.omg.CosNaming.*;
     :
try {
    // get NamingService object references
    NSObj = Orb.resolve_initial_references( "NameService" );
    NamingContextExt Cos  = NamingContextExtHelper.narrow( NSObj );
    String NCid   = new String( "test" );    // object name
    String NCkind = new String( "" );        // object type
    NameComponent nc =  new NameComponent( NCid, NCkind );
    NameComponent NCo[] = { nc };
    try {
        Cos.unbind( NCo ); //delete if there are old object references under the same name
    }
    catch( java.lang.Exception e ) {}
    Cos.bind( NCo, c_obj ); //object references registration
}
catch( java.lang.Exception e ) {
      ...  //exception handling
}
```

6.14.7 Object Reference Pre-creation/Registration

The creation and registration with a NamingService of object references is done in an application in the above example, but you may also create object references and register them in advance by using the OD_or_adm CORBA Service control command. In this case, creation and registration in an application becomes unnecessary. (You would use POA constructions "POA Construction Method 2" or "POA Construction Method 4.")

Example

OD_or_adm Usage Example

```
OD_or_adm -c IDL:ODsample/intf:1.0  -a Imple_sample1  -n ODsample::sample1
```

-c IDL:ODsample/intf:1.0

Registers the object references with the specified Interface Repository ID.

-a Imple_sample1

Registers the object references with the specified implementation repository ID.

-n ODsample::sample1

Specifies the object name to be registered in the NamingService.

6.14.8 POAManager Activation and Termination Standby

By running activate() for the related POAManager in POA, your configuration goes into standby for a reception from the client. You will do the ORB-class run() method immediately after running activate(). By doing this, the server application goes into standby configuration without terminating.

[POAManager activation]

```
try {
    // activate POAManager
    rootPOA.the_POAManager().activate();
    Orb.run();
}
```

```
catch( java.lang.Exception e ) {
      ... // exception handling
}
```

Note

It is never into run() method because it is into standby for a termination when issuing activate(), and run() method does nothing. But it is recommended that run () method is described considering of application portability.

6.15 Server Application Exception Handling

This section contains information on server application exception handling.

Setting Exception Information

The following is an example server application program based on the IDL-language definitions shown:

IDL Mapping

IDL Language

```
module ODsample {
    interface  exptest{
        exception testException { string reason; };
        void    op1() raises( testException );
    };
};
```

Written in Java language, this comes out as follows:

Java Language

<Interface>

```
package ODsample;
public interface exptestOperations{
    public void op1()
        throws ODsample.exptestPackage.testException;
}
```

<User Exception Class>

```
package ODsample.exptestPackage;
public final class testException
    extends org.omg.CORBA.UserException
{
    public java.lang.String reason;
    public testException(){}
    public testException( java.lang.String _reason ){
        reason = _reason;
    }
}
```

Server Application Processing

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

public class expServer {
    public static void main( String args[] ) {
        try {
            // ORB pro-processing
                ...
            // create POA object
                ...
            // Servant creation and registering it to POA
                ...
            // POAManager activation
                ...
        }
        catch ( java.lang.Exception e ) {
            //error processing
                ...
        }
    }
}

// Servant class
// server application method
class expServant extends exptestPOA{
    public void op1() throws ODsample.exptestPackage.testException  {
        throw new ODsample.exptestPackage.testException( "Test of UserException" );
    }
}
```

Obtaining Exception Information

Obtain exception information for server applications in the same way as for client applications. Refer to "6.9 Exception Handling for Client Applications" for details.

6.16 Server Application Implementation Approaches

There are two approaches to implementing a server application with a static skeleton interface:

- Inheritance-Based Implementation
- Delegation-Based Implementation.

We will explain here these two implementation methods and their characteristics.

6.16.1 Inheritance-based and Delegation-based Implementation

This section provides an overview of these two methods.

Inheritance-based Implementation

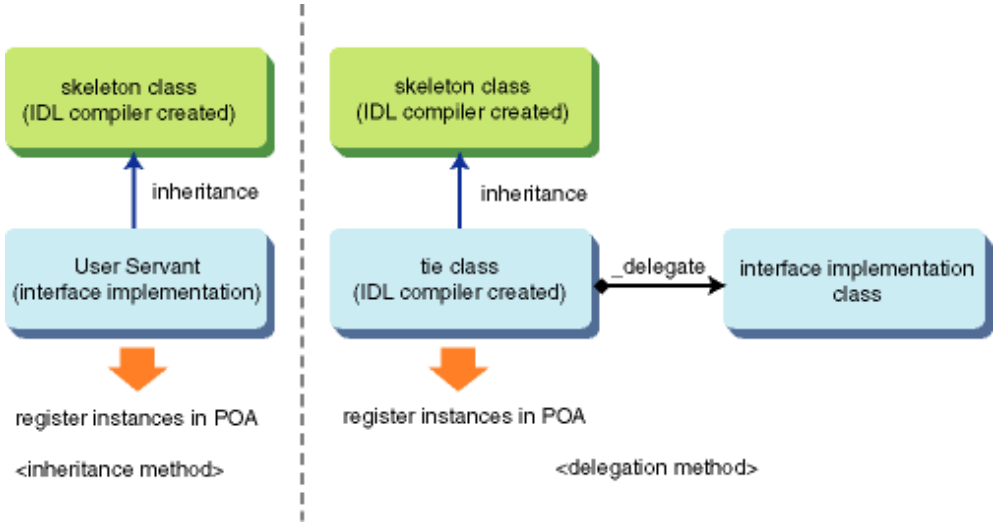
This is a typical implementation method used for creating new server applications. The skeleton class inherits from the Servant. The result of this is that necessary functions are inherited by Servant at the time of launch. It registers Servant itself in respect to POA.

Delegation-based Implementation

This is an effective method for situations in which you would prefer to use preexisting user-developed classes. A tie class is provided as the POA registration object that becomes a Servant (created as <interface name>POATie.java with an IDL compiler).

The instances of a user-constructed interface implementation class are registered as a member object in a tie class. Operation launch is carried out in part via the tie class in response to the registered class instances. The following figure shows inheritance-based and delegation-based methods.

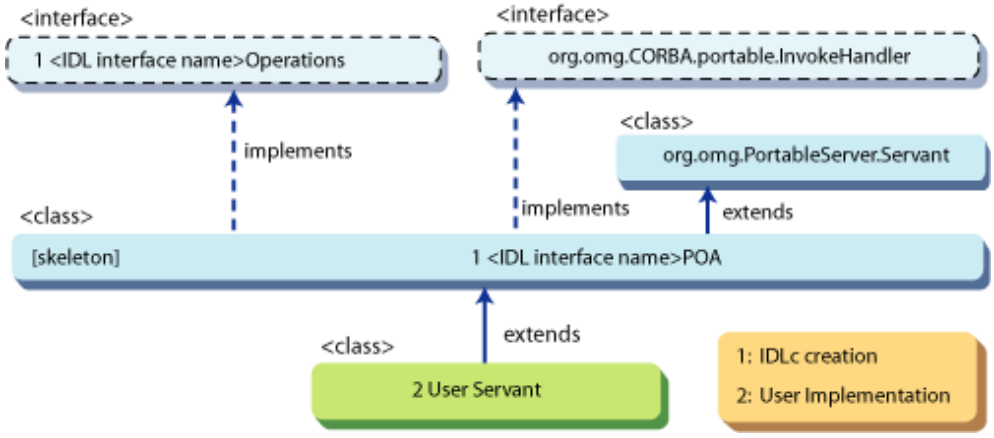
Figure 6.45 Inheritance-Based and Delegation-Based Methods



6.16.2 Inheritance-based Method Servant Implementation

A Servant set up on an Inheritance basis has inheritance relationships like those shown below. This Servant is registered in POA by the server application's main section.

Figure 6.46 Inheritance-based Method Class Inheritance Relations



As displayed, with the inheritance-based method, the Servant implements in a format that inherits the skeleton class in the following manner.

IDL Definition

```
module ODsample{
    interface   intf{
        long    add(in long a, in long b);
    };
};
```

Servant Example

```
class UserServant extends ODsample.  intfPOA{
    public int add( int a, int b ) {
        return( a + b );
    }
}
```

This Servant is registered in POA with the main processor. In the following example, it is registered as default Servant.

Example of Main

```
// Servant creation
Servant svt = new UserServant();
// set as default Servant
aPOA.set_servant( svt );
```

Note

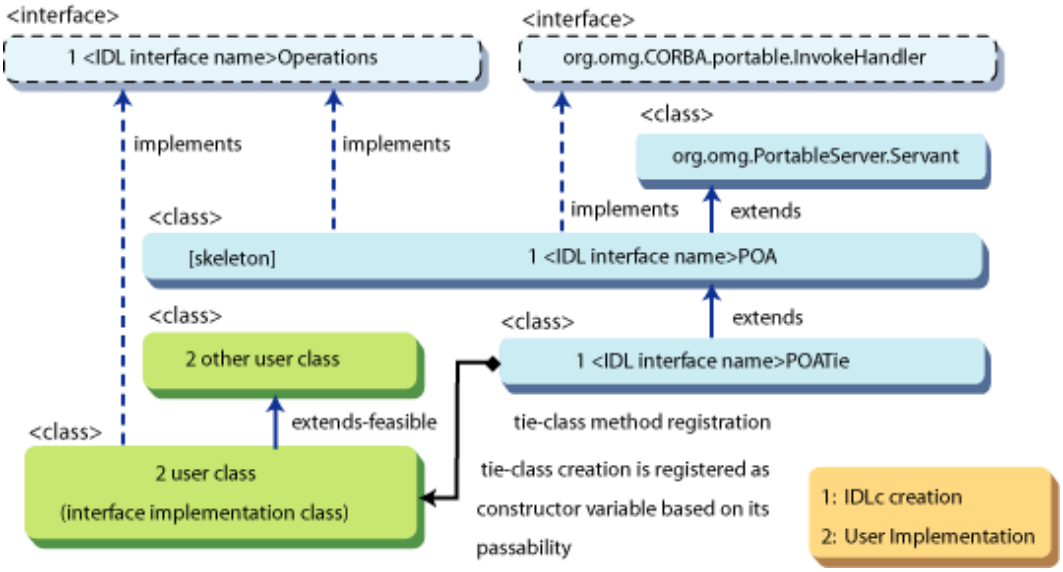
aPOA is a POA class instance.

6.16.3 Delegation-based Method Servant Implementation

A user class (interface implementation class) set up on the delegation-based method is implemented using the following types of inheritance relationships. In the server application's main section, it registers the tie-class instances (<interface name>POATie) in POA. Since it is opened from normal inherited relationships, similar to those used in an Inheritance-Based Method, the degree of freedom in implementation is enhanced. But, the interface implementation class needs to implement the <interface name>Operation interface in order to register as a tie-class method. The figure below shows class inheritance relations in the delegation-based method

Refer to "6.18 Programming Examples of Server Applications" for example applications of the Delegation-Based Method.

Figure 6.47 Class Inheritance Relations in Delegation-based Method



As in the figure above, with the a Delegation-Based Method, the user class is described as follows to implement the <interface name>Operation interface.

IDL Definition

```
module ODsample{
    interface   intf{
```

```
        long    add(in long a, in long b);
    };
};
```

User Class Example

```
class UserClass extends OtherUserClass
implements ODsample.intfOperations
{
    public int add( int a, int b ){
        return( a + b );
    }
}
```

With the main processor, register this user class interface in part in the tie-class instances (passing user-class instances as tie-class constructor attributes). Additionally, register the tie-class instances as Servant in POA. In the following example, they are registered as default Servant.

Example of Main Section

```
// UserClass instance creation
UserClass uc = new UserClass();
// create tie object and register UserClass
intfPOATie  tie = new intfPOATie( uc );
// set in POA default Servant
aPOA.set_servant( tie );
```

Note

aPOA is a POA-class instance.

6.16.4 A Comparison of Inheritance-based and Delegation-based Methods

The following table shows the characteristics of inheritance-based and delegation-based Methods.

In the settings for server applications, it is up to the user which of the implementation methods to use.

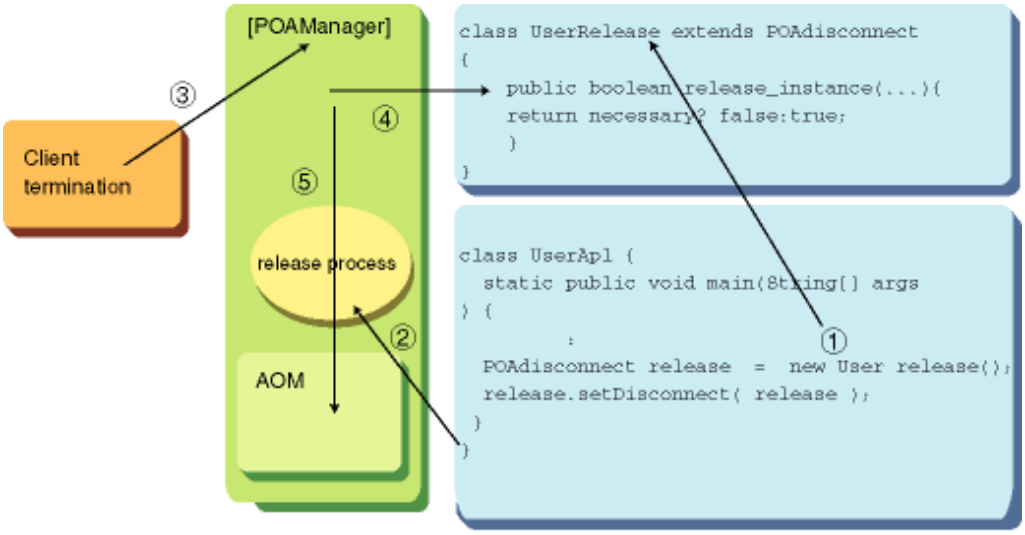
Table 6.12 Comparisons of Inheritance- & Delegation-based Methods

	Inheritance-Based Method	Delegation-Based Method
Principal Applications	new development	utilization of existing resources
Degree of freedom in implementation	Poor. Servant inherits the skeleton, and cannot inherit other classes.	Very Good. With no need for the interface implementation class to inherit a skeleton, it can inherit other classes.
Coding complexity	Relatively simple.	Can be complex.
Performance	As the required skeleton function is inherited by the Servant itself at time of launch, it is comparatively good.	As operation launch passes through tie class, the object references expand. So some overhead is likely.

6.17 Instance Analysis at Client Termination

We will now explain methods for analyzing unnecessary instances on a server application when its client application shuts down. This function utilizes the CORBA Service's instance control feature and may be used only in cases when POA is running instance control using the ActiveObjectMap. The following figure shows instance analysis methods.

Figure 6.48 Instance Analysis Methods



1. Create instances for Server instance analysis class.
2. Register the created instances.
3. Termination alert is received from client.
4. Inquire about propriety of Servant instance
5. When servant instances are not required, delete from AOM.

6.17.1 Constructing a Class to Run Instance Analysis Processing

Construct a class to run instance analysis at the time of client application termination. This class is created by inheriting the com.fujitsu.ObjectDirector.PortableServer.POAdisconnect class

6.17.2 Implementing Instance Analysis Processing

At the time a client application is terminated, a release instance() method for the class that was created is invoked. The release_instance() method for the new class (POAdisconnect class) usually returns false (do not analyze).

When the release_instance() method is invoked, the POA and server instances as well as the object ID used by terminated client are reported as parameters. Based on this information, the alerted Servant, with the overwritten release_instance() method, determines where they are necessary or not and returns true (analyze) or false (do not analyze).

6.17.3 Registering Class Instances for Running Instance Analysis Processing

Register the user-created instance analysis class in POAManager. Registration invokes the static method setDisconnect() of the POAdisconnect class.

6.17.4 Modifying/Deleting Class Instances for Running Instance Analysis Processing

Invoke the static method resetDisconnect() of the POA disconnect class in situations where you modify or delete the previously registered instance class.

6.18 Programming Examples of Server Applications

This section provides server application programming examples.

6.18.1 Default Servant (Default Instances Method)

This section provides a default instances method example.

IDL Definition

Example

```
module ODsample{
    interface  intf{
        long  add( in  long  a,  in  long  b );
    };
};
```

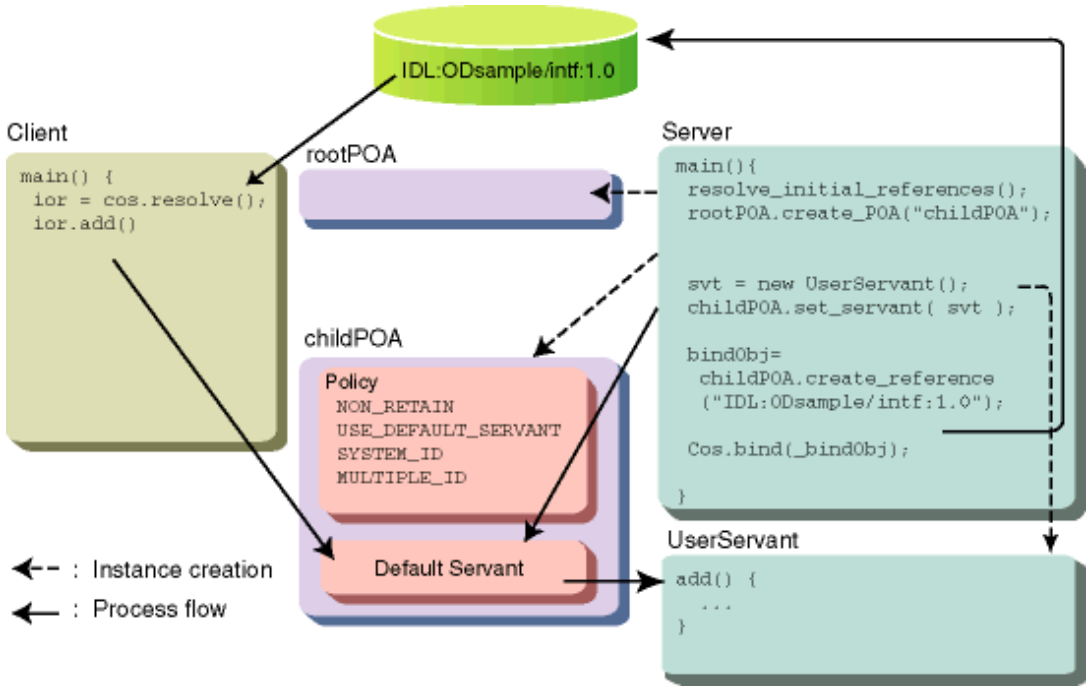
NamingService Registration

Dynamic registration.

Application Construction Overview

The Figure below outlines the default instances method application construction process.

Figure 6.49 Application Construction for Default Instances Method



Client Applications

Example

```
import java.io.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import ODsample.*;
public class Client {
    public static void main( String args[] ) {

    try {
```

```
        // create and initialize ORB
        ORB Orb = ORB.init( args, null );

        // get NamingService object references
        org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "NameService" );
        NamingContextExt Cos = NamingContextExtHelper.narrow( _tmpObj );

        // run NamingService resolve method
        // get server application object references
        String NCid = new String( "ODsample::POAsample1" ); // object name
        String NCkind = new String( "" );   // object type
        NameComponent nc =  new NameComponent( NCid, NCkind );
        NameComponent NCo[] = { nc };
        org.omg.CORBA.Object  Obj = Cos.resolve( NCo );

        // get server application object references
        intf target = intfHelper.narrow( Obj );

        int  in1 = 0;     // in-parameter attribute
        int  in2 = 0;     // in-parameter attribute
        int  result;      // multiple-value attribute
        String line = null;

        try {
            System.out.print( "in1 => " );
            line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
            in1 = Integer.parseInt( line );

            System.out.print( "in2 => " );
            line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
            in2 = Integer.parseInt( line );
        }
        catch ( java.lang.NumberFormatException e ) {
            System.exit( 255 );
        }

        // invoke server application method
        result = target.add( in1, in2 );

        // display method results
        System.out.println( in1 + " + " + in2 + " = "+ result );
    }
    catch ( Exception e ) {
        System.err.println( "\nERROR : " + e );
        System.exit( 255 );
    }
    }
}
```

Server Applications

Example

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.CosNaming.*;
import ODsample.*;

// user application: main process class
public class Server {
    public static void main( String args[] ) {

        try {
```

```
            // create and initialize ORB
            ORB Orb = ORB.init( args, null );

            // get RootPOA object references
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "RootPOA" );
            // get POA objects of RootPOA
            POA rootPOA = POAHelper.narrow( _tmpObj );

            // construct interface POA
            // construct policy list
            org.omg.CORBA.Policy policies[] = new org.omg.CORBA.Policy[4];
            policies[0] = rootPOA.create_servant_retention_policy(
                            ServantRetentionPolicyValue.NON_RETAIN );
            policies[1] = rootPOA.create_request_processing_policy(
                            RequestProcessingPolicyValue.USE_DEFAULT_SERVANT);
            policies[2] = rootPOA.create_id_assignment_policy(
                            IdAssignmentPolicyValue.SYSTEM_ID );
            policies[3] = rootPOA.create_id_uniqueness_policy(
                            IdUniquenessPolicyValue.MULTIPLE_ID );

            // create POA objects for default Servant
            POA childPOA = rootPOA.create_POA( "childPOA", null, policies );

            // create Servant
            Servant svt = new UserServant();
            // set in default Servant
            childPOA.set_servant( svt );

            org.omg.CORBA.Object _bindObj =
                        childPOA.create_reference( "IDL:ODsample/intf:1.0" );

            // get object references of NamingService
            _tmpObj = Orb.resolve_initial_references( "NameService" );
            NamingContextExt Cos = NamingContextExtHelper.narrow(_tmpObj);

            // run NamingService resolve method,
            // get object references of server application
            String NCid   =
                 new String( "ODsample::POAsample1" ); // object name
            String NCkind = new String( "" );  // object type
            NameComponent nc =  new NameComponent( NCid, NCkind );
            NameComponent NCo[] = { nc };
            try {
                Cos.unbind( NCo );
            } catch( Exception e ) {
                  ;
            }
            Cos.bind( NCo, _bindObj );

            // get POAManager
            POAManager poamanager = childPOA.the_POAManager();
            // activate POAManager
            poamanager.activate();
            Orb.run();
        }
        catch ( Exception e ) {
            System.err.println( "\nERROR: " + e );
            System.exit( 255 );
        }
    }
}

// Servant method implementation class (inherit skeleton)
```

```
class UserServant extends intfPOA {
    public int add( int a, int b ) {
        return( a + b );
    }
}
```

Obtaining Exception Information

Obtain exception information for server applications in the same way as for client applications. Refer to "6.9 Exception Handling for Client Applications" for details.

6.18.2 Active Object Map (AOM) Sample Usage (Factory-1 Method)

This section provides an AOM Factory-1 Method example.

IDL Definition

Example

```
module ODsample{
    interface intf{
        readonly  attribute  long  value;
        void  add( in  long  a );
        void  sub( in  long  b );
    };
    interface Factory {
        intf   create();
        void   destroy( in  intf  obj );
    };
};
```

NamingService Registration

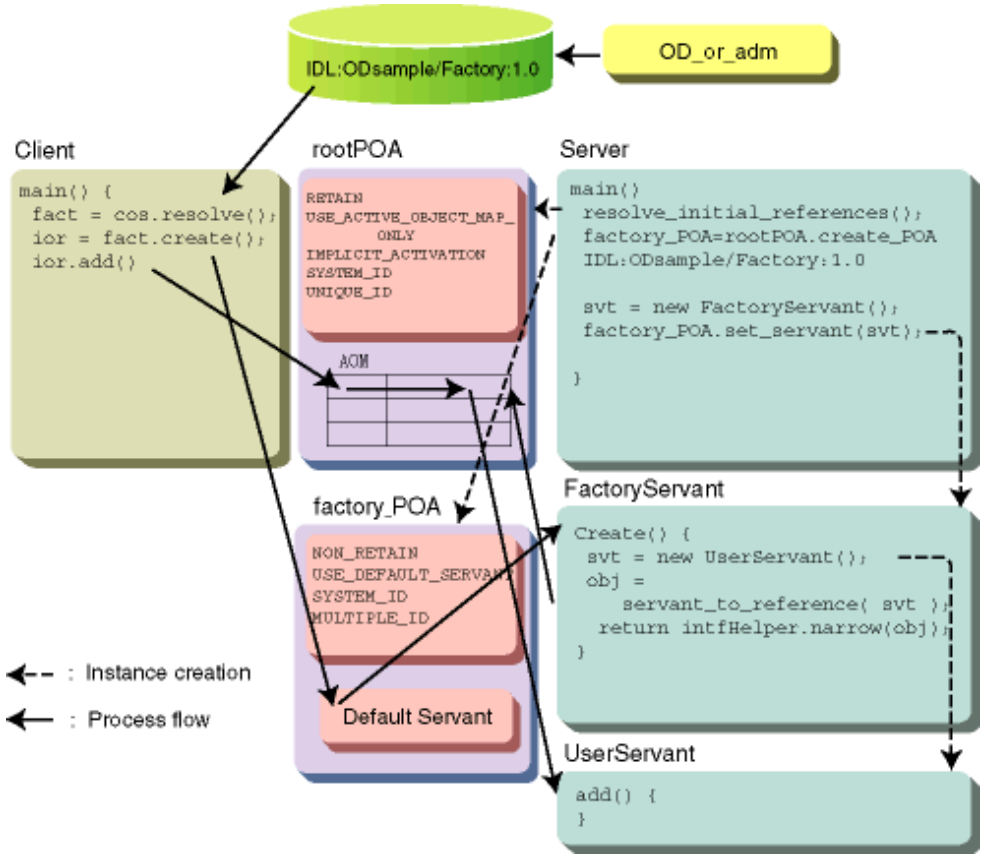
Example

```
OD_or_adm -c IDL:ODsample/Factory:1.0  -n ODsample::POAsample5
```

Application Construction Overview

The figure below outlines the application construction process for Factory-1 method application.

Figure 6.50 Block Diagram for Factory-1 Method Applications



Client Applications

Example

```
import java.io.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import ODsample.*;

public class Client {
    public static void main( String args[] ) {
        try {
            // create and initialize ORB
            ORB Orb = ORB.init( args, null );

            // get NamingService object references
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "NameService" );
            NamingContextExt Cos = NamingContextExtHelper.narrow( _tmpObj );

            // run NamingService resolve method
            // get server application object references
            String NCid   = new String( "ODsample::POAsample5" ); // object name
            String NCkind = new String( "" );                     // object type
            NameComponent nc =  new NameComponent( NCid, NCkind );
            NameComponent NCo[] = { nc };
            org.omg.CORBA.Object  Obj = Cos.resolve( NCo );

            // get Factory object references
            Factory target = FactoryHelper.narrow( Obj );

            // create interface object references with Factory
```

```
            intf _intf = target.create();

            int    in   = 0;    // in-parameter attribute
            String line = null;

            try {
                // invoke server application method
                System.out.println( "value = " + _intf.value() );

                System.out.print( "add => " );
                line = new BufferedReader( new InputStreamReader( System.in )
            ).readLine();
            in = Integer.parseInt( line );

            // invoke server application method
            _intf.add( in );
            System.out.println( "value = " + _intf.value() );

                System.out.print( "sub => " );
                line = new BufferedReader( new InputStreamReader( System.in )
                ).readLine();
                in = Integer.parseInt( line );

                // invoke server application method
                _intf.sub( in );
                System.out.println( "value = " + _intf.value() );

                // analyze instances
                target.destroy( _intf );
            }
            catch ( java.lang.NumberFormatException e ){
                System.exit(255);
            }
        }
        catch ( Exception e ) {
            System.err.println( "\nERROR : " + e );
            System.exit(255);
        }
    }
}
```

Server Applications

Example

```
import org.omg.CORBA.*;
    import org.omg.PortableServer.*;
    import ODsample.*;

    // user application process class
    public class Server {
    public static void main( String args[] ) {

        try {
            // create and initialize ORB
            ORB Orb = ORB.init( args, null );

            // get RootPOA object references
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "RootPOA" );
            // get POA objects of RootPOA
            POA rootPOA  = POAHelper.narrow( _tmpObj );

            // create POA for Factory interface
```

```
            // construct policy list
            org.omg.CORBA.Policy factory_policies[] = new org.omg.CORBA.Policy[4];
            factory_policies[0] = rootPOA.create_servant_retention_policy(
                                  ServantRetentionPolicyValue.NON_RETAIN );
            factory_policies[1] = rootPOA.create_request_processing_policy(
                                  RequestProcessingPolicyValue.USE_DEFAULT_SERVANT);
            factory_policies[2] = rootPOA.create_id_assignment_policy(
                                  IdAssignmentPolicyValue.SYSTEM_ID );
            factory_policies[3] = rootPOA.create_id_uniqueness_policy(
                                  IdUniquenessPolicyValue.MULTIPLE_ID );
            POA factory_POA = rootPOA.create_POA( "IDL:ODsample/Factory:1.0",
                                                  null,
                                                  factory_policies );

            // create FactoryServant
            Servant svt = new FactoryServant( rootPOA );
            // set POA default Servant for Factory interface
            factory_POA.set_servant( svt );

            // get POAManager
            POAManager poamanager = rootPOA.the_POAManager();

            // activate POAManager
            poamanager.activate();
            Orb.run();
        }
        catch ( Exception e ) {
            System.err.println( "\nERROR: " + e );
            System.exit( 255 );
        }
    }
}

// FactoryServant Factory method implementation class (inherit skeleton)
class FactoryServant extends FactoryPOA {

    private POA poa = null;

    // constructor
    public FactoryServant( POA poa ) {
        this.poa = poa;
    }

    public intf create() {
        intf       ior;           // UserServant object references

        try {
            // create Servant
            Servant svt = new UserServant();

            // create object references from Servant
            // due to IMPLICIT_ACTIVATION policy is automatically registered to AOM
            org.omg.CORBA.Object _tmpObj = this.poa.servant_to_reference( svt );
            ior = intfHelper.narrow( _tmpObj );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.err.println( "create error: " + e );
            return( null );
        }

        return( ior );

    }
```

```
    public void destroy( intf obj ) {
        try {
            // request ObjectID from object references
            byte oid[] = this.poa.reference_to_id( obj );
            // deactivate Servant
            this.poa.deactivate_object( oid );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.err.println( "destroy error: " + e );
        }
    }
}

// Servant method implementation class (inherit skeleton)
class UserServant extends intfPOA {

    private int    value = 0;

    public int value() {
        return( this.value );
    }

    public void add( int a ) {
        this.value += a;
    }

    public void sub( int b ) {
        this.value -= b;
    }
}
```

Obtaining Exception Information

Obtain exception information for server applications in the same way as for client applications. Refer to "6.9 Exception Handling for Client Applications" for details.

6.18.3 Servant Activator Sample Usage (Factory-2 Method)

This section provides a servant activator Factory-2 Method example.

IDL Definition

Example

```
module ODsample{
    interface intf{
        readonly  attribute  long  value;
        void  add( in  long  a );
        void  sub( in  long  b );
    };
    interface Factory {
        intf   create();
        void   destroy( in  intf  obj );
    };
};
```

NamingService Registration

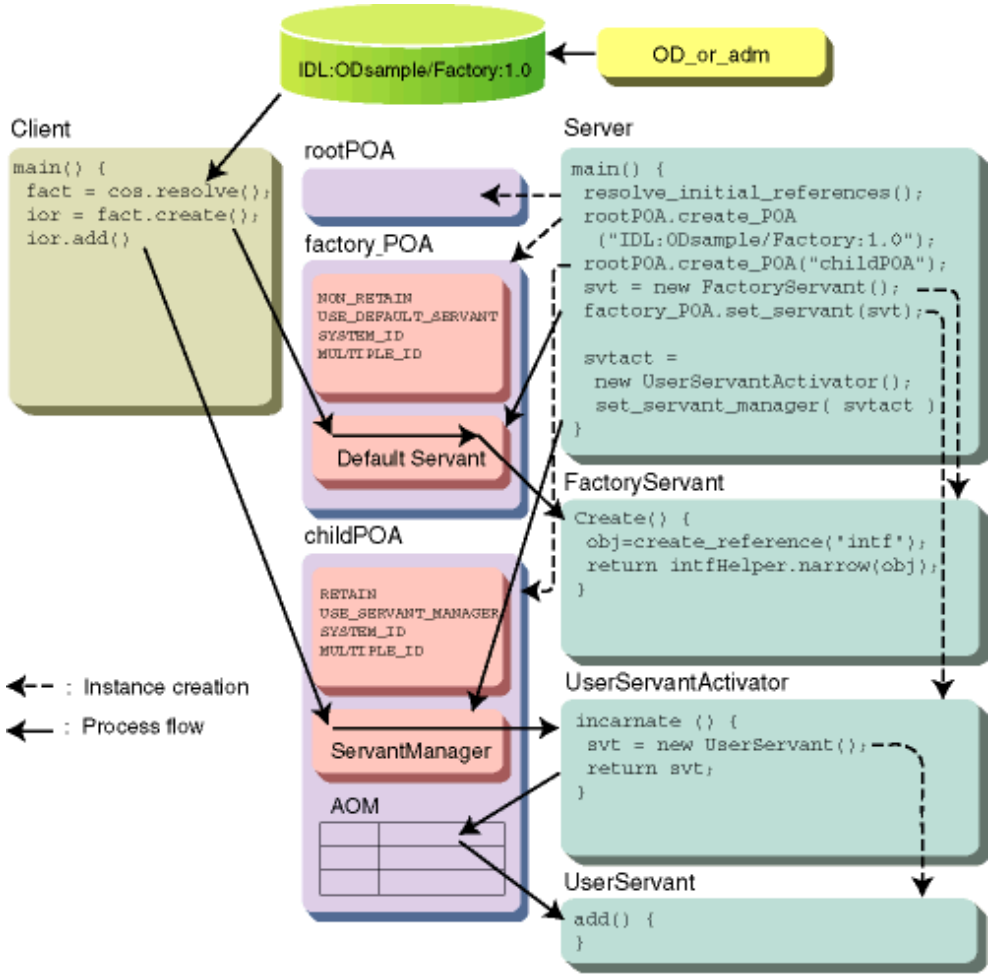
Example

```
OD_or_adm -c IDL:ODsample/Factory:1.0  -n ODsample::POAsample3
```

Application Construction Overview

The figure below outlines the Factory-2 method application construction process.

Figure 6.51 Block Diagram for Factory-2 Method Application



Client Application

Example

```
import java.io.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import ODsample.*;

public class Client {
    public static void main(String args[]) {
        try {
            // create and initialize ORB
            ORB Orb = ORB.init( args, null );

            // get NamingService object references
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "NameService" );
            NamingContextExt Cos = NamingContextExtHelper.narrow( _tmpObj );
```

```
            // run NamingService resolve method,
            // get server application object references
            String NCid   = new String( "ODsample::POAsample3" ); // object name
            String NCkind = new String( "" );  // object type
            NameComponent nc =  new NameComponent( NCid, NCkind );
            NameComponent NCo[] = { nc };
            org.omg.CORBA.Object  Obj = Cos.resolve( NCo );

            // get Factory object references
            Factory target = FactoryHelper.narrow( Obj );

            // create object references for interface with Factory
            intf _intf = target.create();

            int    in   = 0;    // in-parameter attribute
            String line = null;

            try {
                // invoke server application method
                System.out.println( "value = " + _intf.value() );

                System.out.print( "add => " );
                line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
                in = Integer.parseInt( line );

                // invoke server application method
                _intf.add( in );
                System.out.println( "value = " + _intf.value() );

                System.out.print( "sub => " );
                line = new BufferedReader( new InputStreamReader ( System.in ) ).readLine();
                in = Integer.parseInt( line );

                // invoke server application method
                _intf.sub( in );
                System.out.println( "value = " + _intf.value() );

                // analyze instances
                target.destroy( _intf );
            }
            catch ( java.lang.NumberFormatException e ) {
                System.exit( 255 );
            }
        }
        catch ( Exception e ) {
            System.err.println( "\nERROR : " + e );
            System.exit( 255 );
        }
    }
}
```

Server Application

Example

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// server application process class
public class Server {
    public static void main( String args[] ) {
```

```
    try {
        // create and initialize ORB
        ORB Orb = ORB.init( args, null );

        // get RootPOA object references
        org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "RootPOA" );
        // get POA object of RootPOA
        POA rootPOA = POAHelper.narrow( _tmpObj );

        // create POA for Factory interface
        // construct policy list
        org.omg.CORBA.Policy factory_policies[] = new org.omg.CORBA.Policy[4];
        factory_policies[0] = rootPOA.create_servant_retention_policy(
                              ServantRetentionPolicyValue.NON_RETAIN );
        factory_policies[1] = rootPOA.create_request_processing_policy(
                              RequestProcessingPolicyValue.USE_DEFAULT_SERVANT );

        factory_policies[2] = rootPOA.create_id_assignment_policy(
                              IdAssignmentPolicyValue.SYSTEM_ID );
        factory_policies[3] = rootPOA.create_id_uniqueness_policy(
                              IdUniquenessPolicyValue.MULTIPLE_ID );
        POA factory_POA = rootPOA.create_POA( "IDL:ODsample/Factory:1.0",
                                              null,
                                              factory_policies );

        // create POA for interface
        // construct policy list
        org.omg.CORBA.Policy inf_policies[] = new org.omg.CORBA.Policy[4];
        inf_policies[0] = rootPOA.create_servant_retention_policy(
                              ServantRetentionPolicyValue.RETAIN );
        inf_policies[1] = rootPOA.create_request_processing_policy(
                              RequestProcessingPolicyValue.USE_SERVANT_MANAGER );

        inf_policies[2] = rootPOA.create_id_assignment_policy(
                                IdAssignmentPolicyValue.SYSTEM_ID );
        inf_policies[3] = rootPOA.create_id_uniqueness_policy(
                                IdUniquenessPolicyValue.MULTIPLE_ID );
        POA childPOA = rootPOA.create_POA( "childPOA", null, inf_policies );

        // create FactoryServant
        Servant svt = new FactoryServant( childPOA );
        // set default Servant for Factory interface POA
        factory_POA.set_servant( svt );

        // create ServantActivator
        ServantActivator svtact = new UserServantActivator();
        // register as ServantManager for interface POA
        childPOA.set_servant_manager( svtact );

        // get POAManager
        POAManager poamanager = rootPOA.the_POAManager();

        // activate POAManager
        poamanager.activate();
        Orb.run();
    }
    catch ( Exception e )
    {
        System.err.println( "\nERROR: " + e );
        System.exit( 255 );
    }
}
```

```
}

// FactoryServant Factory method implementation class (inherit skeleton)
class FactoryServant extends FactoryPOA {
    private POA poa = null;

    // constructor
    public FactoryServant( POA poa ) {
        this.poa = poa;
    }

    public intf create() {
        intf       ior;        // UserServant object references

        try {
            // create object references
            org.omg.CORBA.Object _tmpObj = this.poa.create_reference( "IDL:ODsample/intf:1.0" );
            ior = intfHelper.narrow( _tmpObj );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.err.println( "create error: " + e );
            return( null );
        }

        return( ior );

    }

    public void destroy( intf obj ) {
        try {
            // request objectID from object references
            byte oid[] = this.poa.reference_to_id( obj );

            // deactivate Servant
            this.poa.deactivate_object( oid );
        }
        catch( org.omg.CORBA.UserException e ) {
              System.err.println( "destroy error: " + e );
        }
    }
}

// ServantActivator Servant create class inherit ServantActivator
class UserServantActivator extends LocalObject implements ServantActivator {
    public Servant incarnate( byte[] oid,
                              POA adapter ) {

        // create Servant
        Servant svt = new UserServant();
        return( svt );
    }

    public void etherealize( byte[] oid,
                             POA adapter,
                             Servant serv,
                             boolean cleanup_in_progress,
                             boolean remaining_activations ) {

        // run attribute initialization.
        serv = null;

    }
}
```

```
// Servant method implementation class (inherit skeleton)
class UserServant extends intfPOA {

    private int    value = 0;

    public int value() {
        return( this.value );
    }

    public void add( int a ) {
          this.value += a;
    }

    public void sub( int b ) {
        this.value -= b;
    }

}
```

Obtaining Exception Information

Obtain exception information for server applications in the same way as for client applications. Refer to "6.9 Exception Handling for Client Applications" for details.

6.18.4 Servant Locator Sample Usage (User Instance Control Method)

This section provides a User Instance Control Method example.

IDL Definition

Example

```
Module ODsample{
    interface    intf{
        readonly  attribute  long  value;
        void add( in  long  a );
        void sub( in  long  b );
    };
    interface Factory{
        intf   create( in  string  userid );
        void   destroy( in  intf  obj );
    };
};
```

NamingService Registration

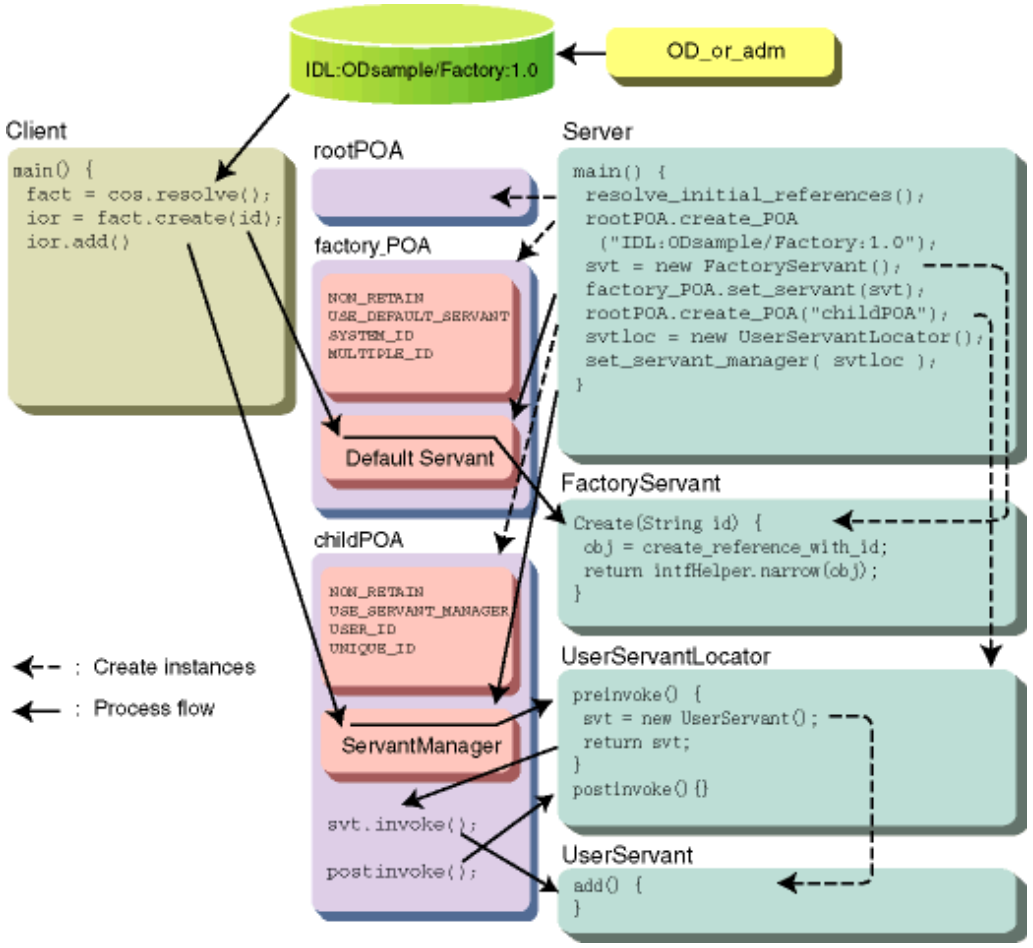
Example

```
OD_or_adm -c IDL:ODsample/Factory:1.0 -n ODsample::POAsample4
```

Application Construction Overview

The figure below outlines the user instance control method application construction process.

Figure 6.52 Block Diagram for User Interface Control Method Application



Client Application

Example

```
import java.io.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import ODsample.*;

public class Client {
    public static void main( String args[] ) {

        try {
            // create and initialize ORB
            ORB Orb = ORB.init( args, null );

            // get NamingService object references
            org.omg.CORBA.Object _tmpObj =
                          Orb.resolve_initial_references( "NameService" );
            NamingContextExt Cos = NamingContextExtHelper.narrow( _tmpObj );

            // run NamingService resolve method,
            // get server application object references
            String NCid   = new String( "ODsample::POAsample4" ); // object name
            String NCkind = new String( "" );         // object type
            NameComponent nc =  new NameComponent( NCid, NCkind );
            NameComponent NCo[] = { nc };
            org.omg.CORBA.Object  Obj = Cos.resolve( NCo );
```

```
            // get server application object references
            Factory target = FactoryHelper.narrow( Obj );

            int  in;
            String line = null;
            String userid = null;

            try {
                System.out.print( "USERID => " );
                userid = new BufferedReader( new InputStreamReader( System.in )).readLine();
                if( userid.length() == 0 )
                    System.exit( 255 );

                // create interface abbreviation IOR with Factory
                intf _intf = target.create( userid );

                // invoke server application method
                System.out.println( "value = " + _intf.value() );

                System.out.print( "add => " );
                line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
                in = Integer.parseInt( line );

                // invoke server application method
                _intf.add( in );
                System.out.println( "value = " + _intf.value() );

                System.out.print( "sub => " );
                line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
                in = Integer.parseInt( line );

                // invoke server application method
                _intf.sub( in );
                System.out.println( "value = " + _intf.value() );

                target.destroy( _intf );
            }
            catch ( java.lang.NumberFormatException e ) {
                System.exit(255);
            }
        }
        catch ( Exception e ) {
            System.err.println( "\nERROR : " + e );
            System.exit( 255 );
        }
    }
}
```

Server Application

Example

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import java.util.*;
import ODsample.*;

// user application process class
public class Server {
    public static void main( String args[] ) {
```

```
        try {
            // create and initialize ORB
            ORB Orb = ORB.init( args, null );

            // get RootPOA object references
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references ( "RootPOA" );
            // get POA objects of RootPOA
            POA rootPOA = POAHelper.narrow( _tmpObj );

            // create POA for Factory interface
            // construct policy list
            org.omg.CORBA.Policy factory_policies[] = new org.omg.CORBA.Policy[4];
            factory_policies[0] = rootPOA.create_servant_retention_policy(
                                  ServantRetentionPolicyValue.NON_RETAIN );
            factory_policies[1] = rootPOA.create_request_processing_policy(
                                  RequestProcessingPolicyValue.USE_DEFAULT_SERVANT );
            factory_policies[2] = rootPOA.create_id_assignment_policy(
                                  IdAssignmentPolicyValue.SYSTEM_ID );
            factory_policies[3] = rootPOA.create_id_uniqueness_policy(
                                  IdUniquenessPolicyValue.MULTIPLE_ID );
            POA factory_POA = rootPOA.create_POA( "IDL:ODsample/Factory:1.0",
                                                  null,
                                                  factory_policies );

            // construct interface POA
            // construct policy list
            org.omg.CORBA.Policy inf_policies[] = new org.omg.CORBA.Policy[4];
            inf_policies[0] = rootPOA.create_servant_retention_policy(
                                  ServantRetentionPolicyValue.NON_RETAIN );
            inf_policies[1] = rootPOA.create_request_processing_policy(
                                  RequestProcessingPolicyValue.USE_SERVANT_MANAGER );
            inf_policies[2] = rootPOA.create_id_assignment_policy(
                                  IdAssignmentPolicyValue.USER_ID );
            inf_policies[3] = rootPOA.create_id_uniqueness_policy(
                                  IdUniquenessPolicyValue.UNIQUE_ID );
            String inf_adapter_name = "IDL:ODsample/intf:1.0";
            //Interface Repository ID
            POA childPOA = rootPOA.create_POA( "childPOA", null, inf_policies );

            // create FactoryServant
            Servant svt = new FactoryServant( childPOA );
            // set to Factory interface default Servant
            factory_POA.set_servant( svt );

            // create ServantLocator
            ServantLocator svtloc = new UserServantLocator();
            // register as ServantManager in interface POA
            childPOA.set_servant_manager( svtloc );

            // get POAManager
            POAManager poamanager = rootPOA.the_POAManager();

            // activate POAManager
            poamanager.activate();
            Orb.run();
        }
        catch ( Exception e ) {
            System.err.println( "\nERROR: " + e );
            System.exit( 255 );
        }
    }
}
```

```
// FactoryServant Factory method implementation class (inherit skeleton)
class FactoryServant extends FactoryPOA {
    static java.util.Hashtable tbl = new java.util.Hashtable();  // control table
    private POA poa = null;

    // constructor
    public FactoryServant( POA poa ) {
        this.poa = poa;
    }

    public intf create( java.lang.String userid ) {
        intf       ior;   // UserServant object references

        // create object references
        org.omg.CORBA.Object _tmpObj = this.poa.create_reference_with_id(
            userid.getBytes(),
            "IDL:ODsample/intf:1.0" );
            ior = intfHelper.narrow( _tmpObj );

            return( ior );
    }

    public void destroy( intf obj )
    {
        try{
            // request ObjectID from object references
            byte oid[] = this.poa.reference_to_id( obj );

            // delete Servant from control table
            FactoryServant.tbl.remove(  new ObjKey( oid )  );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.err.println( "destroy error: " + e );
        }
    }
}

// ServantActivator:Servant create class (inherit ServantActivator)
  class UserServantLocator extends LocalObject implements ServantLocator{

    public Servant preinvoke(
        byte[] oid,
        org.omg.PortableServer.POA adapter,
        java.lang.String operation,
        org.omg.PortableServer.ServantLocatorPackage.CookieHolder cookie ) {

        // search for Servant.  If not found, create with new.
        Servant svt =  (Servant)FactoryServant.tbl.get( new ObjKey( oid ) );
        if ( svt == null ) {
            svt = new UserServant();
            FactoryServant.tbl.put( new ObjKey(oid), svt );
        }

        // create Cookie
        cookie.value = new ObjKey(oid);

        return( svt );
    }

    public void postinvoke( byte[] oid,
                            org.omg.PortableServer.POA adapter,
                            java.lang.String operation,
                            java.lang.Object cookie,
```

```
                            Servant the_servant ) {

        // run initialization attribute.
        the_servant = null;
    }
}

// ObjKey
class ObjKey{
    private byte[]  key;

    public ObjKey( byte[] _key ) {
        key = _key;
    }

    public int hashCode() {
          return (int)key[0];
    }

    public boolean equals( java.lang.Object comp ) {
        int i = key.length;
        if ( i != ((ObjKey)comp).key.length ) {
            return false;
        }
        for ( i--; i>=0; i-- ) {
            if ( key[i] != ((ObjKey)comp).key[i] ) {
                return false;
            }
        }

        return true;
    }
}

// Servant method implementation class (inherit skeleton)
class UserServant extends intfPOA {

    private int    value = 0;

    public int value() {
        return( this.value );
    }

    public void add( int a ) {
        this.value += a;
    }

    public void sub( int b ) {
        this.value -= b;
    }
}
```

Obtaining Exception Information

Obtain exception information for server applications in the same way as for client applications. Refer to "6.9 Exception Handling for Client Applications" for details.

6.18.5 Sample AdapterActivator Applications (find_POA)

This section provides an AdapterActivator Applications find_POA example.

IDL Definition

Example

```
module ODsample{
    interface intf1{
        attribute string value;
    };
    interface intf2{
        readonly attribute long value;
        void add( in long a );
        void sub( in long b );
    };
    interface Factory {
        intf1  create1 ();
        intf2  create2 ();
        void   destroy1( in intf1 obj );
        void   destroy2( in intf2 obj );
    };
};
```

NamingService Registration

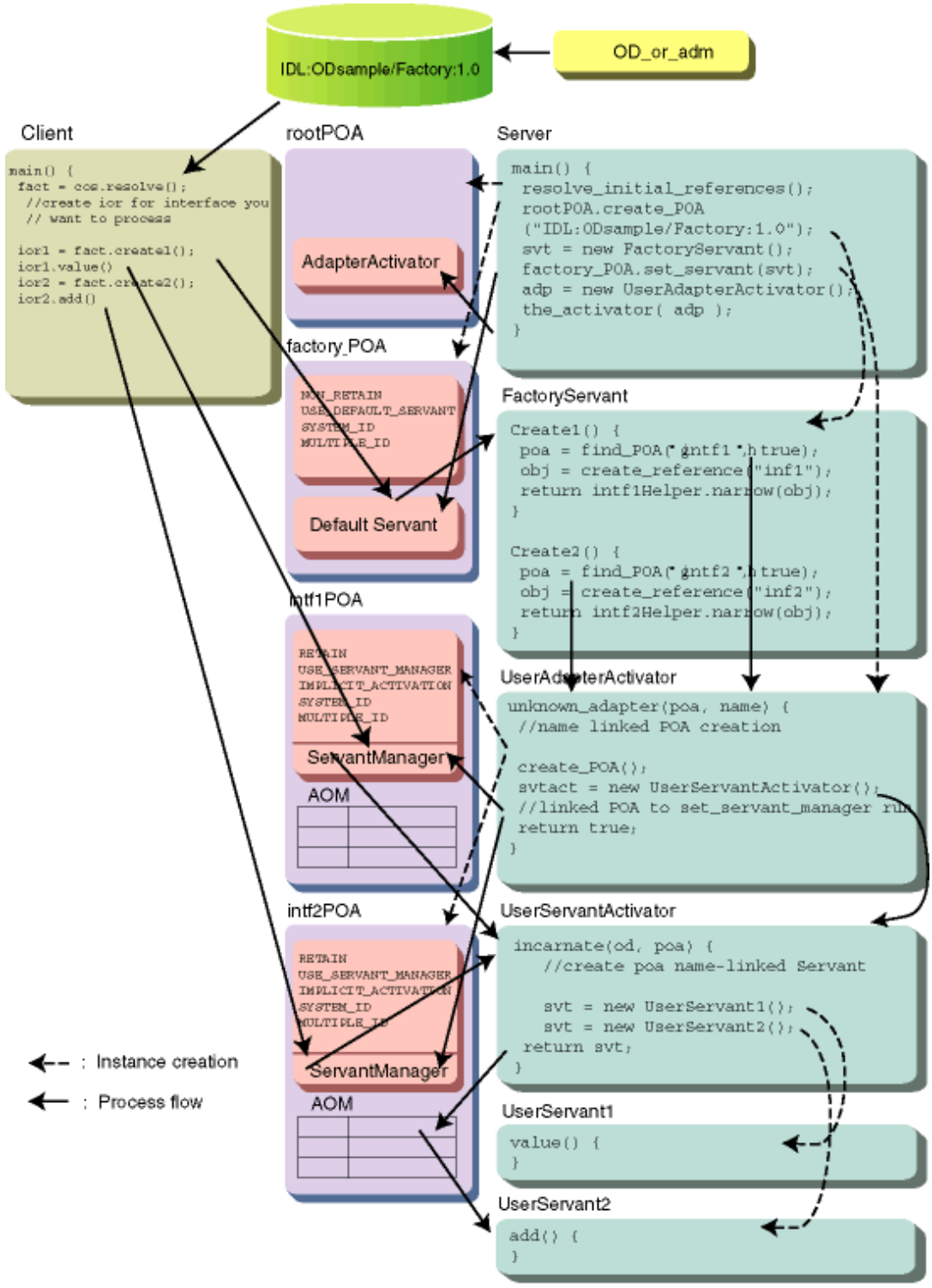
Example

```
OD_or_adm -c IDL:ODsample/Factory:1.0 -n ODsample::POAsample9
```

Application Construction Overview

The figure below outlines the application construction process associated with the AdapterActivator Applications find_POA method.

Figure 6.53 Block Diagram of Sample AdapterActivator (findPOA) Applications



Client Application

Example

```
import java.io.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import ODsample.*;
```

```
public class Client {
    public static void main( String args[] ) {

        try {
            // create and initialize ORB
            ORB Orb = ORB.init( args, null );

            // get NamingService object references
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "NameService" );
            NamingContextExt Cos = NamingContextExtHelper.narrow( _tmpObj );

            // run NamingService resolve method,
            // get server application object references
            String NCid   = new String( "ODsample::POAsample9" ); // object name
            String NCkind = new String( "" );                     // object type
            NameComponent nc =  new NameComponent( NCid, NCkind );
            NameComponent NCo[] = { nc };
            org.omg.CORBA.Object  Obj = Cos.resolve( NCo );

            int    in   = 0;    // in-parameter attribute
            String line = null;

            // get Factory object references
            Factory target = FactoryHelper.narrow( Obj );

            // construct object references for interface1 with Factory
            intf1 _intf1 = target.create1();

            // invoke server application method
            System.out.println( "String = " + _intf1.value() );

            System.out.print( "String => " );
            line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();

            // invoke server application method
            _intf1.value( line );
            System.out.println( "String = " + _intf1.value() );

            // interface analysis
            target.destroy1( _intf1 );

            // construct object references for interface2 with Factory
            intf2 _intf2 = target.create2();

            try {
                // invoke server application method
                System.out.println( "value = " + _intf2.value() );

                System.out.print( "add => " );
                line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
                in = Integer.parseInt( line );

                // invoke server application method
                _intf2.add( in );
                System.out.println( "value = " + _intf2.value() );

                System.out.print( "sub => " );
                line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
                in = Integer.parseInt( line );

                // invoke server application method
                _intf2.sub( in );
                System.out.println( "value = " + _intf2.value() );
```

```
                // instance analysis
                target.destroy2( _intf2 );
            }
            catch ( java.lang.NumberFormatException e ){
                System.exit( 255 );
            }
        }
        catch ( Exception e ) {
            System.err.println( "\nERROR : " + e );
            e.printStackTrace( System.err );
            System.exit( 255 );
        }
    }
}
```

Server Application

Example

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// user application: main process class
public class Server {
    public static void main( String args[] ) {

        try {
            // create and initialize ORB
            ORB Orb = ORB.init( args, null );

            // get RootPOA object references
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references
            ( "RootPOA" );
            // get RootPOA POA objects
            POA rootPOA = POAHelper.narrow( _tmpObj );

            // create POA for Factory instances
            // construct policy list
            org.omg.CORBA.Policy factory_policies[] = new org.omg.CORBA.Policy[4];
            factory_policies[0] = rootPOA.create_servant_retention_policy(
                                  ServantRetentionPolicyValue.NON_RETAIN );
            factory_policies[1] = rootPOA.create_request_processing_policy(
                                  RequestProcessingPolicyValue.USE_DEFAULT_SERVANT );
            factory_policies[2] = rootPOA.create_id_assignment_policy(
                                  IdAssignmentPolicyValue.SYSTEM_ID );
            factory_policies[3] = rootPOA.create_id_uniqueness_policy(
                                  IdUniquenessPolicyValue.MULTIPLE_ID );
            POA factory_POA = rootPOA.create_POA( "IDL:ODsample/Factory:1.0",
                                                  null,
                                                  factory_policies);

            // create FactoryServant (pass implementation POA to constructor)
            Servant svt = new FactoryServant( rootPOA );
            // set to default Servant for Factory interface POA
            factory_POA.set_servant( svt );

            // create AdapterActivator
            AdapterActivator adp = new UserAdapterActivator();
            // set to AdapterActivator for implementation POA
            rootPOA.the_activator( adp );
```

```
            // get POAManager
            POAManager poamanager = rootPOA.the_POAManager();

            // activate POAManager
            poamanager.activate();
            Orb.run();
        }
        catch ( Exception e ) {
            System.err.println( "\nERROR: " + e );
            System.exit( 255 );
        }
    }
}

// FactoryServant Factory method implementation class (inherit skeleton)
class FactoryServant extends FactoryPOA {

    private POA poa = null;

    // constructor
    public FactoryServant( POA poa ) {
        this.poa = poa;
    }

    public intf1 create1() {
        intf1      ior;      // UserServant object references

        try {
            // get POA objects corresponding to Interface Repository ID
            POA intf1POA = this.poa.find_POA( "intf1", true );

            // create object references
            org.omg.CORBA.Object _tmpObj = intf1POA.create_reference(
                                                "IDL:ODsample/intf1:1.0" );
            ior = intf1Helper.narrow( _tmpObj );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.err.println( "create error: " + e );
            return( null );
        }

        return( ior );

    }

    public intf2 create2() {
        intf2      ior;                  // UserServant object references

        try {
            // get POA object corresponding to Interface Repository ID
            POA intf2POA = this.poa.find_POA( "intf2", true );

            // create object reference
            org.omg.CORBA.Object _tmpObj = intf2POA.create_reference(
                                      "IDL:ODsample/intf2:1.0" );
            ior = intf2Helper.narrow( _tmpObj );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.err.println( "create error: " + e );
            return( null );
        }
```

```
        return( ior );

    }

    public void destroy1( intf1 obj ) {
        try {
            // get POA object corresponding to interface ID
            POA intf1POA = this.poa.find_POA( "intf1", false );

            // request objectID from object references
            byte oid[] = intf1POA.reference_to_id( obj );

            // deactivate Servant
            intf1POA.deactivate_object( oid );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.err.println( "destroy error: " + e );
        }
    }

    public void destroy2( intf2 obj ) {
        try{
            // get POA object corresponding to interface ID
            POA intf2POA = this.poa.find_POA( "intf2", false );

            // request ObjectID from object references
            byte oid[] = intf2POA.reference_to_id( obj );

            // deactivate Servant
            intf2POA.deactivate_object( oid );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.err.println( "destroy error: " + e );
        }
    }
}

// AdapterActivator POA creator class (inherit AdapterActivator)
class UserAdapterActivator extends LocalObject implements AdapterActivator {

    public boolean unknown_adapter( POA parent, String name ) {

        // construct policy list
        org.omg.CORBA.Policy policies[] = new org.omg.CORBA.Policy[4];
        policies[0] = parent.create_servant_retention_policy(
                                ServantRetentionPolicyValue.RETAIN );
        policies[1] = parent.create_request_processing_policy(
                                RequestProcessingPolicyValue.USE_SERVANT_MANAGER );
        policies[2] = parent.create_id_assignment_policy(
                                IdAssignmentPolicyValue.SYSTEM_ID );
        policies[3] = parent.create_id_uniqueness_policy(
                                IdUniquenessPolicyValue.MULTIPLE_ID );

        // create ServantActivator
        ServantActivator svtact = new UserServantActivator();

        try {
            if( name.equals( "intf1" ) ) {
                // create POA for intf1
                POA intf1POA = parent.create_POA( name, null, policies );

                // register ServantManager
                intf1POA.set_servant_manager( svtact );
```

```
                return( true );
            }
            else if( name.equals( "intf2") ) {
                // create POA for intf2
                POA intf2POA = parent.create_POA( name, null, policies );

                // register ServantManager
                intf2POA.set_servant_manager( svtact );

                return( true );
            }
        }
        catch( Exception e ) {
            System.err.println( "unknown_adapter error: " + e );
            return( false );
        }

        return( true );
    }
}

// ServantActivator Servant creation class (inherit ServantActivator)
class UserServantActivator extends LocalObject implements ServantActivator {
    public Servant incarnate( byte[] oid, POA adapter ) {

        Servant svt = null;

        if ( adapter.the_name().equals( "intf1") )
            svt = new UserServant1();
        else if ( adapter.the_name().equals( "intf2") )
            svt = new UserServant2();

        return( svt );
    }

    public void etherealize( byte[] oid,
                             POA adapter,
                             Servant serv,
                             boolean cleanup_in_progress,
                             boolean remaining_activations ) {

        // run attribute initialization.
        serv = null;
    }
}

// Servant method implementation class (inherit skeleton)
class UserServant1 extends intf1POA {

    private String  value = "";

    public java.lang.String value() {
        return( this.value );
    }

    public void value( java.lang.String value ) {
        this.value = value;
    }
}

class UserServant2 extends intf2POA {
```

```
    private int    value = 0;

    public int value() {
        return( this.value );
    }

    public void add( int a ) {
        this.value += a;
    }

    public void sub( int b ) {
        this.value -= b;
    }
}
```

Obtaining Exception Information

Obtain exception information for server applications in the same way as for client applications. Refer to "6.9 Exception Handling for Client Applications" for details.

6.18.6 AdapterActivator Sample Usage (Request Reception)

This section provides an AdapterActivator Usage Request Reception example.

IDL Definition

Example

```
module ODsample{
    interface    intf1{
        long  add( in long a, in long b );
    };
    interface  intf2{
        long  sub( in long a, in long b );
    };
};
```

NamingService Registration

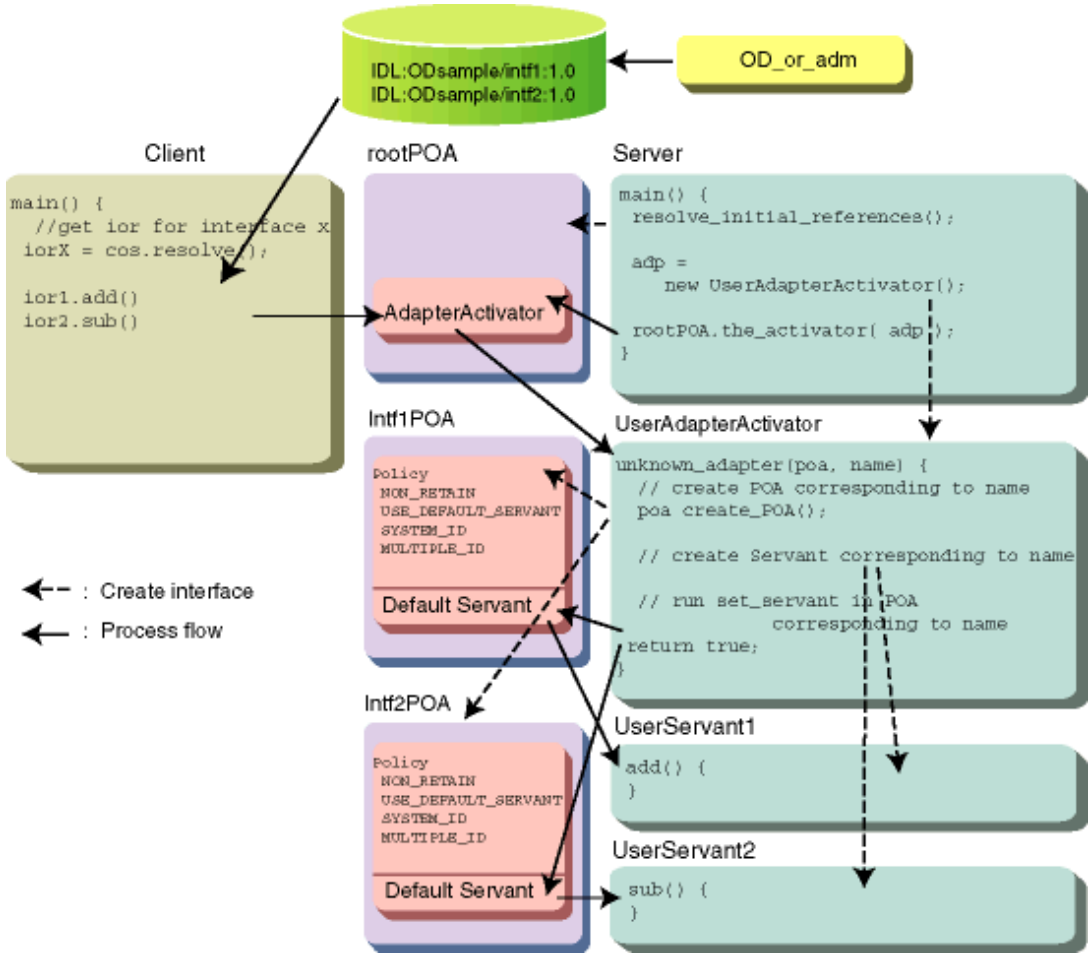
Example

```
OD_or_adm -c IDL:ODsample/intf1:1.0  -a Imple_POAsample -n ODsample::POAsample1
OD_or_adm -c IDL:ODsample/intf2:1.0  -a Imple_POAsample -n ODsample::POAsample2
```

Application Construction Overview

The figure below outlines the application construction process associated with the AdapterActivator (Request Reception) method.

Figure 6.54 Block Diagram for AdapterActivator (Request Reception) Sample Applications



Client Application

Example

```
import java.io.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import ODsample.*;

public class Client {
    public static void main( String args[] ) {

    int  in1 = 0;      // in-parameter attribute
    int  in2 = 0;      // in-parameter attribute
    String line = null;

    try {
        // create and initialize ORB
        ORB Orb = ORB.init( args, null );

        // get NamingService object references
        org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "NameService" );
        NamingContextExt Cos = NamingContextExtHelper.narrow( _tmpObj );

        // run NamingService resolve method,
        // get server application object references
        String NCid   =
```

```
            new String( "ODsample::POAsample10-1" ); // object name
        String NCkind = new String( "" );               // object type
        NameComponent nc =  new NameComponent( NCid, NCkind );
        NameComponent NCo[] = { nc };
        org.omg.CORBA.Object  Obj = Cos.resolve( NCo );

        // get server application object references
        intf1 _intf1 = intf1Helper.narrow( Obj );

        try {
            System.out.print( "in1 => " );
            line = new BufferedReader( new InputStreamReader( System.in )).readLine();
            in1 = Integer.parseInt( line );

            System.out.print( "in2 => " );
            line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
            in2 = Integer.parseInt( line );
        }
        catch ( java.lang.NumberFormatException e ) {
            System.exit( 255 );
        }

        // invoke server application method
        int result = _intf1.add( in1, in2 );

        // display method result
        System.out.println( in1 + " + " + in2 + " = "+ result );

        // run NamingService resolve method,
        // get server application object references
        String NCid2   = new String( "ODsample::POAsample10-2" ); // object name
        String NCkind2 = new String( "" );                        // object type
        NameComponent nc2 =  new NameComponent( NCid2, NCkind2 );
        NameComponent NCo2[] = { nc2 };
        org.omg.CORBA.Object  Obj2 = Cos.resolve( NCo2 );

        // get server application object references
        intf2 _intf2 = intf2Helper.narrow( Obj2 );

        try {
            System.out.print( "in1 => " );
            line = new BufferedReader( new InputStreamReader( System.in )).readLine();
            in1 = Integer.parseInt( line );

            System.out.print( "in2 => " );
            line = new BufferedReader( new InputStreamReader( System.in )).readLine();
            in2 = Integer.parseInt( line );
        }
        catch ( java.lang.NumberFormatException e ) {
            System.exit( 255 );
        }

        // invoke server application method
        result = _intf2.sub( in1, in2 );

        // display method result
        System.out.println( in1 + " - " + in2 + " = "+ result );
    }
    catch ( Exception e ) {
        System.err.println( "\nERROR : " + e );
        System.exit( 255 );
        }
```

```
    }
}
```

Server Application

Example

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// user application: main process class
public class Server {
    public static void main(String args[]) {

        try {
            // create and initialize ORB
            ORB Orb = ORB.init( args, null );

            // get RootPOA object references
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "RootPOA" );
            // get POA objects of RootPOA
            POA rootPOA = POAHelper.narrow( _tmpObj );

            // create AdapterActivator
            AdapterActivator adp = new UserAdapterActivator();
            // set in AdapterActivator of implementation POA
            rootPOA.the_activator( adp );
            // get POAManager
            POAManager poamanager = rootPOA.the_POAManager();
            // activate POAManager
            poamanager.activate();
            Orb.run();
        }
        catch ( Exception e ) {
            System.err.println( "ERROR: " + e );
            System.exit( 255 );
        }
    }
}

// AdapterActivator: POA create class (inherit AdapterActivator)
class UserAdapterActivator extends LocalObject implements AdapterActivator {
    public boolean unknown_adapter( POA parent, String name ) {

    POA        inf_POA;            // interface POA
    // construct policy list
    org.omg.CORBA.Policy policies[] = new org.omg.CORBA.Policy[4];
    policies[0] = parent.create_servant_retention_policy(
                                ServantRetentionPolicyValue.NON_RETAIN );
    policies[1] = parent.create_request_processing_policy(
                                RequestProcessingPolicyValue.USE_DEFAULT_SERVANT );
    policies[2] = parent.create_id_assignment_policy(
                                IdAssignmentPolicyValue.SYSTEM_ID );
    policies[3] = parent.create_id_uniqueness_policy(
                                IdUniquenessPolicyValue.MULTIPLE_ID );
    try {
        if( name.equals( "IDL:ODsample/intf1:1.0" ) ) {
            // construct intf1 POA
            inf_POA = parent.create_POA( name, null, policies );

            // create Servant
```

```
            Servant svt = new UserServant1();
            // set in default Servant
            inf_POA.set_servant( svt );

            return( true );
        }
        else if( name.equals( "IDL:ODsample/intf2:1.0" ) ) {
            // create intf2 POA
            inf_POA = parent.create_POA( name, null, policies );

            // create Servant
            Servant svt = new UserServant2();
            // set in default Servant
            inf_POA.set_servant( svt );

            return( true );
        }
    }
    catch( Exception e ) {
        System.err.println( "unknown_adapter error: " + e );
        return( false );
    }

        return( true );
    }
}

// Servant method implementation class (inherit skeleton)
class UserServant1 extends intf1POA {
    public int add( int a, int b ) {
        return( a + b );
    }
}

class UserServant2 extends intf2POA {
    public int sub( int a, int b ) {
        return( a - b );
    }
}
```

Obtaining Exception Information

Obtain exception information for server applications in the same way as for client applications. Refer to "6.9 Exception Handling for Client Applications" for details.

6.18.7 Delegation-Based Method Sample Usage (Default Instances)

This section provides a delegation-based method example.

IDL Definition

Example

```
module ODsample{
    interface  intf{
        long  add( in  long  a,  in  long  b );
    };
};
```

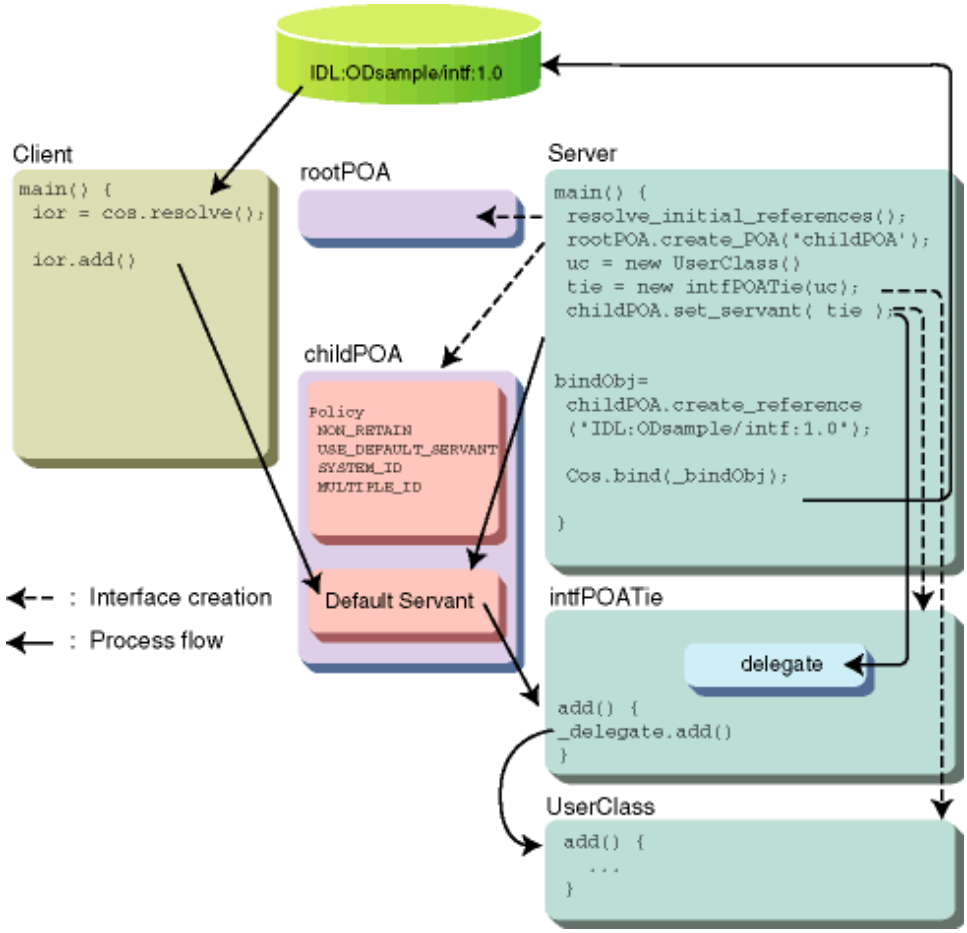
NamingService Registration

Dynamic registration.

Application Construction Overview

The figure below outlines the application construction process associated with the delegation-based method (default instances).

Figure 6.55 Default-Instance Applications Implemented by Delegation-based Method



Client Application

Example

```
import java.io.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import ODsample.*;

public class Client {
    public static void main( String args[] ) {

        try {
            // create and initialize ORB
            ORB Orb = ORB.init( args, null );

            // get NamingService object references
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references("NameService");
            NamingContextExt Cos = NamingContextExtHelper.narrow(_tmpObj);
```

```
            // run NamingService resolve method,
            // get server application object references
            String NCid   = new String( "ODsample::POAsample1" );   // object name
            String NCkind = new String( "" );                       // object type
            NameComponent nc =  new NameComponent( NCid, NCkind );
            NameComponent NCo[] = { nc };
            org.omg.CORBA.Object  Obj = Cos.resolve( NCo );

            // get server application object references
            intf target = intfHelper.narrow( Obj );

            int  in1 = 0;     // in-parameter attribute
            int  in2 = 0;     // in-parameter attribute
            int  result;      // return-value attribute
            String line = null;

            try {
                System.out.print( "in1 => " );
                line = new BufferedReader(new InputStreamReader( System.in ) ).readLine();
                in1 = Integer.parseInt( line );

                System.out.print( "in2 => " );
                line = new BufferedReader( new InputStreamReader( System.in ) ).readLine();
                in2 = Integer.parseInt( line );
            }
            catch ( java.lang.NumberFormatException e ) {
                System.exit( 255 );
            }

            // invoke server application method
            result = target.add( in1, in2 );

            // display method result
            System.out.println( in1 + " + " + in2 + " = "+ result );
        }
        catch ( Exception e ) {
              System.err.println( "\nERROR : " + e );
              System.exit( 255 );
        }
    }
}
```

Server Application

Example

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.CosNaming.*;
import ODsample.*;

// user application:main process class
public class Server {
    public static void main( String args[] ) {

        try {
            // create and initialize ORB
            ORB Orb = ORB.init( args, null );

            // get RootPOA object references
            org.omg.CORBA.Object _tmpObj =
            Orb.resolve_initial_references( "RootPOA" );
```

```
            // get RootPOA POA object
            POA rootPOA = POAHelper.narrow( _tmpObj );

            // construct interface POA
            // construct policy list
            org.omg.CORBA.Policy policies[] = new org.omg.CORBA.Policy[4];
            policies[0] = rootPOA.create_servant_retention_policy(
                          ServantRetentionPolicyValue.NON_RETAIN );
            policies[1] = rootPOA.create_request_processing_policy(
                          RequestProcessingPolicyValue.USE_DEFAULT_SERVANT );
            policies[2] = rootPOA.create_id_assignment_policy(
                          IdAssignmentPolicyValue.SYSTEM_ID );
            policies[3] = rootPOA.create_id_uniqueness_policy(
                          IdUniquenessPolicyValue.MULTIPLE_ID );

            // construct default Servant POA objects
            POA childPOA = rootPOA.create_POA( "childPOA", null, policies );
            // create user class instances
            UserClass uc = new UserClass();
            // construct tie object and register user class
            intfPOATie  tie = new intfPOATie( uc );
            // set in POA default Servant
            childPOA.set_servant( tie );

            org.omg.CORBA.Object _bindObj =
            childPOA.create_reference( "IDL:ODsample/intf:1.0" );

            // get NamingService object references
            _tmpObj = Orb.resolve_initial_references( "NameService" );
            NamingContextExt Cos = NamingContextExtHelper.narrow(_tmpObj);

            // get server application object references
            String NCid   = new String( "ODsample::POAsample1" );
            String NCkind = new String( "" );
            NameComponent nc =  new NameComponent( NCid, NCkind );
            NameComponent NCo[] = { nc };
            try {
                Cos.unbind( NCo );
            } catch( Exception e ) {
                ;
            }
            Cos.bind( NCo, _bindObj );

            // get POAManager
            POAManager poamanager = childPOA.the_POAManager();

            // activate POAManager
            poamanager.activate();
            Orb.run();
        }
        catch ( Exception e ) {
            System.err.println( "\nERROR: " + e );
            System.exit( 255 );
        }
    }
}

//user class (interface implementation class)
class UserClass
extends OtherUserClass
implements ODsample.intfOperations
{
    public int add( int a, int b ){
```

```
      return( a + b );
    }
}

//other user class
class OtherUserClass
{
    public int sub( int a, int b ){
        return( a - b );
    }
}
```

Obtaining Exception Information

Obtain exception information for server applications in the same way as for client applications. Refer to "6.9 Exception Handling for Client Applications" for details.

6.18.8 Active Object Map (AOM) Sample Usage (Factory-1 Method Instance Analysis)

This section provides an AOM Factory-1 Method Client Termination Instance Analysis example.

IDL Definition

Example

```
module ODsample{
    interface intf{
        readonly  attribute  long  value;
        void  add( in  long  a );
        void  sub( in  long  b );
    };
    interface Factory {
        intf   create( in string username );
        void   destroy( in  intf  obj );
    };
};
```

NamingService Registration

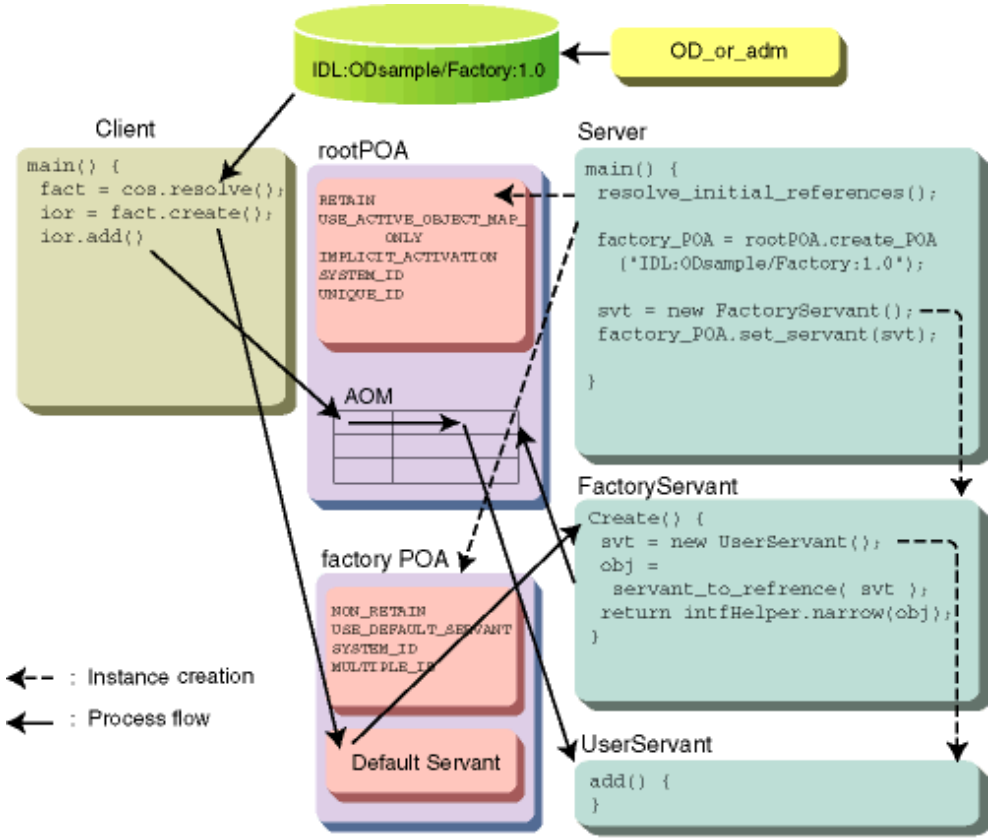
Example

```
OD_or_adm -c IDL:ODsample/Factory:1.0  -n ODsample::POAsample5
```

Application Construction Overview

The figure below outlines the application construction process associated with the Factory-1 (Client Termination Instance Analysis) method.

Figure 6.56 Block Diagram of Factory-1 Method Applications



Client Application

Example

```
import java.io.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import ODsample.*;

public class Client
{
    public static void main(String args[])
    {
        try {
            // create and initialize ORB
            ORB Orb = ORB.init( args, null );

            // get NamingService object references
            org.omg.CORBA.Object _tmpObj =
                        Orb.resolve_initial_references( "NameService" );
            NamingContextExt Cos = NamingContextExtHelper.narrow( _tmpObj );

            // run NamingService resolve method,
            // get server application object references
            String NCid   = new String( "ODsample::POAsample5" );
            String NCkind = new String( "" );
            NameComponent nc =  new NameComponent( NCid, NCkind );
            NameComponent NCo[] = { nc };
            org.omg.CORBA.Object  Obj = Cos.resolve( NCo );

            // get Factory object references
```

```
            Factory target = FactoryHelper.narrow( Obj );

            int  in;
            String line = null;

            try {
                // input user ID
                System.out.print( "Input your Name => " );
                line = new BufferedReader( new InputStreamReader( System.in )).readLine();
                if( line.length() == 0 ) {
                    line = new String( "guest" );
                }

                // construct interface  object references with Factory
                intf _intf = target.create( line );

                // invoke server application method
                System.out.println( "value = " + _intf.value() );

                System.out.print( "add => " );
                line = new BufferedReader( new InputStreamReader( System.in )).readLine();
                in = Integer.parseInt( line );

                // invoke server application method
                _intf.add( in );
                System.out.println( "value = " + _intf.value() );

                System.out.print( "sub => " );
                line = new BufferedReader( new InputStreamReader( System.in )).readLine();
                in = Integer.parseInt( line );

                // invoke server application method
                _intf.sub( in );
                System.out.println( "value = " + _intf.value() );
            }
            catch (java.lang.NumberFormatException e){
                System.exit(255);
            }
        }
        catch ( Exception e ) {
            System.err.println( "\nERROR : " + e );
            System.exit(255);
        }
    }
}
```

Server Application

Example

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// user application:main process class
public class Server
{
    public static void main(String args[])
    {

        try {
            // create and initialize ORB
```

```
            ORB Orb = ORB.init( args, null );

            // get RootPOA object references
            org.omg.CORBA.Object _tmpObj = Orb.resolve_initial_references( "RootPOA" );

            // get RootPOA POA objects
            POA rootPOA  = POAHelper.narrow( _tmpObj );

            // construct Factory interface POA
            // construct policy list
            org.omg.CORBA.Policy factory_policies[] = new org.omg.CORBA.Policy[4];

            factory_policies[0] = rootPOA.create_servant_retention_policy(
                                  ServantRetentionPolicyValue.NON_RETAIN );
            factory_policies[1] = rootPOA.create_request_processing_policy(
                                  RequestProcessingPolicyValue.USE_DEFAULT_SERVANT );
            factory_policies[2] = rootPOA.create_id_assignment_policy(
                                  IdAssignmentPolicyValue.SYSTEM_ID );
            factory_policies[3] = rootPOA.create_id_uniqueness_policy(
                                  IdUniquenessPolicyValue.MULTIPLE_ID );
            POA factory_POA = rootPOA.create_POA( "IDL:ODsample/Factory:1.0", null,
                                                  factory_policies );

            // create FactoryServant
            Servant svt = new FactoryServant( rootPOA );

            // set interface POA in default Servant
            factory_POA.set_servant( svt );

            // get POAManager
            POAManager poamanager = rootPOA.the_POAManager();

            // register instance analysis process
            com.fujitsu.ObjectDirector.PortableServer.POAdisconnect clrel =
                                                                  new instrelease();
            clrel.setDisconnect( clrel );

            // activate POAManager
            poamanager.activate();
            Orb.run();
        }
        catch (Exception e)
        {
            System.err.println("ERROR: " + e);
            e.printStackTrace(System.err);
            System.exit(255);
        }
    }
}

// FactoryServant Factory method implementation class (inherit skeleton)
class FactoryServant extends FactoryPOA
{
    private POA poa = null;
    static public java.util.Hashtable table;

    // constructor
    public FactoryServant() {
        table = new java.util.Hashtable();
    }
    public FactoryServant( POA param_poa ) {
        this();
        poa = param_poa;
```

```
    }

    public intf create( String UserName )
    {
        intf ior;

        try {
            // search for Servant
            Servant svt = (Servant)table.get( UserName );

            // create Servant if one not found (create guest always)
            if( null == svt ) {
                svt = new UserServant( UserName );
                table.put( UserName, svt );

                // if number of instances exceeds 100, modify to unconditional analysis
                if( table.size() > 100 ) {
                    com.fujitsu.ObjectDirector.PortableServer.POAdisconnect clrel =
                      new ALLinstrelease();
                    clrel.resetDisconnect( clrel );
                }
            }

            // create object references from Servant
            // since IMPLICIT_ACTIVATION policy is specified, it is auto-registered
            //in AOM
            org.omg.CORBA.Object _tmpObj = this.poa.servant_to_reference( svt );
            ior = intfHelper.narrow( _tmpObj );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.out.println( "create error: " + e );
            e.printStackTrace(System.err);
            throw new org.omg.CORBA.COMM_FAILURE();
        }

        return( ior );
    }

    public void destroy( intf obj )
    {
        try{
            // request objectID from object references
            byte oid[] = this.poa.reference_to_id( obj );

            // deactivate Servant
            this.poa.deactivate_object( oid );
        }
        catch( org.omg.CORBA.UserException e ) {
            System.out.println( "destroy error: " + e );
        }
    }
}

// Servant method implementation class (inherit skeleton)
class UserServant extends intfPOA
{
    private int    value = 0;
    public String UserName = null;

    public UserServant( String UserName ) {
        this.UserName = UserName;
    }
```

```
    public int value() {
        return( this.value );
    }

    public void add( int a ) {
        this.value += a;
    }

    public void sub( int b ) {
        this.value -= b;
    }

}

// conditional instance analysis class (inherit POAdisconnect)
class instrelease
    extends com.fujitsu.ObjectDirector.PortableServer.POAdisconnect {

    public boolean release_instance( org.omg.PortableServer.POA         POA,
                                     org.omg.PortableServer.Servant servant,
                                     byte[]                         oid ) {
        UserServant svt = (UserServant)servant;
        if( svt.UserName.equals( "guest" ) ) {
            FactoryServant.table.remove( svt.UserName );
            return true;
        }

        return false;
    }
}

// unconditional instance analysis class (inherit POAdisconnect)
class ALLinstrelease
    extends com.fujitsu.ObjectDirector.PortableServer.POAdisconnect {

    public boolean release_instance( org.omg.PortableServer.POA         POA,
                                     org.omg.PortableServer.Servant servant,
                                     byte[]                         oid ) {

        UserServant svt = (UserServant)servant;

        FactoryServant.table.remove( svt.UserName );
        return true;
    }
}
```

Obtaining Exception Information

Obtain exception information for server applications in the same way as for client applications. Refer to "6.9 Exception Handling for Client Applications" for details.

6.19 Mapping to Data Types

This section describes Java data types used in the ObjectDirector application.

6.19.1 Basic Data Types

In this section we will explain the data types to use when constructing client/server applications with Java language. When you use a basic data type defined by CORBA with a Java program, the corresponding data types are defined as shown in the following table.

Table 6.13 Definition Methods for Basic Data Types (Java Language)

CORBA Data Types		Java	Comment
integer type	long unsigned long short unsigned short long long	int int short short long	
floating-point type	float double	float double	
character type	char wchar	char (see Note below)	
octet type	octet	byte	
boolean type	boolean	boolean	
character string type	string wstring	java.lang.String	Refer to "6.19.2 String Type" Refer to "6.19.3 Wide String Type"
enumerator type	enum	<enum name>class	Refer to "6.19.4 Enumerator Type"
any type	any	org.omg.CORBA.Any	Refer to "6.19.5 Any Type"
object reference	Object	org.omg.CORBA.Object	
type code	TypeCode	org.omg.CORBA.TypeCode	

Note: char type cannot handle characters of 2-byte codes. To handle them, use wchar.

6.19.2 String Type

This section provides information relating to string type data.

IDL Mapping

When a character string is specified string in IDL language, it becomes a java.lang.String class and org.omg.CORBA.StringHolder class if you write it in Java language. You will use java.lang.String class for handling in parameters and return values. The org.omg.CORBA.StringHolder class comes out in the following manner. You would use this class for passing an out or inout parameter.

<org.omg.CORBA.StringHolder class>

```
final public class StringHolder {
    public java.lang.String value;
    public StringHolder() { }
    public StringHolder( java.lang.String initial ) {...}
}
```

The following table shows the significance of the members that are defined in the org.omg.CORBA.StringHolder class.

Table 6.14 org.omg.CORBA.StringHolder Class Method

org.omg.CORBA.StringHolder Class Members	Significance
value	Element name
default constructor	Use for creating instances without a setting to a value member

org.omg.CORBA.StringHolder Class Members	Significance
constructor	Sets element value specified by parameters in a value member

Under conditions in which IDL-language definitions were defined in the manner below, we offer this example of a server application program.

IDL

```
module ODsample{
    interface  stringtest{
        string op1( in string str1, out string str2, inout string str3 );
    };
};
```

Written in Java language, this comes out as follows.

Java Language

<Interface>

```
package ODsample;
    public interface stringtestOperations {
        public java.lang.String op1(java.lang.String str1,
                                    org.omg.CORBA.StringHolder str2,
                                    org.omg.CORBA.StringHolder str3 );
}
```

Processing with Client Applications

Example

```
import org.omg.CORBA.*;
import ODsample.*;

public class stringClient {
    public static void main( String args[] ) {
        // preprocess ORB
        // get object references

        try{
            // in parameter attribute
            String  in               = "in_str:data";
            // out parameter attribute
            StringHolder outHolder   = new StringHolder();
            // inout parameter attribute
            StringHolder inoutHolder = new StringHolder( "inout_str:data" );
            // return-value attribute
            String       result;

            // invoke server application method
            result = target.op1( in, outHolder, inoutHolder );

            // display method result
            System.out.println( result );
            System.out.println( outHolder.value );
            System.out.println( inoutHolder.value );
        }
        catch ( java.lang.Exception e ) {
            //error processing

        }
```

```
    }
}
```

When you deliver an in parameter to a server application in Java language, you set the data directly into the java.lang.String class instances. Also, you create org.omg.CORBA.StringHolder class instances for each data type using the new operator when you are delivering parameters to server applications as out or inout parameters. You set java.lang.String class data in the inout parameter. There are two methods for doing the settings, as follows.

Default Constructor

Example

```
StringHolder a = new StringHolder();   //default constructor
a.value = "in_str:data";               //data assignment
```

Constructor

Example

```
StringHolder b = new StringHolder( "in_str:data" );  //constructor
```

Moreover, for the server's processing results, the values are set in the value of each of the out and inout parameter instances. Return values are set directly in java.lang.String class instances.

Processing with Server Applications

Example

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// Servant class
// server application method
class stringServant extends stringtestPOA {

    public String op1( String in, StringHolder outHolder,
                               StringHolder inoutHolder ) {
        // display input parameter
        System.out.println( in );
        System.out.println( inoutHolder.value );

        // set output parameter
        outHolder.value = "out_str:data";
        inoutHolder.value = "inout-res_str:data";

        // return
        return( "result_str:data" );
    }
}

public class stringServer {
    public static void main( String args[] ) {
        try {
            // preprocess ORB

            // create POA objects

            // create Servant and register with its POA
```

```
            // activate POAManager

        }
        catch ( java.lang.Exception e ) {
            //error processing

        }
    }
}
```

When you reference an in parameter in Java language, you reference the in parameter's instances. For referencing inout parameters, retrieve the java.lang.String class instances from the org.omg.CORBA.StringHolder class and reference this instance. Also, when returning parameters to a client application as out or inout parameters, you will retrieve the java.lang.String class instances from org.omg.CORBA.StringHolder class and make the settings directly in these instances.

6.19.3 Wide String Type

This section provides information relating to wide string type data.

IDL Mapping

When a wstring is specified in IDL language, it becomes a java.lang.String class and org.omg.CORBA.StringHolder class if written in Java language. The java.lang.String class is used for handling in parameters and return values. The org.omg.CORBA.StringHolder class comes out in the following manner, and is used for passing an out or inout parameter.

<org.omg.CORBA.StringHolder class>

```
final public class StringHolder {
    public java.lang.String value;
    public StringHolder() { }
    public StringHolder( java.lang.String initial ) {...}
}
```

The following table shows the significance of the members that are defined in the org.omg.CORBA.StringHolder class.

Table 6.15 org.omg.CORBA.StringHolder Class Method

org.omg.CORBA.StringHolder Class Members	Significance
value	Element name
default constructor	Creates instances without a setting to a value member
constructor	Sets the element value specified by parameters in a value member

An example of a server application is provided, based on IDL-language definitions being defined in the following manner.

IDL

```
module ODsample{
    interface  stringtest{
        wstring op1( in swtring str1, out wstring str2, inout wstring str3 );
    };
};
```

Written in Java language, this comes out as follows.

Java Language

<Interface>

```
package ODsample;
    public interface wstringtestOperations {
        public java.lang.String op1(java.lang.String str1,
                                    org.omg.CORBA.StringHolder str2,
                                    org.omg.CORBA.StringHolder str3 );
}
```

Processing with Client Applications

Example

```
import org.omg.CORBA.*;
import ODsample.*;

public class stringClient {
    public static void main( String args[] ) {
        // preprocess ORB
        // get object references

        try{
            // in parameter attribute
            String  in              = "in_str:data";
            // out parameter attribute
            StringHolder outHolder   = new StringHolder();
            // inout parameter attribute
            StringHolder inoutHolder = new StringHolder( "inout_str:data" );
            // return-value attribute
            String       result;

            // invoke server application method
            result = target.op1( in, outHolder, inoutHolder );

            // display method result
            System.out.println( result );
            System.out.println( outHolder.value );
            System.out.println( inoutHolder.value );
        }
        catch ( java.lang.Exception e ) {
            //error processing
        }
    }
}
```

When you deliver an in parameter to a server application in Java language, you set the data directly into the java.lang.String class instances. In addition, you create org.omg.CORBA.StringHolder class instances for each data type using the new operator when you are delivering parameters to server applications as out or inout parameters. You set java.lang.String class data in the inout parameter. There are two methods for carrying out the settings, as shown below.

Default Constructor

Example

```
StringHolder a = new StringHolder();   //default constructor
a.value = "in_str:data";               //data assignment
```

Constructor

Example

```
StringHolder b = new StringHolder( "in_str:data" );  //constructor
```

Moreover, for the server's processing results, the values are set in the value of each of the out and inout parameter instances. Return values are set directly in java.lang.String class instances.

Processing with Server Applications

Example

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// Servant class
// server application method
class stringServant extends stringtestPOA {
    public stringServant(String[] args, java.util.Properties props) {
        super( args, props );
    }

    public String op1( String in, StringHolder outHolder,
                               StringHolder inoutHolder ) {
        // display input parameter
        System.out.println( in );
        System.out.println( inoutHolder.value );

        // set output parameter
        outHolder.value = "out_str:data";
        inoutHolder.value = "inout-res_str:data";

        // return
        return( "result_str:data" );
    }
}

public class stringServer {
    public static void main( String args[] ) {
        try {
            // preprocess ORB

            // create POA objects

            // create Servant and register with its POA

            // activate POAManager

        }
        catch ( java.lang.Exception e ) {
            //error processing

        }
    }
}
```

When you reference an in parameter in Java language, you reference the in parameter's instances. For referencing inout parameters, retrieve the java.lang.String class instances from the org.omg.CORBA.StringHolder class and reference this instance. When returning parameters

to a client application as out or inout parameters, retrieve the java.lang.String class instances from the org.omg.CORBA.StringHolder class and make the settings directly in these instances.

6.19.4 Enumerator Type

This section describes enumerator type data.

IDL Mapping

When an Enumerator Type is defined as enum in IDL language, it is equivalent to enum, enumHolder, and enumHelper classes in Java language. In the enum class, methods for identifying elements, like the one below, and methods for accessing members are defined. This class is used for handling in parameters and return values. In the following example, the enum class name is equivalent to the enumName as defined in IDL.

<enum class>

```
[package <packageName>]
public final class <enumName>{
        public static final int _<label_1> = <labelValue>;
        public static final <enumName> <label_1>
               = new <enumName>(_<label_1>);

        public static final int _<label_n> = <label Value>;
        public static final <enumName> <label_n>
               = new <enumName>(_<label_n>);

        public int value() {...}
        public static <enumName> from_int(int value) {...}
    }
    private <enumName>(int value) { }
}
```

The following table shows the significance of the members that are defined in enum class.

Table 6.16 enum Class Members

enum Class Members	Significance
value()	Extracts element value
from_int()	Specifies element value and extracts element name

The enumHolder class name is equivalent to the enum name as defined in IDL + Holder, as below. This class is used for delivering out or inout parameters.

<enumHolder class>

```
[package <packageName>;]
final public class <enumName>Holder
implements org.omg.CORBA.portable.Streamable  {
    public [<packageName>.]<enumName> value;
    public <enumName>Holder() { }
    public <enumName>Holder([<packageName>.]<enumName> value){...}
    public void _read (org.omg.CORBA.portable.InputStream i){...}
    public void _write(org.omg.CORBA.portable.OutputStream o){...}
```

```
    public org.omg.CORBA.TypeCode _type(){...}
}
```

The following table shows the significance of the members that are defined in enumHolder class.

Table 6.17 enumHolder Class Members

enumHolder Class Members	Significance
value	Element value
default constructor	Used for creating instances without settings for a value member.
constructor	Sets element value specified by parameters in a value member
_read()	Reads element value
_write()	Writes element value
_type()	Extracts TypeCode

The enumHelper class name is equivalent to the enum name as defined in IDL + Helper, as below. The Helper class provides the functions such as outputting the data of user-defined to data streams, inputting the data of user-defined from data streams, TypeCode of user-defined and referring to the repository ID.

<enumHelper class>

```
[package <packageName>;]
public class <enumName>Helper {
    public static void
        insert (org.omg.CORBA.Any a,[<packageName>.]<enumName> value){...}
    public static [<packageName>.]<enumName>
        extract (org.omg.CORBA.Any a){...}
    public static org.omg.CORBA.TypeCode
        type(){...}
    public static [<packageName>.]<enumName>
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, [<packageName>.]<enumName> value){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
}
```

The following table shows the significance of the members that are defined in the enumHelper class.

Table 6.18 enumHelper Class Members

enumHelper Class Members	Significance
insert()	Inserts element value
extract()	Extracts element value
type()	Extracts TypeCode

enumHelper Class Members	Significance
id()	Returns repository IDs

Using the IDL-language definitions defined below, we offer this example of a server application program.

IDL

```
module ODsample{
    enum EnumType { a, b, c };

    interface  enumtest{
        EnumType op( in EnumType in_p, out EnumType out_p, inout EnumType io_p );
    };
};
```

Mapping this in Java language, it comes out like the following.

Java Language

<Interface>

```
package ODsample;
public interface enumtestOperations {
    public ODsample.EnumType op(ODsample.EnumType in_p,
                            ODsample.EnumTypeHolder out_p,
                            ODsample.EnumTypeHolder io_p );
..}
```

<enum class>

```
package ODsample;
public final class EnumType
implements org.omg.CORBA.portable.IDLEntity
{
    public static final int _a = (int)0;
    public static final EnumType a = new EnumType(_a);
    public static final int _b = (int)1;
    public static final EnumType b = new EnumType(_b);
    public static final int _c = (int)2;
    public static final EnumType c = new EnumType(_c);
    public static final EnumType MAX_VALUE = new EnumType(Integer.MAX_VALUE);
    public int value(){...}
    public static EnumType from_int(int i){...}
    private EnumType(int value){ }
}
```

<enumHolder class>

```
package ODsample;
public class EnumTypeHolder
implements org.omg.CORBA.portable.Streamable {
    public ODsample.EnumType value;
    public EnumTypeHolder() { }
    public EnumTypeHolder(ODsample.EnumType value){...}
    public void _read(org.omg.CORBA.portable.InputStream i){...}
```

```
    public void _write(org.omg.CORBA.portable.OutputStream o){...}
    public org.omg.CORBA.TypeCode _type(){...}
}
```

<enumHelper class>

```
package ODsample;
public class EnumTypeHelper
{
    public static void
        insert(org.omg.CORBA.Any a, ODsample.EnumType value){...}
    public static ODsample.EnumType
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static ODsample.EnumType
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, ODsample.EnumType value){...}
}
```

Processing with Client Applications

Example

```
import org.omg.CORBA.*;
import ODsample.*;

public class enumClient {
    public static void main(String args[]) {
        // preprocess ORB
        // get object references

        try{
            // in parameter attribute
            EnumType  in   = EnumType.a;
            //out parameter attribute
            EnumTypeHolder outHolder = new EnumTypeHolder();
            // inout parameter attribute
            EnumTypeHolder inoutHolder = new EnumTypeHolder( EnumType.b );
            // return-value attribute
            EnumType       result;

            // invoke server application method
            result = target.op( in, outHolder, inoutHolder );

            // display method result
            System.out.println( result.value() );
            System.out.println( outHolder.value.value() );
            System.out.println( inoutHolder.value.value() );
        }
```

```
        catch ( java.lang.Exception e ) {
            //error processing

        }
    }
}
```

When you pass an in parameter to a server application, data is set directly in the enum class instances. Also, when you deliver parameters to a server application as out or inout parameters, you will create enumHolder class instances for each data type using the new operator. The inout parameter does the settings for enum class data. There are two methods, as below, for doing the settings.

Default Constructor

Example

```
EnumTypeHolder a = new EnumTypeHolder();  //default constructor
a.value = EnumType.c;                     //data assignment
```

Constructor

Example

```
EnumTypeHolder b = new EnumTypeHolder(EnumType.c);  //constructor
```

Also, for the results of the server process, the values are set in each instance's value for out and inout parameters and return values.

Processing with Server Applications

Example

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// Servant class
// server application method
class enumServant extends enumtestPOA {

    public EnumType op( EnumType in,
                        EnumTypeHolder outHolder,
                        EnumTypeHolder inoutHolder ) {
        // display input parameter
        System.out.println( in.value() );
        System.out.println( inoutHolder.value.value() );

        // set output parameter
        outHolder.value = EnumType.b;
        inoutHolder.value = EnumType.c;

        // return
        return( EnumType.a );
    }
}

public class enumServer {
    public static void main(String args[]) {
        try {
            // preprocess ORB

            // create POA objects

            // create Servant and register with its POA
```

```
            // activate POAManager

        }
        catch (java.lang.Exception e) {
            //error processing

        }
    }
}
```

When referencing an in parameter in Java language, you reference the value in the in parameter's instances. In referencing an inout parameter, you extract enum class instances from enumHolder class and reference with the value of this instance. Also, when delivering a parameter to a client application as an out or inout parameter, you extract enum class instances from the enum class and make settings directly in these instances.

6.19.5 Any Type

This section describes any type data.

IDL Mapping

When a type is defined as Any in IDL language, it is equivalent to org.omg.CORBA.Any class and org.omg.CORBA.AnyHolder class in Java language.

For the org.omg.CORBA.Any class, the methods for handling the Any Type and instance members have been defined. This class is used for handling in parameters and return values.

The following table shows the significance of the members that are defined under org.omg.CORBA.Any class.

Table 6.19 org.omg.CORBA.Any Class Members

org.omg.CORBA.Any Class Members	Significance
insert_xxx()	Inserts xxx-type element value
extract_xxx()	Extracts xxx-type element value
type()	Refer/sets TypeCode

For the org.omb.CORBA.AnyHolder class, instance members, default constructors, and constructors are defined as below. This class is utilized for delivering out or inout parameters.

<org.omg.CORBA.AnyHolder class>

```
final public class AnyHolder {
    public Any value;
    public AnyHolder() { }
    public AnyHolder(Any initial){...}
}
```

The following table shows the significance of the members that are defined under org.omg.CORBA.AnyHolder class.

Table 6.20 org.omg.CORBA.AnyHolder Class Members

org.omg.CORBA.AnyHolder Class Members	Significance
value	Element value
default constructor	Used to create interfaces without settings for a value member

org.omg.CORBA.AnyHolder Class Members	Significance
constructor	Sets element value specified by parameters in a value member

Using IDL-language definitions defined below, we offer this example of a server application program.

IDL

```
module ODsample{
    struct sample1 {
        long    para1;
        string  para2;
    };
    struct sample2 {
        char    para1;
        float   para2;
    };
    struct sample3 {
        char    para1;
        double  para2;
    };
    interface   anytest{
        any op1(in any any1, out any any2, inout any any3 );
    };
};
```

Written in Java language, this comes out as follows.

Java Language

<Interface>

```
package ODsample;
public interface anytestOperations {
    public org.omg.CORBA.Any op1(org.omg.CORBA.Any any1,
                                 org.omg.CORBA.AnyHolder any2,
                                 org.omg.CORBA.AnyHolder any3 );
}
```

<User Definition Type class>

```
package ODsample;
final public class sample1
  implements org.omg.CORBA.portable.IDLEntity
{
    public int para1;
    public java.lang.String para2;
    public sample1() {}
    public sample1(int para1, java.lang.String para2){...}
}

package ODsample;
final public class sample2
  implements org.omg.CORBA.portable.IDLEntity
{
    public char para1;
    public float para2;
```

```
    public sample2() {}
    public sample2(char para1, float para2) {...}
}

package ODsample;
final public class sample3
    implements org.omg.CORBA.portable.IDLEntity
{
    public char para1;
    public double para2;
    public sample3() {}
    public sample3(char para1, double para2){...}
}
```

<User Definition Type Holder class>

```
package ODsample;
public class sample1Holder  implements org.omg.CORBA.portable.Streamable {
    public ODsample.sample1 value;
    public sample1Holder(){}
    public sample1Holder(ODsample.sample1 value) {...}
    public void _read(org.omg.CORBA.portable.InputStream i) {...}
    public void _write(org.omg.CORBA.portable.OutputStream o) {...}
    public org.omg.CORBA.TypeCode _type() {...}
}

package ODsample;
public class sample2Holder  implements org.omg.CORBA.portable.Streamable {
    public ODsample.sample2 value;
    public sample2Holder() {}
    public sample2Holder(ODsample.sample2 value) {...}
    public void _read(org.omg.CORBA.portable.InputStream i) {...}
    public void _write(org.omg.CORBA.portable.OutputStream o) {...}
    public org.omg.CORBA.TypeCode _type() {...}
}

package ODsample;
public class sample3Holder  implements org.omg.CORBA.portable.Streamable {
    public ODsample.sample3 value;
    public sample3Holder() {}
    public sample3Holder(ODsample.sample3 value) {...}
    public void _read(org.omg.CORBA.portable.InputStream i) {...}
    public void _write(org.omg.CORBA.portable.OutputStream o) {...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

<User Definition Type Helper class>

```
package ODsample;
public class sample1Helper
{
    public static void
        insert(org.omg.CORBA.Any a, ODsample.sample1 value){...}
    public static ODsample.sample1
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
```

```
        get_id(){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static ODsample.sample1
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, ODsample.sample1 value){...}
}

package ODsample;
public class sample2Helper
{
    public static void
        insert(org.omg.CORBA.Any a, ODsample.sample2 value){...}
    public static ODsample.sample2
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static ODsample.sample2
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, ODsample.sample2 value){...}
}

package ODsample;
public class sample3Helper
{
    public static void
        insert(org.omg.CORBA.Any a, ODsample.sample3 value){...}
    public static ODsample.sample3
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static ODsample.sample3
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, ODsample.sample3 value){...}
}
```

Processing by Client Application

The data domain for an AnyType of the in parameter is initialized and accessed by the org.omg.CORBA.ORB.create_any() method. Also, for inout or out parameters, you will allocate the data domain using the org.omg.CORBA.AnyHolder class. In order to assign data to AnyType, use the basic data type insert method of the org.omg.CORBA.Any class in situations where you have basic data types such as Long, Short, etc., and in situations where you have other user-defined types, you would use the insert method of the user-defined type Helper class. This way, you are able to set both the values and TypeCode at the same time.

In order to extract data from an AnyType, you would use the basic data type extract method of org.omg.CORBA.Any class when you have the basic data types such as Long, Short, etc. In situations with other user-defined type data, you would utilize the user-definition type instances extracted by the extract method of the user-definition type Helper class.

```
import org.omg.CORBA.*;
import ODsample.*;

public class anyClient {
    public static void main(String args[]) {
        // preprocess ORB
        // get object references

        try{
            int       int_tmp =  1;
            String    str_tmp = "in_str:data";
            // create structureODsample.sample1 instances
            sample1   smp1    = new sample1( int_tmp, str_tmp );
            // in parameter attribute
            org.omg.CORBA.Any in  = org.omg.CORBA.ORB.init().create_any();
            // set sample 1 value, type in org.omg.CORBA.Any class
            sample1Helper.insert( in, smp1 );

            // out parameter attribute
            org.omg.CORBA.AnyHolder outHolder = new org.omg.CORBA.AnyHolder();

            char      char_tmp  = 'x';
            float     float_tmp = 0.003f;
            // create structureODsample.sample2 instances
            sample2   smp2      = new sample2( char_tmp, float_tmp );
            org.omg.CORBA.Any any_tmp  = org.omg.CORBA.ORB.init().create_any();
            // set sample2 value, type in org.omg.CORBA.Any class
            sample2Helper.insert( any_tmp, smp2 );
            // inout parameter attribute
            org.omg.CORBA.AnyHolder inoutHolder =
                           new org.omg.CORBA.AnyHolder( any_tmp );

            // return value attribute
            org.omg.CORBA.Any result;

            // invoke server application method
            result = target.op1( in, outHolder, inoutHolder );

            // display method result
            // extract sample3 value from org.omg.CORBA.Any class
            System.out.println( sample3Helper.extract(result).para1 );
            System.out.println( sample3Helper.extract(result).para2 );
            // extract sample 1 value from org.omg.CORBA.AnyHolder class
            System.out.println( sample1Helper.extract(outHolder.value).para1 );
            System.out.println( sample1Helper.extract(outHolder.value).para2 );
            // extract sample2 value from org.omg.CORBA.AnyHolder class
            System.out.println(sample2Helper.extract(inoutHolder.value).para1 );
            System.out.println(sample2Helper.extract(inoutHolder.value).para2 );
```

```
        }
        catch (java.lang.Exception e) {
            //error processing

        }
    }
}
```

Processing by Server Applications

When referencing inout or out parameters you would use the extract method for basic data types from the org.omg.CORBA.Any class, in the case of a basic data type such as Long, Short, etc. For other situations when the type is user defined, you would utilize the user-definition type instances extracted by the extract method of the user-defined Helper class.

In order to assign data to inout or out parameters, use the basic data type insert method from org.omg.CORBA.Any class when the data are the basic data types such as Long or Short. When it is other user defined type data you would make the settings for the Any type class in the Holder class's value of the inout and out parameters using the user defined type Helper class' insert method.

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// Servant class
// server application method
class anyServant extends anytestPOA {

    public org.omg.CORBA.Any op1( org.omg.CORBA.Any in,
                                  org.omg.CORBA.AnyHolder outHolder,
                                  org.omg.CORBA.AnyHolder inoutHolder ) {
            // display input parameter
            // extract sample1 value from org.omg.CORBA.Any class
            System.out.println( sample1Helper.extract(in).para1 );
            System.out.println( sample1Helper.extract(in).para2 );

            // extract sample2 value from org.omg.CORBA.AnyHolder class
            System.out.println( sample2Helper.extract(inoutHolder.value).para1 );
            System.out.println( sample2Helper.extract(inoutHolder.value).para2 );

            // set output parameter
            int     int_tmp =  12345;
            String  str_tmp = "out_str:data";
            // create structureODsample.sample1 instances
            sample1 smp1    = new sample1( int_tmp, str_tmp );
            org.omg.CORBA.Any any_tmp1  = org.omg.CORBA.ORB.init().create_any();
            // set sample1 value, type in org.omg.CORBA.Any class
            sample1Helper.insert( any_tmp1, smp1 );
            // set out parameter
            outHolder.value = any_tmp1;

            // extract inout parameter, create structureODsample.sample2 instances
            sample2 smp2 = sample2Helper.extract(inoutHolder.value);
            smp2.para1   = 'z';                  // modify inout parameter value
            smp2.para2   = smp2.para2 + 0.007f;   // modify inout parameter value
            org.omg.CORBA.Any any_tmp2  = org.omg.CORBA.ORB.init().create_any();
            // set sample2 value, type in org.omg.CORBA.Any class
            sample2Helper.insert( any_tmp2, smp2 );
            // set inout parameter value
            inoutHolder.value = any_tmp2;

            // set return
            char      char_tmp   = 'R';
```

```
            double    double_tmp = 9E38;
            // create structureODsample.sample3 instances
            sample3   smp3      = new sample3( char_tmp, double_tmp );
            org.omg.CORBA.Any result  = org.omg.CORBA.ORB.init().create_any();
            // set sample3 value, type in org.omg.CORBA.Any class
            sample3Helper.insert( result, smp3 );

            // return
            return( result );
        }
    }

    public class anyServer {
    public static void main( String args[] ) {
        try {
            // preprocess ORB

            // create POA objects

            // create Servant and register with its POA

            // initialize POAManager

        }
        catch ( java.lang.Exception e ) {
            //error processing

        }
    }
}
```

6.19.6 Sequence Type

This section describes sequence type data.

IDL Mapping

When a sequence type is defined as a sequence in IDL language, it is equivalent to the array for data type of elements of the sequence, the sequenceHolder class and sequenceHelper class in Java language. The array for data type is used to handle in parameters and return values. The sequenceHolder class name is equivalent to the sequence type name as defined in IDL + Holder, as below. This class is used for handling out and inout parameters.

<sequenceHolder class>

```
[package <packageName>;]
final public class <sequence_class>Holder
implements org.omg.CORBA.portable.Streamable  {
    public <sequence_element_type>[] value;
    public <sequence_class>Holder() {};
    public <sequence_class>Holder(<sequence_element_type>[] value) {...}
    public void _read(org.omg.CORBA.portable.InputStream i){...}
    public void _write(org.omg.CORBA.portable.OutputStream o){...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

The following table shows the significance of the members that are defined under the sequenceHolderclass.

Table 6.21 sequenceHolder Class Members

sequenceHolder Class Members	Significance
default constructor	Used to create interfaces without making settings for the value member
constructor	Sets specified parameter in the value member
_read()	Reads element value
_write()	Writes element value
_type()	Extracts TypeCode

The sequenceHelper class name is equivalent to the Sequence Type Name as defined in IDL + Helper, as below.

The Helper class provides the functions such as outputting the data of user-defined to data streams, inputting the data of user-defined from data streams, TypeCode of user-defined and referring to the repository ID.

<sequenceHelper class>

```
[package <packageName>;]
public class <sequence_class>Helper {
    public static void
        insert(org.omg.CORBA.Any a, <sequence_element_type>[] value){...}
    public static <sequence_element_type>[]
        extract (org.omg.CORBA.Any a){...}
    public static org.omg.CORBA.TypeCode
        type(){...}
    public static String
        id(){...}
    public static int[]
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, int[] value){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
}
```

The following table shows the significance of the members that are defined under the sequenceHelper class.

Table 6.22 sequenceHelper Class Members

sequenceHelper Class Members	Significance
insert()	Inserts element value
extract()	Extracts element value
type()	Extracts TypeCode
id()	Returns repository ID

Using IDL-language definitions below, we offer this example of a server application program.

IDL

```
module ODsample{
    interface   seqtest{
        typedef sequence<long>  sampleseq;
        sampleseq op1(in sampleseq seq1,
                      out sampleseq seq2,
                      inout sampleseq seq3 );
    };
};
```

Written in Java language, this comes out as follows:

Java Language

<Interface>

```
package ODsample;
public interface seqtestOperations {
    public int[] op1(int[] seq1,
        ODsample.seqtestPackage.sampleseqHolder seq2,
        ODsample.seqtestPackage.sampleseqHolder seq3 );
}
```

<sequenceHolder class>

```
package ODsample.seqtestPackage;
public class sampleseqHolder  implements org.omg.CORBA.portable.Streamable {
    public int[] value;
    public sampleseqHolder() {}
    public sampleseqHolder(int[] value) {...}
    public void _read ( org.omg.CORBA.portable.InputStream _i) {...}
    public void _write(org.omg.CORBA.portable.OutputStream _o) {...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

<sequenceHelper class>

```
package ODsample.seqtestPackage;
public class sampleseqHelper
{
    public static void
        insert(org.omg.CORBA.Any a, int[] value){...}
    public static int[]
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    public static org.omg.CORBA.TypeCode
        type(){...}
    public static int[]
        read(org.omg.CORBA.portable.InputStream istream){...}
```

```
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, int[] value){...}
}
```

Processing by Client Applications

A Sequence Type data domain for an in parameter is acquired using new. For inout or out parameters, the data domain is allocated using the sequenceHolder class.

In allocating data to sequence in an inout parameter, the domain is allocated to the instance data value of sequence by means of the necessary count designation with the constructor's parameters. Data is specified to that value. The sequence data references the instance data value.

```
import org.omg.CORBA.*;
import ODsample.*;

public class seqClient {
    public static void main(String args[]) {
        // preprocess ORB
        // get object references

        try{
            int i;

            // in parameter attribute
            int in[] = new int[10];
            for( i = 0; i < in.length; i++ )
                in[i] = i;

            // out parameter attribute
            ODsample.seqtestPackage.sampleseqHolder outHolder =
                     new ODsample.seqtestPackage.sampleseqHolder();
            // inout parameter attribute
            ODsample.seqtestPackage.sampleseqHolder inoutHolder =
                     new ODsample.seqtestPackage.sampleseqHolder( new int[20] );
            for( i = 0; i < inoutHolder.value.length; i++ )
                inoutHolder.value[i] = i * 2;

            // return value attribute
            int  result[];

            // invoke server application method
            result = target.op1( in, outHolder, inoutHolder );

            // display method result
            // display return value
            for( i = 0; i < result.length; i++ )
                System.out.println( "result[" + i + "]=" + result[i] );

            // display out parameter
            for( i = 0; i < outHolder.value.length; i++ )
                System.out.println( "outHolder.value[" + i + "]=" +
                                                outHolder.value[i] );

            // display inout parameter
            for( i = 0; i < inoutHolder.value.length; i++ )
                System.out.println( "inoutHolder.value[" + i + "]=" +
                                                inoutHolder.value[i] );
        }
        catch (java.lang.Exception e) {
            //error processing
```

```
        }
    }
}
```

Processing by Server Applications

In assigning data to inout or out parameters, set in the Holder class value for inout or out.

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// Servant class
// server application method
class seqServant extends seqtestPOA {

    public int[] op1( int[ in,
                      ODsample.seqtestPackage.sampleseqHolder outHolder,
                      ODsample.seqtestPackage.sampleseqHolder inoutHolder ) {
        int i;

        // display input parameter
        // display in parameter
        for( i = 0; i < in.length; i++ )
            System.out.println( "in[" + i + "]=" + in[i] );

        // display inout parameter
        for( i = 0; i < inoutHolder.value.length; i++ )
            System.out.println( "inoutHolder.value[" + i + "]=" +
                                                    inoutHolder.value[i] );

        // set output parameter
        // set out parameter
        outHolder.value = new int[5];
        for( i = 0; i < outHolder.value.length; i++ )
            outHolder.value[i] = in[i] * 100;

        // set inout parameter
        for( i = 0; i < inoutHolder.value.length; i++ )
            inoutHolder.value[i] = inoutHolder.value[i] * 2;

        // set return
        int result[] = new int[5];
        for( i = 0; i < result.length; i++ )
            result[i] = in[i] + 100;

        // return
        return( result );
    }
}

public class seqServer {
    public static void main(String args[]) {
        try {
            // preprocess ORB

            // create POA objects

            // create Servant and register with its POA

            // activate POAManager

        }
```

```
        catch (java.lang.Exception e) {
            //error processing
        }
    }
}
```

6.19.7 Structures

This section describes structure type data.

IDL Mapping

When a Structure Type is specified as struct in IDL language, that becomes struct class, structHolder class, and structHelper class in Java language. The struct class name is equivalent to the structure name as defined in IDL language. This class is used when in parameters and return values are handled.

The following table shows the significance of the members that are defined under struct class.

Table 6.23 struct Class Members

struct Class Members	Significance
default constructor	Does nothing
constructor	Sets specified parameters in instance members

The structHolder class name is equivalent to the structure name as defined in IDL language + Holder, as below. This class is used for handling out and inout parameters.

<structHolder class>

```
[package <packageName>;]
final public class <struct_class>Holder implements
                     org.omg.CORBA.portable.Streamable {
    public [<packageName>.]<struct_class>  value;
    public <struct_class>Holder() {}
    public <struct_class>Holder([<packageName>.]<struct_class> value) {...}
    public void _read(org.omg.CORBA.portable.InputStream i) {...}
    public void _write(org.omg.CORBA.portable.OutputStream o){...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

The following table shows the significance of the members that are defined by structHolder class.

Table 6.24 structHolder Class Members

structHolder Class Members	Significance
default constructor	Used to create interfaces without making settings for the value member
constructor	Sets specified parameter in the value member
_read()	Reads element value
_write()	Writes element value
_type()	Extracts TypeCode

The structHelper class name is equivalent to the structure name as defined in IDL + Helper, as below.

The Helper class provides the functions such as outputting the data of user-defined to data streams, inputting the data of user-defined from data streams, TypeCode of user-defined and referring to the repository ID.

<structHelper class>

```
[package <packageName>;]
public class <struct_class>Helper {
    public static void
        insert(org.omg.CORBA.Any a, <packageName>.<struct_class>  value){...}
    public static <packageName>.<struct_class>
        extract(org.omg.CORBA.Any a){...}
    public static org.omg.CORBA.TypeCode
        type(){...}
    public static String
        id(){...}
    public static <packageName>.<struct_class>
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream,
                        <packageName>.<struct_class>  value){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
}
```

The following table shows the significance of the members that are defined under the structHelper class.

Table 6.25 structHelper Class Members

structHelper Class Members	Significance
insert()	Inserts element value
extract()	Extracts elements value
type()	Extracts TypeCode
id()	Returns repository IDs

Under the conditions in which IDL-language definitions were defined in the manner below, we offer this example of a server application program.

IDL

```
module ODsample{
    struct samplefix {      // structure (fixed length)
        long    para1;
        long    para2;
    };
    struct samplevar {      // structure (variable length)
        long    para1;
        string  para2;
    };
    interface   structtest{
```

```
        samplefix   op2(
            in samplefix str1,
            out samplefix str2,
            inout samplefix str3 );
        samplevar   op1(
            in samplevar str1,
            out samplevar str2,
            inout samplevar str3 );
    };
};
```

Written in Java language, this comes out as follows:

Java Language

<Interface>

```
package ODsample;
public interface structtestOperations {
    public ODsample.samplefix op2(ODsample.samplefix str1,
                                  ODsample.samplefixHolder str2,
                                  ODsample.samplefixHolder str3 );
    public ODsample.samplevar op1(ODsample.samplevar str1,
                                  ODsample.samplevarHolder str2,
                                  ODsample.samplevarHolder str3 );
}
```

<struct class>

```
package ODsample;
final public class samplefix
  implements org.omg.CORBA.portable.IDLEntity
{
    public int para1;
    public int para2;
    public samplefix() {}
    public samplefix(int para1, int para2) {...}
}

package ODsample;
final public class samplevar
  implements org.omg.CORBA.portable.IDLEntity
{
    public int para1;
    public java.lang.String para2;
    public samplevar() {}
    public samplevar( int para1, java.lang.String para2) {...}
}
```

<structHolder class>

```
package ODsample;
public class samplefixHolder
    implements org.omg.CORBA.portable.Streamable {
    public ODsample.samplefix value;
    public samplefixHolder()  {}
    public samplefixHolder(ODsample.samplefix value) {...}
    public void _read(org.omg.CORBA.portable.InputStream i) {...}
    public void _write(org.omg.CORBA.portable.OutputStream o) {...}
    public org.omg.CORBA.TypeCode _type() {...}
```

```
}

package ODsample;
public class samplevarHolder
      implements org.omg.CORBA.portable.Streamable {
    public ODsample.samplevar value;
    public samplevarHolder()  {}
    public samplevarHolder(ODsample.samplevar value) {...}
    public void _read(org.omg.CORBA.portable.InputStream i) {...}
    public void _write(org.omg.CORBA.portable.OutputStream o) {...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

<structHelper class>

```
package ODsample;
public class samplefixHelper
{
    public static void
        insert(org.omg.CORBA.Any a, ODsample.samplefix value){...}
    public static ODsample.samplefix
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static ODsample.samplefix
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, ODsample.samplefix value){...}
}

package ODsample;
public class samplevarHelper
{
    public static void
        insert(org.omg.CORBA.Any a, ODsample.samplevar value){...}
    public static ODsample.samplevar
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static ODsample.samplevar
```

```
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream, ODsample.samplevar value){...}
}
```

Processing by Client Applications

The struct Type data domain for an in parameter is acquired using new. For inout or out parameters, the data domain is allocated using the structHolder class.

Client applications set the values in struct class instance members as in or inout parameter input values. The struct class instances thus constructed are set in structHolder class members by an inout parameter.

In referencing the output values of inout or out parameters, you will reference the value of the structHolder class instances.

```
import org.omg.CORBA.*;
import ODsample.*;

public class structClient {
    public static void main(String args[]) {
        // preprocess ORB
        // get object references

        try {

            // fixed-length structure

            // in parameter attribute
            int        in_fix_para1 = 3;
            int        in_fix_para2 = 10;
            samplefix  in_fix       = new samplefix( in_fix_para1,
                                                     in_fix_para2 );

            // out parameter attribute
            samplefixHolder outHolder_fix = new samplefixHolder();

            // inout parameter attribute
            int        inout_fix_para1 = 20;
            int        inout_fix_para2 = 50;
            samplefix  inout_fix_tmp   = new samplefix(inout_fix_para1,
                                                       inout_fix_para2);
            samplefixHolder inoutHolder_fix =
                              new samplefixHolder( inout_fix_tmp );

            // return value attribute
            samplefix  ret_fix;

            // invoke server application method
            ret_fix = target.op2( in_fix, outHolder_fix, inoutHolder_fix );

            // display method result
            // display return value
            System.out.println( ret_fix.para1 );
            System.out.println( ret_fix.para2 );

            // display out parameter
            System.out.println( outHolder_fix.value.para1 );
            System.out.println( outHolder_fix.value.para2 );

            // display inout parameter
            System.out.println( inoutHolder_fix.value.para1 );
            System.out.println( inoutHolder_fix.value.para2 );
```

```
            // variable-length structure

            // in parameter attribute
            int        in_var_para1 = 3;
            String     in_var_para2 = "test";
            samplevar  in_var       = new samplevar( in_var_para1,
                                                     in_var_para2 );

            // out parameter attribute
            samplevarHolder outHolder_var = new samplevarHolder();

            // inout parameter attribute
            int        inout_var_para1 = 20;
            String     inout_var_para2 = "dummy";
            samplevar  inout_var_tmp   = new samplevar(inout_var_para1,
                                                       inout_var_para2);

            samplevarHolder inoutHolder_var =
                              new samplevarHolder( inout_var_tmp );

            // return value attribute
            samplevar  ret_var;

            // invoke server application method
            ret_var = target.op1( in_var, outHolder_var, inoutHolder_var );

            // display method result
            // display return value
            System.out.println( ret_var.para1 );
            System.out.println( ret_var.para2 );

            // display out parameter
            System.out.println( outHolder_var.value.para1 );
            System.out.println( outHolder_var.value.para2 );

            // display inout parameter
            System.out.println( inoutHolder_var.value.para1 );
            System.out.println( inoutHolder_var.value.para2 );
        }
        catch (java.lang.Exception e) {
            //error processing
        }
    }
}
```

Processing by Server Applications

In assigning data to inout or out, make the settings in the Holder class value of inout or out parameters.

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// Servant class
// server application method
class structServant extends structtestPOA {

    // fixed-length structure
    public samplefix op2( samplefix in_fix,
                          samplefixHolder outHolder_fix,
                          samplefixHolder inoutHolder_fix ) {
```

```
        // display input parameter
        // display in parameter
        System.out.println( in_fix.para1 );
        System.out.println( in_fix.para2 );

        // display inout parameter
        System.out.println( inoutHolder_fix.value.para1 );
        System.out.println( inoutHolder_fix.value.para2 );

        // set output parameter
        // set out parameter
        int        out_fix_para1 = 123;
        int        out_fix_para2 = 456;
        samplefix  out_fix_tmp   =
                         new samplefix( out_fix_para1, out_fix_para2 );
        outHolder_fix.value = out_fix_tmp;

        // set inout parameter
        inoutHolder_fix.value.para1 = inoutHolder_fix.value.para1 * 10;
        inoutHolder_fix.value.para2 = inoutHolder_fix.value.para2 * 10;

        // set return value
        int        ret_fix_para1 = 333;
        int        ret_fix_para2 = 777;
        samplefix  ret_fix = new samplefix( ret_fix_para1, ret_fix_para2 );

        // return
        return( ret_fix );
    }

    // variable -length structure
    public samplevar op1( samplevar in_var,
                          samplevarHolder outHolder_var,
                          samplevarHolder inoutHolder_var ) {
        // display input parameter
        // display in parameter
        System.out.println( in_var.para1 );
        System.out.println( in_var.para2 );

        // display inout parameter
        System.out.println( inoutHolder_var.value.para1 );
        System.out.println( inoutHolder_var.value.para2 );

        // set output parameter
        // set out parameter
        int        out_var_para1 = 11111;
        String     out_var_para2 = "out-str";
        samplevar  out_var_tmp   =
                            new samplevar( out_var_para1, out_var_para2 );
        outHolder_var.value = out_var_tmp;

        // set inout parameter
        inoutHolder_var.value.para1 = inoutHolder_var.value.para1 + 2000;
        inoutHolder_var.value.para2 =
                              inoutHolder_var.value.para2 + "-inout-str";

        // set return
        int        ret_var_para1 = 0;
        String     ret_var_para2 = "return-str";
        samplevar  ret_var       =
                            new samplevar( ret_var_para1, ret_var_para2 );

        // return
```

```
        return( ret_var );
    }
}


public class structServer {
    public static void main(String args[]) {
        try {
            // preprocess ORB

            // create POA objects

            // create Servant and register with its POA

            // activate POAManager

        }
        catch (java.lang.Exception e) {
            //error processing

        }
    }
}
```

6.19.8 Unions

This section describes union type objects.

IDL Mapping

When a Union Type object is specified as union in IDL language, it becomes union class, unionHolder class, and unionHelper class in Java language. The union class name is equivalent to the union name as defined in IDL language. This class is used for the handling of in parameters and return values.

The following table shows the significance of the members that are defined under the union class.

Table 6.26 union Class Members

union Class Members	Significance
default constructor	Does nothing
discriminator access method	Discriminator access method is called "discriminator"
member settings method	Member settings method is called member name
member referencing method	Member referencing method is called member name

The unionHolder class name is equivalent to the union name as defined in IDL language + Holder, as below. This class is used for the handling of out or inout parameters.

<unionHolder class>

```
[package <packageName>;]
final public class <union_class>Holder implements
                  org.omg.CORBA.portable.Streamable {
    public [<packageName>.]< union _class>  value;
    public < union _class>Holder() {}
    public < union _class>Holder([<packageName>.]< union _class> value) {...}
    public void _read(org.omg.CORBA.portable.InputStream i) {...}
    public void _write(org.omg.CORBA.portable.OutputStream o){...}
```

```
    public org.omg.CORBA.TypeCode _type() {...}
}
```

The following table shows the significance of the members that are defined under unionHolder class.

Table 6.27 unionHolder Class Members

unionHolder Class Members	Significance
default constructor	Used to create instances without settings for the value member
constructor	Sets specified parameter in the value member
_read()	Reads element value
_write()	Writes element value
_type()	Extracts TypeCode

The unionHelper class name is equivalent to the union type name as defined in IDL + Helper, as below.

The Helper class provides the functions such as outputting the data of user-defined to data streams, inputting the data of user-defined from data streams, TypeCode of user-defined and referring to the repository ID.

<unionHelper class>

```
[package <packageName>;]
public class <union_class>Helper {
    public static void
        insert(org.omg.CORBA.Any a, <packageName>.<union_class>  value){...}
    public static <packageName>.<union_class>
        extract(org.omg.CORBA.Any a){...}
    public static org.omg.CORBA.TypeCode
        type(){...}
    public static String
        id(){...}
    public static <packageName>.<union_class>
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream,
                        <packageName>.<union_class>  value){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
}
```

The following table shows the significance of the members that are defined under UnionHelper class.

Table 6.28 UnionHelper Class Members

unionstructHelper Class Members	Significance
insert()	Inserts element value
extract()	Extracts element value

unionstructHelper Class Members	Significance
type()	Extracts TypeCode
id()	Returns repository IDs

We display examples of application programs under situations in which definitions in IDL language are defined in the manner below.

IDL

```
module ODsample{
    union samplefix switch(long){       // union (fixed length)
        case 1: long    para1;
        case 2: long    para2;
    };
    union samplevar switch(long){       // union (variable length)
        case 1: long    para1;
        case 2: string  para2;
    };
    interface uniontest{
        samplefix  op2(
            in samplefix uni1,
            out samplefix uni2,
            inout samplefix uni3
        );
        samplevar  op1(
            in samplevar uni1,
            out samplevar uni2,
            inout samplevar uni3
        );
    };
};
```

Written in Java language, this comes out as follows.

Java Language

<Interface>

```
package ODsample;
public interface uniontestOperations {
    public ODsample.samplefix op2( ODsample.samplefix uni1,
                                   ODsample.samplefixHolder uni2,
                                   ODsample.samplefixHolder uni3 );
    public ODsample.samplevar op1( ODsample.samplevar uni1,
                                   ODsample.samplevarHolder uni2,
                                   ODsample.samplevarHolder uni3 );
}
```

<union class>

```
package ODsample;
final public class samplefix
  implements org.omg.CORBA.portable.IDLEntity
{
    private int _discriminator;
    private java.lang.Object value;
    public samplefix() {}
```

```
    public int discriminator(){...}
    public int para1()
        throws org.omg.CORBA.BAD_OPERATION{...}
    public void para1( int value ){...}
    public int para2()
        throws org.omg.CORBA.BAD_OPERATION{...}
    public void para2( int value ){...}
}

package ODsample;
final public class samplevar
  implements org.omg.CORBA.portable.IDLEntity
{
    private int _discriminator;
    private java.lang.Object value;
    public samplevar() {}
    public int discriminator(){...}
    public int para1()
        throws org.omg.CORBA.BAD_OPERATION{...}
    public void para1( int value ){...}
    public java.lang.String para2()
        throws org.omg.CORBA.BAD_OPERATION{...}
    public void para2( java.lang.String value ){...}
}
```

<unionHolder class>

```
package ODsample;
public class samplefixHolder     implements org.omg.CORBA.portable.Streamable {
    public ODsample.samplefix value;
    public samplefixHolder()  {}
    public samplefixHolder(ODsample.samplefix value) {...}
    public void _read(org.omg.CORBA.portable.InputStream i) {...}
    public void _write(org.omg.CORBA.portable.OutputStream o) {...}
    public org.omg.CORBA.TypeCode _type() {...}
}

package ODsample;
public class samplevarHolder  implements org.omg.CORBA.portable.Streamable {
    public ODsample.samplevar value;
    public samplevarHolder()  {}
    public samplevarHolder(ODsample.samplevar value) {...}
    public void _read(org.omg.CORBA.portable.InputStream i) {...}
    public void _write(org.omg.CORBA.portable.OutputStream o) {...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

<unionHelper class>

```
package ODsample;
public class samplefixHelper
{
    public static void
        insert(org.omg.CORBA.Any a, ODsample.samplefix value){...}
    public static ODsample.samplefix
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream,
                     Object value){...}
    public java.lang.String
```

```
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static ODsample.samplefix
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream,
            ODsample.samplefix value){...}
}

package ODsample;
public class samplevarHelper
{
    public static void
        insert(org.omg.CORBA.Any a, ODsample.samplevar value){...}
    public static ODsample.samplevar
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream,
                    Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static ODsample.samplevar
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream,
            ODsample.samplevar value){...}
}
```

Processing by Client Applications

The union type data domain for an in parameter is acquired using new. For inout or out parameters, the data domain is allocated using unionHolder. For referencing/setting of union data you would utilize a member referencing function or a discriminator access function.

```
import org.omg.CORBA.*;
import ODsample.*;

public class unionClient {
    public static void main(String args[]) {
        // preprocess ORB
        // get object references

        try{
            // fixed length structure

            // in parameter attribute
            samplefix  in_fix   = new samplefix();
            in_fix.para1( 100 ); // initialize union attribute with para1 type

            // out parameter attribute
```

```
samplefixHolder outHolder_fix = new samplefixHolder();

// inout parameter attribute
samplefix  inout_fix_tmp = new samplefix();
inout_fix_tmp.para1( 200 ); // initialize union attribute with para2 type
samplefixHolder inoutHolder_fix = new samplefixHolder(inout_fix_tmp);

// return value attribute
samplefix ret_fix;

// invoke server application
ret_fix = target.op2( in_fix, outHolder_fix, inoutHolder_fix );

// display method result
// display return value
switch( ret_fix.discriminator() ) {
    case 1:
        System.out.println( ret_fix.para1() );
        break;
    case 2:
        System.out.println( ret_fix.para2() );
        break;
}

// display out parameter
switch( outHolder_fix.value.discriminator() ) {
    case 1:
        System.out.println( outHolder_fix.value.para1() );
        break;
    case 2:
        System.out.println( outHolder_fix.value.para2() );
        break;
}

// display inout parameter
switch( inoutHolder_fix.value.discriminator() ) {
    case 1:
        System.out.println( inoutHolder_fix.value.para1() );
        break;
    case 2:
        System.out.println( inoutHolder_fix.value.para2() );
        break;
}

// variable length structure

// in parameter attribute
samplevar  in_var   = new samplevar();
in_var.para1( 500 ); // initialize union attribute with para1 type

// out parameter attribute
samplevarHolder outHolder_var = new samplevarHolder();

// inout parameter attribute
samplevar  inout_var_tmp = new samplevar();
inout_var_tmp.para1( 200 ); // initialize union attribute with para2 type
samplevarHolder inoutHolder_var
        =  new samplevarHolder(inout_var_tmp);

// return value attribute
samplevar ret_var;

// invoke server application
```

```
            ret_var = target.op1( in_var, outHolder_var, inoutHolder_var );

            // display method result
            // display return value
            switch( ret_var.discriminator() ) {
                case 1:
                    System.out.println( ret_var.para1() );
                    break;
                case 2:
                    System.out.println( ret_var.para2() );
                    break;
            }

            // display out parameter
            switch( outHolder_var.value.discriminator() ) {
                case 1:
                    System.out.println( outHolder_var.value.para1() );
                    break;
                case 2:
                    System.out.println( outHolder_var.value.para2() );
                    break;
            }

            // display inout parameter
            switch( inoutHolder_var.value.discriminator() ) {
                case 1:
                    System.out.println( inoutHolder_var.value.para1() );
                    break;
                case 2:
                    System.out.println( inoutHolder_var.value.para2() );
                break;
            }

        }
        catch (java.lang.Exception e) {
            //error processing

        }
    }
}
```

Processing by Server Applications

In assigning data to inout or out parameters, you would make the settings in the Holder class's value of inout and out.

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// Servant class
// server application method
class unionServant extends uniontestPOA {

    public samplefix op2( samplefix in_fix,
                          samplefixHolder outHolder_fix,
                          samplefixHolder inoutHolder_fix ) {
        // display input parameter
        // display in parameter
        switch( in_fix.discriminator() )
        {
            case 1:
```

```
            System.out.println( in_fix.para1() );
            break;
        case 2:
            System.out.println( in_fix.para2() );
            break;
    }

    // display and set inout parameter
    switch( inoutHolder_fix.value.discriminator() )
    {
        case 1:
            System.out.println( inoutHolder_fix.value.para1() );

            // modify union attribute from para1 type to para2 type
            inoutHolder_fix.value.para2( inoutHolder_fix.value.para1() * 100 );
            break;
        case 2:
            System.out.println( inoutHolder_fix.value.para2() );

            // modify union attribute from para2 type to para1 type
            inoutHolder_fix.value.para1( inoutHolder_fix.value.para2() * 100 );
            break;
    }

    // set output parameter
    // set out parameter
    samplefix  out_fix_tmp = new samplefix();
    out_fix_tmp.para2( 999 ); // initialize union attribute with para2 type
    outHolder_fix.value = out_fix_tmp;

    // set return
    samplefix  ret_fix   = new samplefix();
    ret_fix.para1( 0 );   // initialize union attribute with para1 type

    // return
    return( ret_fix );
}


public samplevar op1( samplevar in_var,
                      samplevarHolder outHolder_var,
                      samplevarHolder inoutHolder_var ) {
    // display input parameter
    // display in parameter
    switch( in_var.discriminator() )
    {
        case 1:
            System.out.println( in_var.para1() );
            break;
        case 2:
            System.out.println( in_var.para2() );
            break;
    }

    // display and set inout parameter
    switch( inoutHolder_var.value.discriminator() )
    {
        case 1:
            System.out.println( inoutHolder_var.value.para1() );

            // modify union attribute from para1 type to para2 type
            inoutHolder_var.value.para2( "inout-str-add-ret" );
            break;
```

```
            case 2:
                System.out.println( inoutHolder_var.value.para2() );

                // modify union attribute from para2 type to para1 type
                inoutHolder_var.value.para1( 5000 );
                break;
        }

        // set output parameter
        // set out parameter
        samplevar  out_var_tmp = new samplevar();
        out_var_tmp.para2( "out-str" ); // initialize union attribute with para2 type
        outHolder_var.value = out_var_tmp;

        // set return
        samplevar  ret_var   = new samplevar();
        ret_var.para2( "ret-str" );   // initialize union attribute with para1 type

        // return
        return( ret_var );
    }
}

public class unionServer {
    public static void main(String args[]) {
        try {
            // preprocess ORB

            // create POA object

            // create Servant and register to its POA

            // activate POAManager

        }
        catch ( java.lang.Exception e ) {
            //error processing

        }
    }
}
```

6.19.9 Arrays

This section describes arrays.**Error! Bookmark not defined.**

IDL Mapping

When these are defined as arrays in IDL language, the Java-language equivalent becomes the array for data type of elements of the sequence, arraynameHolder class and arraynameHelper class.

The array for data type is used to handle in parameters and return values.

The arraynameHolder class name is equivalent to the IDL-defined array name + Holder. You would use this class for handling an out or inout parameter.

<arraynameHolder class>

```
[package <packageName>;]
final public class <array_class>Holder
    implements org.omg.CORBA.portable.Streamable  {
    public < array _element_type>[] value;
    public < array _class>Holder() {};
    public < array _class>Holder(<array _element_type>[] value) {...}
```

```
    public void _read(org.omg.CORBA.portable.InputStream i){...}
    public void _write(org.omg.CORBA.portable.OutputStream o){...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

The following table shows the significance of the members that are defined under arraynameHolder class.

Table 6.29 arraynameHolder Class Members

arraynameHolder Class Members	Significance
default constructor	Used to create instances without settings for the value member
constructor	Sets specified parameter in value member
_read()	Reads element value
_write()	Writes element value
_type()	Extract TypeCode

The arraynameHelper class name is equivalent to the IDL-defined array name + Helper, as below.

The Helper class provides the functions such as outputting the data of user-defined to data streams, inputting the data of user-defined from data streams, TypeCode of user-defined and referring to the repository ID.

<arraynameHelper class>

```
[package <packageName>;]
public class <array_class>Helper
{
    public static void
        insert(org.omg.CORBA.Any a, <array _element_type>[] value){...}
    public static <array _element_type>[]
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream,
                  Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static <array _element_type>[]
        read( org.omg.CORBA.portable.InputStream istream ){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream,
        <array _element_type>[] value)
        throws  org.omg.CORBA.MARSHAL{...}
}
```

The following table shows the significance of the members that are defined under the arraynameHelper class.

Table 6.30 arraynameHelper Class Member

ArraynameHelper Class Members	Significance
insert()	Inserts element value
extract()	Extracts element value
type()	Extracts TypeCode
id()	Returns repository IDs

The following example programs use the definition in the IDL language below.

IDL

```
Module ODsample{
    interface arraytest{
        typedef long fix[4][3][2];
        typedef string str[2][3][4];
        fix op1( in fix para1, out fix para2, inout fix para3 );
        str op2( in str para1, out str para2, inout str para3 );
    };
};
```

Written in Java language, this comes out as follows.

Java Language

<Interface>

```
package ODsample;
public interface arraytestOperations {
    public int[][][] op1(int[][][] para1,
                         ODsample.arraytestPackage.fixHolder para2,
                         ODsample.arraytestPackage.fixHolder para3 );
    public java.lang.String[][][] op2(java.lang.String[][][] para1,
                         ODsample.arraytestPackage.strHolder para2,
                         ODsample.arraytestPackage.strHolder para3 );
}
```

<arraynameHolder class>

```
package ODsample.arraytestPackage;
public final class fixHolder  implements org.omg.CORBA.portable.Streamable {
    public int[][][] value;
    public fixHolder()  {}
    public fixHolder(int[][][] value) {...}
    public void _read (org.omg.CORBA.portable.InputStream istream) {...}
    public void _write (org.omg.CORBA.portable.OutputStream ostream) {...}
    public org.omg.CORBA.TypeCode _type () {...}
}

package ODsample.arraytestPackage;
public final class strHolder implements org.omg.CORBA.portable.Streamable {
    public java.lang.String[][][] value;
    public strHolder()  {}
    public strHolder(java.lang.String[][][] value) {...}
    public void _read (org.omg.CORBA.portable.InputStream istream) {...}
```

```
    public void _write (org.omg.CORBA.portable.OutputStream ostream) {...}
    public org.omg.CORBA.TypeCode _type () {...}
}
```

<arraynameHelper class>

```
package ODsample.arraytestPackage;
public class fixHelper
{
    public static void
        insert(org.omg.CORBA.Any a, int[][][] value){...}
    public static int[][][]
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static int[][][]
        read( org.omg.CORBA.portable.InputStream istream ){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream,
            int[][][] value) throws  org.omg.CORBA.MARSHAL{...}
}

package ODsample.arraytestPackage;
public class strHelper
{
    public static void
        insert(org.omg.CORBA.Any a, java.lang.String[][][] value){...}
    public static java.lang.String[][][]
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static java.lang.String[][][]
        read( org.omg.CORBA.portable.InputStream istream ){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream,
              java.lang.String[][][] value)
              throws  org.omg.CORBA.MARSHAL{...}
}
```

Processing by Client Applications

The array type data domain for an in parameter is acquired using new. For inout or out parameters, you would allocate the data domain using the arraynameHolder class. In referencing inout or out parameters, reference the value of the instance data.

```
import org.omg.CORBA.*;
import ODsample.*;

public class arrayClient {
public static void main(String args[]) {
    // preprocess ORB
    // get object references

    try{

        int i, j, k;

        // int array

        // in parameter attribute
        int[][][] in_fix = new int[4][3][2];
        for( i = 0; i < in_fix.length; i++ )
            for( j = 0; j < in_fix[i].length; j++ )
                for( k = 0; k < in_fix[i][j].length; k++ )
                    in_fix[i][j][k]  =  i + j + k;

        // out parameter attribute
        ODsample.arraytestPackage.fixHolder outHolder_fix =
            new ODsample.arraytestPackage.fixHolder();

        // inout parameter attribute
        int[][][] inout_fix_tmp = new int[4][3][2];
        for( i = 0; i < inout_fix_tmp.length; i++ )
            for( j = 0; j < inout_fix_tmp[i].length; j++ )
                for( k = 0; k < inout_fix_tmp[i][j].length; k++ )
                    inout_fix_tmp[i][j][k]  =  i + j + k + 1;
        ODsample.arraytestPackage.fixHolder inoutHolder_fix =
            new ODsample.arraytestPackage.fixHolder( inout_fix_tmp );

        // return value attribute
        int[][][] ret_fix;

        // invoke server application method
        ret_fix = target.op1( in_fix, outHolder_fix, inoutHolder_fix );

        // display method result
        // display return value
        for( i = 0; i < ret_fix.length; i++ )
            for( j = 0; j < ret_fix[i].length; j++ )
                for( k = 0; k < ret_fix[i][j].length; k++ )
                    System.out.println( "ret_fix[" + i + "][" + j + "][" + k + "]="
                            + ret_fix[i][j][k] );

        // display out parameter
        for( i = 0; i < outHolder_fix.value.length; i++ )
            for( j = 0; j < outHolder_fix.value[i].length; j++ )
                for( k = 0; k < outHolder_fix.value[i][j].length; k++ )
                    System.out.println(
                        "outHolder_fix.value[" + i + "][" + j + "][" + k + "]="
                            +  outHolder_fix.value[i][j][k] );

        // display inout parameter
        for( i = 0; i < inoutHolder_fix.value.length; i++ )
```

```
            for( j = 0; j < inoutHolder_fix.value[i].length; j++ )
                for( k = 0; k < inoutHolder_fix.value[i][j].length; k++ )
                    System.out.println(
                      "inoutHolder_fix.value[" + i + "][" + j + "][" + k + "]="
                               + inoutHolder_fix.value[i][j][k] );

        // String array

        // in parameter attribute
        String[][][] in_str = new String[2][3][4];
        for( i = 0; i < in_str.length; i++ )
        for( j = 0; j < in_str[i].length; j++ )
        for( k = 0; k < in_str[i][j].length; k++ )
            in_str[i][j][k]  =  "indata" + i + j + k;

        // out parameter attribute
        ODsample.arraytestPackage.strHolder outHolder_str =
                      new ODsample.arraytestPackage.strHolder();

        // inout parameter attribute
        String[][][] inout_str_tmp = new String[2][3][4];
        for( i = 0; i < inout_str_tmp.length; i++ )
            for( j = 0; j < inout_str_tmp[i].length; j++ )
                for( k = 0; k < inout_str_tmp[i][j].length; k++ )
                    inout_str_tmp[i][j][k]  =  "inoutdata" + i + j + k + "snd";
        ODsample.arraytestPackage.strHolder inoutHolder_str =
            new ODsample.arraytestPackage.strHolder( inout_str_tmp );

        // return value attribute
        String[][][] ret_str;

        // invoke server application method
        ret_str = target.op2( in_str, outHolder_str, inoutHolder_str );

        // display method result
        // display return value
        for( i = 0; i < ret_str.length; i++ )
            for( j = 0; j < ret_str[i].length; j++ )
                for( k = 0; k < ret_str[i][j].length; k++ )
                    System.out.println(
                        "ret_str[" + i + "][" + j + "][" + k + "]="
                            + ret_str[i][j][k] );

        // display out parameter
        for( i = 0; i < outHolder_str.value.length; i++ )
            for( j = 0; j < outHolder_str.value[i].length; j++ )
                for( k = 0; k < outHolder_str.value[i][j].length; k++ )
                    System.out.println(
                        "outHolder_str.value[" + i + "][" + j + "][" + k + "]="
                            + outHolder_str.value[i][j][k] );

        // display inout parameter
        for( i = 0; i < inoutHolder_str.value.length; i++ )
            for( j = 0; j < inoutHolder_str.value[i].length; j++ )
                for( k = 0; k < inoutHolder_str.value[i][j].length; k++ )
                    System.out.println(
                        "inoutHolder_str.value[" + i + "][" + j + "][" + k + "]="
                            + inoutHolder_str.value[i][j][k] );
    }
    catch ( java.lang.Exception e ) {
          //error processing

        }
```

```
    }
}
```

Processing by Server Applications

When you assign data to an inout or out parameter, set it in the inout or out Holder class value.

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// Servant class
// server application method
class arrayServant extends arraytestPOA {

    public int[][][] op1( int[][][] in_fix,
                        ODsample.arraytestPackage.fixHolder outHolder_fix,
                        ODsample.arraytestPackage.fixHolder inoutHolder_fix )
    {
        int i, j, k;

        // display input parameter
        // display in parameter
        for( i = 0; i < in_fix.length; i++ )
            for( j = 0; j < in_fix[i].length; j++ )
                for( k = 0; k < in_fix[i][j].length; k++ )
                    System.out.println(
                        "in_fix[" + i + "][" + j + "][" + k + "]=" + in_fix[i][j][k] );

        // display inout parameter
        for( i = 0; i < inoutHolder_fix.value.length; i++ )
            for( j = 0; j < inoutHolder_fix.value[i].length; j++ )
                for( k = 0; k < inoutHolder_fix.value[i][j].length; k++ )
                    System.out.println(
                        "inoutHolder_fix.value[" + i + "][" + j + "][" + k + "]="
                            + inoutHolder_fix.value[i][j][k] );

        // set output parameter
        // set out parameter
        int[][][] out_fix_tmp = new int[4][3][2];
        for( i = 0; i < out_fix_tmp.length; i++ )
            for( j = 0; j < out_fix_tmp[i].length; j++ )
                for( k = 0; k < out_fix_tmp[i][j].length; k++ )
                    out_fix_tmp[i][j][k]  =  ( i + j + k ) * 10;
        outHolder_fix.value = out_fix_tmp;

        // set inout parameter
        for( i = 0; i < inoutHolder_fix.value.length; i++ )
            for( j = 0; j < inoutHolder_fix.value[i].length; j++ )
                for( k = 0; k < inoutHolder_fix.value[i][j].length; k++ )
                    inoutHolder_fix.value[i][j][k]  =
                        inoutHolder_fix.value[i][j][k] * 20;

        // set return value
        int[][][] ret_fix = new int[4][3][2];
        for( i = 0; i < ret_fix.length; i++ )
            for( j = 0; j < ret_fix[i].length; j++ )
                for( k = 0; k < ret_fix[i][j].length; k++ )
                    ret_fix[i][j][k]  =  ( i + j + k ) * 100;

        // return value
        return( ret_fix );
```

```
    }

    public String[][][] op2( String[][][] in_str,
                             ODsample.arraytestPackage.strHolder outHolder_str,
                             ODsample.arraytestPackage.strHolder inoutHolder_str )
    {
        int i, j, k;

        // display input parameter
        // display in parameter
        for( i = 0; i < in_str.length; i++ )
            for( j = 0; j < in_str[i].length; j++ )
                for( k = 0; k < in_str[i][j].length; k++ )
                    System.out.println(
                        "in_str[" + i + "][" + j + "][" + k + "]=" + in_str[i][j][k] );

        // display inout parameter
        for( i = 0; i < inoutHolder_str.value.length; i++ )
            for( j = 0; j < inoutHolder_str.value[i].length; j++ )
                for( k = 0; k < inoutHolder_str.value[i][j].length; k++ )
                    System.out.println(
                        "inoutHolder_str.value[" + i + "][" + j + "][" + k + "]="
                            + inoutHolder_str.value[i][j][k] );

        // set output parameter
        // set out parameter
        String[][][] out_str_tmp = new String[2][3][4];
        for( i = 0; i < out_str_tmp.length; i++ )
            for( j = 0; j < out_str_tmp[i].length; j++ )
                for( k = 0; k < out_str_tmp[i][j].length; k++ )
                    out_str_tmp[i][j][k]  =  "outdata" + i + j + k;
        outHolder_str.value = out_str_tmp;

        // set inout parameter
        for( i = 0; i < inoutHolder_str.value.length; i++ )
            for( j = 0; j < inoutHolder_str.value[i].length; j++ )
                for( k = 0; k < inoutHolder_str.value[i][j].length; k++ )
                    inoutHolder_str.value[i][j][k]  =
                                    inoutHolder_str.value[i][j][k] + "-rcv";

        // set return value
        String[][][] ret_str = new String[2][3][4];
        for( i = 0; i < ret_str.length; i++ )
            for( j = 0; j < ret_str[i].length; j++ )
                for( k = 0; k < ret_str[i][j].length; k++ )
                    ret_str[i][j][k]  =  "retdata" + i + j + k;

        // return value
        return( ret_str );
    }
}


public class arrayServer {
    public static void main(String args[]) {
        try {
            // preprocess ORB

            // create POA object

            // create Servant and register to its POA

            // activate POAManager
```

```
        }
        catch (java.lang.Exception e) {
            //error processing

        }
    }
}
```

6.19.10 Mapping Attribute Declaration (Attribute)

This section provides information on mapping attribute declaration.

IDL Mapping

When an object is specified as attribute in the IDL language, it is mapped in Java language to member attributes that have the same name as the attribute name and methods for setting/getting data for those attributes are generated.

The following is a sample application program with the IDL defined below.

IDL

```
module ODsample{
    interface attrtest{
        attribute long para1;
        attribute string para2;
        readonly attribute long para3;
    };
};
```

Written in Java language, this comes out as follows:

Java Language

<Interface>

```
package ODsample;
public interface attrtestOperations
{
    public int para1();
    public void para1( int value );
    public java.lang.String para2();
    public void para2( java.lang.String value );
    public int para3();
}
```

<Holder class>

```
package ODsample;
public final class attrtestHolder  implements org.omg.CORBA.portable.Streamable {
    public ODsample.attrtest value;
    public attrtestHolder() {...}
    public attrtestHolder(ODsample.attrtest __arg) {...}
    public void _write(org.omg.CORBA.portable.OutputStream out) {...}
    public void _read(org.omg.CORBA.portable.InputStream in) {...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

<Helper class>

```
package ODsample;
public class attrtestHelper
{
    public static void
        insert(org.omg.CORBA.Any a, ODsample.attrtest value){...}
    public static ODsample.attrtest
        extract(org.omg.CORBA.Any a){...}
    public static String
        id(){...}
    public java.lang.Object
        read_Object(org.omg.CORBA.portable.InputStream istream){...}
    public void
        write_Object(org.omg.CORBA.portable.OutputStream ostream, Object value){...}
    public java.lang.String
        get_id( ){...}
    public org.omg.CORBA.TypeCode
        get_type(){...}
    private static org.omg.CORBA.TypeCode _type;
    synchronized public static org.omg.CORBA.TypeCode
        type(){...}
    public static ODsample.attrtest
        read(org.omg.CORBA.portable.InputStream istream){...}
    public static void
        write(org.omg.CORBA.portable.OutputStream ostream,
            ODsample.attrtest value){...}
    public static ODsample.attrtest
        narrow( org.omg.CORBA.Object obj)
            throws org.omg.CORBA.BAD_PARAM{...}
    private static ODsample.attrtest
        narrow( org.omg.CORBA.Object obj, boolean ignore_check)
            throws org.omg.CORBA.BAD_PARAM{...}
}
```

Processing by Client Applications

Run direct setting and extraction of server object data.

```
import org.omg.CORBA.*;
import ODsample.*;

public class attributeClient {
    public static void main(String args[]) {
        // preprocess ORB
        // get object references

        try{
            // set server application attributes
            java.util.Date  dt = new java.util.Date();
            target.para1( (int)dt.getTime() );
            target.para2( dt.toString() );

            // reference server application attributes
            System.out.println( target.para1() );
            System.out.println( target.para2() );
            System.out.println( target.para3() );

        }
        catch ( Exception e ) {
            //error processing

        }
```

```
    }
}
```

Processing by Server Applications

You implement referencing/setting processing for interface class attributes with the Servant class of a server application.

```
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import ODsample.*;

// Servant class
// server application method
class attributeServant extends attrtestPOA {

    // initialize attributes
    private int    value1 = 0;
    private String value2 = "";
    private int    value3 = 999;

    public int para1(){
        return( value1 );
    }
    public void para1(int arg){
        value1 = arg;
    }
    public String para2(){
        return( value2 );
    }
    public void para2( String arg ){
        value2 = arg;
    }
    public int para3(){
        return( value3 );
    }
}

public class attributeServer {
    public static void main(String args[]) {
        try {
            // preprocess ORB

            // create POA object

            // create Servant and register with its POA

            // activate POAManager

        }
        catch (java.lang.Exception e) {
            //error processing

        }
    }
}
```

6.19.11 Mapping Constant Declaration (const)

This section provides information on mapping constant declaration.

IDL Mapping

When constant (const) is specified in IDL language within an interface declaration, it is mapped in Java language as a public static final method (constant method) with a name identical to the constant name within the public interface of the interface name.

```
[package <packageName>];
public interface <interfaceName> {
    public static final <type> <constName> = (<type>)(<value>);
}
```

IDL

```
module Example{
    interface Face{
        const long aLongOne = -321;
    };
};
```

Written in Java language, this comes out as follows:

Java Language

```
package Example;
public interface FaceOperations {
    public final int aLongOne = (int) -321;
}
```

When specified outside of the interface declarations, the public static final method is defined with the name of value on the inside of the public interface with the same name as the constant name.

```
[package <moduleName>];
public interface <constName> {
    public static final <type> value = (<type>)(<value>);
}
```

Refer to the following example.

IDL

```
module Example{
    const long aLongOne = -321;
};
```

Written in Java language, this comes out as follows:

Java Language

```
package Example;
    public interface aLongOne {
    final public static int value = (int) -321;
}
```

6.19.12 Passing Parameters to Server Applications

The following table lists the valid data types for passing parameters between client and server applications.

Table 6.31 Valid Data Types for Passing Parameters

CORBA Data Type	In	Out	Inout	Return
Long	int	org.omg.CORBA. IntHolder	org.omg.CORBA. IntHolder	int
unsigned long	int	org.omg.CORBA. IntHolder	org.omg.CORBA. IntHolder	int
Short	short	org.omg.CORBA. ShortHolder	org.omg.CORBA. ShortHolder	short
unsigned short	short	org.omg.CORBA. ShortHolder	org.omg.CORBA. ShortHolder	short
long long	long	org.omg.CORBA. LongHolder	org.omg.CORBA LongHolder	long
Float	float	org.omg.CORBA. FloatHolder	org.omg.CORBA. FloatHolder	float
Double	double	org.omg.CORBA. DoubleHolder	org.omg.CORBA. DoubleHolder	double
Boolean	boolean	org.omg.CORBA. BooleanHolder	org.omg.CORBA. BooleanHolder	boolean
Char	char	org.omg.CORBA. CharHolder	org.omg.CORBA. CharHolder	char
Wchar	char	org.omg.CORBA. CharHolder	org.omg.CORBA. CharHolder	char
Octet	byte	org.omg.CORBA. ByteHolder	org.omg.CORBA. ByteHolder	byte
enum	data name class	data name Holderclass	data name Holderclass	data name class
struct	data name class	data name Holderclass	data name Holderclass	data name class
union	data name class	data name Holderclass	data name Holderclass	data name class
string	java.lang.String	org.omg.CORBA. StringHolder	org.omg.CORBA. StringHolder	java.lang.String
wstring	java.lang.String	org.omg.CORBA. StringHolder	org.omg.CORBA. StringHolder	java.lang.String
any	org.omg.CORBA. Any	org.omg.CORBA. CORBA.AnyHolder	org.omg.CORBA. CORBA.AnyHolder	org.omg.CORBA. CORBA.Any
sequence	elements type	dataname Holderclass	dataname Holderclass	elements type

CORBA Data Type	In	Out	Inout	Return
array	array of elements type	dataname Holderclass	dataname Holderclass	array of elements type
Object	org.omg.CORBA. Object	org.omg.CORBA. ObjectHolder	org.omg.CORBA. ObjectHolder	org.omg.CORBA. Object
TypeCode	org.omg.CORBA. TypeCode	org.omg.CORBA. TypeCodeHolder	org.omg.CORBA. TypeCodeHolder	org.omg.CORBA. TypeCode

Handling String Type Objects

When handling a string type in Java you need to set not just "null" but "null string" as an object value at the time you set null as a character string. Refer to the example below:

The following is an incorrect example:

```
java.lang.String str = null;
```

The following is a correct example:

```
java.lang.String str2 = new java.lang.String("");
```

Handling Attributes

When handling attributes in Java you need to be careful, even when handling attributes made unsigned by IDL definition, that the value you assign falls within the following ranges because there is no true unsigned type in Java.

- int: -2147483648 to +2147483647
- short: -32768 to +32767
- long: -9223372036854775808 to +9223372036854775807

Handling char Type

The char type in Java is equivalent to Unicode (16bit).

Handling Japanese Characters

To use char and string in IDL definition, characters and character-strings of 1-byte codes are available to deliver. To deliver Japanese characters and character-strings, wchar and wstring must be defined in IDL.

Setting Null Objects

Null objects cannot be set to string type, sequence type, structs, unions, arrays and any in the cases of in parameter and inout parameter in client applications and out parameter, inout parameter and return value in server applications.

Holder-class Handling

To pass parameters using the inout parameter in a client application or out parameter in a server application, assign a value to the value variable in the Holder class instance before passing the parameters.

6.20 Notes

6.20.1 Notes on Application Development

Refer to the "Notes on Developing CORBA Applications" chapter.

6.20.2 Notes on Applet Operation in the Client Environment

- If you are using Microsoft(R) Internet Explorer 7.0 on Windows Server(R) 2008, Windows Server(R) 2012, Windows Vista(R), Windows(R) 7 and Windows(R) 8 protection mode must be disabled in the client used for storing log information.
- When the JBK plug-in is used, the HTTPS protocol can be used.
- When repeating activation/termination of the same applet (applet downloaded from the same URL) without terminating the Java VM that was started while using Portable-ORB, use org.omg.CORBA.ORB.destroy() to explicitly release resources being held by ORB instances.

  If the resources are not explicitly released, an exception may occur during communication with the server application.

  When activating/terminating a browser using JavaScript or when using Internet Explorer with an active desk top installed, be sure to free the communication resources because Java VM will not be terminated by termination of the browser.
- When applets are used, digital signatures are required. For an example of applying a digital signature, refer to "6.7.3 Digital Signature Procedure".
- When using Portable-ORB applets, note the following:
  - When an Any type is generated using org.omg.CORBA.ORB.create_any() in a user application, an object reference created by org.omg.CORBA.ORB.init() with an argument must be used.
  - When a union type is used, a default member cannot be specified in the IDL definition.
  - An enum type cannot be specified as a union type discriminator type. Specify a type other than enum.

6.20.3 Other Precautions

The following precautions should be taken:

- Application classes compiled (with javac) using a different JDK execution environment and version may not operate correctly in a JDK/JRE execution environment. These should be re-compiled (with javac) using the same JDK version and environment as the execution environment, before re-execution.

Windows32/64

- If ODjava4.jar set in the CLASSPATH environment variable is changed to an alias when this product is installed, the setting is not deleted even if the product is uninstalled. If the name is not required, delete it from the CLASSPATH setting.
- If you start an application without the -Xrs option, the application will terminate when you log off Windows. To leave CORBA-Java applications and EJB applications running on the CORBA WorkUnit even after log off, specify the -Xrs option.

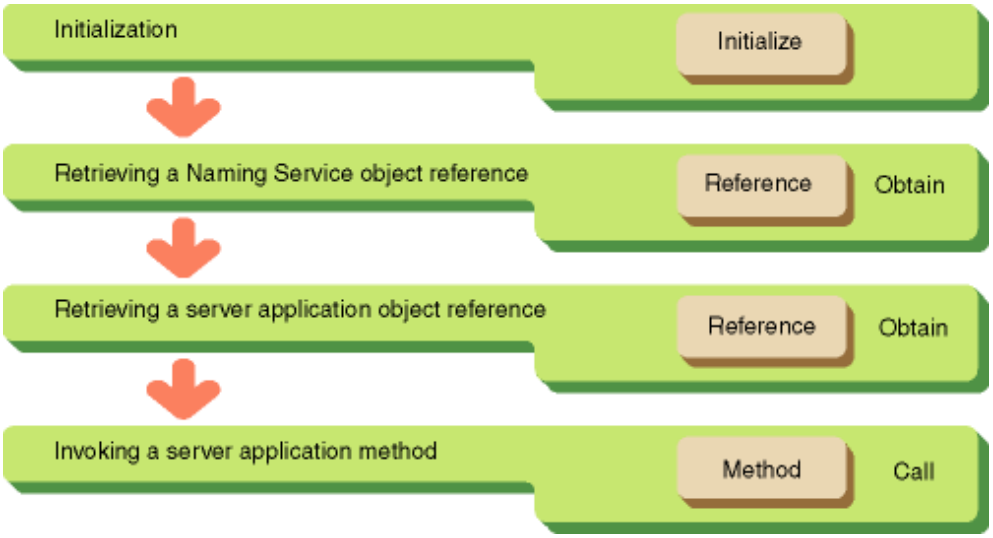
Chapter 7 COBOL Programming Guide

This chapter explains how to develop CORBA applications in COBOL.

7.1 Client Application Programming (Static Invocation Interface)

The following figure shows the actions of a client application using static invocation.

Figure 7.1 Client Application (Static Invocation Interface) Process Flow



7.1.1 Initialization

To initialize ObjectDirector, invoke *CORBA-ORB-INIT*. An ORB Object Reference will be returned. This object reference will be used whenever an ORB interface is invoked. Then, define the environment division (ENVIRONMENT DIVISION) and the data division (DATA DIVISION).

IDENTIFICATION DIVISION.

```
PROGRAM-ID.  "CLIENT-MAIN".
 AUTHOR.  OD/IDLCOMPILER VER.2.0.
 INSTALLATION.  IDL FILE NAME IS COBSAMPLE.IDL.
 SECURITY.  THIS SOURCE CODE WAS GENERATED BASE ON YOUR IDL FILE.
           WHEN THIS STUB/SKELETON SOURCE CODE IS CHANGED, THE OPERATION.
           GURANTEED IS NOT DONE.
 DATE-WRITTEN.  TUE MAY  6 11:03:40 1997
```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.

```
     ARGUMENT-NUMBER   IS ARG-C
     ARGUMENT-VALUE    IS ARG-V
     SYMBOLIC CONSTANT
     COPY SYMBOL-CONST IN CORBA.
     .
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
 01 TEMP-BUF USAGE POINTER.
 01 MESS PIC X(30).
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
```

```
 01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
 01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
 01 COPY BOA IN CORBA REPLACING CORBA-BOA BY BOA.
 01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
 01 COPY BOOLEAN IN CORBA REPLACING CORBA-BOOLEAN BY RET.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY I.
 01 BUFFER USAGE POINTER.
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY BUF-LENGTH.
 01 STR-BUF PIC X(30).
 01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING
        COSNAMING-NAMINGCONTEXT BY COS-NAMING.
 01 COPY COSNAMING-NAME IN CORBA REPLACING COSNAMING-NAME BY NAME.
 01 NAME-A USAGE POINTER.
 01 COPY COSNAMING-NAMECOMPONENT IN CORBA REPLACING
        COSNAMING-NAMECOMPONENT BY NAME-COMPONENT.
 01 NAME-COMPONENT-A USAGE POINTER.
*########## ORB SETTING PARAMETER #########
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY CURRENT-ARG-C.
 01 CURRENT-ARG-V.
    02 FILLER OCCURS 6.
       03 CURRENT-ARG-V-VALUE USAGE POINTER.
 01 APLI-NAME PIC X(8) VALUE "simple_c".
 01 TMP-STRING-BUF PIC X(20).
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY ARG-COUNT.
```

PROCEDURE DIVISION.

```
MAIN.
*  Setting an ObjectDirector object ID
     DISPLAY "CLIENT START!!".
* argument set : CURRENT-ARG-V-VALUE
     ACCEPT CURRENT-ARG-C FROM ARG-C.
     COMPUTE CURRENT-ARG-C = CURRENT-ARG-C + 1.
     PERFORM VARYING ARG-COUNT FROM 1 BY 1 UNTIL ARG-COUNT > CURRENT-ARG-C
        IF ARG-COUNT = 1
           MOVE APLI-NAME TO TMP-STRING-BUF
        ELSE
           ACCEPT TMP-STRING-BUF FROM ARG-V
        END-IF
        MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH
        CALL "CORBA-STRING-SET" USING
              CURRENT-ARG-V-VALUE (ARG-COUNT)
              STRING-LENGTH
              TMP-STRING-BUF
     END-PERFORM.
     SET CURRENT-ARG-V-VALUE (ARG-COUNT) TO NULL.
*  Initializing ObjectDirector
     MOVE 12 TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           FJ-OM-ORB-ID.
     CALL "CORBA-ORB-INIT" USING
           CURRENT-ARG-C
           CURRENT-ARG-V
           TEMP-BUF
           ENV
           ORB.
     CALL "CORBA-FREE" USING TEMP-BUF.
     MOVE "CORBA-ORB-INIT" TO MESS.
     PERFORM ENV-CHECK
```

```
     MOVE 15 TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           CORBA-BOA-OA-ID.
     CALL "CORBA-ORB-BOA-INIT" USING
           ORB
           CURRENT-ARG-C
           CURRENT-ARG-V
           TEMP-BUF
           ENV
           BOA.
*  Releasing the area stored in ORBID
     CALL "CORBA-FREE" USING TEMP-BUF.
     MOVE "CORBA-ORB-BOA-INIT" TO MESS.
     PERFORM ENV-CHECK.
```

Refer to "7.3 Client Application Exception Processing" for details of exception processing.

7.1.2 Retrieving a Naming Service Object Reference

To retrieve a Naming Service object reference use *CORBA-ORB-RESOLVE-INITIAL-REFERENCE*. To retrieve the object reference, specify *CORBA-ORB-OBJECTID-NAMESERVICE* as a parameter. This constant is defined in the include file, or in other files provided by your ORB vendor.

```
*  Setting a naming service object ID
     MOVE FUNCTION LENG ( CORBA-ORB-OBJECTID-NAMESERVICE ) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           CORBA-ORB-OBJECTID-NAMESERVICE.
*  Retrieving a naming service object reference
     CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
           ORB
           TEMP-BUF
           ENV
           COS-NAMING.
*  Releasing the area stored in NAMESERVICEID
     CALL "CORBA-FREE" USING TEMP-BUF.
     MOVE "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" TO MESS.
     PERFORM ENV-CHECK.
```

7.1.3 Retrieving a Server Application Object Reference

To extract a target object use the Naming Service method *COSNAMING-NAMINGCONTEXT-RESOLVE*.retrieveobject reference(). Specify the target object name as a method parameter.

```
*  Object name
     MOVE FUNCTION LENG (STR-BUF) TO STRING-LENGTH.
     MOVE "ODdemo::calculator" TO STR-BUF.
     CALL "CORBA-STRING-SET" USING
           IDL-ID OF NAME-COMPONENT
           STRING-LENGTH
           STR-BUF.
*  Object type
     MOVE " " TO STR-BUF.
     CALL "CORBA-STRING-SET" USING
           KIND OF NAME-COMPONENT
           STRING-LENGTH
           STR-BUF.
*  Number of object names
     MOVE 1 TO SEQ-LENGTH OF NAME.
```

```
     MOVE 1 TO SEQ-MAXIMUM OF NAME.
*  Number of sequence elements
     MOVE 1 TO NUM.
*  Allocating a sequence area for storing CosNaming access information
     CALL "CORBA-SEQUENCE-COSNAMING-NAMECOMPONENT-ALLOCBUF" USING
           SEQ-MAXIMUM OF NAME
           SEQ-BUFFER OF NAME.
     MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
     MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
*  Setting a CosNaming element
     CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
           NAME-A
           NUM
           NAME-COMPONENT-A.
*  Retrieving a server application object reference
     CALL "COSNAMING-NAMINGCONTEXT-RESOLVE" USING
              COS-NAMING
              NAME
              ENV
              OBJ.
     MOVE "COSNAMING-NAMINGCONTEXT-RESOLVE" TO MESS.
     PERFORM ENV-CHECK.
```

7.1.4 Invoking a Method

To invoke a method implemented in the server application, specify the method name with the following:

Module name

Interface name

Method names specified in IDL

using hyphen ("-") to concatenate the three elements.

In the following example, ODDEMO, CALCULATOR, and CALCULATE are the module, interface, and method names, respectively.

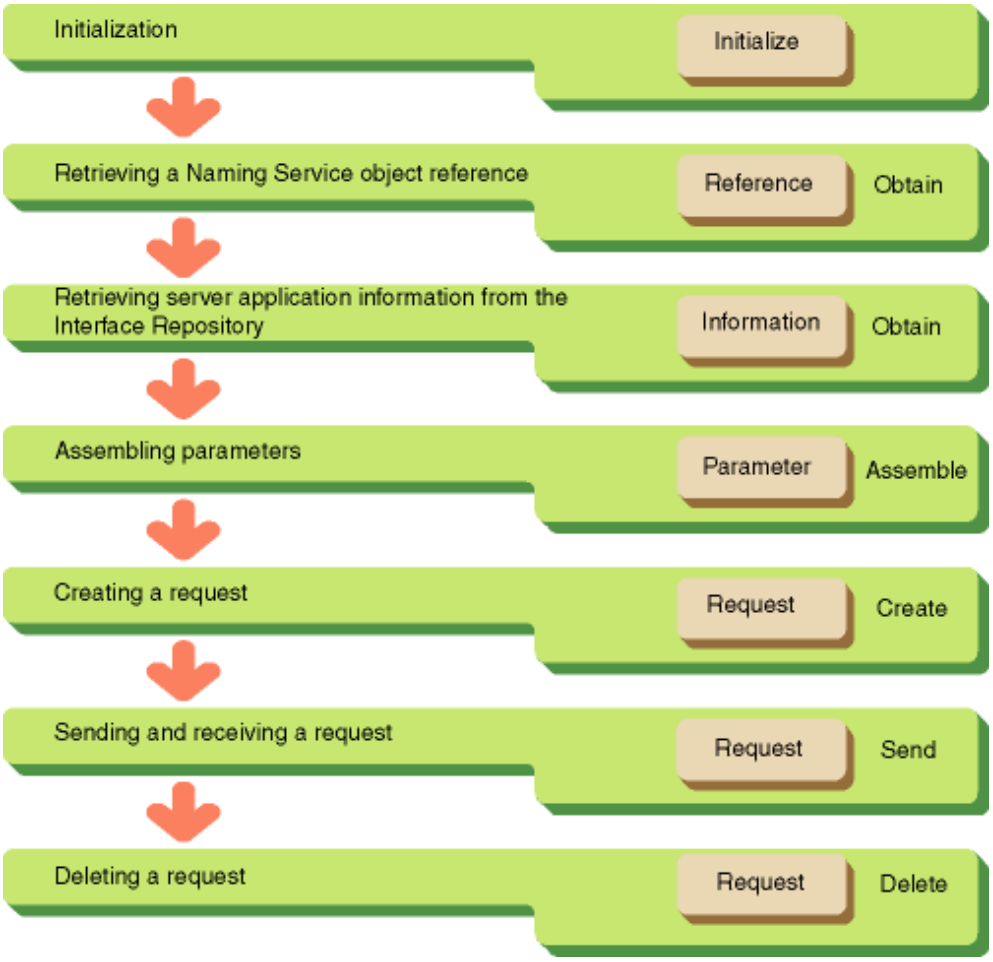
Specify the CORBA::Environment structure to which any exception information will be returned, in the event of an error occurring when invoking the method.object reference.

```
     MOVE 100 TO PARAM1.
     MOVE  20 TO PARAM2.

     CALL "ODDEMO-CALCULATOR-CALCULATE" USING
           OBJ
           PARAM1
           PARAM2
           ENV
           A-RESULT.
     MOVE "ODdemo_calculator_calculate" TO MESS.
     PERFORM ENV-CHECK.
```

7.2 Client Application Programming (Dynamic Invocation Interface)

The following figure shows client application processing using a dynamic invocation interface.

Figure 7.2 Client Application (Dynamic Invocation Interface) Process Flow



7.2.1 Initialization

Invoke *CORBA-ORB-INIT* to initialize the application. An ORB object reference is returned. This object reference will be used whenever an ORB interface is invoked.

IDENTIFICATION DIVISION.

```
PROGRAM-ID.  "CLIENT-MAIN".
 AUTHOR.  OD/IDLCOMPILER VER.2.0.
 INSTALLATION.  IDL FILE NAME IS COBSAMPLE.IDL.
 SECURITY.  THIS SOURCE CODE WAS GENERATED BASE ON YOUR IDL FILE.
           WHEN THIS STUB/SKELETON SOURCE CODE IS CHANGED, THE OPERATION.
           GURANTEED IS NOT DONE.
 DATE-WRITTEN.  TUE MAY  6 11:03:40 1997
```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.

```
     ARGUMENT-NUMBER   IS ARG-C
     ARGUMENT-VALUE    IS ARG-V
     SYMBOLIC CONSTANT
     COPY SYMBOL-CONST IN CORBA.
     .
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
 01 TEMP-BUF USAGE POINTER.
 01 MESS PIC X(30).
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
 01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
 01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
 01 COPY BOA IN CORBA REPLACING CORBA-BOA BY BOA.
 01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
 01 COPY BOOLEAN IN CORBA REPLACING CORBA-BOOLEAN BY RET.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY I.
 01 BUFFER USAGE POINTER.
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY BUF-LENGTH.
 01 STR-BUF PIC X(30).
 01 COPY COSNAMING-NAMINGCONTEXT IN CORBA
    REPLACING COSNAMING-NAMINGCONTEXT BY COS-NAMING.
 01 COPY COSNAMING-NAME IN CORBA REPLACING COSNAMING-NAME BY NAME.
 01 NAME-A USAGE POINTER.
 01 COPY COSNAMING-NAMECOMPONENT IN CORBA
    REPLACING COSNAMING-NAMECOMPONENT BY NAME-COMPONENT.
 01 NAME-COMPONENT-A USAGE POINTER.
 01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY INTF-INTF.
 01 COPY DEFINITIONKIND IN CORBA REPLACING CORBA-DEFINITIONKIND BY DK-FLAG.
 01 INTF-OPR USAGE POINTER.
 01 DESCRIPTION USAGE POINTER.
*########## ORB SETTING PARAMETER #########
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY CURRENT-ARG-C.
 01 CURRENT-ARG-V.
 02 FILLER OCCURS 6.
 03 CURRENT-ARG-V-VALUE USAGE POINTER.
 01 APLI-NAME PIC X(8) VALUE "simple_c".
 01 TMP-STRING-BUF PIC X(20).
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY ARG-COUNT.
```

PROCEDURE DIVISION.

```
MAIN.
*  Setting an ObjectDirector object ID
     DISPLAY "CLIENT START!!".
* argument set : CURRENT-ARG-V-VALUE
     ACCEPT CURRENT-ARG-C FROM ARG-C.
     COMPUTE CURRENT-ARG-C = CURRENT-ARG-C + 1.
     PERFORM VARYING ARG-COUNT FROM 1 BY 1 UNTIL ARG-COUNT > CURRENT-ARG-C
        IF ARG-COUNT = 1
           MOVE APLI-NAME TO TMP-STRING-BUF
        ELSE
           ACCEPT TMP-STRING-BUF FROM ARG-V
        END-IF
        MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH
        CALL "CORBA-STRING-SET" USING
              CURRENT-ARG-V-VALUE (ARG-COUNT)
              STRING-LENGTH
              TMP-STRING-BUF
     END-PERFORM.
     SET CURRENT-ARG-V-VALUE (ARG-COUNT) TO NULL.
*  Initializing ObjectDirector
     MOVE 12 TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           FJ-OM-ORB-ID.
     CALL "CORBA-ORB-INIT" USING
           CURRENT-ARG-C
```

```
           CURRENT-ARG-V
           TEMP-BUF
           ENV
           ORB.
*  Releasing the area stored in ORBID
     CALL "CORBA-FREE" USING TEMP-BUF.
     MOVE "CORBA-ORB-INIT" TO MESS.
     PERFORM ENV-CHECK
```

Refer to "7.3 Client Application Exception Processing" for details of exception processing.

7.2.2 Retrieving a Naming Service Object Reference

To retrieve a Naming Service object reference use *CORBA-ORB-OBJECTID-NAMESERVICEretrieveobject reference*. Specify *CORBA-ORB-OBJECTID-NAMESERVICE* as a parameter. This is shown in the following example:

```
*  Setting a naming service object ID
     MOVE FUNCTION LENG ( CORBA-ORB-OBJECTID-NAMESERVICE ) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           CORBA-ORB-OBJECTID-NAMESERVICE.
*  Retrieving a naming service object reference
     CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
           ORB
           TEMP-BUF
           ENV
           COS-NAMING.
*  Releasing the area stored in NAMESERVICEID
     CALL "CORBA-FREE" USING TEMP-BUF.
     MOVE "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" TO MESS.
     PERFORM ENV-CHECK.
```

7.2.3 Retrieving Server Application Information from the Interface Repository

Server application information for the entire domain is stored in the Interface Repository. Module names, interface names, method names and parameters defined in IDL are stored hierarchically in the Interface Repository.

To obtain server application information, follow the procedures outlined in this section.

Retrieving an InterfaceDef Object Reference

To Retrieve interface information from the Interface Repository, an *INTERFACEDEF* object reference is needed. To obtain this, use the object name of the server application to search Naming Service for the server application object reference. Then, use *OBJECT-GET-INTERFACE* to retrieve the *INTERFACEDEF* object reference.

```
*  Object name
     MOVE FUNCTION LENG (STR-BUF) TO STRING-LENGTH.
     MOVE "ODdemo::calculator" TO STR-BUF.
     CALL "CORBA-STRING-SET" USING
           IDL-ID OF NAME-COMPONENT
           STRING-LENGTH
           STR-BUF.
*  Object type
     MOVE " " TO STR-BUF.
     CALL "CORBA-STRING-SET" USING
           KIND OF NAME-COMPONENT
           STRING-LENGTH
           STR-BUF.
*  Number of object names
     MOVE 1 TO SEQ-LENGTH OF NAME.
```

```
     MOVE 1 TO SEQ-MAXIMUM OF NAME.
*  Number of sequence elements
     MOVE 1 TO NUM.
*  Allocating a sequence area for storing CosNaming access information
     CALL "CORBA-SEQUENCE-COSNAMING-NAMECOMPONENT-ALLOCBUF" USING
           SEQ-MAXIMUM OF NAME
           SEQ-BUFFER OF NAME.
     MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
     MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
*  Setting a CosNaming element
     CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
           NAME-A
           NUM
           NAME-COMPONENT-A.
*  Retrieving a server application object reference
     CALL "COSNAMING-NAMINGCONTEXT-RESOLVE" USING
              COS-NAMING
              NAME
              ENV
              OBJ.
     MOVE "COSNAMING-NAMINGCONTEXT-RESOLVE" TO MESS.
     PERFORM ENV-CHECK.
*  Collecting a server application interfaceDef
   CALL "CORBA-OBJECT-GET-INTERFACE" USING
         OBJ
         ENV
         INTF-INTF.
```

Retrieving an OperationalDef Object Reference

To search for a specified method in the Interface Repository use *CORBA-INTERFACEDEF-LOOKUP-NAME*. Retrieve the information from the Interface Repository. Specify the server application method name as a parameter. An *OPERATIONDEF* object reference will be returned, containing information about the specified method.

```
*  Entering a method name
   DISPLAY " Please input method name==>".
   ACCEPT BUF.
*  Retrieving an object reference to collect operation information
   SET CORBA-DK-OPERATION OF DK-FLAG TO TRUE.
   MOVE -1 TO NUM.
   MOVE CORBA-FALSE TO BL.
   CALL "CORBA-INTERFACEDEF-LOOKUP-NAME" USING
        INTF-INTF
        BUF
           NUM
           DK-FLAG
           BL
           ENV
           INTF-OPR.
```

Retrieving Parameter Information

To search the Interface Repository for the parameter information of a method implemented in the server application, use *CORBA-OPERATIONDEF-DESCRIBE*. Retrieve the information from the Interface Repository. Parameter information includes parameter names, number of parameters, and parameter types. Specify an *OPERATIONDEF* object reference as a parameter.

```
*  Collecting a parameter information structure
      MOVE 1 TO NUM.
      CALL "CORBA-SEQUENCE-ELEMENT-GET" USING
            INTF-OPR
            NUM
            WORK-POINTER.
      SET ADDRESS OF TMP-CONT TO WORK-POINTER.
```

```
    CALL "CORBA-OPERATIONDEF-DESCRIBE" USING
          TMP-CONT
             ENV
             DESCRIPTION.
```

7.2.4 Assembling Parameters

This section provides information on assembling parameters.

(1) Creating a Parameter List

To create a list object, which stores the parameters to be passed to the server, use *CORBA-ORB-CREATE-LIST*. Specify the parameters to be stored as a parameter. An *NVList* object reference is then returned. Refer to "NVList Object" in the "CORBA Interface" chapter for further details.

```
*  Extracting a parameter structure from the Contained-Description structure
   SET ADDRESS OF TMP-DESC TO DESCRIPTION.
   MOVE IDL-VALUE OF TMP-DESC TO PARAM-ANY.
*  Extracting parameter information
   SET ADDRESS OF TMP-OPR-DESC TO ANY-VALUE OF PARAM-ANY.
   MOVE PARAMETERS OF TMP-OPR-DESC TO PARAMS.
   MOVE LENGTH OF PARAMS TO NUM.
*  Creating a list object
   CALL "CORBA-ORB-CREATE-LIST" USING
     ORB
     NUM
     ARG-LIST
     ENV
     RET-VAL.
```

(2) Assigning Parameters to the List

To assign a parameter to the list to be passed to the server use *CORBA-NVLIST-ADD-ITEM()*. Specify the object reference for *CORBA::NVLIST*, the name of the server application, type, value, and length as parameters.

```
   MOVE FUNCTION ADDR(PARAMS) TO SEQ-POINTER.
                 MOVE "in_p" TO STR-BUF.
                 MOVE FUNCTION LENG (STR-BUF) TO STRING-LENGTH.
                 CALL "CORBA-STRING-SET" USING
                    STRING-POINTER
                    STRING-LENGTH
                    STR-BUF.
                 MOVE FUNCTION LENG (TC-LONG) TO STRING-LENGTH.
                 CALL "CORBA-STRING-SET" USING
                    TEMP-BUF
                    STRING-LENGTH
                    TC-LONG.
                 CALL "CORBA-ORB-TYPECODE-FROM-CGEN-TC" USING
                    TEMP-BUF
                    TMP-TYPE.
                 CALL "CORBA-FREE" USING
                    TEMP-BUF.
                 MOVE 1234 TO IN-P.
                 MOVE FUNCTION ADDR( IN-P ) TO PARAM-POINTER.
                 MOVE LSIZE OF TMPDATA TO NUM.
                 SET CORBA-ARG-IN OF FLAG TO TRUE.
                 CALL "CORBA-NVLIST-ADD-ITEM" USING
                 ARG-LIST
                 STRING-POINTER
                    TMP-TYPE
                    PARAM-POINTER
                    NUM
```

```
                    FLAG
                    ENV
                    RET-VAL.
```

7.2.5 Creating a Request

To create a request object use *CORBA-OBJECT-CREATE-REQUEST*. Specify an object reference for the target object, the *NVList* and NamedValue to store the returned value. The object reference for the request object will be returned. Return values from the server are stored in the NamedValue location.

```
     MOVE FUNCTION ADDR(RESULT) TO PARAM-POINTER.
     MOVE "calculate" TO STR-BUF.
     MOVE FUNCTION LENG (STR-BUF) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           STRING-POINTER
           STRING-LENGTH
           STR-BUF.

     SET CORBA-OUT-LIST-MEMORY OF FLAG TO TRUE.

     CALL "CORBA-OBJECT-CREATE-REQUEST" USING
           OBJ
           CTX
           STRING-POINTER
           ARG-LIST
           PARAM-POINTER
           REQ
           FLAG
           ENV
           RET-VAL.
```

The following is the IDL file definition for NamedValue.

```
module CORBA {
     typedef string          Identifier;
     enum Flags {
          ARG_IN,
          ARG_OUT,
          ARG_INOUT,
          OUT_LIST_MEMORY,
          IN_COPY_VALUE,
          INV_NO_RESPONSE,
          INV_TERM_ON_ERR,
          RESP_NO_WAIT,
          DEPENDENT_LIST,
          CTX_RESTRICT_SCOPE,
          CTX_DELETE_DESCENDENTS
     };
     struct NamedValue {
          identifier     name;       // Parameter name
          any     argument;          // Parameter value
          long     len;              // Parameter length
          Flags     arg_modes;       // Parameter delivery method (in, out, or inout)
     };
};
```

The following is the library in COBOL.

```
*  NamedValue structure
     CORBA-NAMED-VALUE.
*  Parameter name
     07  NAME     USAGE   POINTER.
*  Parameter value
```

```
     07  ARGUMENT.
         09 ANY-TYPE      USAGE POINTER.
         09 ANY-VALUE     USAGE POINTER.
*  Parameter length
     07  LEN      PIC     S9(9) COMP-5.
*  Parameter delivery method (in, out, or inout)
     07  ARG-MODES        PIC     9(9) COMP-5.
         88  CORBA-ARG-IN                 VALUE   1.
         88  CORBA-ARG-OUT                VALUE   2.
         88  CORBA-ARG-INOUT              VALUE   4.
         88  CORBA-OUT-LIST-MEMORY        VALUE   8.
         88  CORBA-IN-COPY-VALUE          VALUE   16.
         88  CORBA-INV-NO-RESPONSE        VALUE   32.
         88  CORBA-INV-TERM-ON-ERR        VALUE   64.
         88  CORBA-RESP-NO-WAIT           VALUE   128.
         88  CORBA-DEPENDENT-LIST         VALUE   256.
         88  CORBA-CTX-RESTRICT-SCOPE     VALUE   512.
         88  CORBA-CTX-DELETE-DESCENDENTS VALUE   1024.
```

7.2.6 Sending a Request

To send a request to server applications, use either synchronous or asynchronous communication.

Synchronous Communication

To send a request to the server application with the synchronous communication method use *CORBA-REQUEST-INVOKE*. The server application processing result is the return value. Specify the object reference of the request object as a parameter.

```
    MOVE 0 TO  FLAG.
    CALL "CORBA-REQUEST-INVOKE" USING
      REQ
             FLAG
             ENV
             RET-VAL.
```

Asynchronous Communication

To send a request to the server application with the asynchronous communication method use *CORBA-REQUEST-SEND*. Specify the object reference of the request object as a parameter. Then, invoke *CORBA-REQUEST-GET-RESPONSE* to receive the response from the server application. Specify the object reference of the request object as a parameter.

```
*  Requesting processing
    SET CORBA-INV-NO-RESPONSE OF FLAG TO TRUE.
    CALL "CORBA-REQUEST-SEND" USING
      REQ
             FLAG
             ENV
             RET-VAL.
*  Receiving the processing result
    MOVE CORBA-RESP-NO-WAIT TO  FLAG.
    CALL "CORBA-REQUEST-GET-RESPONSE" USING
      REQ
             FLAG
             ENV
             RET-VAL.
    IF RET-VAL = CORBA-TRUE
*  User processing
    END-IF
```

If *CORBA-REQUEST-GET-RESPONSE* is invoked and the request is not completed in the server application, invoke the *CORBA-REQUEST-GET-RESPONSE* method again.

7.2.7 Deleting a Request

To delete a request object use *CORBA-REQUEST-DELETE*. Specify the object reference of the request object as a parameter.

```
*  Deleting a request object
    CALL "CORBA-REQUEST-DELETE" USING
        REQ
              ENV
              RET-VAL.
```

7.3 Client Application Exception Processing

Exceptions are used to let the client application know if a request has failed. The client can know if the exception occurred in the System or the server application. The former is called system exception and the latter is called user exception.

Error information is stored in the CORBA_ENVIRONMENT structure that you specify when a method is invoked. To know CORBA_ENVIRONMENT structure, refer to "ENVIRONMENT.cbl" in the COBOL library.

```
typedef struct  {
     CORBA_enum _major;
          .
          .
} CORBA_Environment;
```

The following values are set in MAJOR:

- *CORBA-NO-EXCEPTION:*

  Normal termination

- CORBA-SYSTEM-EXCEPTION:

  System exception

- CORBA-USER-EXCEPTION:

  User exception

System Exception

The following table shows the system exception values that can be assigned. When a character-string data type is posted as a system exception, use the CORBA-EXCEPTION-ID function to retrieve a character-string from the CORBA_ENVIRONMENT structure. Refer to the COBOL library for texts beginning with "EX-" for details of the system exception character-string.

Refer to the Reference Manual (API Edition) for the meaning of each exception.

Table 7.1 Exception Codes

Exception Information	Exception Code
BAD_CONTEXT	EX-CORBA-STEXCEP-BAD-CONTEXT
BAD_INV_ORDER	EX-CORBA-STEXCEP-BAD-INV-ORDER
BAD_OPERATION	EX-CORBA-STEXCEP-BAD-OPERATION
BAD_PARAM	EX-CORBA-STEXCEP-BAD-PARAM
BAD_QOS	EX-CORBA-STEXCEP-BAD-QOS
BAD_TYPECODE	EX-CORBA-STEXCEP-BAD-TYPECODE
CODESET_INCOMPATIBLE	EX-CORBA-STEXCEP-CODESET-INCOM (refer to Note below)
COMM_FAILURE	EX-CORBA-STEXCEP-COMM-FAILURE
DATA_CONVERSION	EX-CORBA-STEXCEP-DATA-CONVERSI (refer to Note below)
FREE_MEM	EX-CORBA-STEXCEP-FREE-MEM

Exception Information	Exception Code
IMP_LIMIT	EX-CORBA-STEXCEP-IMP-LIMIT
INITIALIZE	EX-CORBA-STEXCEP-INITIALIZE
INTERNAL	EX-CORBA-STEXCEP-INTERNAL
INTF_REPOS	EX-CORBA-STEXCEP-INTF-REPOS
INV_FLAG	EX-CORBA-STEXCEP-INV-FLAG
INV_IDENT	EX-CORBA-STEXCEP-INV-IDENT
INV_OBJREF	EX-CORBA-STEXCEP-INV-OBJREF
INV_POLICY	EX-CORBA-STEXCEP-INV-POLICY
MARSHAL	EX-CORBA-STEXCEP-MARSHAL
NO_IMPLEMENT	EX-CORBA-STEXCEP-NO-IMPLEMENT
NO_MEMORY	EX-CORBA-STEXCEP-NO-MEMORY
NO_PERMISSION	EX-CORBA-STEXCEP-NO-PERMISSION
NO_RESOURCES	EX-CORBA-STEXCEP-NO-RESOURCES
NO_RESPONSE	EX-CORBA-STEXCEP-NO-RESPONSE
OBJ_ADAPTER	EX-CORBA-STEXCEP-OBJ-ADAPTER
PERSIST_STORE	EX-CORBA-STEXCEP-PERSIST-STORE
REBIND	EX-CORBA-STEXCEP-REBIND
TIMEOUT	EX-CORBA-STEXCEP-TIMEOUT
TRANSIENT	EX-CORBA-STEXCEP-TRANSIENT
UNKNOWN	EX-CORBA-STEXCEP-UNKNOWN
INVALID_TRANSACTION	EX-CORBA-STEXCEP-INVALID-TRANS (refer to Note below)
TRANSACTION_MODE	EX-CORBA-STEXCEP-TRANSACTION-M (refer to Note below)
TRANSACTION_REQUIRED	EX-CORBA-STEXCEP-TRANSACTION-R (refer to Note below)
TRANSACTION_ROLLEDBACK	EX-CORBA-STEXCEP-TRANSACTION-R (refer to Note below)
TRANSACTION_UNAVAILABLE	EX-CORBA-STEXCEP-TRANSACTION-U (refer to Note below)

Note

Because the maximum number of characters that can be used in COBOL is 30, the Exception Code is abbreviated.

A minor code is set in _MINOR in the CORBA_ENVIRONMENT structure when a system exception occurs. Refer to information on CORBA Service Minor Codes in the Reference Manual (API Edition) for details of minor code values.

User Exception

When error information is posted as a character-string in a user exception, invoke the *CORBA-EXCEPTION-ID function* to retrieve a character-string from the *CORBA_ENVIRONMENT* structure. Invoke *CORBA-EXCEPTION-VALUE* to retrieve details of user definition information defined in IDL from the *CORBA_ENVIRONMENT* structure.

Obtaining Exception Information

The following example shows how to obtain exception information.

```
LINKAGE SECTION.
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
*  Invoking a method
     MOVE 100 TO PARAM-1.
```

```
     MOVE 20 TO PARAM-2.
     INVOKE "ODDEMO-CALCULATE-CALCULATE" USING
           OBJ
           PARAM-1
           PARAM-2
           ENV
           RESULT.
     EVALUATE TRUE
           WHEN CORBA-NO-EXCEPTION OF MAJOR OF ENV
                CONTINUE
*  System exception error processing
           WHEN CORBA-SYSTEM-EXCEPTION OF MAJOR OF ENV
                MOVE FUNCTION LENG (ID) TO STRING-LENGTH
                CALL "CORBA-STRING-GET" USING
                      IDL-ID OF ENV
                      STRING-LENGTH
                      ID
* Obtaining minor code information
                      DISPLAY "MINOR OF ENV :" MINOR OF ENV.
                     ...
*  User exception error processing
           WHEN CORBA-USER-EXCEPTION OF MAJOR OF ENV
                MOVE FUNCTION LENG (UEXC) TO STRING-LENGTH
                CALL "CORBA-STRING-GET" USING
                      IDL-ID OF ENV
                      STRING-LENGTH
                      UEXC
      END-EVALUATE.
```

7.4 Server Application Programming (Static Invocation Interface)

This is not valid for Linux (64 bit).

The server application is composed of initialization and implementation components. The initialization component process is shown in the following figure.

Figure 7.3 Server Application (Static Invocation Interface) Process Flow

7.4.1 Initialization

Initialization is carried out by invoking *CORBA-ORB-INIT*. An ORB object reference is returned by this function. This object reference is specified whenever the ORB interface is invoked.

IDENTIFICATION DIVISION

```
PROGRAM-ID.  "SERVER-MAIN".
 AUTHOR.  OD/IDLcompiler Ver.2.0.
 INSTALLATION.  IDL FILE NAME IS COBsample.idl.
 SECURITY.  THIS SOURCE CODE WAS GENERATED BASE ON YOUR IDL FILE.
           WHEN THIS STUB/SKELETON SOURCE CODE IS CHANGED, THE OPERATION.
           GURANTEED IS NOT DONE.
 DATE-WRITTEN.  Tue May  6 11:03:40 1997
```

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
     ARGUMENT-NUMBER IS ARG-C
     ARGUMENT-VALUE  IS ARG-V
     SYMBOLIC CONSTANT
     COPY SYMBOL-CONST IN CORBA.
     .
DATA DIVISION.
WORKING-STORAGE SECTION.
COPY CONST IN CORBA.
 01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
 01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
 01 COPY BOA  IN CORBA REPLACING CORBA-BOA BY BOA.
 01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
 01 COPY REPOSITORYID IN CORBA REPLACING CORBA-REPOSITORYID BY INTF-REP.
 01 COPY FJ-IMPLEMENTATIONDEF IN CORBA REPLACING
 FJ-IMPLEMENTATIONDEF BY IMPL-REP.
 01 COPY INTERFACEDEF IN CORBA REPLACING CORBA-INTERFACEDEF BY INTF.
 01 COPY IMPLEMENTATIONDEF IN CORBA REPLACING CORBA-IMPLEMENTATIONDEF BY IMPL.
 01 COPY REFERENCEDATA IN CORBA REPLACING CORBA-REFERENCEDATA BY REF.
 01 REF-P USAGE POINTER.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
 01 NAME-A USAGE POINTER.
 01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING
         COSNAMING-NAMINGCONTEXT BY COS-NAMING.
 01 COPY COSNAMING-NAME IN CORBA REPLACING COSNAMING-NAME BY NAME.
 01 COPY COSNAMING-NAMECOMPONENT IN CORBA REPLACING
         COSNAMING-NAMECOMPONENT BY NAME-COMPONENT.
 01 NAME-COMPONENT-A USAGE POINTER.
 01 STR-BUF PIC X(30).
 01 INTF-INTF-A PIC X(25) VALUE "IDL:ODdemo/calculator:1.0".
 01 IMPL-INTF-A PIC X(25) VALUE "IDL:ODdemo/calculator:1.0".
 01 TEMP-BUF USAGE POINTER.
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
 01 MESS PIC X(30).
*########## ORB SETTING PARAMETER #########
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY CURRENT-ARG-C.
 01 CURRENT-ARG-V.
    02 FILLER OCCURS 6.
       03 CURRENT-ARG-V-VALUE USAGE POINTER.
 01 APLI-NAME PIC X(8) VALUE "simple_s".
 01 TMP-STRING-BUF PIC X(20).
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY ARG-COUNT.
*
 LINKAGE SECTION.
PROCEDURE DIVISION.
MAIN.
*  Setting an ObjectDirector object ID
* argument set : CURRENT-ARG-V-VALUE
     ACCEPT CURRENT-ARG-C FROM ARG-C.
     COMPUTE CURRENT-ARG-C = CURRENT-ARG-C + 1.

     PERFORM VARYING ARG-COUNT FROM 1 BY 1 UNTIL ARG-COUNT > CURRENT-ARG-C
        IF ARG-COUNT = 1
           MOVE APLI-NAME TO TMP-STRING-BUF
        ELSE
           ACCEPT TMP-STRING-BUF FROM ARG-V
```

```
        END-IF

        MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH
        INVOKE "CORBA-STRING-SET" USING
                          CURRENT-ARG-V-VALUE (ARG-COUNT)
                          STRING-LENGTH
                          TMP-STRING-BUF
     END-PERFORM.
     SET CURRENT-ARG-V-VALUE (ARG-COUNT) TO NULL.
*  Initializing ObjectDirector
     MOVE FUNCTION LENG (FJ-OM-ORB-ID) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
               TEMP-BUF
               STRING-LENGTH
               FJ-OM-ORB-ID.
     CALL "CORBA-ORB-INIT" USING
               CURRENT-ARG-C
               CURRENT-ARG-V
               TEMP-BUF
               ENV
               ORB.
     CALL "CORBA-FREE" USING TEMP-BUF.
     MOVE "CORBA-ORB-INIT" TO MESS.
     PERFORM ENV-CHECK
```

Initialize the primitive object adapter.

```
MOVE FUNCTION LENG (CORBA-BOA-OA-ID) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
               TEMP-BUF
               STRING-LENGTH
               CORBA-BOA-OA-ID.
     CALL "CORBA-ORB-BOA-INIT" USING
               ORB
               CURRENT-ARG-C
               CURRENT-ARG-V
               TEMP-BUF
               ENV
               BOA.
     CALL "CORBA-FREE" USING TEMP-BUF.
     MOVE "CORBA-ORB-BOA-INIT" TO MESS.
     PERFORM ENV-CHECK.
```

Then initialize the server application, if required.

7.4.2 Activating the Server

When a server application has been initialized, a completion result is sent to ORB. ORB sends a client request to the server application when this instruction is issued. The activation method varies with server type as shown in the following table.

Table 7.2 Server Activation Methods

Server type	Method
shared server	CORBA-BOA-IMPL-IS-READY
unshared server	CORBA-BOA-OBJ-IS-READY
persistent server	CORBA-BOA-IMPL-IS-READY
server per method	CORBA-BOA-IMPL-IS-READY

(1) Retrieving an Implementation Repository Object Reference

To retrieve an Implementation Repository object reference, use CORBA-ORB-RESOLVE-INITIAL-REFERENCES method to specify CORBA-ORB-OBJECTID-IMPLREP as a parameter.

(2) Searching for an IMPLEMENTATIONDEF Object Reference

To obtain an IMPLEMENTATIONDEF object reference, use FJ-IMPLEMENTATIONREP-LOOKUP-ID to specify the server application IMPLEMENTATIONREP object for the server application for a parameter.

(3) Activating the Server

To activate the server, use CORBA-BOA-IMPL-IS-READY or CORBA-BOA-OBJ-IS-READY.

Note

After a return for a method is received, the server must be deactivated. Files opened at initialization must be closed, and memory location must be released. Refer to "7.4.4 Deactivating the Server" for further details.

```
*  Retrieving an implementation information object reference
     MOVE FUNCTION LENG (CORBA-ORB-OBJECTID-IMPLREP) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
               TEMP-BUF
               STRING-LENGTH
               CORBA-ORB-OBJECTID-IMPLREP.
     CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
               ORB
               TEMP-BUF
               ENV
               IMPL-REP.
     CALL "CORBA-FREE" USING TEMP-BUF.
     MOVE "CORBA-ORB-RESOLVE-INITIAL-REFERENCES-2" TO MESS.
     PERFORM ENV-CHECK
*  Retrieving an ImplementationRep object reference
     MOVE FUNCTION LENG (IMPL-INTF-A) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
               TEMP-BUF
               STRING-LENGTH
               IMPL-INTF-A.
     CALL "FJ-IMPLEMENTATIONREP-LOOKUP-ID" USING
               IMPL-REP
               TEMP-BUF
               ENV
               IMPL.
     CALL "CORBA-FREE" USING TEMP-BUF.
     MOVE "FJ-IMPLEMENTATIONREP-LOOKUP-ID" TO MESS.
     PERFORM ENV-CHECK
*  Activating the server
     CALL "CORBA-BOA-IMPL-IS-READY" USING
           BOA
           IMPL
           ENV.
     MOVE "CORBA-BOA-IMPL-IS-READY" TO MESS.
     PERFORM ENV-CHECK.
```

7.4.3 Interface Installation Functions

After initialization, processing of the interface implemented in the server application is described. The object reference, parameters defined in IDL and pointers to the *CORBA_Environment* structure are sent to the interface implementation function as parameters.

IDENTIFICATION DIVISION.

```
 *  Calculate method installation function
PROGRAM-ID.  "ODDEMO-CALCULATOR-CALCULATE".
```

```
 AUTHOR.  OD/IDLcompiler Ver.2.0.
 INSTALLATION.  IDL FILE NAME IS COBsample.idl.
 SECURITY.  THIS SOURCE CODE WAS GENERATED BASE ON YOUR IDL FILE.
           WHEN THIS STUB/SKELETON SOURCE CODE IS CHANGED, THE OPERATION.
           GURANTEED IS NOT DONE.
 DATE-WRITTEN.  Tue May  6 11:03:40 1997
```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.

```
     ARGUMENT-NUMBER IS ARG-C
     ARGUMENT-VALUE  IS ARG-V
     SYMBOLIC CONSTANT
     COPY SYMBOL-CONST IN CORBA.
     .
*
 INPUT-OUTPUT SECTION.
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
 01 MESS PIC X(30).
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
 01 COPY ORB IN CORBA REPLACING CORBA-ORB  BY ORB.
 01 COPY BOA IN CORBA REPLACING CORBA-BOA  BY BOA.
 01 STRING-TMP  PIC X(10).
 01 COPY ULONG  IN CORBA REPLACING CORBA-UNSIGNED-LONG BY FLAGS.
 01 TEMP-BUF USAGE POINTER.
 01 EXCEP.
    03  COPY LONG IN CORBA REPLACING CORBA-LONG BY EXCEP-L.
 01 EXCEP-A  USAGE POINTER.
 01 EX-ODDEMO-CALCULATOR-ZEROPARAM PIC X(35)
    VALUE "IDL:ODdemo/calculator/ZEROPARAM:1.0".
 01 COPY EXCEPTION-TYPE IN CORBA REPLACING CORBA-EXCEPTION-TYPE BY EX-STATUS.
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY C-ARG-C.
 01 C-ARG-V.
    02 FILLER OCCURS 2.
       03 ARG-V-VALUE USAGE POINTER.
 01 APLI-NAME PIC X(8) VALUE "simple_s".

 LINKAGE SECTION.
 01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARAM1.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARAM2.
 01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
 01 A-RESULT.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY ADD-RESULT.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SUBTRACT-RESULT.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY MULTIPLE-RESULT.
    02 COPY FLOAT IN CORBA REPLACING CORBA-FLOAT BY DEVIDE-RESULT.
```

PROCEDURE DIVISION

```
USING
*  Server object reference
         OBJ
*  Input value
         PARAM1
*  Input value
         PARAM2
*  Error value
         ENV
```

```
*  Operation result
         A-RESULT.
*
 MAIN.

     COMPUTE C-ARG-C = 1.
     MOVE FUNCTION LENG (APLI-NAME) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           ARG-V-VALUE (1)
           STRING-LENGTH
           APLI-NAME.
     SET ARG-V-VALUE(2) TO NULL.

     MOVE  12 TO STRING-LENGTH.
     CALL  "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           FJ-OM-ORB-ID.
     CALL "CORBA-ORB-INIT" USING
           C-ARG-C
           C-ARG-V
           TEMP-BUF
           ENV
           ORB.
     MOVE "CORBA-ORB-INIT" TO MESS.
     PERFORM ENV-CHECK

     CALL "CORBA-FREE" USING
           TEMP-BUF.

     MOVE 15 TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           CORBA-BOA-OA-ID.
     CALL "CORBA-ORB-BOA-INIT" USING
           ORB
           C-ARG-C
           C-ARG-V
           TEMP-BUF
           ENV
           BOA.
     MOVE "CORBA-ORB-BOA-INIT" TO MESS.
     PERFORM ENV-CHECK.
*  0 division check
     IF PARAM2 = 0
         MOVE 1 TO FLAGS
         MOVE FUNCTION LENG(EX-ODDEMO-CALCULATOR-ZEROPARAM) TO STRING-LENGTH
         CALL "CORBA-STRING-SET" USING
               TEMP-BUF
               STRING-LENGTH
               EX-ODDEMO-CALCULATOR-ZEROPARAM
         MOVE  0 TO  EXCEP-L OF EXCEP
         MOVE  FUNCTION ADDR (EXCEP) TO EXCEP-A

         CALL "CORBA-BOA-SET-EXCEPTION" USING
               BOA
               FLAGS
               TEMP-BUF
               EXCEP-A
               ENV
     ELSE
* Calculate
```

```
        COMPUTE  ADD-RESULT      OF A-RESULT = PARAM1 + PARAM2
        COMPUTE  SUBTRACT-RESULT OF A-RESULT = PARAM1 - PARAM2
        COMPUTE  MULTIPLE-RESULT OF A-RESULT = PARAM1 * PARAM2
        COMPUTE  DEVIDE-RESULT   OF A-RESULT = PARAM1 / PARAM2
    END-IF.
    EXIT PROGRAM.
MAIN-END.
```

7.4.4 Deactivating the Server

When a server application receives a stop request from a user, it notifies ORB that subsequent requests from clients will not be accepted. The method of issuing a stop request for server application users, and the method for receiving such requests in the application, varies with different ORB vendors. When the notification has been issued, ORB will stop dispatching requests to the server, and will return exceptions to clients. The following table shows deactivation methods for various server types.

Table 7.3 Server Deactivation Methods

Server type	Method
shared server	CORBA-BOA-DEACTIVATE-IMPL
unshared server	CORBA-BOA-DEACTIVATE-OBJ
persistent server	CORBA-BOA-DEACTIVATE-IMPL
server per method	Not issued

To specify a server application ImplementationRep object as a parameter, invoke the *CORBA-BOA-DEACTIVATE-IMPL method*. To specify an object reference as a parameter, invoke the *CORBA-BOA-DEACTIVATE-OBJ method.*

```
*  Deactivating the server
   CALL "CORBA-BOA-DEACTIVATE-IMPL" USING
          BOA
          IMPL
          ENV
          RET-VAL.
```

If the *isstopwu* command or the Interstage Management Console is used to stop an operating WorkUnit, the server application notifies ORB that subsequent requests from the client should not be accepted. Client applications no longer need to issue the deactivation method.

If the *odcntlque* command is used to stop server applications when the CORBA application program is operated under control of the WorkUnit, client applications no longer need to issue the deactivation method.

7.5 Server Application Exception Handling

This is not valid for Linux (64 bit).

This section contains information on programming server application exceptions.

Setting Exception Information

To assign a user exception, invoke CORBA-STRING-SET. This will retrieve the location where the identifying character-string is to be set. Copy the character-string to this area. To retrieve the area in which the user exception is set, invoke the ALLOC function generated by the IDL compiler. The ALLOC function name using the interface name, exception ID, and ALLOC as specified in IDL, concatenated with hyphen ("-") must be specified.

Exception information, which is defined in IDL, becomes a structure in COBOL. A value to the structure member variable must be assigned. In the following example, information identifying a user exception is not set. To set exception information in the CORBA-ENVIRONMENT structure, invoke the CORBA-BOA-SET-EXCEPTION function.

IDENTIFICATION DIVISION.

```
PROGRAM-ID.  "ODDEMO-CALCULATOR-CALCULATE".
```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.

```
SYMBOLIC CONSTANT
 COPY SYMBOL-CONST IN CORBA.
  INPUT-OUTPUT SECTION.
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
   01 COPY FLOAT IN CORBA REPLACING CORBA-FLOAT FY FA.
   01 COPY FLOAT IN CORBA REPLACING CORBA-FLOAT BY FB.
 LINKAGE SECTION.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY A.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY B.
   01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
   01 COPY FLOAT IN CORBA REPLACING CORBA-FLOAT BY FRET.
```

PROCEDURE DIVISION

```
USING
         OBJ
         A
         B
         ENV
         FRET.
*
 MAIN.
      MOVE A TO FA.
      MOVE B TO FB.
*  Setting error ZEROPARAM in env.
      IF B = 0
         MOVE CORBA-USER-EXCEPTION OF FLAG TO TRUE
         MOVE EX-ODDEMO-CALCULATOR-ZEROPARAM OF USER-EX TO TRUE
         CALL "CORBA-BOA-SET-EXCEPTION" USING
               BOA
               FLAG
               USER-EX
               NULL
               ENV
               RET
         MOVE 0 TO FRET
      GO TO MAIN-END
      END-IF
      COMPUTE FRET = FA / FB.
      EXIT PROGRAM.
 MAIN-END.
END PROGRAM  "ODDEMO-CALCULATOR-CALCULATE"
```

Note

To call another method from the inside of a server method, use another CORBA_Environment structure. Do not use the CORBA_Environment structure passed to a parameter of a server method as a parameter for calling another method. If this is done, exception information for the inside of the server method that is generated by another method will be set as the exception information.

The following sections show a correct and an incorrect programming example.

Correct Programming Example

IDENTIFICATION DIVISION.

```
   PROGRAM-ID.  "ODDEMO-CALCULATOR-CALCULATE".
```

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
  SYMBOLIC CONSTANT
  COPY SYMBOL-CONST IN CORBA.

  INPUT-OUTPUT SECTION.
```

```
DATA DIVISION.
WORKING-STORAGE SECTION.
    01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ-A.
    01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING COSNAMING-NAMINGCONTEXT BY COS-NAMING.
    01 COPY COSNAMING-NAME          IN CORBA REPLACING COSNAMING-NAM BY NAME.
    01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY LOCAL-ENV.

  LINKAGE SECTION.
    01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
    01 COPY LONG IN CORBA REPLACING CORBA-LONG BY A.
    01 COPY LONG IN CORBA REPLACING CORBA-LONG BY B.
    01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
    01 COPY FLOAT IN CORBA REPLACING CORBA-FLOAT BY FRET.
*
```

```
PROCEDURE DIVISION
  USING
          OBJ
          A
          B
          ENV
          FRET.
*
  MAIN.
       :

  * The exception information for COSNAMING-NAMINGCONTEXT-BIND
  * is set in LOCAL_ENV.  Thus it is not handled as the exception
  * information for ODDEMO-CALCULATOR-CALCULATE.
  *
      CALL "COSNAMING-NAMINGCONTEXT-BIND" USING
              COS-NAMING
              NAME
              OBJ-A
              LOCAL-ENV.
          :
```

Incorrect Programming Example

```
IDENTIFICATION DIVISION.
  PROGRAM-ID.  "ODDEMO-CALCULATOR-CALCULATE".
```

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
  SYMBOLIC CONSTANT
  COPY SYMBOL-CONST IN CORBA.

  INPUT-OUTPUT SECTION.
```

```
DATA DIVISION.
WORKING-STORAGE SECTION.
```

```
    01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ-A.
    01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING COSNAMING-NAMINGCONTEXT BY COS-NAMING.
    01 COPY COSNAMING-NAME          IN CORBA REPLACING COSNAMING-NAM BY NAME.

  LINKAGE SECTION.
    01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
    01 COPY LONG IN CORBA REPLACING CORBA-LONG BY A.
    01 COPY LONG IN CORBA REPLACING CORBA-LONG BY B.
    01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
    01 COPY FLOAT IN CORBA REPLACING CORBA-FLOAT BY FRET.
 *
```

PROCEDURE DIVISION

```
  USING
          OBJ
          A
          B
          ENV
          FRET.
 *
  MAIN.
      :
     * When an exception is returned to COSNAMING-NAMINGCONTEXT-BIND,
 * the exception information for COSNAMING-NAMINGCONTEXT-BIND
 * is incorrectly handled as the exception information for
 * ODDEMO-CALCULATOR-CALCULATE.
 *
     CALL "COSNAMING-NAMINGCONTEXT-BIND" USING
             COS-NAMING
             NAME
             OBJ-A
             ENV.
     :
```

Obtaining Exception Information

Exception information for server applications is obtained in the same way as for client applications. Refer to "7.3 Client Application Exception Processing" for details.

7.6 Registering a Server Application

This is not valid for Linux (64 bit).

Register the created server application to ImplementationRepository and NamingService.

7.6.1 Registering in the Implementation Repository

Server applications must be registered in the Implementation Repository. Use the OD_impl_inst command to register in, or delete from, the Implementation Repository. The command syntax is shown below.

```
OD_impl_inst -ax def
```

-ax def

Indicates that server application information is registered, using the definition information specified in the definition file.

Example *def* File Contents Windows32/64

```
rep_id                  = IDL:ODdemo/calculator:1.0
lang                    = COBOL
type                    = shared
binary                  = D:\server\simple_s.exe
```

```
IDL:ODdemo/calculator:1.0 = D:\server\libODDEMO-CALCULATOR.dll,DSI
ior                       = 1.1
```

Example *def* File Contents Solaris32/64

```
rep_id                    = IDL:ODdemo/calculator:1.0
lang                      = COBOL
type                      = shared
binary                    = /home/guest/simple_s
IDL:ODdemo/calculator:1.0 = /home/guest/libODDEMO-CALCULATOR.so,DSI
uid                       = user
gid                       = group
ior                       = 1.1
env                       = LD_LIBRARY_PATH=/opt/FJSVcbl/lib:/opt/FSUNod/lib;LANG=ja
```

rep_id = IDL:ODdemo/calculator:1.0

Specify the Implementation Repository ID.

lang = COBOL

Specify the server application language type. Specify COBOL for a COBOL application.

type = shared

Specify the server type. The server types available are persistent, shared, unshared, and server per method.

binary = D:\server\simple_s.exe Windows32/64

binary = /home/guest/simple_s Solaris32/64

Specify the pathname of the server application storage location.

IDL:ODdemo/calculator:1.0 = D:\server\lib0DDEMO-CALCULATOR.dll, DSI Windows32/64

IDL:ODdemo/calculator:1.0 = /home/guest/libODDEMO-CALCULATOR.so,DSI Solaris32/64

Specify the pathname of the storage location of the server application library. If the dynamic skeleton interface is used, define ".DSI" after the library.

uid = user Solaris32/64

Specify the User ID used when executing the server application.

gid = group Solaris32/64

Specify the Group ID used when executing the server application.

ior=1.1

Specify IOR version. Available versions are 1.0 and 1.1.

env= LD_LIBRARY_PATH=/opt/FJSVcbl/lib:/opt/FSUNod/lib;LANG=ja

The environment variable is specified when the server application is executed. Specify the library path that is required for the execution of the application.

7.6.2 Registering in the Naming Service

To make a server application available as an object to other applications, create an object reference to identify it. At the same time, register the object reference in the Naming Service.

The following methods can be used to create an object:

- Use the OD_or_adm command
- Use BOA functionality to create an object reference and register it in Naming Service.

Using the OD_or_adm Command

Once an object reference has been created, use OD_or_adm command to register it in the Naming Service. The following example deals with registration using the OD_or_adm command and information that needs to be specified:

```
OD_or_adm -c IDL:ODdemo/calculator:1.0 -n ODdemo::calculator
```

-c IDL:ODdemo/calculator:1.0

Registers an object reference using the specified interface repository ID.

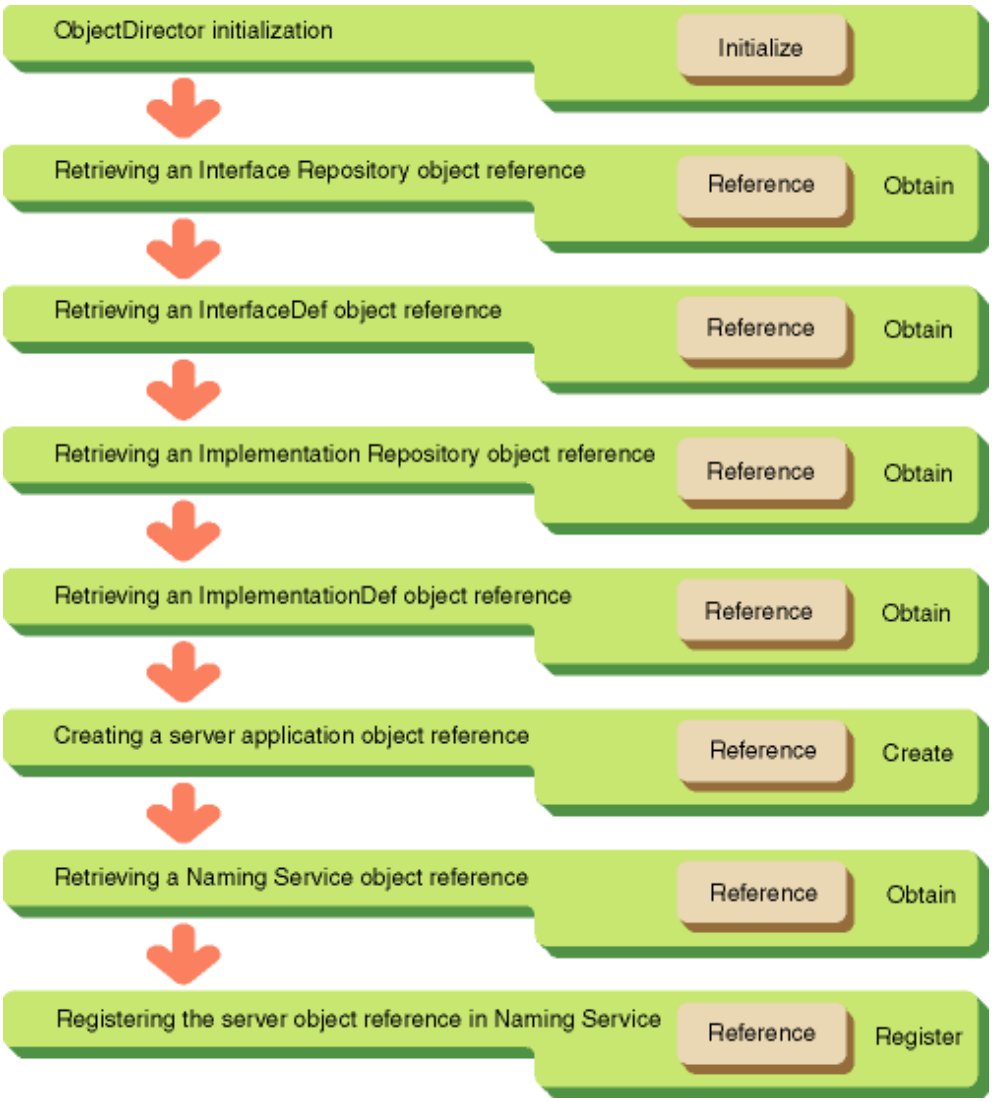
-n ODdemo::calculator

Specifies the name of the object to be registered in the Naming Service.

Creation Method in a Server Application

The following figure describes the process.

Figure 7.4 Creation Method in a Server Application




```
* Object returned from ORB-INIT
   01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
* Object returned from ORB-BOA-INIT    01 COPY BOA IN CORBA REPLACING CORBA-BOA BY BOA.
* Object reference for interface repository
   01 COPY REPOSITORYID IN CORBA REPLACING CORBA-REPOSITORYID BY INTF-REP.
* Object reference for implementation repository
```

```
   01 COPY FJ-IMPLEMENTATIONDEF IN CORBA
      REPLACING FJ-IMPLEMENTATIONDEF BY IMPL-REP.
*  InterfaceDef for interface repository
   01 COPY INTERFACEDEF IN CORBA REPLACING CORBA-INTERFACEDEF BY INTF.
*  ImplementationDef for implementation repository
   01 COPY IMPLEMENTATIONDEF IN CORBA REPLACING
 CORBA-IMPLEMENTATIONDEF BY IMPL.
*  ReferenceData storage area
   01 COPY REFERENCEDATA IN CORBA REPLACING CORBA-REFERENCEDATA BY R-ID.

*  ObjectDiretor Initialization (omitted)
       :

*  Getting an Interface Repository object reference
     MOVE FUNCTION LENG (CORBA-ORB-OBJECTID-LIGHTINTFR) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TEMP-BUF
          STRING-LENGTH
          CORBA-ORB-OBJECTID-LIGHTINTFR.
    CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
          ORB
          TEMP-BUF
          ENV
          INTF-REP.
    CALL "CORBA-FREE" USING TEMP-BUF.

*  Getting an InterfaceDef object reference
*  INTF-INTF-A : Implementation Repository ID
    MOVE FUNCTION LENG (INTF-INTF-A) TO STRING-LENGTH.
    CALL "CORBA-STRING-SET" USING
          TEMP-BUF
          STRING-LENGTH
          INTF-INTF-A.
    CALL "CORBA-REPOSITORY-LOOKUP-ID" USING
          INTF-REP
          TEMP-BUF
          ENV
          INTF.
    CALL "CORBA-FREE" USING TEMP-BUF.

*  Getting an Implementation Repository object reference
    MOVE FUNCTION LENG (CORBA-ORB-OBJECTID-IMPLREP) TO STRING-LENGTH.
    CALL "CORBA-STRING-SET" USING
          TEMP-BUF
          STRING-LENGTH
          CORBA-ORB-OBJECTID-IMPLREP.
    CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
          ORB
          TEMP-BUF
          ENV
          IMPL-REP.
    CALL "CORBA-FREE" USING TEMP-BUF.

* Getting an ImplementationDef object reference
    MOVE FUNCTION LENG (IMPL-INTF-A) TO STRING-LENGTH.
    CALL "CORBA-STRING-SET" USING
          TEMP-BUF
          STRING-LENGTH
          IMPL-INTF-A.
    CALL "FJ-IMPLEMENTATIONREP-LOOKUP-ID" USING
          IMPL-REP
          TEMP-BUF
          ENV
```

```
          IMPL.
    CALL "CORBA-FREE" USING TEMP-BUF.

*  Creating an object reference
    MOVE FUNCTION ADDR(R-ID) TO R-ID-P.
    CALL "CORBA-BOA-CREATE" USING
          BOA
          R-ID-P
          INTF
          IMPL
          ENV
          NEW-OBJ.

*  Setting a naming service object ID
    MOVE FUNCTION LENG ( CORBA-ORB-OBJECTID-NAMESERVICE ) TO STRING-LENGTH.
    CALL  "CORBA-STRING-SET" USING
          NAMESERVICEID
          STRING-LENGTH
          CORBA-ORB-OBJECTID-NAMESERVICE.
*  Retrieve a naming service object reference
    CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
          ORB
          NAMESERVICEID
          ENV
          COS-NAMING.
*  Releasing the area stored in NAMESERVICEID
    CALL "CORBA-FREE" USING NAMESERVICEID.

* Setting a object name
    MOVE FUNCTION LENG (STR-BUF) TO STRING-LENGTH.
    MOVE "ODdemo::calculator" TO STR-BUF.
    CALL "CORBA-STRING-SET" USING
          IDL-ID OF NAME-COMPONENT
          STRING-LENGTH
          STR-BUF.

* Setting a object type
    MOVE " " TO STR-BUF.
    CALL "CORBA-STRING-SET" USING
          KIND OF NAME-COMPONENT
          STRING-LENGTH
          STR-BUF.

*  Number of object names
    MOVE 1 TO SEQ-LENGTH OF NAME.
    MOVE 1 TO SEQ-MAXIMUM OF NAME.
    MOVE 1 TO NUM.
*  Collecting and setting an object name storage area
    CALL "CORBA-SEQUENCE-COSNAMING-NAMECOMPONENT-ALLOCBUF" USING
          SEQ-MAXIMUM OF NAME
          SEQ-BUFFER OF NAME.
    MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
    MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
    CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
          NAME-A
          NUM
          NAME-COMPONENT-A.
    CALL "COSNAMING-NAMINGCONTEXT-BIND" USING
          COS-NAMING
          NAME
          OBJ
          ENV
          .
```

```
*  Setting an object name
    MOVE FUNCTION LENG (STR-BUF) TO STRING-LENGTH.
    MOVE "ODdemo::calculator" TO STR-BUF.
    CALL "CORBA-STRING-SET" USING
          IDL-ID OF NAME-COMPONENT
          STRING-LENGTH
          STR-BUF.
*  Setting an object type
    MOVE " " TO STR-BUF.
    CALL "CORBA-STRING-SET" USING
          KIND OF NAME-COMPONENT
          STRING-LENGTH
          STR-BUF.
*  Object of server application
    CALL "COSNAMING-NAMINGCONTEXT-BIND" USING
          COS-NAMING
          NAME
          NEW-OBJ
          ENV.
```

CORBA-BOA-CREATE creates an ObjectReference with its default character set registered in Implementation Repository. The default character set is specified by OD_impl_inst or OD_set_env command.

7.7 Handling Data Types

COBOL provides the following libraries to handle CORBA data types:

- For nonnumeric constants: CONST.cbl
- For SYMBOLIC CONSTANT: SYMBOL-CONST.cbl
- Other data types: data-type-name.cbl

Refer to the following table for details.

Table 7.4 Data Types (COBOL)

Data type			Type Declaration	Type Declaration
Primitive data type	Integer	long	CORBA-LONG	PIC S9(9) COMP-5.
		short	CORBA-SHORT	PIC S9(4) COMP-5.
		unsigned long	CORBA-UNSIGNED-LONG	PIC 9(9) COMP-5.
		unsigned short	CORBA-UNSIGNED-SHORT	PIC 9(4) COMP-5.
		long long	CORBA-LONG-LONG	PIC S9(18) COMP-5.
	Floating-point	float	CORBA-FLOAT	COMP-1.
		double	CORBA-DOUBLE	COMP-2.
	Character	char	CORBA-CHAR	PIC X(1).
		wchar	CORBA-WCHAR	PIC N(1)
	Octet	octet	CORBA-OCTET	PIC 9(4) COMP-5.
	Boolean	boolean	CORBA-BOOLEAN	PIC 9(9) COMP-5.
	Character-string	string	Refer to "7.7.1 String Type"	PIC X(n).
		wstring	Refer to "7.7.2 Wide String Type"	PIC N(n)

Data type			Type Declaration	Type Declaration
	Enum	enum	CORBA-ENUM	PIC 9(9) COMP-5.
	Any	any	Refer to "7.7.3 Any Type"	Group item
Sequence		sequence	Refer to "7.7.4 Sequence Type"	OCCURS clause
Structure		struct	Refer to "7.7.5 Structure Type"	Group item
Union		union	Refer to "7.7.6 Union"	REDEFINES phrase
Fixed-point		fixed	Refer to "7.7.7 Fixed-point"	PIC xx (n) PACKED-DECIMAL.
Array		array	Refer to "7.7.8 Array"	OCCURS clause
Object Reference		Object	CORBA-OBJECT	USAGE POINTER
Type code		TypeCode	CORBA-TYPECODE	USAGE POINTER

Notes on Handling SHORT and USHORT

- These data types are managed as double-byte internally in COBOL, but are defined as PIC S9(4) COMP-5 and PIC 9(4) COMP-5 data.
- If the data types are displayed unchanged, values of more than four digits cannot be displayed.
- To display values of more than four digits, including the sign, temporarily assign them to a variable such as CORBA-LONG beforehand.
- If a value of five digits or more is assigned to SHORT or USHORT, a size error warning is displayed at compilation time.
- To use a value of five digits or more, use the LONG type. Note that a size error occurs when a value of nine digits or more is assigned to LONG or ULONG.

Notes on Interpreting Union Types

- If there are common type members with different data lengths, the JMN2232I-W message is output during the COBOL interpretation.
- In CORBA Service, "long" and "string" are offered as common type samples with different data lengths. The JMN2232I-W message is output when these samples are interpreted.

7.7.1 String Type

This section describes string type data.

IDL Mapping

When a character-string type string is specified in IDL, data is declared using CORBA-STRING in COBOL.

This is explained using the following IDL definition example.

IDL

```
module ODsample{
    interface     stringtest{
        string op1(in string str1, out string str2, inout string str3);
    };
};
```

COBOL

IDENTIFICATION DIVISION.

```
PROGRAM-ID.  "ODSAMPLE-STRINGTEST-OP1".
```

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
*  Object reference
```

```
  01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
*  in parameter
  01 STR1 USAGE POINTER.
*  out parameter
  01 STR2 USAGE POINTER.
*  inout parameter
  01 STR3 USAGE POINTER.
*  Exception information
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
*  Return value
  01 RET  USAGE POINTER.
```

PROCEDURE DIVISION.

```
  CALL "ODSAMPLE-STRINGTEST-OP1" USING
        ORB
        STR1
        STR2
        STR3
        ENV
        RET.
```

Parameters Handled by Client Applications

The following table shows how the client application parameters are handled.

Table 7.5 Client Parameter Area (Character String Type)

Parameter	Parameter passed to server	Parameter passed from server
in	The CORBA-STRING-SET function is used to allocate an area required for the character string + termination character '\0', and to set the character string.	_
inout	Same as the *in* parameter	The area is automatically allocated by the stub.
out return	_	Same as the *inout* parameter

When an area that has been allocated by the client and stub is no longer required, it must be released using the CORBA-FREE function.

The following is an example of DATA DIVISION and PROCEDURE DIVISION processing in a client application.

DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
  01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
  01 STR1 USAGE POINTER.
  01 STR2 USAGE POINTER.
  01 STR3 USAGE POINTER.
  01 RET USAGE POINTER.
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
  01 RET  USAGE POINTER.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
  01 STR-WORK PIC X(30).
```

PROCEDURE DIVISION.

```
*  Setting the in parameter
  MOVE "IN" TO STR-WORK.
  MOVE 3 TO LSIZE.
  CALL "CORBA-STRING-SET" USING
        STR1
        LSIZE
        STR-WORK.
*  Setting the inout parameter
  MOVE "INOUT:1" TO STR-WORK.
  MOVE 8 TO LSIZE.
  CALL "CORBA-STRING-SET" USING
        STR2
        LSIZE
        STR-WORK.
  CALL "ODSAMPLE-STRINGTEST-OP1" USING
        ORB
        STR1
        STR2
        STR3
        ENV
        RET.

*  Releasing the return value area
  CALL "CORBA-FREE" USING
        RET.
*  Releasing the in parameter area
  CALL "CORBA-FREE" USING
        STR1.
*  Releasing the out parameter area
  CALL "CORBA-FREE" USING
        STR2.
*  Releasing the inout parameter area
  CALL "CORBA-FREE" USING
        STR3.
```

Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

The following table shows how the server application parameters are handled.

Table 7.6 Server Parameter Area (Character String Type)

Parameter	Parameter passed from client	Parameter passed to client
in	The character string area is automatically allocated or released by the skeleton.	_
inout	The character string area is automatically allocated by the skeleton.	When the character string to be returned is shorter than the passed parameter, the character string is set in the passed character string area. When the character string to be returned is longer than the passed parameter, the passed character string area is released once using the CORBA-FREE function; then the area is allocated and the character string is set using the CORBA-STRING-SET function. The character string area is automatically released by the skeleton.
out return	_	The area is allocated and the character string is set using the CORBA-STRING-SET function. The character string area is automatically released by the skeleton.

The following is an example of server application processing.

IDENTIFICATION DIVISION.

```
 PROGRAM-ID.  "ODSAMPLE-STRINGTEST-OP1".
```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.

```
    SYMBOLIC CONSTANT
    COPY SYMBOL-CONST IN CORBA.
    .
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
  COPY CONST IN CORBA.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
  01 STR-WORK PIC X(30).
 LINKAGE SECTION.
*  Object reference
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
*  in parameter
  01 STR1 USAGE IS POINTER.
*  out parameter
  01 STR2 USAGE IS POINTER.
*  inout parameter
  01 STR3 USAGE IS POINTER.
  01 STR USAGE IS POINTER.
*  Exception information
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
```

PROCEDURE DIVISION

```
USING OBJ STR1 STR2 STR3 ENV STR.
 MAIN.
*  out parameter processing
  MOVE OUT TO STR-WORK.
  MOVE 4 TO LSIZE.
*  Setting the out parameter
  CALL "CORBA-STRING-SET" USING
        STR2
        LSIZE
        STR-WORK.
*  inout parameter processing
*  Releasing the area sent from the client
  CALL "CORBA-FREE" USING
         STR3.
  MOVE 8 TO LSIZE.
*  Setting the output parameter
  MOVE "INOUT:2" TO STR-WORK.
  CALL "CORBA-STRING-SET" USING
        STR3
        LSIZE
        STR-WORK.
*  Return value processing
  MOVE 7 TO LSIZE.
*  Setting the return value
  MOVE "RETURN" TO STR-WORK.
  CALL "CORBA-STRING-SET" USING
        STR
        LSIZE
        STR-WORK.
```

```
 MAIN-END.
END PROGRAM "ODSAMPLE-STRINGTEST-OP1".
```

7.7.2 Wide String Type

This section describes wstring type data.

IDL Mapping

When a wide string type is specified in IDL, data is declared using POINTER in COBOL.

This is explained using the following IDL definition example.

IDL

```
module ODsample{
     interface     wstringtest{
        wstring op1(in wstring str1, out wstring str2, inout wstring str3);
     };
};
```

COBOL

IDENTIFICATION DIVISION.

```
 PROGRAM-ID.  "ODSAMPLE-WSTRINGTEST-OP1".
```

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
*  Object reference
  01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
*  in parameter
  01 STR1 USAGE POINTER.
*  out parameter
  01 STR2 USAGE
POINTER.
*  inout parameter
  01 STR3 USAGE POINTER.
*  Exception information
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
*  Return value
  01 RET  USAGE POINTER.
```

PROCEDURE DIVISION.

```
  CALL "ODSAMPLE-WSTRINGTEST-OP1" USING
        ORB
        STR1
        STR2
        STR3
        ENV
        RET.
```

Parameters Handled by Client Applications

The following table shows how the client application parameters are handled.

Table 7.7 Client Parameter Area (Wide String Type)

Parameter	Parameter passed to server	Parameter passed from server
in	The CORBA-WSTRING-SET function is used to allocate an area required for the character string + termination character '\0', and to set the character string.	_
inout	Same as the *in* parameter	The area is allocated automatically by the stub.
out return	_	Same as the *inout* parameter

Note

When an area that has been allocated by the client and stub is no longer required, it must be released using the CORBA-FREE function.

The following is an example of DATA DIVISION and PROCEDURE DIVISION processing in a client application.

DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
  01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
  01 STR1 USAGE POINTER.
  01 STR2 USAGE POINTER.
  01 STR3 USAGE POINTER.
  01 RET USAGE POINTER.
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
  01 RET  USAGE POINTER.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
  01 STR-WORK PIC N(30).
```

PROCEDURE DIVISION.

```
*  Setting the in parameter
  MOVE N"IN" TO STR-WORK.
  COMPUTE LSIZE = FUNCTION LENGTH(STR-WORK).
  CALL "CORBA-WSTRING-SET" USING
        STR1
        LSIZE
        STR-WORK.
*  Setting the inout parameter
  MOVE N"INOUT" TO STR-WORK.
  COMPUTE LSIZE = FUNCTION LENGTH(STR-WORK).
  CALL "CORBA-WSTRING-SET" USING
        STR2
        LSIZE
        STR-WORK.
  CALL "ODSAMPLE-WSTRINGTEST-OP1" USING
        ORB
        STR1
        STR2
        STR3
        ENV
        RET.

*  Releasing the return value area
  CALL "CORBA-FREE" USING
        RET.
*  Releasing the in parameter area
  CALL "CORBA-FREE" USING
        STR1.
```

```
*  Releasing the out parameter area
  CALL "CORBA-FREE" USING
        STR2.
*  Releasing the inout parameter area
  CALL "CORBA-FREE" USING
        STR3.
```

Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

The following table shows how the server application parameters are handled.

Table 7.8 Server Parameter Area (Wide String Type)

Parameter	Parameter passed from client	Parameter passed to client
in	The character string area is automatically allocated or released by the skeleton.	_
inout	The character string area is automatically allocated by the skeleton.	When the character string to be returned is shorter than the passed parameter, the character string is set in the passed character string area. When the character string to be returned is longer than the passed parameter, the passed character string area is released once using the CORBA-FREE function. The area is then allocated and the character string is set using the CORBA-WSTRING-SET function. The character string area is automatically released by the skeleton.
out return	_	The area is allocated and the character string is set using the CORBA-WSTRING-SET function. The character string area is automatically released by the skeleton.

The following is an example of server application processing.

IDENTIFICATION DIVISION.

```
PROGRAM-ID.  "ODSAMPLE-WSTRINGTEST-OP1".
```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.

```
   SYMBOLIC CONSTANT
   COPY SYMBOL-CONST IN CORBA.
   .
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
  COPY CONST IN CORBA.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
  01 STR-WORK PIC N(30).
 LINKAGE SECTION.
*  Object reference
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
*  in parameter
  01 STR1 USAGE IS POINTER.
*  out parameter
  01 STR2 USAGE IS POINTER.
*  inout parameter
  01 STR3 USAGE IS POINTER.
  01 STR USAGE IS POINTER.
```

```
*  Exception information
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
```

PROCEDURE DIVISION

```
USING OBJ STR1 STR2 STR3 ENV STR.
 MAIN.
*  out parameter processing
  MOVE N"OUT" TO STR-WORK.
  COMPUTE LSIZE = FUNCTION LENGTH(STR-WORK).
*  Setting the out parameter
  CALL "CORBA-WSTRING-SET" USING
        STR2
        LSIZE
        STR-WORK.
*  inout parameter processing
*  Releasing the area sent from the client
  CALL "CORBA-FREE" USING
         STR3.
*  Setting the output parameter
  MOVE N"INOUT" TO STR-WORK.
  COMPUTE LSIZE = FUNCTION LENGTH(STR-WORK).
  CALL "CORBA-WSTRING-SET" USING
        STR3
        LSIZE
        STR-WORK.
*  Return value processing
  MOVE N"RETURN" TO STR-WORK.
  COMPUTE LSIZE = FUNCTION LENGTH(STR-WORK).
*  Setting the return value
  MOVE "RETURN" TO STR-WORK.
  CALL "CORBA-WSTRING-SET" USING
        STR
        LSIZE
        STR-WORK.
  MAIN-END.
 END PROGRAM "ODSAMPLE-WSTRINGTEST-OP1".
```

7.7.3 Any Type

This section provides information on *Any* type data.

IDL Mapping

When Any from IDL Any type is specified, declare data using the *CORBA-ANY structure* in COBOL. An allocation function for any data area (any-value area) is generated by the IDL compiler. The function is given a name "module name", "interface name", "struct name", and "alloc" concatenating with hyphen ("-"). ANY-TYPE is a type code that identifies the data type. ANY-VALUE is a pointer to a location containing data. The *CORBA-ANY structure* consists of the following:

```
CORBA-ANY.
*  Data identification information
 49 ANY-TYPE USAGE POINTER.
*  Data storage area
 49 ANY-VALUE USAGE POINTER.
```

This is explained using the following IDL definition example.

IDL

```
module ODsample{
        struct sample1 {
                long        para1;
                string      para2;
```

```
        };
        struct sample2 {
                char        para1;
                float       para2;
        };
        struct sample3 {
                char        para1;
                double      para2;
        };
        interface   anytest{
                    any         op1(in any any1, out any any2, inout any any3 );
        };
};
```

Parameters Handled by Client Applications

The following table shows how the client application parameters are handled.

Table 7.9 Client Parameter Area (Any Type)

Parameter	Parameter passed to server	Parameter passed from server
in	The TypeCode information ("TC-module name-interface name-data name-IMPL-SEQ" defined in IDL file name _h.cbl) is set in ANY-TYPE, and the pointer for the data area (allocated using the XX-ALLOC function) is set in ANY-VALUE. The CORBA-ANY-ALLOC function or the XX-ALLOC function is used to dynamically allocate the area (CORBA-ANY or ANY-VALUE area).	_
inout	(Same as the *in* parameter")	The area is automatically allocated by the stub.
out return	_	Same as the *inout* parameter

When an area that has been allocated by the client and stub is no longer required, it must be released using the CORBA-FREE function. Whether or not the area indicated by ANY-VALUE is to be released can be specified in the release flag.

The following functions and flags are used to reference and set the release flag.

Function

```
CALL "CORBA-ANY-GET-RELEASE" USING
      BUFFER
      RET-VAL.
CALL "CORBA-ANY-SET-RELEASE" USING
      BUFFER
      CORBA-TRUE-VALUE.
```

Flag

- CORBA-TRUE-VALUE:

  When the CORBA-FREE function is issued, the area indicated by ANY-VALUE is also released.

- CORBA-FALSE-VALUE:

  When the CORBA-FREE function is issued, the area indicated by ANY-VALUE is not released. (Default)

The release flag of the out parameter or return value allocated by the stub is set in CORBA-TRUE-VALUE.

The following is an example of client application processing.

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
  01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
  01 COPY ANY IN CORBA REPLACING CORBA-ANY BY SMP1.
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY T-LONG.
  01 TYPE-P USAGE POINTER.
  01 TC-ODSAMPLE-SAMPLE2-IMPL-SEQ.
     03 MEMBER1 PIC X(100) VALUE "69,69,0,0,0,0,15,4,0,8,0,25,73,
        68,76,58,79,68,115,97,109,112,108,101,47,115,97,109,112,
        108,101,50,58".
     03 MEMBER2 PIC X(100) VALUE ",49,46,48,0,0,8,115,97,109,112,
        108,101,50,0,0,2,0,6,112,97,114,97,49,0,0,1,9,0,6,112,97,
        114,97,50,0,".
     03 MEMBER3 PIC X(100) VALUE "0,1,6".
  01 TC-ODSAMPLE-SAMPLE3-IMPL-SEQ.
     03 MEMBER1 PIC X(100) VALUE "69,69,0,0,0,0,15,8,0,16,0,25,73,
        68,76,58,79,68,115,97,109,112,108,101,47,115,97,109,112,
        108,101,51,5".
     03 MEMBER2 PIC X(100) VALUE "8,49,46,48,0,0,8,115,97,109,112,
        108,101,51,0,0,2,0,6,112,97,114,97,49,0,0,1,9,0,6,112,97,
        114,97,50,0".
     03 MEMBER3 PIC X(100) VALUE ",0,1,7".
  01 COPY ANY IN CORBA REPLACING CORBA-ANY BY ANY1.
  01 ANY2 USAGE POINTER.
  01 ANY3 USAGE POINTER.
  01 ANY0 USAGE POINTER.
 LINKAGE SECTION.
  01 SMP2.
     02 COPY CHAR IN CORBA REPLACING CORBA-CHAR BY PARA1.
     02 COPY FLOAT IN CORBA REPLACING CORBA-FLOAT BY PARA2.
  01 SMP3.
     02 COPY CHAR IN CORBA REPLACING CORBA-CHAR BY PARA1.
     02 COPY DOUBLE IN CORBA REPLACING CORBA-DOUBLE BY PARA2.
  01 COPY ANY IN CORBA REPLACING CORBA-ANY BY ANY2-P.
  01 COPY ANY IN CORBA REPLACING CORBA-ANY BY ANY3-P.
  01 COPY ANY IN CORBA REPLACING CORBA-ANY BY ANY0-P.
```

PROCEDURE DIVISION.

```
*  Setting typecode
  MOVE FUNCTION LENG (TC-ODSAMPLE-SAMPLE3-IMPL-SEQ) TO T-LONG.
  CALL "CORBA-STRING-SET" USING
    TYPE-P
    T-LONG
    TC-ODSAMPLE-SAMPLE3-IMPL-SEQ.
  CALL "CORBA-ORB-TYPECODE-FROM-CGEN-TC" USING
      TYPE-P
      ANY-TYPE OF ANY1.
  CALL "CORBA-FREE" USING
    TYPE-P.
  CALL "ODSAMPLE-SAMPLE3-ALLOC" USING
      ANY-VALUE OF ANY1.
  SET ADDRESS OF SMP3 TO ANY-VALUE OF ANY1.
*  Setting the input parameteru
  MOVE "a" TO PARA1 OF SMP3.
*  Setting the input parameter
  MOVE 0.00001 TO PARA2 OF SMP3.
```

```
*  Allocating an input-output parameter area
  CALL "CORBA-ANY-ALLOC" USING
      ANY3.
  SET ADDRESS OF ANY3-P TO ANY3.
*  Setting TypeCode
  MOVE FUNCTION LENG (TC-ODSAMPLE-SAMPLE2-IMPL-SEQ) TO T-LONG.
  CALL "CORBA-STRING-SET" USING
    TYPE-P
    T-LONG
    TC-ODSAMPLE-SAMPLE2-IMPL-SEQ.
  CALL "CORBA-ORB-TYPECODE-FROM-CGEN-TC" USING
      TYPE-P
      ANY-TYPE OF ANY3-P.
  CALL "CORBA-FREE" USING
    TYPE-P.

  CALL "ODSAMPLE-SAMPLE2-ALLOC" USING
      ANY-VALUE OF ANY3-P.
  SET ADDRESS OF SMP2 TO ANY-VALUE OF ANY3-P.
*  Setting the input-output parameter
  MOVE "c" TO PARA1 OF SMP2.
*  Setting the input-output parameter
  MOVE 0.0001 TO PARA2 OF SMP2.
  CALL "ODSAMPLE-ANYTEST-OP1" USING
      OBJ
      ANY1
      ANY2
      ANY3
      ENV
      ANY0.
*  Releasing the return value area
  CALL "CORBA-FREE" USING
      ANY0.
*  Releasing the input parameter area
  CALL "CORBA-FREE" USING
      ANY-VALUE OF ANY1.
*  Releasing the output parameter area
  CALL "CORBA-FREE" USING
      ANY2.
*  Releasing the input-output parameter area
  CALL "CORBA-FREE" USING
      ANY3.
```

Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

The following table shows how the server application parameters are handled.

Table 7.10 Server Parameter Area (any type)

Parameter	Parameter passed from client	Parameter passed to client
in	Each area (CORBA-ANY or ANY-VALUE area) is automatically allocated or released by the skeleton.	_
inout	Each area (CORBA-ANY or ANY-VALUE area) is automatically allocated by the skeleton.	The ANY-VALUE area is released once using the CORBA-FREE function and re-allocated using the XX-ALLOC function. The area is automatically released by the skeleton.

Parameter	Parameter passed from client	Parameter passed to client
out return	_	Each area (CORBA-ANY or ANY-VALUE area) is allocated using the CORBA-ANY-ALLOC function or XX-ALLOC function. The area is automatically released by the skeleton.

The following is an example of server application processing.

IDENTIFICATION DIVISION.

```
 PROGRAM-ID.  "ODSAMPLE-ANYTEST-OP1".
```

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY T-LONG.
  01 TYPE-P USAGE POINTER.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
  01 STR-WORK PIC X(20).
  01 TC-ODSAMPLE-SAMPLE1-IMPL-SEQ.
     03 MEMBER1 PIC X(100) VALUE "71,71,0,0,0,0,15,4,0,8,0,25,73,68,76,
        58,79,68,115,97,109,112,108,101,47,115,97,109,112,108,101,49,58".
     03 MEMBER2 PIC X(100) VALUE ",49,46,48,0,0,8,115,97,109,112,108,101,
        49,0,0,2,0,6,112,97,114,97,49,0,0,1,3,0,6,112,97,114,97,50,0,".
     03 MEMBER3 PIC X(100) VALUE "0,3,18,0,0".
  01 TC-ODSAMPLE-SAMPLE2-IMPL-SEQ.
     03 MEMBER1 PIC X(100) VALUE "69,69,0,0,0,0,15,4,0,8,0,25,73,68,76,58,
        79,68,115,97,109,112,108,101,47,115,97,109,112,108,101,50,58".
     03 MEMBER2 PIC X(100) VALUE ",49,46,48,0,0,8,115,97,109,112,108,101,
        50,0,0,2,0,6,112,97,114,97,49,0,0,1,9,0,6,112,97,114,97,50,0,".
     03 MEMBER3 PIC X(100) VALUE "0,1,6".
  01 TC-ODSAMPLE-SAMPLE3-IMPL-SEQ.
     03 MEMBER1 PIC X(100) VALUE "69,69,0,0,0,0,15,8,0,16,0,25,73,68,76,58,
        79,68,115,97,109,112,108,101,47,115,97,109,112,108,101,51,5".
     03 MEMBER2 PIC X(100) VALUE "8,49,46,48,0,0,8,115,97,109,112,108,101,
        51,0,0,2,0,6,112,97,114,97,49,0,0,1,9,0,6,112,97,114,97,50,0".
   03 MEMBER3 PIC X(100) VALUE ",0,1,7".

 LINKAGE SECTION.
  01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
  01 SMP1.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
     02 PARA2 USAGE IS POINTER.
  01 SMP2.
     02 COPY CHAR IN CORBA REPLACING CORBA-CHAR BY PARA1.
     02 COPY FLOAT IN CORBA REPLACING CORBA-FLOAT BY PARA2.
  01 SMP3.
     02 COPY CHAR IN CORBA REPLACING CORBA-CHAR BY PARA1.
     02 COPY DOUBLE IN CORBA REPLACING CORBA-DOUBLE BY PARA2.
  01 SMP2 USAGE POINTER.
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
  01 COPY ANY IN CORBA REPLACING CORBA-ANY BY ANY1.
  01 ANY2 USAGE POINTER.
  01 COPY ANY IN CORBA REPLACING CORBA-ANY BY ANY2-P.
  01 ANY3 USAGE POINTER.
  01 COPY ANY IN CORBA REPLACING CORBA-ANY BY ANY3-P.
  01 ANY0 USAGE POINTER.
  01 COPY ANY IN CORBA REPLACING CORBA-ANY BY ANY0-P.
```

PROCEDURE DIVISION

```
OBJ ANY1 ANY2 ANY3 ENV ANY0.
MAIN.
*  Allocating an out parameter area
  CALL "CORBA-ANY-ALLOC"  USING
      ANY2.
  SET ADDRESS OF ANY2-P TO ANY2.
*  Setting typecode
  MOVE FUNCTION LENG (TC-ODSAMPLE-SAMPLE2-IMPL-SEQ) TO T-LONG.
  CALL "CORBA-STRING-SET" USING
    TYPE-P
    T-LONG
    TC-ODSAMPLE-SAMPLE2-IMPL-SEQ.
  CALL "CORBA-ORB-TYPECODE-FROM-CGEN-TC" USING
      TYPE-P
      ANY-TYPE OF ANY2-P.
  CALL "CORBA-FREE" USING
    TYPE-P.
*  Allocating an output parameter data area
  CALL "ODSAMPLE-SAMPLE2-ALLOC" USING
      ANY-VALUE OF ANY2-P.
  SET ADDRESS OF SMP2 TO ANY-VALUE OF ANY2-P.
*  Setting the output parameter
  MOVE "x" TO PARA1 OF SMP2.
*  Setting the output parameter
  MOVE 0.001 TO PARA2 OF SMP2.
  CALL "CORBA-ANY-SET-RELEASE" USING
      ANY2
      CORBA-TRUE-VALUE.

*  inout parameter processing
  SET ADDRESS OF ANY3-P TO ANY3.
  CALL "CORBA-FREE" USING ANY-VALUE OF ANY3-P.
  CALL "CORBA-ANY-SET-RELEASE" USING
      ANY3
      CORBA-TRUE-VALUE.
*  Setting typecode
  MOVE FUNCTION LENG (TC-ODSAMPLE-SAMPLE3-IMPL-SEQ) TO T-LONG.
  CALL "CORBA-STRING-SET" USING
    TYPE-P
    T-LONG
    TC-ODSAMPLE-SAMPLE3-IMPL-SEQ.
  CALL "CORBA-ORB-TYPECODE-FROM-CGEN-TC" USING
      TYPE-P
      ANY-TYPE OF ANY3-P.
  CALL "CORBA-FREE" USING
    TYPE-P.
  CALL "ODSAMPLE-SAMPLE3-ALLOC" USING
      ANY-VALUE OF ANY3-P.
  SET ADDRESS OF SMP3 TO ANY-VALUE OF ANY3-P.
  MOVE "y" TO PARA1 OF SMP3.
  MOVE 0.0001 TO PARA2 OF SMP3.

*  Return value processing
  CALL "CORBA-ANY-ALLOC" USING ANY0.
  SET ADDRESS OF ANY0-P TO ANY0.

  MOVE FUNCTION LENG (TC-ODSAMPLE-SAMPLE1-IMPL-SEQ) TO T-LONG.
  CALL "CORBA-STRING-SET" USING
    TYPE-P
    T-LONG
    TC-ODSAMPLE-SAMPLE1-IMPL-SEQ.
  CALL "CORBA-ORB-TYPECODE-FROM-CGEN-TC" USING
      TYPE-P
```

```
      ANY-TYPE OF ANY0-P.
  CALL "CORBA-FREE" USING
    TYPE-P.

  CALL "ODSAMPLE-SAMPLE1-ALLOC" USING
      ANY-VALUE OF ANY0-P.
  SET ADDRESS OF SMP1 TO ANY-VALUE OF ANY0-P.
  MOVE 300 TO PARA1 OF SMP1.
  MOVE 5 TO LSIZE.
  MOVE "test" TO STR-WORK.
  CALL "CORBA-STRING-SET" USING
      PARA2 OF SMP1
      LSIZE
      STR-WORK.
  CALL "CORBA-ANY-SET-RELEASE" USING
      ANY0
      CORBA-TRUE-VALUE.

 MAIN-END.
END PROGRAM " ODSAMPLE-ANYTEST-OP1".
```

7.7.4 Sequence Type

This section provides information about sequence type data.

IDL Mapping

When the sequence type (*sequence*) is specified in IDL, data is declared in COBOL using the following structure (sequence structure). A function for allocating the sequence structure area (the function name is CORBA-SEQUENCE-datatype-ALLOC, hereafter referred to as the XX-ALLOC function) is generated by the IDL compiler. A function for allocating the sequence data area (the function name is CORBA-SEQUENCE-datatype-ALLOCBUF, hereafter referred to as the XX-ALLOCBUF function) is also generated by the IDL compiler.

```
01 CORBA-SEQUENCE.
*  Maximum sequence length
 03 SEQ-MAXIMUM PIC S9(9) COMP-5.
*  Sequence length
 03 SEQ-LENGTH PIC S9(9) COMP-5.
*  Sequence data
 03 SEQ-BUFFER USAGE POINTER.
```

This is explained using the following IDL definition example.

IDL

Sequence (basic type)

```
module ODsample{
  interface  seqtest{
             typedef sequence<long>        sampleseq;
             sampleseq op1(in sampleseq seq1, out sampleseq seq2, inout sampleseq seq3 );
       };
};
```

Sequence (structure type)

```
module ODsample{
  interface  seqtest{
             struct structV { string st; };
             typedef sequence<structV>  sampleseq;
             sampleseq op1(in sampleseq seq1, out sampleseq seq2, inout sampleseq seq3 );
```

```
        };
};
```

COBOL

```
  01 ODSAMPLE-SEQTEST-SAMPLESEQ.
*  Maximum sequence length
  02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
*  Sequence length
  02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
*  Sequence data
  02 SEQ-BUFFER USAGE IS POINTER.
```

Parameters Handled by Client Applications

The following table shows how the client application parameters are handled.

Table 7.11 Client Parameter Area (Sequence Type)

Parameter	Parameter passed to server	Parameter passed from server
in	The maximum number of arrays, the number of arrays to be used, and the data area address are set as sequence structure members. The XX-ALLOC function or the XX-ALLOCBUF function is used to allocate the sequence structure area or SEQ-BUFFER area.	_
inout	Same as the *in* parameter	The area is automatically allocated by the stub. (Each member is also set.)
out return	_	Same as the *inout* parameter

When an area that has been allocated by the client and stub is no longer required, it must be released using the CORBA-FREE function. Whether or not the data area (allocated using the XX-ALLOCBUF function) is to be released is specified in the release flag.

The following functions and flags are used to reference and set the release flag.

Function

```
CALL "CORBA-SEQUENCE-GET-RELEASE" USING
      BUFFER
      RET-VAL.
CALL "CORBA-SEQUENCE-SET-RELEASE" USING
      BUFFER
      CORBA-TRUE-VALUE.
```

Flag

- CORBA-TRUE-VALUE:

  When the CORBA-FREE function is issued, the area indicated by SEQ-BUFFER is also released.

- CORBA-FALSE-VALUE:

  When the CORBA-FREE function is issued, the area indicated by SEQ-BUFFER is not released. (Default)

The release flag of the out parameter or return value allocated by the stub is set in CORBA-TRUE-VALUE.

The following is an example of client application processing.

Sequence (basic type)

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
  01 SEQ1.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
     02 SEQ-BUFFER USAGE IS POINTER.
     01 SEQ2 USAGE POINTER.


  01 SEQ3-P USAGE POINTER.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY I.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LONG-V.
  01 BUF-IN-P USAGE POINTER.
  01 LONG-IN-P USAGE POINTER.
  01 BUF-IO-P USAGE POINTER.
  01 LONG-IO-P USAGE POINTER.
LINKAGE SECTION.
  01 SEQ3.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
      02 SEQ-BUFFER USAGE IS POINTER.
```

PROCEDURE DIVISION.

```
*  in parameter processing
  MOVE 2 TO SEQ-MAXIMUM OF SEQ1.
  MOVE 2 TO SEQ-LENGTH OF SEQ1.
  CALL "CORBA-SEQUENCE-LONG-ALLOCBUF" USING
        SEQ-LENGTH OF SEQ1
        SEQ-BUFFER OF SEQ1.
  MOVE 10 TO LONG-V.
  MOVE FUNCTION ADDR (SEQ1) TO BUF-IN-P.
  PERFORM VARYING I FROM 1 BY 1 UNTIL I > SEQ-LENGTH OF SEQ1
        COMPUTE LONG-V = 20 + LONG-V
        MOVE FUNCTION ADDR(LONG-V) TO LONG-IN-P
        CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
                                    BUF-IN-P
                                    I
                                    LONG-IN-P
  END-PERFORM
* inout parameter processing
  CALL "CORBA-SEQUENCE-LONG-ALLOC" USING
      SEQ3-P.
  SET ADDRESS OF SEQ3 TO SEQ3-P.
  MOVE 3 TO SEQ-MAXIMUM OF SEQ3.
  MOVE 3 TO SEQ-LENGTH OF SEQ3.
  CALL "CORBA-SEQUENCE-LONG-ALLOCBUF" USING
        SEQ-LENGTH OF SEQ3
        SEQ-BUFFER OF SEQ3.
  MOVE 5 TO LONG-V.
  MOVE FUNCTION ADDR (SEQ3) TO BUF-IO-P.
  PERFORM VARYING I FROM 1 BY 1 UNTIL I > SEQ-LENGTH OF SEQ3
```

```
       COMPUTE LONG-V = 30 + LONG-V
       MOVE FUNCTION ADDR(LONG-V) TO LONG-IO-P
       CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
                                   BUF-IO-P
                                   I
                                   LONG-IO-P
  END-PERFORM
  CALL "CORBA-SEQUENCE-SET-RELEASE" USING
        SEQ3-P
        CORBA-TRUE-VALUE.
  CALL "ODSAMPLE-SEQTEST-OP1" USING
      OBJ
      SEQ1
      SEQ2
      SEQ3-P
      ENV
      SEQ0.
  CALL "CORBA-FREE" USING
      SEQ0.
  CALL "CORBA-FREE" USING
      SEQ1.
  CALL "CORBA-FREE" USING
      SEQ-BUFFER OF SEQ1.
  CALL "CORBA-FREE" USING
      SEQ3-P.
```

Sequence (basic type)

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
   COPY CONST IN CORBA.
    01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
    01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
    01OP1-IN-P.
       02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
       02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
       02 SEQ-BUFFER USAGE IS POINTER.
    01 OP1-OUT-P USAGE POINTER.
    01 OP1-IO-P USAGE POINTER.
    01 OP1-RESULT USAGE IS POINTER.
    01 TMP-ODSAMPLE-SEQTEST-STRUCTV..
      02 ST USAGE IS POINTER.
    01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY WS-NUM.
    01 TMP-POINTER USAGE IS POINTER.
    01 TMP-SEQ-ADDR USAGE IS POINTER.
    01 TMP-BUF    PIC X(20).
01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY TMP-LEN.
  LINKAGE SECTION.
    01 TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
        02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
        02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
        02 SEQ-BUFFER USAGE IS POINTER.
```

PROCEDURE DIVISION.

```
  * in parameter processing
    MOVE 10 TO SEQ-MAXIMUM OF OP1-IN-P.
    MOVE 10 TO SEQ-LENGTH OF OP1-IN-P.
    CALL "CORBA-SEQUENCE-ODSAMPLE-SEQTEST-STRUCTV-ALLOCBUF" USING
          SEQ-MAXIMUM OF OP1-IN-P
```

```
      SEQ-BUFFER OF OP1-IN-P.
  MOVE 10 TO LONG-V.
  MOVE FUNCTION ADDR( OP1-IN-P ) TO TMP-SEQ-ADDR.
  PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH OF OP1-IN-P
      MOVE "ABC" TO TMP-BUF
      MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
      CALL "CORBA-STRING-SET" USING
          ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV
          TMP-LEN
          TMP-BUF
      MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-STRUCTV ) TO TMP-POINTER
      CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
          TMP-SEQ-ADDR
          WS-NUM
          TMP-POINTER
  END-PERFORM.
  MOVE FUNCTION ADDR( OP1-IN-P ) TO TMP-SEQ-ADDR
  PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH OF OP1-IN-P
      CALL "CORBA-SEQUENCE-ELEMENT-GET" USING
          TMP-SEQ-ADDR
          WS-NUM
          TMP-POINTER
      SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L TO TMP-POINTER
      MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
      CALL "CORBA-STRING-GET" USING
          ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L
          TMP-LEN
          TMP-BUF
  END-PERFORM.
* inout parameter processing
  CALL "CORBA-SEQUENCE-ODSAMPLE-SEQTEST-STRUCTV-ALLOC" USING
      OP1-IO-P.
  SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ TO OP1-IO-P.
  MOVE 10 TO SEQ-MAXIMUM OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
  MOVE 10 TO SEQ-LENGTH OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
  CALL "CORBA-SEQUENCE-ODSAMPLE-SEQTEST-STRUCTV-ALLOCBUF" USING
          SEQ-MAXIMUM OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
          SEQ-BUFFER OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
  MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-SAMPLESEQ ) TO TMP-SEQ-ADDR
  PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH
      OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
      MOVE "DEF" TO TMP-BUF
      MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
      CALL "CORBA-STRING-SET" USING
          ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV
          TMP-LEN
          TMP-BUF
      MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-STRUCTV ) TO TMP-POINTER
      CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
          TMP-SEQ-ADDR
          WS-NUM
          TMP-POINTER
  END-PERFORM.
  SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ TO OP1-IO-P.
  MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-SAMPLESEQ ) TO TMP-SEQ-ADDR
  PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH
      OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
      CALL "CORBA-SEQUENCE-ELEMENT-GET" USING
         TMP-SEQ-ADDR
          WS-NUM
        TMP-POINTER
      SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L TO TMP-POINTER
      MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
```

```
        CALL "CORBA-STRING-GET" USING
            ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L
            TMP-LEN
            TMP-BUF
    END-PERFORM.
    CALL "CORBA-SEQUENCE-SET-RELEASE" USING
          OP1-IO-P
          CORBA-TRUE-VALUE.
    CALL "ODSAMPLE-SEQTEST-OP1" USING
        OBJ
        OP1-IN-P
        OP1-OUT-P
        OP1-IO-P
        ENV
        OP1-RESULT.
    CALL "CORBA-FREE" USING
        OP1-RESULT.
    CALL "CORBA-FREE" USING
        SEQ-BUFFER OF OP1-IN-P.
    CALL "CORBA-FREE" USING
       OP1-OUT-P.
    CALL "CORBA-FREE" USING
       OP1-IO-P.
```

Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

The maximum number of arrays, the number of arrays to be used, and the data area address are set as sequence structure members.

The XX-ALLOC function or the XX-ALLOCBUF function is used to allocate the sequence structure area or SEQ-BUFFER area.

The following table shows how the server application parameters are handled.

Table 7.12 Server Parameter Area (Sequence Type)

Parameter	Parameter passed from client	Parameter passed to client
in	Each area (sequence structure area or data area) is automatically allocated or released by the skeleton.	_
inout	Each area (sequence structure area or data area) is automatically allocated by the skeleton.	The SEQ-BUFFER area is released once using the CORBA-FREE function, and re-allocated using the XX-ALLOCBUF function. The area is automatically released by the skeleton.
out return	_	The sequence structure area or SEQ-BUFFER area is allocated using the XX-ALLOC function or the XX-ALLOCBUF function. The area is automatically released by the skeleton.

The following is an example of server application processing.

Sequence (basic type)

IDENTIFICATION DIVISION.

```
PROGRAM-ID.  "ODSAMPLE-SEQTEST-OP1".
```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.

```
    SYMBOLIC CONSTANT
    COPY SYMBOL-CONST IN CORBA.
    .
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
  COPY CONST IN CORBA.
  01 STR  USAGE POINTER.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
  01 STR-WORK PIC X(30).
   03 IDL-ID USAGE POINTER.
   03 MINOR PIC 9(9) COMP-5.
   03 IDL-STATUS PIC 9(9) COMP-5.
   03 PARAM USAGE POINTER.
   03 MAGIC PIC 9(9) COMP-5.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY I.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
  01 LONG-OUT-P USAGE POINTER.
  01 LONG-IO-P  USAGE POINTER.
  01 LONG-RET-P USAGE POINTER.
  01 BUF-OUT-P  USAGE POINTER.
  01 BUF-IO-P   USAGE POINTER.
  01 BUF-RET-P  USAGE POINTER.
 LINKAGE SECTION
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
  01 SEQ.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
    02 SEQ-BUFFER USAGE POINTER.
  01 SEQ1.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
    02 SEQ-BUFFER USAGE POINTER.
  01 SEQ2.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
    02 SEQ-BUFFER USAGE POINTER.
  01 SEQ3.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
    02 SEQ-BUFFER USAGE POINTER.
  01 SEQ1-P USAGE POINTER.
  01 SEQ2-P USAGE POINTER.
  01 SEQ3-P USAGE POINTER.
  01 ENV.
   03 MAJOR PIC 9(9) COMP-5.
     88 CORBA-NO-EXCEPTION VALUE 0.
     88 CORBA-USER-EXCEPTION VALUE 1.
     88 CORBA-SYSTEM-EXCEPTION VALUE 2.
```

PROCEDURE DIVISION USING

```
      OBJ
      SEQ1
      SEQ2-P
      SEQ3-P
      ENV
      SEQ-P.

MAIN.
*  out parameter processing
  INVOKE "CORBA-SEQUENCE-LONG-ALLOC" USING
```

```
      SEQ2-P
  SET ADDRESS OF SEQ2 TO SEQ2-P.
  MOVE 4 TO SEQ-MAXIMUM OF SEQ2.
  MOVE 4 TO SEQ-LENGTH OF SEQ2.
  CALL "CORBA-SEQUENCE-LONG-ALLOCBUF" USING
        SEQ-LENGTH OF SEQ2
        SEQ-BUFFER OF SEQ2.
  MOVE FUNCTION ADDR (SEQ2) TO BUF-OUT-P.
  PERFORM VARYING I FROM 1 BY 1 UNTIL I > SEQ-LENGTH OF SEQ2
        COMPUTE NUM = I * 100
        MOVE FUNCTION ADDR(NUM) TO LONG-OUT-P
        CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
             BUF-OUT-P
             I
             LONG-OUT-P
  END-PERFORM
  CALL "CORBA-SEQUENCE-SET-RELEASE" USING
      SEQ2-P
        CORBA-TRUE-VALUE.

*  inout parameter processing
  CALL "CORBA-FREE" USING
        SEQ-BUFFER OF SEQ3.
  MOVE 5 TO SEQ-MAXIMUM OF SEQ3.
  MOVE 5 TO SEQ-LENGTH OF SEQ3.
  CALL "CORBA-SEQUENCE-LONG-ALLOCBUF" USING
        SEQ-LENGTH OF SEQ3
        SEQ-BUFFER OF SEQ3.
  PERFORM VARYING I FROM 1 BY 1 UNTIL I > SEQ-LENGTH OF SEQ3
        COMPUTE NUM = I * 1000
        MOVE FUNCTION ADDR(NUM) TO LONG-IO-P
        CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
            BUF-IO-P
            I
            LONG-IO-P
  END-PERFORM
  CALL "CORBA-SEQUENCE-SET-RELEASE" USING
      SEQ3-P
        CORBA-TRUE-VALUE.

*  Return value processing
  CALL "CORBA-SEQUENCE-LONG-ALLOC" USING
      SEQ-P.
  SET ADDRESS OF SEQ TO SEQ-P.
  MOVE 6 TO SEQ-MAXIMUM OF SEQ.
  MOVE 6 TO SEQ-LENGTH OF SEQ.
  MOVE FUNCTION ADDR(SEQ) TO BUF-RET-P.
  CALL "CORBA-SEQUENCE-LONG-ALLOCBUF" USING
        SEQ-LENGTH OF SEQ
        SEQ-BUFFER OF SEQ.
  PERFORM VARYING I FROM 1 BY 1 UNTIL I > SEQ-LENGTH OF SEQ
        COMPUTE NUM = I * 10000
        MOVE FUNCTION ADDR(NUM) TO LONG-RET-P
        CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
            BUF-RET-P
            I
            LONG-RET-P
  END-PERFORM
  CALL "CORBA-SEQUENCE-SET-RELEASE" USING
      SEQ-P
        CORBA-TRUE-VALUE.
  MAIN-END.
 END PROGRAM "ODSAMPLE-SEQTEST-OP1".
```

Sequence (basic type)

IDENTIFICATION DIVISION.

```
 PROGRAM-ID.  "ODSAMPLE-SEQTEST-OP1".
```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.

```
     SYMBOLIC CONSTANT
     COPY SYMBOL-CONST IN CORBA.
     .
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
   COPY CONST IN CORBA.
   01 TMP-POINTER USAGE IS POINTER.
   01 TMP-SEQ-ADDR USAGE IS POINTER.
   01 TMP-BUF    PIC X(20).
   01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY TMP-LEN.
   01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY WS-NUM.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY WS-LONG.
   01 TMP-ODSAMPLE-SEQTEST-STRUCTV.
      02 ST USAGE IS POINTER.
  LINKAGE SECTION.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
   01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
   01 OP1-IN-P.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
     02 SEQ-BUFFER USAGE IS POINTER.
   01 OP1-OUT-P USAGE IS POINTER.
   01 OP1-IO-P     USAGE IS POINTER.
   01 OP1-RESULT USAGE IS POINTER.
   01 TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
     02 SEQ-BUFFER USAGE IS POINTER.
   01 TMP-ODSAMPLE-SEQTEST-STRUCTV-L.
     02 ST USAGE IS POINTER.
```

PROCEDURE DIVISION USING

```
       OBJ
       OP1-IN-P
       OP1-OUT-P
       OP1-IO-P
       ENV
       OP1-RESULT.
 MAIN.
 * out parameter processing
   CALL "CORBA-SEQUENCE-ODSAMPLE-SEQTEST-STRUCTV-ALLOC" USING OP1-OUT-P.
   CALL "CORBA-SEQUENCE-SET-RELEASE" USING OP1-OUT-P CORBA-TRUE-VALUE.
   SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ TO OP1-OUT-P.
   MOVE 10 TO SEQ-MAXIMUM OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
   MOVE 10 TO SEQ-LENGTH OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
   CALL "CORBA-SEQUENCE-ODSAMPLE-SEQTEST-STRUCTV-ALLOCBUF" USING
           SEQ-MAXIMUM OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
           SEQ-BUFFER OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
   MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-SAMPLESEQ ) TO TMP-SEQ-ADDR
   PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH
       OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
```

```
        MOVE "JKL" TO TMP-BUF
        MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
        CALL "CORBA-STRING-SET" USING
            ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV
            TMP-LEN
            TMP-BUF
        MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-STRUCTV ) TO TMP-POINTER
        CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
            TMP-SEQ-ADDR
            WS-NUM
            TMP-POINTER
    END-PERFORM.
    SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ TO OP1-OUT-P.
    MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-SAMPLESEQ ) TO TMP-SEQ-ADDR
    PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH
        OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
        CALL "CORBA-SEQUENCE-ELEMENT-GET" USING
            TMP-SEQ-ADDR
            WS-NUM
            TMP-POINTER
        SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L TO TMP-POINTER
        MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
        CALL "CORBA-STRING-GET" USING
            ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L
            TMP-LEN
            TMP-BUF
    END-PERFORM.
    CALL "CORBA-SEQUENCE-SET-RELEASE" USING
       OP1-OUT-P
       CORBA-TRUE-VALUE.
*   inout parameter processing
    CALL "CORBA-SEQUENCE-ODSAMPLE-SEQTEST-STRUCTV-ALLOC" USING OP1-IO-P.
    CALL "CORBA-SEQUENCE-SET-RELEASE" USING OP1-IO-P CORBA-TRUE-VALUE.
    SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ TO OP1-IO-P.
    MOVE 10 TO SEQ-MAXIMUM OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
    MOVE 10 TO SEQ-LENGTH OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
    CALL "CORBA-SEQUENCE-ODSAMPLE-SEQTEST-STRUCTV-ALLOCBUF" USING
            SEQ-MAXIMUM OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
            SEQ-BUFFER OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
    MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-SAMPLESEQ ) TO TMP-SEQ-ADDR
    PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH
        OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
        MOVE "GHI" TO TMP-BUF
        MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
        CALL "CORBA-STRING-SET" USING
            ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV
            TMP-LEN
            TMP-BUF
        MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-STRUCTV ) TO TMP-POINTER
        CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
           TMP-SEQ-ADDR
            WS-NUM
           TMP-POINTER
    END-PERFORM.
    SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ TO OP1-IO-P.
    MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-SAMPLESEQ ) TO TMP-SEQ-ADDR
    PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH
        OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
        CALL "CORBA-SEQUENCE-ELEMENT-GET" USING
            TMP-SEQ-ADDR
            WS-NUM
            TMP-POINTER
        SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L TO TMP-POINTER
```

```
        MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
        CALL "CORBA-STRING-GET" USING
            ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L
            TMP-LEN
            TMP-BUF
    END-PERFORM.
    CALL "CORBA-SEQUENCE-SET-RELEASE" USING
       OP1-IO-P
       CORBA-TRUE-VALUE.
  * return value processing
    CALL "CORBA-SEQUENCE-ODSAMPLE-SEQTEST-STRUCTV-ALLOC" USING OP1-RESULT.
    CALL "CORBA-SEQUENCE-SET-RELEASE" USING OP1-RESULT CORBA-TRUE-VALUE.
    SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ TO OP1-RESULT.
    MOVE 10 TO SEQ-MAXIMUM OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
    MOVE 10 TO SEQ-LENGTH OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ.
    CALL "CORBA-SEQUENCE-ODSAMPLE-SEQTEST-STRUCTV-ALLOCBUF" USING
            SEQ-MAXIMUM OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
            SEQ-BUFFER OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
    MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-SAMPLESEQ ) TO TMP-SEQ-ADDR
    PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH
        OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
        MOVE "MNO" TO TMP-BUF
        MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
        CALL "CORBA-STRING-SET" USING
            ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV
            TMP-LEN
            TMP-BUF
        MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-STRUCTV ) TO TMP-POINTER
        CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
            TMP-SEQ-ADDR
            WS-NUM
            TMP-POINTER
    END-PERFORM.
    SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ TO OP1-RESULT.
    MOVE FUNCTION ADDR( TMP-ODSAMPLE-SEQTEST-SAMPLESEQ ) TO TMP-SEQ-ADDR
    PERFORM VARYING WS-NUM FROM 1 BY 1 UNTIL WS-NUM > SEQ-LENGTH
        OF TMP-ODSAMPLE-SEQTEST-SAMPLESEQ
        CALL "CORBA-SEQUENCE-ELEMENT-GET" USING
            TMP-SEQ-ADDR
            WS-NUM
            TMP-POINTER
        SET ADDRESS OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L TO TMP-POINTER
        MOVE FUNCTION LENG( TMP-BUF ) TO TMP-LEN
        CALL "CORBA-STRING-GET" USING
            ST OF TMP-ODSAMPLE-SEQTEST-STRUCTV-L
            TMP-LEN
            TMP-BUF
    END-PERFORM.
    CALL "CORBA-SEQUENCE-SET-RELEASE" USING
       OP1-RESULT
       CORBA-TRUE-VALUE.
    MAIN-END.
END PROGRAM "ODSAMPLE-SEQTEST-OP1".
```

7.7.5 Structure Type

This section provides information about structure type data.

IDL Mapping

If structure-type struct is specified in IDL, data is declared as structure type in COBOL.

For a variable-length structure, the function for allocating structure location is generated in the IDL compiler. This function is named using the module name, interface name, and structures and ALLOC concatenated using hyphen ("-").

From now on, this function is invoked using the XX-ALLOCBUF function.

Following is an example of STRUCT defined in IDL:

IDL

```
module ODsample{
        struct samplefix {              // Structure (fixed length)
                long        para1;
                long        para2;
        };
        struct samplevar {              // Structure (variable length)
                long        para1;
                string      para2;
        };
        interface        structtest{
                samplefix   op2(
                            in samplefix str1,
                            out samplefix str2,
                            inout samplefix str3
                );
                samplevar   op1(
                            in samplevar str1,
                            out samplevar str2,
                            inout samplevar str3
                );
        };
};
```

COBOL

```
* Structure (fixed length)
    01 ODSAMPLE-SAMPLEFIX.
       02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
       02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
* Structure (variable length)
    01 ODSAMPLE-SAMPLEVAR.
       02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
       02 PARA2 USAGE IS POINTER.
```

Fixed-length Parameters Handled by Client Applications

When a client application is handling in, out and inout parameters of a structure (fixed length), there is no need to define any special location allocation or release functions. Send the structure address as a function parameter.

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
  COPY CONST IN CORBA.
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
  01 FIX1.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
  01 FIX2 .
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
  01 FIX3.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
  01 FIX0 .
```

```
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
```

PROCEDURE DIVISION.

```
*  Setting the in parameter
  MOVE 10 TO PARA1 OF FIX1.
*  Setting the in parameter
  MOVE 11 TO PARA2 OF FIX1.
*  Setting the inout parameter
  MOVE 20 TO PARA1 OF FIX3.
*  Setting the inout parameter
  MOVE 21 TO PARA2 OF FIX3.

  CALL "ODSAMPLE-STRUCTTEST-OP2" USING
       OBJ
       FIX1
       FIX2
       FIX3
       ENV
       FIX0.
```

Variable-length Parameters Handled by Client Applications

The following table shows how the client application parameters are handled.

Table 7.13 Client Parameter Area (Structure or Variable-length Data Area)

Parameter	Parameter passed to server	Parameter passed from server
in	The XX-ALLOC function or the data area allocation function is used to allocate the structure area or variable-length data area.	_
inout	Same as the *in* parameter	The area is automatically allocated by the stub.
out return	_	Same as the *inout* parameter

Note

When an area that has been dynamically allocated by the client and stub is no longer required, it must be released using the CORBA-FREE function. The variable-length data area is also released when the CORBA-FREE function is issued.

The following is an example of client application processing.

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
   COPY CONST IN CORBA.
   01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
   01 VAR1.
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
      02 PARA2 USAGE IS POINTER.
   01 VAR2 USAGE POINTER.
   01 VAR3-P USAGE POINTER.
   01 VAR0 USAGE POINTER.
   01 STR-WORK PIC X(30).
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
LINKAGE SECTION.
   01 VAR3.
```

```
      02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
      02 PARA2 USAGE IS POINTER.
```

PROCEDURE DIVISION.

```
*  Setting the in parameter
   MOVE 5 TO PARA1 OF VAR1.
   MOVE "test" TO STR-WORK.
   MOVE FUNCTION LENG(STR-WORK) TO LSIZE
   CALL "CORBA-STRING-SET" USING
        PARA2 OF VAR1
        LSIZE
        STR-WORK.
*  Allocating an inout parameter area
   CALL "ODSAMPLE-SAMPLEVAR-ALLOC" USING
         VAR3-P.
   SET ADDRESS OF VAR3 TO VAR3-P.
*  Setting the inout parameter
   MOVE 4 TO PARA1 OF VAR3.
   MOVE "pro" TO STR-WORK.
   MOVE FUNCTION LENG(STR-WORK) TO LSIZE.
   CALL "CORBA-STRING-SET" USING
        PARA2 OF VAR3
        LSIZE
        STR-WORK.
   CALL "ODSAMPLE-STRUCTTEST-OP1" USING
       OBJ
       VAR1
       VAR2
       VAR3-P
       ENV
       VAR0.
   CALL "CORBA-FREE" USING PARA2 OF VAR1.
   CALL "CORBA-FREE" USING VAR2.
   CALL "CORBA-FREE" USING VAR3-P.
   CALL "CORBA-FREE" USING VAR0.
```

Fixed-length Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

When a server application handles the in, out, and inout parameters of the structure (fixed length), there is no need to specify any special location allocation or release function. The processing result in the structure member for the out and inout parameters must be set.

IDENTIFICATION DIVISION.

```
PROGRAM-ID.  "ODSAMPLE-STRUCTTEST-OP2".
```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.

```
    SYMBOLIC CONSTANT
    COPY SYMBOL-CONST IN CORBA.
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
 01 STR-WORK PIC X(30).
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
LINKAGE SECTION.
 01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
 01 STR1.
```

```
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
 01 STR2.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
 01 STR3.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
 01 ENV.
    03 MAJOR PIC 9(9) COMP-5.
       88 CORBA-NO-EXCEPTION VALUE 0.
       88 CORBA-USER-EXCEPTION VALUE 1.
       88 CORBA-SYSTEM-EXCEPTION VALUE 2.
    03 IDL-ID USAGE POINTER.
    03 MINOR PIC 9(9) COMP-5.
    03 IDL-STATUS PIC 9(9) COMP-5.
    03 PARAM USAGE POINTER.
    03 MAGIC PIC 9(9) COMP-5.
 01 FIX.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
```

PROCEDURE DIVISION

```
USING
    OBJ
    STR1
    STR2
    STR3
    ENV
    FIX.

MAIN SECTION.
*  Setting the out parameter
   MOVE 0 TO PARA1 OF STR2.
   MOVE 1 TO PARA2 OF STR2.
*  Setting the inout parameter
   MOVE 2 TO PARA1 OF STR3.
   MOVE 3 TO PARA2 OF STR3.
*  Return value processing
   MOVE 4 TO PARA1 OF FIX.
   MOVE 5 TO PARA2 OF FIX.

 MAIN-END.
END PROGRAM "ODSAMPLE-STRUCTTEST-OP2".
```

Variable-length Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

The following table shows how the server application parameters are handled.

Table 7.14 Server Parameter Area (Structure or Variable-length Data Area)

Parameter	Parameter area passed from client	Parameter area passed to client
in	The structure area or variable-length data area is automatically allocated or released by the skeleton.	_
inout	The structure area or variable-length data area is automatically allocated by the skeleton.	The variable-length data area is released once using the CORBA-FREE function and re-allocated using the data area allocation function. The area is automatically released by the skeleton.

Parameter	Parameter area passed from client	Parameter area passed to client
out return	_	The structure area or variable-length data area is allocated using the XX-ALLOC function or the data area allocation function. The area is automatically released by the skeleton.

The following is an example of server application processing.

IDENTIFICATION DIVISION.

```
PROGRAM-ID.  "ODSAMPLE-STRUCTTEST-OP1".
```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.

```
    SYMBOLIC CONSTANT
    COPY SYMBOL-CONST IN CORBA.
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
 01 STR-WORK PIC X(30).
LINKAGE SECTION.
 01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
 01 STR1.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    02 PARA2 USAGE IS POINTER.
 01 STR2 USAGE IS POINTER.
 01 STR2-P.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    02 PARA2 USAGE IS POINTER.
 01 STR3 USAGE IS POINTER.
 01 STR3-P.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    02 PARA2 USAGE IS POINTER.
 01 STR0-P.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    02 PARA2 USAGE IS POINTER.
 01 STR0 USAGE IS POINTER.
 01 ENV.
    03 MAJOR PIC 9(9) COMP-5.
       88 CORBA-NO-EXCEPTION VALUE 0.
       88 CORBA-USER-EXCEPTION VALUE 1.
       88 CORBA-SYSTEM-EXCEPTION VALUE 2.
    03 IDL-ID USAGE POINTER.
    03 MINOR PIC 9(9) COMP-5.
    03 IDL-STATUS PIC 9(9) COMP-5.
    03 PARAM USAGE POINTER.
    03 MAGIC PIC 9(9) COMP-5.
```

PROCEDURE DIVISION

```
USING
    OBJ
    STR1
    STR2
    STR3
    ENV
    STR0.
```

```
MAIN SECTION.
*  out parameter processing
  CALL "ODSAMPLE-SAMPLEVAR-ALLOC" USING
         STR2.
  SET ADDRESS OF STR2-P TO STR2.
  MOVE 12 TO PARA1 OF STR2-P.
  MOVE "(*str2)->para2" TO STR-WORK.
  MOVE FUNCTION LENG(STR-WORK) TO LSIZE
  CALL "CORBA-STRING-SET" USING
        PARA2 OF STR2-P
        LSIZE
        STR-WORK.

*  inout parameter processing
  SET ADDRESS OF STR3-P TO STR3.
  CALL "CORBA-FREE" USING
         PARA2 OF STR3-P.
  MOVE 12 TO PARA1 OF STR3-P.
  MOVE 16 TO LSIZE.
  MOVE "(*str3)->para2" TO STR-WORK.
  MOVE FUNCTION LENG(STR-WORK) TO LSIZE
  CALL "CORBA-STRING-SET" USING
        PARA2 OF STR3-P
        LSIZE
        STR-WORK.

*  Return value processing
  CALL "ODSAMPLE-SAMPLEVAR-ALLOC" USING
         STR0.
  SET ADDRESS OF STR0-P TO STR0.
  MOVE 11 TO PARA1 OF STR0-P.
  MOVE 16 TO LSIZE.
  MOVE "(*str2)->para2" TO STR-WORK.
  MOVE FUNCTION LENG(STR-WORK) TO LSIZE
  CALL "CORBA-STRING-SET" USING
        PARA2 OF STR0-P
        LSIZE
        STR-WORK.
 MAIN-END.
END PROGRAM "ODSAMPLE-STRUCTTEST-OP1".
```

7.7.6 Union

This section provides information on union type data.

IDL Mapping

When union-type union is specified in IDL, data is declared as a structure in COBOL consisting of identification information D, to identify the data type, and union-type U.

The function for allocating structure location is generated in the IDL compiler. The function name is composed of module name, interface name, union name and ALLOC concatenated using hyphen ("-"). From now on, this is invoked using the XX-ALLOCBUF function.

IDL

```
module ODsample{
              union samplefix switch(long){               // Union (fixed length)
                case 1:     long        para1;
                case 2:     long        para2;
        };
        union samplevar switch(long){               // Union (variable length)
                case 1:     long        para1;
                case 2:     string      para2;
```

```
        };
        interface        uniontest{
                samplefix   op2(
                            in samplefix uni1,
                            out samplefix uni2,
                            inout samplefix uni3
                                );
                samplevar   op1(
                            in samplevar uni1,
                            out samplevar uni2,
                            inout samplevar uni3
                                );
        };
};
```

COBOL

```
* Union (fixed length)
 01 ODSAMPLE-SAMPLEFIX.
   02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
   02 U.
     03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
   02 FILLER REDEFINES U.
     03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.

* Union (variable length)
 01 ODSAMPLE-SAMPLEVAR.
   02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
   02 U.
     03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
   02 FILLER REDEFINES U.
     03 PARA2 USAGE IS POINTER.
```

Fixed-length Parameters Handled by Client Applications

It is not necessary to allocate or release a structure location. Set the identification value of the data type to be set, and the corresponding data type value as identifying information in the structure.

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
  COPY CONST IN CORBA.
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
  01 UNI1.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
    02 U.
      03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
    02 FILLER REDEFINES U.
      03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
  01 UNI2.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
    02 U.
      03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
    02 FILLER REDEFINES U.
      03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
  01 UNI3.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
    02 U.
      03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
    02 FILLER REDEFINES U.
      03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
  01 UNI0.
```

```
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
    02 U.
      03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
    02 FILLER REDEFINES U.
      03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY L-SIZE.
  01 STR-WORK PIC X(10).
```

PROCEDURE DIVISION.

```
*  Setting the in parameter identifier
   MOVE 2 TO D OF UNI1.
*  Setting the in parameter value
   MOVE 100 TO PARA2 OF UNI1.

*  Setting the inout parameter identifier
   MOVE 2 TO D OF UNI3.
*  Setting the inout parameter value
   MOVE 200 TO PARA2 OF UNI3.

   CALL "ODSAMPLE-UNIONTEST-OP2" USING
         OBJ
         UNI1
         UNI2
         UNI3
         ENV
         UNI0.
```

Variable-length Parameters Handled by Client Applications

The following table shows how client application parameters are handled.

Table 7.15 Client Parameter Area (Union Type or Variable-length Data Area)

Parameter	Parameter passed to server	Parameter passed from server
in	The XX-ALLOC function or the data area allocation function is used to allocate the structure area or variable-length data area.	_
inout	Same as the *in* parameter	The area is automatically allocated by the stub.
out return	_	Same as the *inout* parameter

When an area that has been dynamically allocated by the client and stub is no longer required, it must be released using the CORBA-FREE function. The variable-length data area is also released when the CORBA-FREE function is issued.

The following is an example of client application processing.

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
  COPY CONST IN CORBA.
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
  01 UNI1.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
    02 U.
      03 PARA2 USAGE IS POINTER.
```

```
    02 FILLER REDEFINES U.
      03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
  01 UNI2 USAGE POINTER.
  01 UNI3 USAGE POINTER.
  01 UNI0 USAGE POINTER.
  01 STR-WORK PIC X(30).
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.

 LINKAGE SECTION.
  01 UNI-P.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
    02 U.
      03 PARA2 USAGE IS POINTER.
    02 FILLER REDEFINES U.
      03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
```

PROCEDURE DIVISION.

```
*  Setting the in parameter identifier
   MOVE 1 TO D OF UNI1.
*  Setting the in parameter value
   MOVE 10 TO PARA1 OF UNI1.

*  Allocating an inout parameter area
   CALL "ODSAMPLE-SAMPLEVAR-ALLOC" USING
         UNI3.
   SET ADDRESS OF UNI-P TO UNI3.
*  Setting the inout parameter identifier
   MOVE 2 TO D OF UNI1-P.
*  Setting the inout parameter value
   MOVE 12 TO LSIZE.
   MOVE "INOUT:para2" TO STR-WORK.
   MOVE FUNCTION LENG(STR-WORK) TO LSIZE
   CALL "CORBA-STRING-SET" USING
        PARA2 OF UNI-P
        LSIZE
        STR-WORK.

   CALL "ODSAMPLE-UNIONTEST-OP1" USING
       OBJ
       UNI1
       UNI2
       UNI3
       ENV
       UNI0.
*  Releasing the out parameter area
 CALL "CORBA-FREE" USING
       UNI2.
*  Releasing the inout parameter area
 CALL "CORBA-FREE" USING
       UNI3.
*  Releasing the Return value area
 CALL "CORBA-FREE" USING
       UNI0.
```

Fixed-length Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

To set a server application processing result in the union as an out or inout parameter, there is no need to allocate or release a structure location. Instead, set the identification value of the data type to be assigned, and the corresponding data type value in the identification information.

IDENTIFICATION DIVISION.

```
PROGRAM-ID.  "ODSAMPLE-UNIONTEST-OP2".
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
LINKAGE SECTION.
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
  01 UNI1.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
     02 U.
        03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
     02 FILLER REDEFINES U.
        03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
  01 UNI2.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
     02 U.
        03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
     02 FILLER REDEFINES U.
        03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
  01 UNI3.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
     02 U.
        03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
     02 FILLER REDEFINES U.
        03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
  01 ENV.
     03 MAJOR PIC 9(9) COMP-5.
        88 CORBA-NO-EXCEPTION VALUE 0.
        88 CORBA-USER-EXCEPTION VALUE 1.
        88 CORBA-SYSTEM-EXCEPTION VALUE 2.
     03 IDL-ID USAGE POINTER.
     03 MINOR PIC 9(9) COMP-5.
     03 IDL-STATUS PIC 9(9) COMP-5.
     03 PARAM USAGE POINTER.
     03 MAGIC PIC 9(9) COMP-5.
  01 UNI.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
     02 U.
        03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
     02 FILLER REDEFINES U.
       03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA2.
```

PROCEDURE DIVISION

```
USING
      OBJ
      UNI1
      UNI2
      UNI3
      ENV
      UNI.

MAIN SECTION.

*  out parameter processing
   MOVE 1 TO D OF UNI2.
   MOVE 10 TO PARA1 OF UNI2.
*  inout parameter processing
   MOVE 1 TO D OF UNI3.
   MOVE 30 TO PARA1 OF UNI3.
*  Return value processing
   MOVE 1 TO D OF UNI.
```

```
   MOVE 100 TO PARA1 OF UNI.
MAIN-END.
END PROGRAM "ODSAMPLE-UNIONTEST-OP2".
```

Variable-length Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

The following table shows how server application parameters are handled.

Table 7.16 Server Parameter Area (Union or Variable-length Data Area)

Parameter	Parameter area passed from client	Parameter area passed to client
in	The structure area or variable-length data area is automatically allocated or released by the skeleton.	_
inout	The structure area or variable-length data area is automatically allocated by the skeleton.	The variable-length data area is released once using the CORBA-FREE function and re-allocated using the data area allocation function. The area is automatically released by the skeleton.
out return	_	The structure area or variable-length data area is allocated using the XX-ALLOC function or the data area allocation function. The area is automatically released by the skeleton.

The following is an example of server application processing.

IDENTIFICATION DIVISION.

```
PROGRAM-ID.  "ODSAMPLE-UNIONTEST-OP1".
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
  01 STR-WORK PIC X(30).
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
 LINKAGE SECTION.
  01 UNI1.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
    02 U.
      03 PARA2 USAGE IS POINTER.
    02 FILLER REDEFINES U.
     03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
  01 UNI2-P.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
    02 U.
      03 PARA2 USAGE IS POINTER.
    02 FILLER REDEFINES U.
      03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
  01 UNI2 USAGE IS POINTER.
  01 UNI3-P.
    02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
    02 U.
      03 PARA2 USAGE IS POINTER.
    02 FILLER REDEFINES U.
      03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
  01 ENV.
    03 MAJOR PIC 9(9) COMP-5.
      88 CORBA-NO-EXCEPTION VALUE 0.
      88 CORBA-USER-EXCEPTION VALUE 1.
```

```
      88 CORBA-SYSTEM-EXCEPTION VALUE 2.
    03 IDL-ID USAGE POINTER.
    03 MINOR PIC 9(9) COMP-5.
    03 IDL-STATUS PIC 9(9) COMP-5.
    03 PARAM USAGE POINTER.
    03 MAGIC PIC 9(9) COMP-5.
  01 UNI-P.
   02 COPY LONG IN CORBA REPLACING CORBA-LONG BY D.
   02 U.
    03 PARA2 USAGE IS POINTER.
   02 FILLER REDEFINES U.
    03 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARA1.
  01 UNI USAGE POINTER.
```

PROCEDURE DIVISION

```
USING
      OBJ
      UNI1
      UNI2
      UNI3
      ENV
      UNI.

MAIN SECTION.

* out parameter processing
  CALL "ODSAMPLE-SAMPLEVAR-ALLOC" USING
      UNI2.
  SET ADDRESS OF UNI2-P TO UNI2.
  MOVE 2 TO D OF UNI2-P.
  MOVE 10 TO LSIZE.
  MOVE "OUT:param" TO STR-WORK.
  MOVE FUNCTION LENG(STR-WORK) TO LSIZE.
  CALL "CORBA-STRING-SET" USING
       PARA2 OF UNI2-P
       LSIZE
       STR-WORK.

*  inout return value processing
   SET ADDRESS OF UNI3-P TO UNI3.
   EVALUATE D OF UNI3-P
      WHEN 2
         INVOKE "CORBA-FREE" USING
               PARA2 OF UNI3-P
   END-EVALUATE.
   MOVE 1 TO D OF UNI3-P.
   MOVE 10 TO PARA1 OF UNI3-P.

*  Return value processing
   CALL "ODSAMPLE-SAMPLEVAR-ALLOC" USING
      UNI.
   SET ADDRESS OF UNI-P TO UNI.
   MOVE 1 TO D OF UNI-P.
   MOVE 30 TO PARA1 OF UNI-P.

MAIN-END.
END PROGRAM "ODSAMPLE-UNIONTEST-OP1".
```

7.7.7 Fixed-point

This section provides fixed-point information.

IDL Mapping

When a Fixed-point is specified in IDL, data are declared with COBOL due to digits and scale as follows.

IDL

```
// fixed <digits, scale>
typedef fixed <10,2>  fixed1;    // for digits > scale > 0
typedef fixed <10,-2> fixed2;    // for digits-18 <= scale < 0
typedef fixed <2,10>  fixed3;    // for digits < scale <= 18
typedef fixed <10,10> fixed4;    // for digits = scale
typedef fixed <10,0>  fixed5;    // for scale = 0
```

COBOL

```
01 FIXED1 PIC S9(8)V9(2)   PACKED-DECIMAL.
01 FIXED2 PIC S9(10)P(2)  PACKED-DECIMAL.
01 FIXED3 PIC SVP(8)9(2)  PACKED-DECIMAL.
01 FIXED4 PIC SV9(10)     PACKED-DECIMAL.
01 FIXED5 PIC S9(10)      PACKED-DECIMAL
```

The following IDL definition example is explained in the place at the rest.

IDL

```
module ODsample {
    interface fixedtest {
        fixed <10,3>  op1(
            in       fixed <5,10>    fixed1,
            out      fixed <10,-3>   fixed2,
            inout    fixed <10,10>   fixed3 );
    };
};
```

COBOL

IDENTIFICATION DIVISION.

```
PROGRAM-ID.  "EXEC-METHOD".
 AUTHOR.  OD/IDLCOMPILER VER.2.0.
 INSTALLATION.  IDL FILE NAME IS simple.idl.
 SECURITY.  THIS SOURCE CODE WAS GENERATED BASE ON YOUR IDL FILE.
           WHEN THIS STUB/SKELETON SOURCE CODE IS CHANGED, THE OPERATION.
           GURANTEED IS NOT DONE.
 DATE-WRITTEN.  TUE MAY  6 11:03:40 1997
* ALL RIGHTS RESERVED, COPYRIGHT (C) FUJITSU LIMITED 1998
```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.

```
     SYMBOLIC CONSTANT
     COPY SYMBOL-CONST IN CORBA.
     .
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT OBJFILE ASSIGN "obj".
```

DATA DIVISION.
FILE SECTION.

```
 FD  OBJFILE.
 01  FILE-BUF PIC X(1024).
```

WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
 01 MESS PIC X(40).
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
 01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
 01 FIXED1 PIC SVP(5)9(5) PACKED-DECIMAL.
 01 FIXED2 PIC S9(10)P(3) PACKED-DECIMAL.
 01 FIXED3 PIC SV9(10)    PACKED-DECIMAL.
 01 FIXED0 PIC S9(7)V9(3) PACKED-DECIMAL.
*
 LINKAGE SECTION.

 01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
 01 COPY BOOLEAN IN CORBA REPLACING CORBA-BOOLEAN BY RESULT.
```

PROCEDURE DIVISION

```
USING
         OBJ
         NUM
         RESULT.
*
     MOVE 0.  22 TO FIXED1.
     MOVE  0.55555   TO FIXED3.

     CALL "ODSAMPLE-FIXEDTEST-OP1" USING
           OBJ
           FIXED1
           FIXED2
           FIXED3
           ENV
           FIXED0.
     MOVE "ODSAMPLE-FIXEDTEST-OP1" TO MESS.
     PERFORM ENV-CHECK
*
     EXIT PROGRAM.
 MAIN-END.

 ENV-CHECK SECTION.
     EVALUATE TRUE
         WHEN CORBA-NO-EXCEPTION OF MAJOR OF ENV
              CONTINUE
         WHEN CORBA-USER-EXCEPTION OF MAJOR OF ENV
              DISPLAY "USER-EXCEPTION   : " MESS
              MOVE FUNCTION LENG (MESS) TO STRING-LENGTH
              CALL "CORBA-STRING-GET" USING
                                   IDL-ID OF ENV
                                   STRING-LENGTH
                                   MESS
              DISPLAY "ID : " MESS
              EXIT PROGRAM
         WHEN CORBA-SYSTEM-EXCEPTION OF MAJOR OF ENV
              DISPLAY "SYSTEM-EXCEPTION : " MESS
              MOVE FUNCTION LENG (MESS) TO STRING-LENGTH
              CALL "CORBA-STRING-GET" USING
                                   IDL-ID OF ENV
                                   STRING-LENGTH
                                   MESS
```

```
              DISPLAY "ID : " MESS
              EXIT PROGRAM
     END-EVALUATE.
 ENV-CHECK-END.
     EXIT.
*
 END PROGRAM "EXEC-METHOD".
```

Parameters Handled by Client Applications

The following is an example of client application processing.

IDENTIFICATION DIVISION.

```
PROGRAM-ID.  "CLIENT-MAIN".
 AUTHOR.  OD/IDLCOMPILER VER.2.0.
 INSTALLATION.  IDL FILE NAME IS simple.idl.
 SECURITY.  THIS SOURCE CODE WAS GENERATED BASE ON YOUR IDL FILE.
           WHEN THIS STUB/SKELETON SOURCE CODE IS CHANGED, THE OPERATION.
           GURANTEED IS NOT DONE.
 DATE-WRITTEN.  TUE MAY  6 11:03:40 1997
* ALL RIGHTS RESERVED, COPYRIGHT (C) FUJITSU LIMITED 1998
```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.

```
     ARGUMENT-NUMBER   IS ARG-C
     ARGUMENT-VALUE    IS ARG-V
     SYMBOLIC CONSTANT
     COPY SYMBOL-CONST IN CORBA.
     .
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
 01 TEMP-BUF USAGE POINTER.
 01 MESS PIC X(30).
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
 01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
 01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
 01 COPY BOA IN CORBA REPLACING CORBA-BOA BY BOA.
 01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
 01 COPY BOOLEAN IN CORBA REPLACING CORBA-BOOLEAN BY RET.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY I.
 01 BUFFER USAGE POINTER.
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY BUF-LENGTH.
 01 STR-BUF PIC X(30).
 01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING
         COSNAMING-NAMINGCONTEXT BY COS-NAMING.
 01 COPY COSNAMING-NAME IN CORBA REPLACING COSNAMING-NAME BY NAME.
 01 NAME-A USAGE POINTER.
 01 COPY COSNAMING-NAMECOMPONENT IN CORBA REPLACING
         COSNAMING-NAMECOMPONENT BY NAME-COMPONENT.
 01 NAME-COMPONENT-A USAGE POINTER.
*########## ORB SETTING PARAMETER #########
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY CURRENT-ARG-C.
 01 CURRENT-ARG-V.
    02 FILLER OCCURS 6.
       03 CURRENT-ARG-V-VALUE USAGE POINTER.
 01 APLI-NAME PIC X(8) VALUE "simple_c".
```

```
 01 TMP-STRING-BUF PIC X(20).
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY ARG-COUNT.
```

PROCEDURE DIVISION.

```
MAIN.

     DISPLAY "CLIENT START!!".
* argument set : CURRENT-ARG-V-VALUE
     ACCEPT CURRENT-ARG-C FROM ARG-C.
     COMPUTE CURRENT-ARG-C = CURRENT-ARG-C + 1.
     PERFORM VARYING ARG-COUNT FROM 1 BY 1 UNTIL ARG-COUNT > CURRENT-ARG-C
        IF ARG-COUNT = 1
           MOVE APLI-NAME TO TMP-STRING-BUF
        ELSE
           ACCEPT TMP-STRING-BUF FROM ARG-V
        END-IF
        MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH
        CALL "CORBA-STRING-SET" USING
              CURRENT-ARG-V-VALUE (ARG-COUNT)
              STRING-LENGTH
              TMP-STRING-BUF
     END-PERFORM.
     SET CURRENT-ARG-V-VALUE (ARG-COUNT) TO NULL.
* ObjectDirector Initialize
     MOVE 12 TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           FJ-OM-ORB-ID.
     CALL "CORBA-ORB-INIT" USING
           CURRENT-ARG-C
           CURRENT-ARG-V
           TEMP-BUF
           ENV
           ORB.
     CALL "CORBA-FREE" USING TEMP-BUF.
     MOVE "CORBA-ORB-INIT" TO MESS.
     PERFORM ENV-CHECK

     MOVE 15 TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           CORBA-BOA-OA-ID.
     CALL "CORBA-ORB-BOA-INIT" USING
           ORB
           CURRENT-ARG-C
           CURRENT-ARG-V
           TEMP-BUF
           ENV
           BOA.
     CALL "CORBA-FREE" USING TEMP-BUF.
     MOVE "CORBA-ORB-BOA-INIT" TO MESS.
     PERFORM ENV-CHECK.
* NamingService repository get
     MOVE FUNCTION LENG ( CORBA-ORB-OBJECTID-NAMESERVICE ) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           CORBA-ORB-OBJECTID-NAMESERVICE.
     CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
           ORB
```

```
           TEMP-BUF
           ENV
           COS-NAMING.
     CALL "CORBA-FREE" USING TEMP-BUF.
     MOVE "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" TO MESS.
     PERFORM ENV-CHECK.
* ODdemo::calculator repository get
     MOVE FUNCTION LENG (STR-BUF) TO STRING-LENGTH.
     MOVE "ODsample::fixedtest" TO STR-BUF.
     CALL "CORBA-STRING-SET" USING
           IDL-ID OF NAME-COMPONENT
           STRING-LENGTH
           STR-BUF.
     MOVE " " TO STR-BUF.
     CALL "CORBA-STRING-SET" USING
           KIND OF NAME-COMPONENT
           STRING-LENGTH
           STR-BUF.
     MOVE 1 TO SEQ-LENGTH OF NAME.
     MOVE 1 TO SEQ-MAXIMUM OF NAME.
     MOVE 1 TO NUM.
     CALL "CORBA-SEQUENCE-COSNAMING-NAMECOMPONENT-ALLOCBUF" USING
           SEQ-MAXIMUM OF NAME
           SEQ-BUFFER OF NAME.
     MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
     MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
     CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
           NAME-A
           NUM
           NAME-COMPONENT-A.
     CALL "COSNAMING-NAMINGCONTEXT-RESOLVE" USING
              COS-NAMING
              NAME
              ENV
              OBJ.
     MOVE "COSNAMING-NAMINGCONTEXT-RESOLVE" TO MESS.
     PERFORM ENV-CHECK.
* CALL fixed program
     CALL "EXEC-METHOD" USING
                 OBJ
                 NUM
                 RET.
     CALL "CORBA-OBJECT-RELEASE" USING OBJ ENV.
     CALL "CORBA-OBJECT-RELEASE" USING COS-NAMING ENV.
     MOVE "CORBA-OBJECT-RELEASE" TO MESS.
     PERFORM ENV-CHECK.
     CALL "CORBA-FREE" USING SEQ-BUFFER OF NAME.

     STOP RUN.
 MAIN-END.

 ENV-CHECK SECTION.
     EVALUATE TRUE
         WHEN CORBA-NO-EXCEPTION OF MAJOR OF ENV
              CONTINUE
         WHEN CORBA-USER-EXCEPTION OF MAJOR OF ENV
              DISPLAY "USER-EXCEPTION   : " MESS
              MOVE FUNCTION LENG (MESS) TO STRING-LENGTH
              CALL "CORBA-STRING-GET" USING
                                  IDL-ID OF ENV
                                  STRING-LENGTH
                                  MESS
              DISPLAY "ID : " MESS
```

```
             EXIT PROGRAM
         WHEN CORBA-SYSTEM-EXCEPTION OF MAJOR OF ENV
             DISPLAY "SYSTEM-EXCEPTION : " MESS
             MOVE FUNCTION LENG (MESS) TO STRING-LENGTH
             CALL "CORBA-STRING-GET" USING
                                  IDL-ID OF ENV
                                  STRING-LENGTH
                                  MESS
             DISPLAY "ID : " MESS
             EXIT PROGRAM
     END-EVALUATE.
 ENV-CHECK-END.
     EXIT.
*
 END PROGRAM "CLIENT-MAIN".
```

Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

The following is an example of server application processing.

IDENTIFICATION DIVISION.

```
PROGRAM-ID.  "ODSAMPLE-FIXEDTEST-OP1".
 AUTHOR.  OD/IDLcompiler Ver.2.0.
 INSTALLATION.  IDL FILE NAME IS simple.idl.
 SECURITY.  THIS SOURCE CODE WAS GENERATED BASE ON YOUR IDL FILE.
           WHEN THIS STUB/SKELETON SOURCE CODE IS CHANGED, THE OPERATION.
           GURANTEED IS NOT DONE.
 DATE-WRITTEN.  Tue May  6 11:03:40 1997
* ALL RIGHTS RESERVED, COPYRIGHT (C) FUJITSU LIMITED 1998
```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.

```
*
 INPUT-OUTPUT SECTION.
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.

 LINKAGE SECTION.
 01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
 01 FIXED1 PIC SVP(5)9(5) PACKED-DECIMAL.
 01 FIXED2 PIC S9(10)P(3) PACKED-DECIMAL.
 01 FIXED3 PIC SV9(10)    PACKED-DECIMAL.
 01 FIXED0 PIC S9(7)V9(3) PACKED-DECIMAL.
 01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
```

PROCEDURE DIVISION

```
USING
         OBJ
         FIXED1
         FIXED2
         FIXED3
         ENV
         FIXED0.
*
 MAIN.
*
```

```
* SET OUT PARAMETER
     MOVE   00 TO FIXED2.
*
* SET INOUT PARAMETER.
     MOVE 0.88888  TO FIXED3.
*
* SET RETURN.
     MOVE 12345.345 TO FIXED0.
*
 MAIN-END.
*
 END PROGRAM  "ODSAMPLE-FIXEDTEST-OP1".
```

7.7.8 Array

This section provides information on arrays.

IDL Mapping

When an array is specified in IDL, data is declared using an OCCURS clause in COBOL. The index of all arrays begins with 1 and continues up to <array-size>. Define an array name using the module, interface, and array names and "-V" concatenated using hyphen ("-"). Invoke the COPY statement to define the name as required.

The function for retrieving an array location is generated in the IDL compiler. The function name is composed of the module, interface, and structure names and ALLOC concatenated using hyphen ("-"). From now on, this is invoked using the XX-ALLOCBUF function.

IDL

```
module ODsample{
   interface   arraytest{
               typedef   long     fix[4][3][2];       // Array (fixed length)
               typedef   string   str[2][3][4];       // Array (variable length)
               fix       op1(in fix para1, out fix para2, inout fix para3 );
               str       op2(in str para1, out str para2, inout str para3 );
        };
};
```

COBOL

```
01 ODSAMPLE-ARRAYTEST-FIX.
  02 FILLER OCCURS 4.
    03 FILLER OCCURS 3.
      04 FILLER OCCURS 2.
        05 COPY LONG IN CORBA REPLACING CORBA-LONG BY ODSAMPLE-ARRAYTEST-FIX-V.
01 ODSAMPLE-ARRAYTEST-STR.
  02 FILLER OCCURS 2.
    03 FILLER OCCURS 3.
      04 FILLER OCCURS 4.
        05 ODSAMPLE-ARRAYTEST-STR-V USAGE IS POINTER.
```

Fixed-length Parameters Handled by Client Applications

When a client application uses in, out, and inout parameters, it is not necessary to allocate or release a location. When a client application uses a return value, an array location is allocated in the stub using the XX-ALLOC function. When the area is no longer required, the client application must release it using the *CORBA-FREE function*.

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
   COPY CONST IN CORBA.
   01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
```

```
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
  01 PARA1.
    02 FILLER OCCURS 4.
      03 FILLER OCCURS 3.
        04 FILLER OCCURS 2.
          05 COPY LONG IN CORBA
             REPLACING CORBA-LONG BY ODSAMPLE-ARRAYTEST-PARA1-V.
  01 PARA2.
    02 FILLER OCCURS 4.
      03 FILLER OCCURS 3.
        04 FILLER OCCURS 2.
          05 COPY LONG IN CORBA REPLACING CORBA-LONG BY ODSAMPLE-ARRAYTEST-PARA2-V.
  01 PARA3.
    02 FILLER OCCURS 4.
      03 FILLER OCCURS 3.
        04 FILLER OCCURS 2.
          05 COPY LONG IN CORBA
             REPLACING CORBA-LONG BY ODSAMPLE-ARRAYTEST-PARA3-V.
  01 FIX0.
    02 FILLER OCCURS 4.
      03 FILLER OCCURS 3.
        04 FILLER OCCURS 2.
          05 COPY LONG IN CORBA REPLACING CORBA-LONG BY ODSAMPLE-ARRAYTEST-FIX0-V.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY I.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY J.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY K.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
```

PROCEDURE DIVISION.

```
  PERFORM VARYING I FROM 1 BY 1 UNTIL I > 4
        PERFORM VARYING J FROM 1 BY 1 UNTIL J > 3
            PERFORM VARYING K FROM 1 BY 1 UNTIL K > 2
                MOVE I TO ODSAMPLE-ARRAYTEST-PARA1-V( I J K )
                COMPUTE NUM = I * 10
                MOVE NUM TO ODSAMPLE-ARRAYTEST-PARA3-V( I J K )
            END-PERFORM
       END-PERFORM
  END-PERFORM.

CALL "ODSAMPLE-ARRAYTEST-OP1" USING
      OBJ
      PARA1
      PARA2
      PARA3
      ENV
      FIX0.
```

Variable-length Parameters Handled by Client Applications

When a client application uses an inout parameter or an out parameter, or return value, an array location is allocated in the stub using the XX-ALLOC function. When the location is no longer required, the client application must release it using the *CORBA-FREE function*.

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
  COPY CONST IN CORBA.
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
  01 STR1.
    02 FILLER OCCURS 2.
      03 FILLER OCCURS 3.
        04 FILLER OCCURS 4.
```

```
          05 ODSAMPLE-ARRAYTEST-STR1-V USAGE IS POINTER.
   01 STR2 USAGE POINTER.
   01 STR3 USAGE POINTER.
      01 STR0 USAGE POINTER.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY I.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY J.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY K.
   01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
   01 STR1 X(20).
   01 STR2 X(20).
    01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
   LINKAGE SECTION.
   01 STR3-P.
     02 FILLER OCCURS 2.
       03 FILLER OCCURS 3.
         04 FILLER OCCURS 4.
           05 ODSAMPLE-ARRAYTEST-STR3-V USAGE IS POINTER.
```

PROCEDURE DIVISION.

```
CALL "ODSAMPLE-ARRAYTEST-STR-ALLOC" USING  STR3.
SET ADDRESS OF STR3-P TO STR3.
PERFORM VARYING I FROM 1 BY 1 UNTIL I > 2
      PERFORM VARYING J FROM 1 BY 1 UNTIL J > 3
            PERFORM VARYING K FROM 1 BY 1 UNTIL K > 4
                  MOVE "str1[I][J][K]" TO STR1-D
                  MOVE "str3[I][J][K]" TO STR2-D
                  EVALUATE I
                    WHEN 1
                      INSPECT STR1-D REPLACING ALL "I" BY "0"
                      INSPECT STR2-D REPLACING ALL "I" BY "0"
                    WHEN 2
                      INSPECT STR1-D REPLACING ALL "I" BY "1"
                      INSPECT STR2-D REPLACING ALL "I" BY "1"
                  END-EVALUATE
                  EVALUATE J
                    WHEN 1
                      INSPECT STR1-D REPLACING ALL "J" BY "0"
                      INSPECT STR2-D REPLACING ALL "J" BY "0"
                    WHEN  2
                      INSPECT STR1-D REPLACING ALL "J" BY "1"
                      INSPECT STR2-D REPLACING ALL "J" BY "1"
                    WHEN  3
                      INSPECT STR1-D REPLACING ALL "J" BY "2"
                      INSPECT STR2-D REPLACING ALL "J" BY "2"
                  END-EVALUATE
                  EVALUATE K
                    WHEN 1
                      INSPECT STR1-D REPLACING ALL "K" BY "0"
                      INSPECT STR2-D REPLACING ALL "K" BY "0"
                    WHEN 2
                      INSPECT STR1-D REPLACING ALL "K" BY "1"
                      INSPECT STR2-D REPLACING ALL "K" BY "1"
                    WHEN 3
                      INSPECT STR1-D REPLACING ALL "K" BY "2"
                      INSPECT STR2-D REPLACING ALL "K" BY "2"
                    WHEN 4
                      INSPECT STR1-D REPLACING ALL "K" BY "3"
                      INSPECT STR2-D REPLACING ALL "K" BY "3"
                  END-EVALUATE
                  MOVE FUNCTION LENG(STR1-D) TO LSIZE
                  CALL "CORBA-STRING-SET" USING
                     ODSAMPLE-ARRAYTEST-STR1-V( I J K )
```

```
                    LSIZE
                    STR1-D
                 MOVE FUNCTION LENG(STR2-D) TO LSIZE
                 CALL "CORBA-STRING-SET" USING
                    ODSAMPLE-ARRAYTEST-STR3-V( I J K )
                    LSIZE
                    STR2-D
           END-PERFORM
      END-PERFORM
END-PERFORM.
CALL "ODSAMPLE-ARRAYTEST-OP2" USING
  OBJ
  STR1
  STR2
  STR3
  ENV
  STR0.
PERFORM VARYING I FROM 1 BY 1 UNTIL I > 2
     PERFORM VARYING J FROM 1 BY 1 UNTIL J > 3
           PERFORM VARYING K FROM 1 BY 1 UNTIL K > 4
                 CALL "CORBA-FREE" USING  ODSAMPLE-ARRAYTEST-STR1-V( I J K )
           END-PERFORM
   END-PERFORM
END-PERFORM.
CALL "CORBA-FREE"  USING
  STR0.
CALL "CORBA-FREE"  USING
  STR2.
CALL "CORBA-FREE"  USING
  STR3.
```

Fixed-length Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

The following is an example of server application processing.

IDENTIFICATION DIVISION.

```
 PROGRAM-ID.  "ODSAMPLE-ARRAYTEST-OP1"
```

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
  COPY CONST IN CORBA.
LINKAGE SECTION.
   01 ENV.
     03 MAJOR PIC 9(9) COMP-5.
       88 CORBA-NO-EXCEPTION VALUE 0.
       88 CORBA-USER-EXCEPTION VALUE 1.
       88 CORBA-SYSTEM-EXCEPTION VALUE 2.
     03 IDL-ID USAGE POINTER.
     03 MINOR PIC 9(9) COMP-5.
     03 IDL-STATUS PIC 9(9) COMP-5.
     03 PARAM USAGE POINTER.
     03 MAGIC PIC 9(9) COMP-5.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
   01 PARA1.
     02 FILLER OCCURS 4.
       03 FILLER OCCURS 3.
         04 FILLER OCCURS 2.
           05 COPY LONG IN CORBA REPLACING CORBA-LONG BY ODSAMPLE-ARRAYTEST-PARA1-V.
   01 PARA2.
```

```
    02 FILLER OCCURS 4.
      03 FILLER OCCURS 3.
        04 FILLER OCCURS 2.
          05 COPY LONG IN CORBA
             REPLACING CORBA-LONG BY ODSAMPLE-ARRAYTEST-PARA2-V.
   01 PARA3.
    02 FILLER OCCURS 4.
      03 FILLER OCCURS 3.
        04 FILLER OCCURS 2.
           05 COPY LONG IN CORBA
              REPLACING CORBA-LONG BY ODSAMPLE-ARRAYTEST-PARA3-V.
   01 FIX.
    02 FILLER OCCURS 4.
      03 FILLER OCCURS 3.
        04 FILLER OCCURS 2.
          05 COPY LONG IN CORBA REPLACING CORBA-LONG BY ODSAMPLE-ARRAYTEST-FIX-V.
          05 ODSAMPLE-ARRAYTEST-FIX-V USAGE IS POINTER.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY I.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY J.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY K.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
```

PROCEDURE DIVISION

```
USING OBJ PARA1 PARA2 PARA3 ENV FIX.

*  out parameter processing
  PERFORM VARYING I FROM 1 BY 1 UNTIL I > 4
        PERFORM VARYING J FROM 1 BY 1 UNTIL J > 3
              PERFORM VARYING K FROM 1 BY 1 UNTIL K > 2
                    COMPUTE NUM = I + J + K
                    MOVE NUM TO ODSAMPLE-ARRAYTEST-PARA2-V( I J K )
              END-PERFORM
       END-PERFORM
  END-PERFORM.   *  inout parameter processing
  PERFORM VARYING I FROM 1 BY 1 UNTIL I > 4
        PERFORM VARYING J FROM 1 BY 1 UNTIL J > 3
              PERFORM VARYING K FROM 1 BY 1 UNTIL K > 2
                    COMPUTE NUM = (I + J + K) * 10
                    MOVE NUM TO ODSAMPLE-ARRAYTEST-PARA3-V( I J K )
              END-PERFORM
       END-PERFORM
  END-PERFORM.
*  Return value processing
  PERFORM VARYING I FROM 1 BY 1 UNTIL I > 4
        PERFORM VARYING J FROM 1 BY 1 UNTIL J > 3
              PERFORM VARYING K FROM 1 BY 1 UNTIL K > 2
                    COMPUTE NUM = I * J * K
                    MOVE NUM TO ODSAMPLE-ARRAYTEST-FIX-V( I J K )
              END-PERFORM
       END-PERFORM
  END-PERFORM.
```

Variable-length Parameters Handled by Server Applications

This is not valid for Linux (64 bit).

When a server application handles in and inout parameters of an array (variable length), no special location allocation, or release function is needed. When a server application uses an out parameter or return value, the XX-ALLOC function to allocate a location must be used. Then, assign data to that location. For an inout parameter, first use the CORBA-FREE function to release the variable-length data area in the structure sent from the client.

IDENTIFICATION DIVISION.

```
PROGRAM-ID "ODSAMPLE-ARRAYTEST-OP2"
```

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
 COPY CONST IN CORBA.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY I.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY J.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY K.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
  01 STR-OUT PIC X(20).
  01 STR-IO PIC X(20).
  01 STR-RET PIC X(20).
 LINKAGE SECTION.
  01 ENV.
    03 MAJOR PIC 9(9) COMP-5.
      88 CORBA-NO-EXCEPTION VALUE 0.
      88 CORBA-USER-EXCEPTION VALUE 1.
      88 CORBA-SYSTEM-EXCEPTION VALUE 2.
    03 IDL-ID USAGE POINTER.
    03 MINOR PIC 9(9) COMP-5.
    03 IDL-STATUS PIC 9(9) COMP-5.
    03 PARAM USAGE POINTER.
    03 MAGIC PIC 9(9) COMP-5.
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
   01 PARAM1.
    02 FILLER OCCURS 2.
      03 FILLER OCCURS 3.
        04 FILLER OCCURS 4.
          05 ODSAMPLE-ARRAYTEST-PARAM1-V USAGE IS POINTER.
  01 PARAM2-P.
    02 FILLER OCCURS 2.
      03 FILLER OCCURS 3.
        04 FILLER OCCURS 4.
          05 ODSAMPLE-ARRAYTEST-PARAM2-V USAGE IS POINTER.
  01 PARAM2 USAGE POINTER.
  01 PARAM3-P.
    02 FILLER OCCURS 2.
      03 FILLER OCCURS 3.
        04 FILLER OCCURS 4.
          05 ODSAMPLE-ARRAYTEST-PARAM3-V USAGE IS POINTER.
  01 PARAM3 USAGE POINTER.
  01 PARAM0-P.
    02 FILLER OCCURS 2.
      03 FILLER OCCURS 3.
        04 FILLER OCCURS 4.
          05 ODSAMPLE-ARRAYTEST-PARAM0-V USAGE IS POINTER.
  01 PARAM0 USAGE POINTER.
```

PROCEDURE DIVISION

```
USING OBJ PARAM1 PARAM2 PARAM3 ENV PARAM0.
* out parameter processing
CALL "ODSAMPLE-ARRAYTEST-STR-ALLOC" USING
     PARAM2.

SET ADDRESS  OF PARAM2-P TO PARAM2.
PERFORM VARYING I FROM 1 BY 1 UNTIL I > 2
        PERFORM VARYING J FROM 1 BY 1 UNTIL J > 3
              PERFORM VARYING K FROM 1 BY 1 UNTIL K > 4
                   MOVE "str2[I][J][K]" TO STR-OUT
                   EVALUATE I
```

```
                  WHEN 1
                    INSPECT STR-OUT REPLACING ALL "I" BY "0"
                  WHEN 2
                    INSPECT STR-OUT REPLACING ALL "I" BY "1"
                END-EVALUATE
                EVALUATE J
                  WHEN 1
                    INSPECT STR-OUT REPLACING ALL "J" BY "0"
                  WHEN  2
                    INSPECT STR-OUT REPLACING ALL "J" BY "1"
                  WHEN  3
                    INSPECT STR-OUT REPLACING ALL "J" BY "2"
                END-EVALUATE
                EVALUATE K
                  WHEN 1
                    INSPECT STR-OUT REPLACING ALL "K" BY "0"
                  WHEN 2
                    INSPECT STR-OUT REPLACING ALL "K" BY "1"
                  WHEN 3
                    INSPECT STR-OUT REPLACING ALL "K" BY "2"
                  WHEN 4
                    INSPECT STR-OUT REPLACING ALL "K" BY "3"
                END-EVALUATE
              MOVE FUNCTION LENG(STR-OUT) TO LSIZE

              CALL "CORBA-STRING-SET" USING
                    ODSAMPLE-ARRAYTEST-PARAM2-V( I J K )
                    LSIZE
                    STR-OUT
            END-PERFORM
       END-PERFORM
END-PERFORM.
*  inout parameter processing
SET ADDRESS  OF PARAM3-P TO PARAM3.
PERFORM VARYING I FROM 1 BY 1 UNTIL I > 2
        PERFORM VARYING J FROM 1 BY 1 UNTIL J > 3
              PERFORM VARYING K FROM 1 BY 1 UNTIL K > 4
                  CALL "CORBA-FREE" USING ODSAMPLE-ARRAYTEST-PARAM3-V( I J K )
                  MOVE "str3[I][J][K]" TO STR-IO
                  EVALUATE I
                    WHEN 1
                      INSPECT STR-IO REPLACING ALL "I" BY "0"
                    WHEN 2
                      INSPECT STR-IO REPLACING ALL "I" BY "1"
                  END-EVALUATE
                  EVALUATE J
                    WHEN 1
                      INSPECT STR-IO REPLACING ALL "J" BY "0"
                    WHEN  2
                      INSPECT STR-IO REPLACING ALL "J" BY "1"
                    WHEN  3
                      INSPECT STR-IO REPLACING ALL "J" BY "2"
                  END-EVALUATE
                  EVALUATE K
                    WHEN 1
                      INSPECT STR-IO REPLACING ALL "K" BY "0"
                    WHEN 2
                      INSPECT STR-IO REPLACING ALL "K" BY "1"
                    WHEN 3
                      INSPECT STR-IO REPLACING ALL "K" BY "2"
                    WHEN 4
                      INSPECT STR-IO REPLACING ALL "K" BY "3"
                  END-EVALUATE
```

```
               MOVE FUNCTION LENG(STR-IO) TO LSIZE
               CALL "CORBA-STRING-SET" USING
                     ODSAMPLE-ARRAYTEST-PARAM3-V( I J K )
                     LSIZE
                     STR-IO
             END-PERFORM
       END-PERFORM
END-PERFORM.

*  Return value processing
CALL "ODSAMPLE-ARRAYTEST-STR-ALLOC" USING
     PARAM0.
SET ADDRESS  OF PARAM0-P TO PARAM0.
PERFORM VARYING I FROM 1 BY 1 UNTIL I > 2
        PERFORM VARYING J FROM 1 BY 1 UNTIL J > 3
              PERFORM VARYING K FROM 1 BY 1 UNTIL K > 4
                   MOVE "str0[I][J][K]" TO STR-RET
                   EVALUATE I
                     WHEN 1
                       INSPECT STR-RET REPLACING ALL "I" BY "0"
                     WHEN 2
                       INSPECT STR-RET REPLACING ALL "I" BY "1"
                   END-EVALUATE
                   EVALUATE J
                     WHEN 1
                       INSPECT STR-RET REPLACING ALL "J" BY "0"
                     WHEN  2
                       INSPECT STR-RET REPLACING ALL "J" BY "1"
                     WHEN  3
                       INSPECT STR-RET REPLACING ALL "J" BY "2"
                   END-EVALUATE
                   EVALUATE K
                     WHEN 1
                       INSPECT STR-RET REPLACING ALL "K" BY "0"
                     WHEN 2
                       INSPECT STR-RET REPLACING ALL "K" BY "1"
                     WHEN 3
                       INSPECT STR-RET REPLACING ALL "K" BY "2"
                     WHEN 4
                       INSPECT STR-RET REPLACING ALL "K" BY "3"
                   END-EVALUATE
                 MOVE FUNCTION LENG(STR-RET) TO LSIZE
                 CALL "CORBA-STRING-SET" USING
                       ODSAMPLE-ARRAYTEST-PARAM0-V( I J K )
                       LSIZE
                       STR-RET
                 END-PERFORM
             END-PERFORM
        END-PERFORM.
    EXIT PROGRAM.
END PROGRAM "ODSAMPLE-ARRAYTEST-OP2".
```

7.7.9 Mapping Attribute Declaration (attribute)

This section provides information on attribute declarations.

IDL Mapping

When attribute is specified in IDL, set object data can be assigned and retrieved using the SET and GET functions. Declaring attribute generates the related SET and GET functions in the IDL compiler. The SET or GET function name is formed using the module and interface names concatenated using hyphen "-". SET or GET is then added with two hyphens "--" and a variable to assign attribute results.

IDL

```
module ODsample{
   interface    attrtest{
                attribute            long     para1;
                attribute            string   para2;
                readonly attribute   long     para3;
        };
};
```

COBOL

IDENTIFICATION DIVISION.

```
 PROGRAM-ID.  " ODSAMPLE-ATTRTEST--SET-PARA1".
```

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
*  Object reference
  01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
*  Setting value
01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM
*  Exception information
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
```

Parameters Handled by Client Applications

When a string location allocated using the GET function is no longer required the client application must release it using the *CORBA-FREE function*.

IDENTIFICATION DIVISION.

```
 PROGRAM-ID.  " ODSAMPLE-ATTRTEST".
```

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
  01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY RET-NUM.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY LSIZE.
  01 STR1 USAGE POINTER.
  01 STR-WORK PIC X(30).
  01 RET-STR1 USAGE POINTER.
  01 ERR-MSG PIC X(30).
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
```

PROCEDURE DIVISION.

```
MAIN.

  MOVE  2 TO NUM.
CALL "ODSAMPLE-ATTRTEST-SET-PARA1" USING
      OBJ
        NUM
        ENV.
  CALL "ODSAMPLE-ATTRTEST-GET-PARA1" USING
        OBJ
        ENV
```

```
       RET-NUM.
  DISPLAY "ODsample_attrtest_get_para1 returns " RET-NUM.
  MOVE "test" TO STR-WORK.
  MOVE 5 TO LSIZE.
  CALL "CORBA-STRING-SET" USING
       STR1
       LSIZE
       STR-WORK.
  CALL "ODSAMPLE-ATTRTEST-SET-PARA2" USING
      OBJ
       STR1
       ENV.
CALL "ODSAMPLE-ATTRTEST-GET-PARA2" USING
      OBJ
       ENV
       RET-STR1.

  MOVE "ODsample_attrtest__get_para2" TO ERR-MSG.
PERFORM ENV-CHECK.

  DISPLAY "ODsample_attrtest_get_para2 returns "RET-STR1.

CALL "CORBA-FREE" USING
      RET-STR1.
CALL "ODSAMPLE-ATTRTEST-GET-PARA3" USING
      OBJ
       ENV
       RET-NUM.

DISPLAY "ODsample_attrtest_get_para3 returns " RET-NUM.
MAIN-END.

   ENV-CHECK SECTION.
    EVALUATE TRUE
        WHEN CORBA-NO-EXCEPTION OF MAJOR OF ENV
             CONTINUE
        WHEN CORBA-USER-EXCEPTION OF MAJOR OF ENV
DISPLAY "USER-EXCEPTION   : " MESS
MOVE FUNCTION LENG (MESS) TO STRING-LENGTH
CALL "CORBA-STRING-GET" USING
      IDL-ID OF ENV
      STRING-LENGTH
      MESS.
DISPLAY "ID : " MESS
EXIT PROGRAM
WHEN CORBA-SYSTEM-EXCEPTION OF MAJOR OF ENV
     DISPLAY "SYSTEM-EXCEPTION : " MESS
     MOVE FUNCTION LENG (MESS) TO STRING-LENGTH
     INVOKE "CORBA-STRING-GET" USING
           IDL-ID OF ENV
           STRING-LENGTH
           MESS
     DISPLAY "ID : " MESS
     EXIT PROGRAM
END-EVALUATE.
ENV-CHECK-END.
EXIT.
```

Parameters Used by Server Application Processing

This is not valid for Linux (64 bit).

When a string area allocated using the GET function is no longer required the client application must release it using the CORBA-FREE function.

When a server application uses attribute, you do not need any special location allocation or release function for primitive type (such as long) in parameters of the SET and GET functions.

Allocate a location for out parameters of other types (such as string) of the SET and GET functions.

Release and reallocate the location for inout parameters of other types (such as string) of the SET and GET functions.

The allocated locations are released in the skeleton. To send data between functions, invoke a GLOBAL clause to define variables. In the following example, GNUM and GSTR are used as variables.

IDENTIFICATION DIVISION.

```
 PROGRAM-ID.  "ODSAMPLE-ATTRTEST-GET-PARA1".
```

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.

 LINKAGE SECTION.
  01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
01 ENV.
     03 MAJOR PIC 9(9) COMP-5.
        88 CORBA-NO-EXCEPTION VALUE 0.
        88 CORBA-USER-EXCEPTION VALUE 1.
        88 CORBA-SYSTEM-EXCEPTION VALUE 2.
     03 IDL-ID USAGE POINTER.
     03 MINOR PIC 9(9) COMP-5.
     03 IDL-STATUS PIC 9(9) COMP-5.
     03 PARAM USAGE POINTER.
     03 MAGIC PIC 9(9) COMP-5.
01 COPY LONG IN CORBA REPLACING CORBA-LONG BY RET-NUM.
```

PROCEDURE DIVISION

```
USING
      OBJ
      ENV.

 MAIN.
MOVE  GNUM TO RET-NUM.

 MAIN-END.
END PROGRAM "ODSAMPLE-ATTRTEST-GET-PARA1".
```

IDENTIFICATION DIVISION

```
 PROGRAM-ID.  "ODSAMPLE-ATTRTEST-SET-PARA1".
```

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.

 LINKAGE SECTION.
  01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.
  01 ENV.
     03 MAJOR PIC 9(9) COMP-5.
        88 CORBA-NO-EXCEPTION VALUE 0.
        88 CORBA-USER-EXCEPTION VALUE 1.
```

```
        88 CORBA-SYSTEM-EXCEPTION VALUE 2.
     03 IDL-ID USAGE POINTER.
     03 MINOR PIC 9(9) COMP-5.
     03 IDL-STATUS PIC 9(9) COMP-5.
     03 PARAM USAGE POINTER.
     03 MAGIC PIC 9(9) COMP-5.
```

PROCEDURE DIVISION

```
USING
      OBJ
      NUM
     ENV.

 MAIN.

MOVE  NUM TO GNUM.

 MAIN-END.
END PROGRAM "ODSAMPLE-ATTRTEST-SET-PARA1".
```

IDENTIFICATION DIVISION.

```
 PROGRAM-ID.  "ODSAMPLE-ATTRTEST-GET-PARA2".
```

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
01 WSTR USAGE IS POINTER.
 LINKAGE SECTION.
  01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
  01 ENV.
     03 MAJOR PIC 9(9) COMP-5.
        88 CORBA-NO-EXCEPTION VALUE 0.
        88 CORBA-USER-EXCEPTION VALUE 1.
        88 CORBA-SYSTEM-EXCEPTION VALUE 2.
     03 IDL-ID USAGE POINTER.
     03 MINOR PIC 9(9) COMP-5.
     03 IDL-STATUS PIC 9(9) COMP-5.
     03 PARAM USAGE POINTER.
     03 MAGIC PIC 9(9) COMP-5.
01 STR1 USAGE IS POINTER.
```

PROCEDURE DIVISION

```
USING
      OBJ
      ENV
      STR1.
 MAIN.

     MOVE GSTR TO STR1.

 MAIN-END.
 END PROGRAM "ODSAMPLE-ATTRTEST-GET-PARA2".
```

IDENTIFICATION DIVISION.

```
 PROGRAM-ID.  "ODSAMPLE-ATTRTEST-SET-PARA2".
```

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
 LINKAGE SECTION.
  01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
  01 STR1 USAGE IS POINTER.
  01 ENV.
     03 MAJOR PIC 9(9) COMP-5.
        88 CORBA-NO-EXCEPTION VALUE 0.
        88 CORBA-USER-EXCEPTION VALUE 1.
        88 CORBA-SYSTEM-EXCEPTION VALUE 2.
     03 IDL-ID USAGE POINTER.
     03 MINOR PIC 9(9) COMP-5.
     03 IDL-STATUS PIC 9(9) COMP-5.
     03 PARAM USAGE POINTER.
     03 MAGIC PIC 9(9) COMP-5.
```

PROCEDURE DIVISION

```
USING
      OBJ
      STR1
      ENV.

MOVE STR1 TO GSTR

MAIN-END.
END PROGRAM "ODSAMPLE-ATTRTEST-SET-PARA2".
```

IDENTIFICATION DIVISION.

```
PROGRAM-ID.  " ODSAMPLE-ATTRTEST-GET-PARA3".
```

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
LINKAGE SECTION.
  01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
  01 ENV.
     03 MAJOR PIC 9(9) COMP-5.
        88 CORBA-NO-EXCEPTION VALUE 0.
        88 CORBA-USER-EXCEPTION VALUE 1.
        88 CORBA-SYSTEM-EXCEPTION VALUE 2.
     03 IDL-ID USAGE POINTER.
     03 MINOR PIC 9(9) COMP-5.
     03 IDL-STATUS PIC 9(9) COMP-5.
     03 PARAM USAGE POINTER.
     03 MAGIC PIC 9(9) COMP-5.
01 COPY LONG IN CORBA REPLACING CORBA-LONG BY RET-NUM.
```

PROCEDURE DIVISION

```
USING
      OBJ
      ENV
      RET-NUM.

MAIN.

MOVE GNUM TO RET-NUM.
```

```
MAIN-END.
END PROGRAM "ODSAMPLE-ATTRTEST-GET-PARA3".
```

7.7.10 Sending Parameters to the Server Application

The following table shows the data types that may be used to send parameters.

Table 7.17 COBOL Data Types that may be used to Send Parameters

CORBA data type	In	Out	Inout	Return
long	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
short	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
unsigned long	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
unsigned short	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
long long	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
float	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
double	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
Boolean	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
char	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
wchar	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
octet	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
enum	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
string, bound	TEXT	TEXT	TEXT	TEXT
string, unbound	STRING	STRING	STRING	STRING
any	BY REFERENCE	POINTER	POINTER	POINTER
sequence	BY REFERENCE	POINTER	POINTER	POINTER
struct, fixed	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
struct, variable	BY REFERENCE	POINTER	POINTER	POINTER
union, fixed	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
union, variable	BY REFERENCE	POINTER	POINTER	POINTER
array, fixed	BY REFERENCE	BY REFERENCE	BY REFERENCE	BY REFERENCE
array, variable	BY REFERENCE	POINTER	POINTER	POINTER
object	BY REFERENCE	POINTER	POINTER	POINTER
TypeCode	BY REFERENCE	POINTER	POINTER	POINTER

Notes

- POINTER indicates that POINTER is sent using BY REFERENCE.
- TEXT is a COBOL fixed-length text excluding NULL characters.
- STRING is a variable-length character-string pointer that ends with NULL.

Assigning NULL Pointer

Windows32/64 Solaris32/64 Linux32

The NULL pointer cannot be assigned to out, inout and return parameters of server applications, or to in and inout parameters of client applications, for string, sequence, struct, union, and array types.

Linux64

The NULL pointer cannot be assigned to in and inout parameters of client applications, for string, sequence, struct, union, and array types.

Using Each Data Type

For a fixed-length sequence, if a value exceeding maximum is assigned in length, a system exception is posted. A fixed-length sequence variable is mapped as follows:

IDL

```
module mod {
        typedef sequence<long,10> seq_fix;
    };
```

COBOL

DATA DIVISION.
WORKING-STORAGE SECTION.

```
     01 MOD-SEQ-FIX.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
     02 SEQ-BUFFER USAGE IS POINTER.
     02 FILLER OCCURS 10.
     03 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-VALUE.
```

- A variable-length sequence variable is mapped as shown below. To access, invoke the CORBA-SEQUENCE-ELEMENT-SET and CORBA-SEQUENCE-ELEMENT-GET functions.

IDL

```
module mod {
     typedef sequence<long> seq_val;
   };
```

COBOL

DATA DIVISION.
WORKING-STORAGE SECTION.

```
     01 MOD-SEQ-VAL.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-MAXIMUM.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY SEQ-LENGTH.
     02 SEQ-BUFFER USAGE IS POINTER.
```

- For array, each array element is mapped as follows:

IDL

```
module mod {
     typedef long S_array[2][3][4];
  };
```

COBOL

DATA DIVISION.
WORKING-STORAGE SECTION.

```
     01 MOD-S-ARRAY.
     02 FILLER OCCURS 2.
     03 FILLER OCCURS 3.
     04 FILLER OCCURS 4.
     05 COPY LONG IN CORBA REPLACING CORBA-LONG BY MOD-S-ARRAY-V.
```

- For array, exception processing is mapped as follows:

IDL

```
module mod {
     exception foo {
          long ex_val;
     };
  };
```

COBOL

```
DATA DIVISION.
WORKING-STORAGE SECTION.
     01 EX-MOD-FOO PIC X(15) VALUE "IDL:mod/foo:1.0".
     01 MOD-FOO.
     02 COPY LONG IN CORBA REPLACING CORBA-LONG BY EX-VAL.
```

Consistency with COBOL

- *Float* is described using the following syntax:

```
[USAGE IS] [COMP-1]  :  4 bytes
```

- *Double* is described using the following syntax:

```
[USAGE IS] [COMP-2]  :  8 bytes
```

- *Const* is described using the following syntax.
  OMG specification

```
    >> CONSTANT constant-name IS literal
```

  COBOL specification:

```
    SYMBOLIC CONSTANT
    {symbolic-constant - 1 IS-literal - 1}
```

- Typedef is described using the following syntax (primitive-type only).
  For the CORBA library, the following definition is supported (for long):

```
        long-type usage (local long type)/
```

- The user uses *typedef* in the COPY clause:

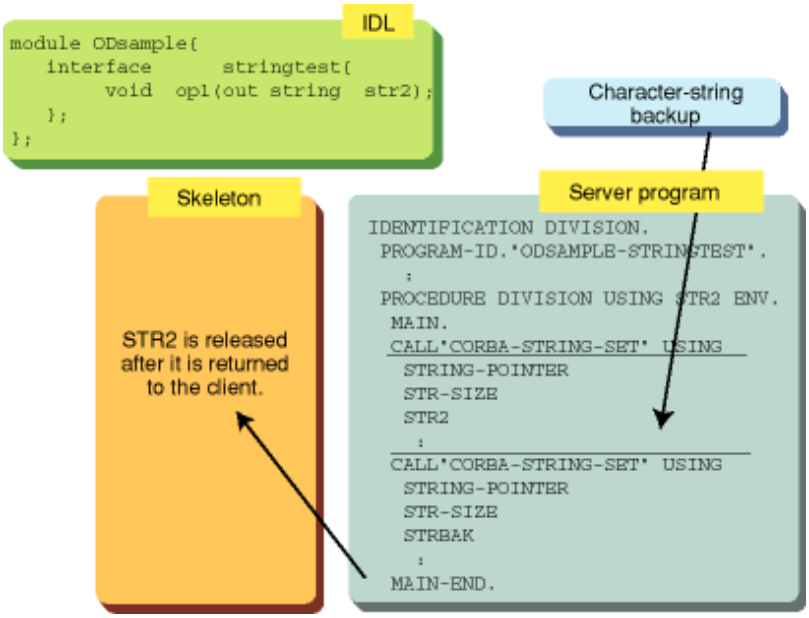
```
WORKING STORAGE SECTION.
           :
    01 COPY LONG IN CORBA
           REPLACING long-type WITH ws-long-1
    01 COPY LONG IN CORBA
           REPLACING long-type WITH ws-long-2
```

- *Typedef* of structure is not supported by Fujitsu COBOL and therefore cannot be used.

7.8 Any Type and Sequence Type Release Flags

Variable-length data is automatically released in the skeleton after it is returned to the client. Data that is not to be released must be copied to another location. The following figure shows an example using string data.

Figure 7.5 Releasing a String-type Location



For complicated types such as any and sequence, copy processing is time consuming. To help to resolve this, set the release flag to specify whether the location is to be released or not. Refer to the following table for details.

Table 7.18 Any Type and Sequence Type Release Flags

Variable Type	Setting Method
any	CORBA-ANY-SET-RELEASE is used. The default is CORBA-FALSE-VALUE. (Example 1)
sequence	CORBA-SEQUENCE-SET-RELEASE is used. The default is CORBA-FALSE-VALUE. (Example 2)

- Example 1

```
    INVOKE "CORBA-ANY-ALLOC" USING
          ANY-VAL.
    INVOKE "CORBA-ANY-SET-RELEASE" USING
          ANY-POINTER
          REL-FLAG.
```

- Example 2

(struct sequence<long> data)

```
    INVOKE "CORBA-DATA-ALLOC" USING
          SEQ-POINTER.
    INVOKE "CORBA-SEQUENCE-SET-RELEASE" USING
          SEQ-POINTER
          REL-FLAG.
```

Note

CORBA-TRUE-VALUE: Released

CORBA-FALSE-VALUE: Not released

7.9 Notes on the Use of COBOL Mapping

This section contains notes on the use of COBOL mapping.

Message

The COBOL runtime library may display a message during COBOL application execution. If a message whose message ID begins with JMP is output, refer to the COBOL User's Manual (Runtime Messages).

Variables and Reserved Words (Handling Function and Variable Names)

Note the following points on variables and reserved words:

- Specify a maximum of 60 characters for the function name. If the name exceeds 60 characters, a compile error will occur in the COBOL compiler. When defining the IDL file, the length of the module, interface, and operation names combined must not exceed 55 characters. It they do, it may cause the function name for each file generated to exceed 60 characters.
- If the truncated form of a function or variable name containing more than 30 characters already exists, replace the 27th and subsequent characters of the name with numbers.
- Underscore ("_") cannot be used in COBOL. Replace them with hyphens ("-"). However, underscoring can be used in operation and interface names.
- If a variable or class name from the IDL file is converted to the same name due to their relationship, the variable type is not defined. Therefore, do not use such variable type names under the following conditions:

```
typedef long foo_bar;
interface foo { typedef short bar; };
```

Using Reserved Words

Note the following points on the use of reserved words:

- When a COBOL reserved word character string is written in the IDL file, "IDL-" is prefixed to supported reserved word strings. For details about reserved words, refer to the COBOL manual. For details on supported reserved words, refer to "List of Supported Reserved Words".
- The const declaration becomes "SYMBOLIC CONSTANT" in COBOL. In the OMG specification, a const declaration is converted to the ">>CONSTANT" phrase. This is not supported in COBOL.

Using NVList

Note the following points on using NVList:

- In COBOL, NVList is a pointer to indicate the storage address of a NamedValue structure (library, NAMEDVALUE.cbl) array.
- Use CORBA-NVLIST-ADD-ITEM to set parameters. Note the following while setting these parameters:

LIST:	Initialization is required (use CORBA-ORB-CREATE-LIST).
ITEM-NAME:	Specify a pointer to a value assigned by CORBA-STRING-SET. When it is omitted, specify NULL.
ITEM-TYPE:	Specify a typecode created by CORBA-TYPECODE-FROM-CGEN-TC. This cannot be omitted (refer to Note below).
VALUE:	Assigns an address to a parameter value. Caution: For STRING and OBJECT data handled using a pointer, retrieve and assign an address with FUNCTION ADDR). To assign CORBA-ARG-OUT and such types to ITEM-FLAG, set NULL (needed).
VALUE-LEN:	Specify the length of the VALUE location. To omit, specify 0.
ITEM-FLAGS:	Use library, FLAGS.cbl. This cannot be omitted.

Note

The parameters that can be specified in CORBA-ORB-TYPECODE-FROM-CGEN-TC are:

- primitive type TC-XXX, written in library CONST.cbl
- entities with prefix TC in the library
- IDL file name_h.cbl generated by the IDL compiler using TC-XXX-IMPL-SEQ.

- Releasing area

  To release NVList itself, invoke CORBA-NVLIST-FREE. To release values set in NVList, invoke CORBA-NVLIST-FREE-MEMBER.

Setting Strings

To include a character string as a parameter in a function, the character string must be set in the CORBA-STRING-SET function. The following is an example.

```
*Get string length
 MOVE FUNCTION LENG (CORBA-ORB-OBJECTID-IMPLREP) TO STRING-LENGTH.
*Store string into temporary TEMP-BUF
 CALL "CORBA-STRING-SET" USING
       TEMP-BUF
       STRING-LENGTH
       CORBA-ORB-OBJECTID-IMPLREP.
*Call function
 CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
       ORB
       TEMP-BUF
       ENV
       IMPL-REP.
```

In all platforms excluding Linux 64, when string<n> (fixed-length character string) is defined in the IDL file, declaration of PIC X(n) is required in the LINKAGE SECTION of the server.

Use the MOVE statement to assign in, out, and input that are declared in PIC X(m). However, the character string length that can be handled by the mapped applications other than COBOL contains "\0."

When a Japanese character string is handled, the string cannot be directly assigned from Japanese item N (or NC) of COBOL to character item X. Use the character string after setting null characters in the CORBA-STRING-SET function or CORBA-WSTRING-SET function. Use the CORBA-STRING-GET function or CORBA-WSTRING-GET function to fetch a character string.

Notes on Building and Running Windows(R) Applications

Windows32/64

- The server application must be created using a dynamic linking structure (i.e., one which loads all called functions when the main program is loaded).

  Do not create it using a dynamic program structure (i.e., one in which the load is performed when the functions are called).

  For this reason, do not specify the DLOAD option as a translation option when the application is translated.

- Do not use DISPLAY statements in the method part of a server application.
- When a pointer is passed to an object while each interface is used with the server application, a user exception may be returned (exception code: "BAD PARAM"). Refer to exception information when including ENVIRONMENT in the parameter.
- When a server application is registered in the implementation repository as shared, unshared, or server-per-method type, use the following settings. Do not use the setting item env for registration in the implementation repository.
  - Add the following setting to the system environment variable.

    @EnvSetWindow=UNUSE

  - Add the path of the DLL file that is the method implementation section of an application to the system environment variable PATH.

- In a client application, the BOA and ServerRequest interfaces cannot be used.
- Do not create client and server applications in the same folder. If you do so, object files created for the client application and for the server application may be given the same names. This may result in abnormal application operation.
- The names of the dynamic libraries used in the COBOL server application must be together with the server application interface declaration, as displayed in the examples below:
  - Module name: oddemo
  - Interface name: calculator
  - Dynamic library name: ODDEMO-CALCULATOR.DLL

Windows64

- UNION type mapping has changed from the 32-bit version. Re-execute the IDL compiler and create the source files again.

Notes on Building and Running Solaris/Linux Applications

Solaris32/64 Linux32

- When operating the applications in the Shift JIS environment, set CBR_CODE_CHECK=no.

Solaris32/64 Linux32/64

- BOA and ServerRequest interfaces cannot be used in client applications.
- The server application must be created using a dynamic linking structure (i.e., one which loads all called functions when the main program is loaded).

  Do not create it using a dynamic program structure (i.e., one in which the load is performed when the functions are called).

  For this reason, do not specify the DLOAD option as a translation option when the application is translated.
- The names of the libraries used in the COBOL server application must be together with the server application interface declaration, as displayed in the examples below:
  - Module name: oddemo
  - Interface name: calculator
  - Dynamic library name: libODDEMO-CALCULATOR.so

Building Windows Applications

Windows32/64

This section describes procedures to build COBOL applications in Windows(R). For details about the translation and linkage methods, refer to the NetCOBOL manual.

Note

Examples used in this section assume that the CORBA Service installation directory is C:\Interstage\ODWIN.

Compiling and Linking a COBOL Client Application

Use the following compilation and linkage procedure:

(1) Perform IDL Compilation

Use the following command:

```
IDLc -cobol 'IDL file name'
```

(2) Register Stub and cdr Source

Register the generated stub and *cdr* source to COBOL source file, and compile with main programs. As the compile option, specify a library under the following directory:

Windows32

```
CORBA=C:\Interstage\ODWIN\include\COBOL
```

Windows64

```
set COB_CORBA=C:\Interstage\ODWIN\include\COBOL
COBOL.EXE  -NM "IDL file name_STUB.cbl"
COBOL.EXE  -NM "IDL file name_CDR.cbl"
COBOL.EXE  -M "XXX_C.cbl"
```

(3) Specify Process or Thread Mode Files

Windows32

Specify the following files in the library to be linked.

For process mode:

```
C:\Interstage\ODWIN\LIB\ODCOBCBL.LIB
```

For thread mode:

```
C:\Interstage\ODWIN\LIB\ODCOBCBLMT.LIB
```

For process mode/thread mode (UNICODE/UTF8 module):

```
C:\Interstage\ODWIN\LIB\ODCOBCBLUC.LIB
```

Windows64

Combine the translated objects and create the client application. Specify the following files in [Link Options].

For common:

```
F4AGCIMP.LIB
libcmt.lib
kernel32.lib
```

For process mode:

```
C:\Interstage\ODWIN\LIB\x64\ODCOBCBL.LIB
```

For thread mode:

```
C:\Interstage\ODWIN\LIB\x64\ODCOBCBLMT.LIB
```

For process mode/thread mode (UNICODE/UTF8 module):

```
C:\Interstage\ODWIN\LIB\x64\ODCOBCBLUC.LIB
```

Target files

```
'IDL file name'_STUB.CBL
'IDL file name'_CDR.CBL
```

Compile Options

For common:

- Specify the NAME option.

For thread mode:

- Specify the THREAD(MULTI) option.

For process mode/thread mode (UNICODE/UTF8 module):

- Specify the RCS option (Use UCS2-Unicode (UCS2)).
- In thread mode, specify the THREAD(MULTI) option.

For details about the compile options, refer to the NetCOBOL manual.

Library

Set the following environment variable.

Windows32

```
CORBA=C:\Interstage\ODWIN\INCLUDE\COBOL
```

Windows64

```
COB_CORBA=C:\Interstage\ODWIN\INCLUDE\COBOL
```

Files Generated

Target objects defined in the stub and CDR source.

Compiling and Linking a COBOL Server Application

The compilation and linkage procedures and file option that can be used for developing server applications are as follows:

(1) Perform IDL Compilation

Use the following command:

```
IDLc -cobol 'IDL file name'
```

(2) Compile Skeleton and cdr source

Compile the generated skeleton and cdr source, and create object files (*.obj). Invoke P-STAFF, select Tool->Compile->Compile Options and specify the file option.

(3) Compile Method Part of Server Application

Perform the same procedure to compile the method part (server processing) of the server application. (Refer to "Notes on Building and Running Windows(R) Applications".)

(4) Create DLL Files

Link compiled objects to create DLL files. The format of DLL filenames is "module name"-interface name".DLL. Select Tool | Link and specify with the following files.

Windows32

For process mode:

```
C:\Interstage\ODWIN\LIB\ODCOBCBLSV.LIB
```

For thread mode:

```
C:\Interstage\ODWIN\LIB\ODCOBCBLMTSV.LIB
```

For process mode/thread mode (UNICODE/UTF8 module):

```
C:\Interstage\ODWIN\LIB\ODCOBCBLSVUC.LIB
```

Windows64

For common:

```
F4AGCIMP.LIB
libcmt.lib
kernel32.lib
```

For process mode:

```
C:\Interstage\ODWIN\LIB\x64\ODCOBCBLSV.LIB
```

For thread mode:

```
C:\Interstage\ODWIN\LIB\x64\ODCOBCBLMTSV.LIB
```

For process mode/thread mode (UNICODE/UTF8 module):

```
C:\Interstage\ODWIN\LIB\x64\ODCOBCBLSVUC.LIB
```

(5) Create Executable Program

Windows32

Create an executable program. Specify with the following files.

For process mode:

```
C:\Interstage\ODWIN\LIB\ODCOBCBLSV.LIB
```

For thread mode:

```
C:\Interstage\ODWIN\LIB\ODCOBCBLMTSV.LIB
```

For UNICODE :

```
C:\Interstage\ODWIN\LIB\ODCOBCBLSVUC.LIB
```

Windows64

Create the execution program of the server application as the main program of the main processing. At this time, specify the following files in [Link Options].

For common:

```
F4AGCIMP.LIB
libcmt.lib
kernel32.lib
```

For process Mode:

```
C:\Interstage\ODWIN\LIB\x64\ODCOBCBLSV.LIB
```

For thread mode:

```
C:\Interstage\ODWIN\LIB\x64\ODCOBCBLMTSV.LIB
```

For process mode/thread mode (UNICODE/UTF8 module):

```
C:\Interstage\ODWIN\LIB\x64\ODCOBCBLSVUC.LIB
```

(6) Register Application

Register the application using the OD_impl_inst command. For a definition file passed to the command, refer to the following sample files.

```
C:\Interstage\ODWIN\SRC\SAMPLE\COMPLEX\SAMPLELIST.COBOL\DATA\*\*.DEF
```

Notes

- Set SYNC_END for mode.
- For DLL, set the DLL name only and add the DLL path to the PATH environment variable.

Target files

```
'IDL file name'_'interface name'_SKEL.CBL
'IDL file name'_CDR.CBL
'IDL file name'_SKEL.CBL
```

Note that these files may not be generated, depending on the IDL files.

Compile Options

For common:

- Specify the NAME option.
- Note specify the DLOAD option.

For thread mode:

- Specify the THREAD(MULTI) option.

For process mode/thread mode (UNICODE/UTF8 module):

- Specify the RCS option (Use UCS2-Unicode (UCS2)).
- In thread mode, specify the THREAD(MULTI) option.

For details about the compile options, refer to the NetCOBOL manual.

Library

Set the following environment variable.

Windows32

```
CORBA=C:\Interstage\ODWIN\INCLUDE\COBOL
```

Windows64

```
COB_CORBA=C:\Interstage\ODWIN\INCLUDE\COBOL
```

Files Generated

```
'module name'-'interface name'-'operation name'.OBJ
target objects for skeleton and cdr source.
```

Building Solaris Applications

Solaris32/64

This section provides notes about building COBOL applications under Solaris.

Note

To develop UNICODE applications using the "NetCOBOL", create the source files and any libraries using UNICODE (UTF-8).

To perform translations/links/execution of the application, set "UNICODE" as the code type.

Point

In this explanation, the CORBA Service installation directory is /opt/FSUNod.

Building Client Applications

The translation and linkage procedures for creating client applications are described below.

Translation and Linkage Procedures

1. Compile IDL

```
IDLc -cobol IDL filename
```

2. Specify the registry. Specify the following directory for the CORBA environment variable:

```
CORBA=/opt/FSUNod/include/COBOL
```

3. Translate the main program. The compilation method is as follows (XXX_c.cbl: Main program filename):

   For process mode:

```
cobol -M -c XXX_c.cbl
```

   For thread mode:

```
cobol -Tm -M -c XXX_c.cbl
```

   For process mode (UNICODE/UTF8 module):

```
cobol -M -c XXX_c.cbl
```

   For thread mode (UNICODE/UTF8 module):

```
cobol -Tm -M -c XXX_c.cbl
```

4. Translate the stub and COR source generated in Step 1. The compilation method is as follows (A_cdr.cbl: COR source filename; B_stub.cbl: Stub filename):

   For process mode:

```
cobol  -G -o libA_cdr.so  A_cdr.cbl
cobol  -G -o libB_stub.so -lA_cdr  B_stub.cbl
```

   For thread mode:

```
cobol  -G -Tm -o libA_cdr.so  A_cdr.cbl
cobol  -G -Tm -o libB_stub.so -lA_cdr  B_stub.cbl
```

   For process mode/thread mode (UNICODE/UTF8 module):

```
cobol  -G -Tm -o libA_cdr.so  A_cdr.cbl
cobol  -G -Tm -o libB_stub.so -lA_cdr  B_stub.cbl
```

5. Link COBOL libraries and create the client application.

   (CAP_c: Name of created client application)

   For process mode:

   Links /opt/FSUNod/lib/libOMcbl.so

```
cobol -L/opt/FSUNod/lib -lOMcbl -o CAP_c -lB_stub -lA_cdr XXX_c.o
```

   For thread mode:

   Links /opt/FSUNod/lib/libOMcblMT.so

```
cobol -L/opt/FSUNod/lib -lOMcblMT -Tm -o CAP_c -lB_stub -lA_cdr XXX_c.o
```

   For process mode/thread mode (UNICODE/UTF8 module):

   Links /opt/FSUNod/lib/libOMcblUC.so

```
cobol -L/opt/FSUNod/lib -lOMcblUC -Tm -o CAP_c -lB_stub -lA_cdr XXX_c.o
```

Building Server Applications

The translation and linkage procedures for creating server applications are described below.

Translation and Linkage Procedures

1. Compile IDL

```
IDLc -cobol IDL filename
```

2. Specify the registry. Specify the following directory for the CORBA environment variable:

```
CORBA=/opt/FSUNod/include/COBOL
```

3. Translate the main program. The compilation method is as follows (XXX_s.cbl: Main program filename):

   For process mode:

```
cobol -M -c XXX_s.cbl
```

   For thread mode:

```
cobol -Tm -M -c XXX_s.cbl
```

   For process mode (UNICODE/UTF8 module):

```
cobol -M -c XXX_s.cbl
```

   For thread mode (UNICODE/UTF8 module):

```
cobol -Tm -M -c XXX_s.cbl
```

4. Create the server application (initialization processing part).

   (SAP_s: Server application name)

   For process mode:

   Links /opt/FSUNod/lib/libOMcbl.so

```
cobol -lOMcbl -o SAP_s XXX_s.o
```

   For thread mode:

   Links /opt/FSUNod/lib/libOMcblMT.so

```
cobol -lOMcblMT -Tm -o SAP_s XXX_s.o
```

   For process mode/thread mode (UNICODE/UTF8 module):

   Links /opt/FSUNod/lib/libOMcblUC.so

```
cobol -lOMcblUC -Tm -o SAP_s XXX_s.o
```

5. Translate the skeleton and COR source generated in Step 1. The compilation method is as follows (A_cdr.cbl: COR source filename; B_skel.cbl: Skeleton filename). If the IDL definition contains data that is not basic data type data, compile and link of the skeleton file for acquiring area is also required.

   For process mode:

```
cobol  -G -o libA_cdr.so  A_cdr.cbl
cobol  -G -o libB_skel.so B_skel.cbl
```

   For thread mode:

```
cobol  -G -Tm -o libA_cdr.so  A_cdr.cbl
cobol  -G -Tm -o libB_skel.so B_skel.cbl
```

   For process mode/thread mode (UNICODE/UTF8 module):

```
cobol  -G -Tm -o libA_cdr.so  A_cdr.cbl
cobol  -G -Tm -o libB_skel.so B_skel.cbl
```

6. Link COBOL libraries and create the server application (interface device part).

   (D_sap.cbl: COBOL filename; libC.so: Library filename)

   For process mode:

Links /opt/FSUNod/lib/libOMcbl.so

```
cobol -G -lcobol -lOMcbl -o libC.so -lA_cdr -lB_skel D_sa.cbl
```

For thread mode:

Links /opt/FSUNod/lib/libOMcblMT.so

```
cobol -G -Tm -lrcobol -lOMcblMT -o libC.so -lA_cdr -lB_skel D_sa.cbl
```

For process mode/thread mode (UNICODE/UTF8 module):

Links /opt/FSUNod/lib/libOMcblUC.so

```
cobol -G -Tm -lrcobol -lOMcblUC -o libC.so -lA_cdr -lB_skel D_sa.cbl
```

Building Linux Applications

Linux32/64

This section provides notes about building COBOL applications under Linux.

Note

To develop UNICODE applications, create the source files and any libraries using UNICODE (UTF-8).

To perform translations/links/execution of the application, set "UNICODE" as the code type.

In this explanation, the CORBA Service installation directory is /opt/FJSVod.

Linux64

To perform COBOL translations using UNICODE, specify "UCS2" in the RCS option. At this time, the endian can be selected. Specify either a big endian (BE) or a little endian (LE).

Building Client Applications

The translation and linkage procedures for creating client applications are described below.

Translation and Linkage Procedures

1. Compile IDL

```
IDLc -cobol IDL filename
```

2. Specify the registry. Specify the following directory for the CORBA environment variable:

```
CORBA=/opt/FJSVod/include/COBOL
```

3. Translate the main program. The compilation method is as follows (XXX_c.cbl: Main program filename):

For process mode:

```
cobol -M -c XXX_c.cbl
```

For thread mode:

```
cobol -Tm -M -c XXX_c.cbl
```

For process mode (UNICODE/UTF8 module):

```
cobol -M -c XXX_c.cbl
```

For thread mode (UNICODE/UTF8 module):

```
cobol -Tm -M -c XXX_c.cbl
```

Linux64

For process mode (Specifying a big endian in the UNICODE/UTF8 module):

```
cobol -WC,"RCS(UCS2,BE)" -M -c XXX_c.cbl
```

For thread mode (Specifying a big endian in the UNICODE/UTF8 module):

```
cobol -Tm -WC,"RCS(UCS2,BE)" -M -c XXX_c.cbl
```

For process mode (Specifying a little endian in the UNICODE/UTF8 module):

```
cobol -WC,"RCS(UCS2,LE)" -M -c XXX_c.cbl
```

For thread mode (Specifying a little endian in the UNICODE/UTF8 module):

```
cobol -Tm -WC,"RCS(UCS2,LE)" -M -c XXX_c.cbl
```

4. Translate the stub and COR source generated in Step 1. The compilation method is as follows (A_cdr.cbl: COR source filename; B_stub.cbl: Stub filename):

   For process mode:

   ```
   cobol  -G -o libA_cdr.so  A_cdr.cbl
   cobol  -G -o libB_stub.so -L. -lA_cdr  B_stub.cbl
   ```

   For thread mode:

   ```
   cobol  -G -Tm -o libA_cdr.so  A_cdr.cbl
   cobol  -G -Tm -o libB_stub.so -L. -lA_cdr  B_stub.cbl
   ```

   For process mode/thread mode (UNICODE/UTF8 module):

   ```
   cobol  -G -Tm -o libA_cdr.so  A_cdr.cbl
   cobol  -G -Tm -o libB_stub.so -L. -lA_cdr  B_stub.cbl
   ```

   Linux64

   For process mode/thread mode (Specifying a big endian in the UNICODE/UTF8 module):

   ```
   cobol  -G -Tm -WC,"RCS(UCS2,BE)" -o libA_cdr.so  A_cdr.cbl
   cobol  -G -Tm -WC,"RCS(UCS2,BE)" -o libB_stub.so -L. -lA_cdr  B_stub.cbl
   ```

   For process mode/thread mode (Specifying a little endian in the UNICODE/UTF8 module):

   ```
   cobol  -G -Tm -WC,"RCS(UCS2,LE)" -o libA_cdr.so  A_cdr.cbl
   ```

   ```
   cobol  -G -Tm -WC,"RCS(UCS2,LE)" -o libB_stub.so -L. -lA_cdr  B_stub.cbl
   ```

5. Link COBOL libraries and create the client application.

   (CAP_c: Name of created client application)

   For process mode:

   Links /opt/FJSVod/lib/libOMcbl.so

   ```
   cobol -L/opt/FJSVod/lib -lOMcbl -o CAP_c -L. -lB_stub -lA_cdr XXX_c.o
   ```

   For thread mode:

   Links /opt/FJSVod/lib/libOMcblMT.so

   ```
   cobol -L/opt/FJSVod/lib -lOMcblMT -Tm -o CAP_c -lB_stub -lA_cdr XXX_c.o
   ```

   For process mode/thread mode (UNICODE/UTF8 module):

   Links /opt/FJSVod/lib/libOMcblUC.so

   ```
   cobol -L/opt/FJSVod/lib -lOMcblUC -Tm -o CAP_c -L. -lB_stub -lA_cdr XXX_c.o
   ```

Linux64

For process mode/thread mode (Specifying a big endian in the UNICODE/UTF8 module):

Links /opt/FJSVod/lib/libOMcblUCBE.so

```
cobol -L/opt/FJSVod/lib -lOMcblUCBE -Tm -o CAP_c -L. -lB_stub -lA_cdr XXX_c.o
```

For process mode/thread mode (Specifying a little endian in the UNICODE/UTF8 module):

Links /opt/FJSVod/lib/libOMcblUCLE.so

```
cobol -L/opt/FJSVod/lib -lOMcblUCLE -Tm -o CAP_c -L. -lB_stub -lA_cdr XXX_c.o
```

Building Server Applications

The translation and linkage procedures for creating server applications are described below.

Translation and Linkage Procedures

1. Compile IDL

   ```
   IDLc -cobol IDL filename
   ```

2. Specify the registry. Specify the following directory for the CORBA environment variable:

   ```
   CORBA=/opt/FJSVod/include/COBOL
   ```

3. Translate the main program. The compilation method is as follows (XXX_s.cbl: Main program filename):

   For process mode:

   ```
   cobol -M -c XXX_s.cbl
   ```

   For thread mode:

   ```
   cobol -Tm -M -c XXX_s.cbl
   ```

   For process mode (UNICODE/UTF8 module):

   ```
   cobol -M -c XXX_s.cbl
   ```

   For thread mode (UNICODE/UTF8 module):

   ```
   cobol -Tm -M -c XXX_s.cbl
   ```

   Linux64

   For process mode (Specifying a big endian in the UNICODE/UTF8 module):

   ```
   cobol -WC,"RCS(UCS2,BE)" -M -c XXX_s.cbl
   ```

   For thread mode (Specifying a big endian in the UNICODE/UTF8 module):

   ```
   cobol -Tm -WC,"RCS(UCS2,BE)" -M -c XXX_s.cbl
   ```

   For process mode (Specifying a little endian in the UNICODE/UTF8 module):

   ```
   cobol -WC,"RCS(UCS2,LE)" -M -c XXX_s.cbl
   ```

   For thread mode (Specifying a little endian in the UNICODE/UTF8 module):

   ```
   cobol -Tm -WC,"RCS(UCS2,LE)" -M -c XXX_s.cbl
   ```

4. Create the server application (initialization processing part).

   (SAP_s: Server application name)

For process mode:

Links /opt/FJSVod/lib/libOMcbl.so

```
cobol -L/opt/FJSVod/lib -lOMcbl -o SAP_s XXX_s.o
```

For thread mode:

Links /opt/FJSVod/lib/libOMcblMT.so

```
cobol -L/opt/FJSVod/lib -lOMcblMT -Tm -o SAP_s XXX_s.o
```

For process mode/thread mode (UNICODE/UTF8 module):

Links /opt/FJSVod/lib/libOMcblUC.so

```
cobol -L/opt/FJSVod/lib -lOMcblUC -Tm -o SAP_s XXX_s.o
```

Linux64

For process mode/thread mode (Specifying a big endian in the UNICODE/UTF8 module):

Links /opt/FJSVod/lib/libOMcblUCBE.so

```
cobol -L/opt/FJSVod/lib -lOMcblUCBE -Tm -o SAP_s XXX_s.o
```

For process mode/thread mode (Specifying a little endian in the UNICODE/UTF8 module):

Links /opt/FJSVod/lib/libOMcblUCLE.so

```
cobol -L/opt/FJSVod/lib -lOMcblUCLE -Tm -o SAP_s XXX_s.o
```

5. Translate the skeleton and COR source generated in Step 1. The compilation method is as follows (A_cdr.cbl: COR source filename; B_skel.cbl: Skeleton filename). If the IDL definition contains data that is not basic data type data, compile and link of the skeleton file for acquiring area is also required.

   For process mode:

   ```
   cobol -G -o libA_cdr.so A_cdr.cbl
   cobol -G -o libB_skel.so B_skel.cbl
   ```

   For thread mode:

   ```
   cobol -G -Tm -o libA_cdr.so A_cdr.cbl
   cobol -G -Tm -o libB_skel.so B_skel.cbl
   ```

   For process mode/thread mode (UNICODE/UTF8 module):

   ```
   cobol -G -Tm -o libA_cdr.so A_cdr.cbl
   cobol -G -Tm -o libB_skel.so B_skel.cbl
   ```

   Linux64

   For process mode/thread mode (Specifying a big endian in the UNICODE/UTF8 module):

   ```
   cobol -G -Tm -WC,"RCS(UCS2,BE)" -o libA_cdr.so A_cdr.cbl
   cobol -G -Tm -WC,"RCS(UCS2,BE)" -o libB_skel.so B_skel.cbl
   ```

   For process mode/thread mode (Specifying a little endian in the UNICODE/UTF8 module):

   ```
   cobol -G -Tm -WC,"RCS(UCS2,LE)" -o libA_cdr.so A_cdr.cbl
   cobol -G -Tm -WC,"RCS(UCS2,LE)" -o libB_skel.so B_skel.cbl
   ```

6. Link COBOL libraries and create the server application (interface device part).

   (D_sap.cbl: COBOL filename; libC.so: Library filename)

   For process mode:

   Links /opt/FJSVod/lib/libOMcbl.so

```
cobol -G -L/opt/FJSVod/lib -lOMcbl -o libD.so -L. -lA_cdr -lB_skel D_sa.cbl
```

For thread mode:

Links /opt/FJSVod/lib/libOMcblMT.so

```
cobol -G -Tm -L/opt/FJSVod/lib -lOMcblMT -o libD.so -L. -lA_cdr -lB_skel D_sa.cbl
```

For process mode/thread mode (UNICODE/UTF8 module):

Links /opt/FJSVod/lib/libOMcblUC.so

```
cobol -G -Tm -L/opt/FJSVod/lib -lOMcblUC -o libD.so -L. -lA_cdr -lB_skel D_sa.cbl
```

Linux64

For process mode/thread mode (Specifying a big endian in the UNICODE/UTF8 module):

Links /opt/FJSVod/lib/libOMcblUCBE.so

```
cobol -G -Tm -WC,"RCS(UCS2,BE)" -L/opt/FJSVod/lib -lOMcblUCBE -o libD.so -L. -lA_cdr -lB_skel
D_sa.cbl
```

For process mode/thread mode (Specifying a little endian in the UNICODE/UTF8 module):

Links /opt/FJSVod/lib/libOMcblUCLE.so

```
cobol -G -Tm -WC,"RCS(UCS2,LE)" -L/opt/FJSVod/lib -lOMcblUCLE -o libD.so -L. -lA_cdr -lB_skel
D_sa.cbl
```

List of Supported Reserved Words

The list of supported reserved words is shown below.

"ACCEPT"	"ACCESS"	"ACTUAL"	"ADD"
"ADDRESS"	"ADVANCING"	"AFTER"	"ALL"
"ALPHABET"	"ALPHABETIC"	"ALPHANUMERIC"	"ALSO"
"ALTER"	"ALTERNATE"	"AND"	"ANY"
"APPLY"	"ARE"	"AREA"	"AREAS"
"ARITHMETIC"	"AS"	"ASCENDING"	"ASSIGN"
"AT"	"AUTHOR"	"AUTO"	"AUTOMATIC"
"BASED"	"BEFORE"	"BEGINNING"	"BELL"
"BINARY"	"BIT"	"BITS"	"BLANK"
"BLINK"	"BLOCK"	"BOTTOM"	"BY"
"CALL"	"CANCEL"	"CBL"	"CD"
"CF"	"CH"	"CHANGED"	"CHARACTER"
"CHARACTERS"	"CLASS"	"CLOSE"	"COBOL"
"CODE"	"COLLATING"	"COLUMN"	"COMMA"
"COMMAND"	"COMMIT"	"COMMON"	"COMMUNICATION"
"COMP"	"COMPLEX"	"COMPUTATIONAL"	"COMPUTE"
"CONFIGURATION"	"CONNECT"	"CONSTANT"	"CONTAINED"
"CONTAINS"	"CONTENT"	"CONTINUE"	"CONTROL"
"CONTROLS"	"CONVERTING"	"COPY"	"CORR"
"CORRESPONDING"	"COUNT"	"CRP"	"CRT"

"CURRENCY"	"CURRENT"	"CURSOR"	"DATA"
"DATE"	"DAY"	"DB"	"DE"
"DEBUGGING"	"DEFAULT"	"DELETE"	"DELIMITED"
"DELIMITER"	"DEPENDING"	"DESCENDING"	"DESTINATION"
"DETAIL"	"DEVICE"	"DIRECT"	"DISABLE"
"DISCONNECT"	"DISJOINING"	"DISPLAY"	"DIVIDE"
"DIVISION"	"DOWN"	"DUPLICATE"	"DUPLICATES"
"DYNAMIC"	"EGCS"	"EGI"	"EJECT"
"ELSE"	"EMI"	"EMPTY"	"ENABLE"
"END"	"ENDCOBOL"	"ENDING"	"ENTER"
"ENTRY"	"ENVIRONMENT"	"EOL"	"EOP"
"EOS"	"EQUAL"	"EQUALS"	"ERASE"
"ERROR"	"ESI"	"EVALUATE"	"EVERY"
"EXACT"	"EXAMINE"	"EXCEEDS"	"EXCEPTION"
"EXCLUSIVE"	"EXEC"	"EXIT"	"EXOR"
"EXTEND"	"EXTERNAL"	"FD"	"FETCH"
"FILE"	"FILES"	"FILLER"	"FINAL"
"FIND"	"FINISH"	"FIRST"	"FLADD"
"FOOTING"	"FOR"	"FORM"	"FORMAT"
"FORMATTED"	"FREE"	"FROM"	"FULL"
"FUNCTION"	"GENERATE"	"GET"	"GIVING"
"GLOBAL"	"GO"	"GOBACK"	"GREATER"
"GRID"	"GROUP"	"HEADING"	"HIGHLIGHT"
"ID"	"IDENTIFICATION"	"IF"	"IN"
"INCLUDE"	"INDEX"	"INDEXED"	"INDICATE"
"INITIAL"	"INITIALIZE"	"INITIATE"	"INPUT"
"INSPECT"	"INSTALLATION"	"INTO"	"INVALID"
"IS"	"JAPANESE"	"JOB"	"JOINING"
"JUST"	"JUSTIFIED"	"KANJI"	"KEEP"
"KEY"	"LABEL"	"LAST"	"LD"
"LEADING"	"LEFT"	"LEFTLINE"	"LENGTH"
"LESS"	"LIMIT"	"LIMITED"	"LIMITS"
"LINAGE"	"LINE"	"LINES"	"LINKAGE"
"LOCALLY"	"LOCK"	"LOWLIGHT"	"MANUAL"
"MEMBER"	"MEMORY"	"MERGE"	"MESSAGE"
"MODE"	"MODIFY"	"MODULES"	"MOVE"
"MULTICON"	"MULTIPLE"	"MULTIPLY"	"NAMED"
"NATIONAL"	"NATIVE"	"NEGATIVE"	"NEXT"
"NO"	"NOMINAL"	"NONE"	"NOT"
"NOTE"	"NULL"	"NULLS"	"NUMBER"

"NUMERIC"	"OCCURS"	"OF"	"OFF"
"OMITTED"	"ON"	"ONLY"	"OPEN"
"OPTIONAL"	"OR"	"ORDER"	"ORGANIZATION"
"OTHER"	"OTHERWISE"	"OUTPUT"	"OVERFLOW"
"OVERLINE"	"OWNER"	"PADDING"	"PAGE"
"PASSWORD"	"PERFORM"	"PF"	"PH"
"PIC"	"PICTURE"	"PLUS"	"POINTER"
"POSITION"	"POSITIONING"	"POSITIVE"	"PREFIX"
"PRESENT"	"PREVIOUS"	"PRINTING"	"PRIOR"
"PROCEDURE"	"PROCEDURES"	"PROCEED"	"PROCESSING"
"PROGRAM"	"PROMPT"	"PROTECTED"	"PURGE"
"QUEUE"	"QUOTE"	"QUOTES"	"RANDOM"
"RANGE"	"RD"	"READ"	"READY"
"REALM"	"RECEIVE"	"RECONNECT"	"RECORD"
"RECORDING"	"RECORDS"	"REDEFINES"	"REEL"
"REFERENCE"	"REFERENCES"	"RELATION"	"RELATIVE"
"RELEASE"	"RELOAD"	"REMAINDER"	"REMARKS"
"REMOVAL"	"RENAMES"	"REPEATED"	"REPLACE"
"REPLACING"	"REPORT"	"REPORTING"	"REPORTS"
"RERUN"	"REQUIRED"	"RESERVE"	"RESET"
"RETAINING"	"RETRIEVAL"	"RETURN"	"REVERSED"
"REWIND"	"REWRITE"	"RF"	"RH"
"RIGHT"	"ROLLBACK"	"ROUNDED"	"RUN"
"SA"	"SAME"	"SCREEN"	"SD"
"SEARCH"	"SECTION"	"SECURITY"	"SECURE"
"SEEK"	"SEGMENT"	"SELECT"	"SELECTED"
"SELECTIVE"	"SEND"	"SENTENCE"	"SEPARATE"
"SEQUENCE"	"SEQUENTIAL"	"SERVICE"	"SESSION"
"SET"	"SHARED"	"SIGN"	"SIMPLE"
"SINGLE"	"SIZE"	"SKIP1"	"SKIP2"
"SKIP3"	"SORT"	"SOURCE"	"SPACE"
"SPACES"	"STANDERD"	"START"	"STATION"
"STATIONS"	"STATUS"	"STOP"	"STORE"
"STRING"	"SUBRANGE"	"SUBTRACT"	"SUCCESSIVE"
"SUFFIX"	"SUM"	"SUPPRESS"	"SYMBOLIC"
"SYNC"	"SYNCHRONIZED"	"TABLE"	"TALLY"
"TALLYING"	"TAPE"	"TENANT"	"TERMINAL"
"TERMINATE"	"TEST"	"TEXT"	"THAN"
"THEN"	"THROUGH"	"THRU"	"TIME"
"TIMES"	"TITLE"	"TO"	"TOP"

"TRACE"	"TRACK"	"TRACKS"	"TRAILING"
"TRANSACTION"	"TYPE"	"UNDERLINE"	"UNEQUAL"
"UNIT"	"UNLOCK"	"UNSTRING"	"UNTIL"
"UP"	"UPDATE"	"UPON"	"USAGE"
"USE"	"USING"	"VALID"	"VALIDATE"
"VALUE"	"VALUES"	"VARYING"	"WAIT"
"WHEN"	"WITH"	"WITHIN"	"WORDS"
"WRITE"	"ZERO"	"ZEROES"	"ZEROS"

7.10 COBOL Library

The library texts for the functions, data declarations, and so on, provided by the CORBA Service are shown below.

Windows32/64

Storage directory (Default installation path)

```
C:\Interstage\ODWIN\INCLUDE\COBOL
```

Solaris32/64

Storage directory (Default installation path)

```
/opt/FSUNod/include/COBOL
```

Linux32

Storage directory (Default installation path)

```
/opt/FJSVod/include/COBOL
```

7.10.1 Use Example

This section provides usage examples.

(1) The declaration in the composition knot of the environment part

A COPY SYMBOL-CONST IN CORBA declaration is done by the SYMBOLIC CONSTANT phrase in the SPECIAL-NAMES inside.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.

```
    SYMBOLIC CONSTANT
    COPY SYMBOL-CONST IN CORBA.
    .
```

(2) The declaration of the function that it is succeeded to and which can be used

Before a necessary function is extracted from the REPLACE source of the library and used, the function declaration which is succeeded to with Interface Repository and which can be used is declared.

```
REPLACE
== "CORBA-CONTAINED--GET-DEF-KIND" == BY
== "CORBA-IROBJECT--GET-DEF-KIND" ==
    .
```

(3) The declaration in the workshop place knot of the data part (1)

A COPY CONST IN CORBA declaration is declared right under WORKING-STORAGE SECTION. (A level number is not set up.)

WORKING-STORAGE SECTION.

```
COPY CONST IN CORBA.
```

(4) The declaration in the workshop place knot of the data part (2)

A variety type declaration declares a territory by the COPY sentence.

Description Example 1

```
01 COPY ULONG IN CORBA REPLACING
        (1)      (2)
CORBA-UNSIGNED-LONG BY STRING-LENGTH.
      (3)              (4)
```

Notes

1. The source name of the registration collection is described.
2. An environment variable name is described. (CORBA is used.)
3. The data name in the source of the registration collection is described.
4. The variable name defined by the user is described.

Description Example 2

```
01 COPY ENVIRONMENT    IN CORBA         REPLACING
                CORBA-ENVIRONMENT       BY ENV.
01 COPY ORB            IN CORBA         REPLACING
                CORBA-ORB               BY ORB.
01 COPY BOA            IN CORBA         REPLACING
                CORBA-BOA               BY BOA.
01 COPY OBJECT         IN CORBA         REPLACING
```

7.10.2 Library Texts

This section describes library texts.

Data Type Declarations

ANY.cbl	CORBA-ANY type structure
BOOLEAN.cbl	CORBA-BOOLEAN type
CHAR.cbl	CORBA-CHAR type
DOUBLE.cbl	CORBA-DOUBLE type
ENUM.cbl	CORBA-ENUM type
FIXED-D-S.cbl	CORBA-FIXED-D-S type
FLOAT.cbl	CORBA_FLOAT type
LONG.cbl	CORBA_LONG type
OBJECT.cbl	CORBA_OBJECT type
OCTET.cbl	CORBA_OCTET type
SHORT.cbl	CORBA-SHORT type

TYPECODE.cbl	CORBA-TYPECODE type
ULONG.cbl	CORBA-UNSIGNED-LONG type
ULLONG.cbl	CORBA-UNSIGNED-LONG-LONG type
USHORT.cbl	CORBA-UNSIGNED-SHORT type
LLONG.cbl	CORBA-LONG-LONG type
WCHAR.cbl	CORBA-WCHAR type

Literal Type Declarations

COMPLETION-STATUS.cbl	Status of env member communication
CONST.cbl	Fixed character-string information
FJ-IMPLEMENTATIONDEF.cbl	Implementation object reference
FLAGS.cbl	Flag
IMPL-BOA.cbl	Implementation Repository ID
IMPL-CONTEXT.cbl	Implementation Repository ID
IMPL-IDLTYPE.cbl	Implementation Repository ID
IMPL-IMPLEMENTATIONDEF.cbl	Implementation Repository ID
IMPL-INTERFACEDEF.cbl	Implementation Repository ID
IMPL-NVLIST.cbl	Implementation Repository ID
IMPL-OBJECT.cbl	Implementation Repository ID
IMPL-OPERATIONDEF.cbl	Implementation Repository ID
IMPL-ORB.cbl	Implementation Repository ID
IMPL-PRINCIPAL.cbl	Implementation Repository ID
IMPL-REQUEST.cbl	Implementation Repository ID
IMPL-SERVERREQUEST.cbl	Implementation Repository ID
IMPL-TYPECODE.cbl	Implementation Repository ID
ORBSTATUS.cbl	ORB information type
STATUS.cbl	Confirmation flag at dynamic return
SYMBOL-CONST.cbl	Constant information

Error Exception Declarations

BAD-CONTEXT.cbl	System exception definition
BAD-INV-ORDER.cbl	System exception definition
BAD-OPERATION.cbl	System exception definition
BAD-PARAM.cbl	System exception definition
BAD-QOS.cbl	System exception definition
BAD-TYPECODE.cbl	System exception definition
CODESET-INCOMPATIBLE.cbl	System exception definition
COMM-FAILURE.cbl	System exception definition
DATA-CONVERSION.cbl	Exception information
ENVIRONMENT.cbl	Exception information structure

EX-ORB-INVALIDNAME.cbl	Exception character-string
EX-STEXCEP-BAD-INV-ORDER.cbl	Exception character-string
EX-STEXCEP-BAD-OPERATION.cbl	Exception character-string
EX-STEXCEP-BAD-PARAM.cbl	Exception character-string
EX-STEXCEP-BAD-QOS.cbl	Exception character-string
EX-STEXCEP-BAD-TYPECODE.cbl	Exception character-string
EX-STEXCEP-CODESET-INCOM.cbl	Exception character-string
EX-STEXCEP-COMM-FAILURE.cbl	Exception character-string
EX-STEXCEP-CONTEXT.cbl	Exception character-string
EX-STEXCEP-DATA-CONVERSI.cbl	Exception character-string
EX-STEXCEP-FREE-MEM.cbl	Exception character-string
EX-STEXCEP-IMP-LIMIT.cbl	Exception character-string
EX-STEXCEP-INITIALIZE.cbl	Exception character-string
EX-STEXCEP-INTERNAL.cbl	Exception character-string
EX-STEXCEP-INTF-REPOS.cbl	Exception character-string
EX-STEXCEP-INV-FLAG.cbl	Exception character-string
EX-STEXCEP-INV-IDENT.cbl	Exception character-string
EX-STEXCEP-INV-OBJREF.cbl	Exception character-string
EX-STEXCEP-INV-POLICY.cbl	Exception character-string
EX-STEXCEP-MARSHAL.cbl	Exception character-string
EX-STEXCEP-NO-IMPLEMENT.cbl	Exception character-string
EX-STEXCEP-NO-MEMORY.cbl	Exception character-string
EX-STEXCEP-NO-PERMISSION.cbl	Exception character-string
EX-STEXCEP-NO-RESOURCES.cbl	Exception character-string
EX-STEXCEP-NO-RESPONSE.cbl	Exception character-string
EX-STEXCEP-OBJ-ADAPTER.cbl	Exception character-string
EX-STEXCEP-PERSIST-STORE.cbl	Exception character-string
EX-STEXCEP-REBIND.cbl	Exception character-string
EX-STEXCEP-TIMEOUT.cbl	Exception character-string
EX-STEXCEP-TRANSACTION-M.cbl	Exception character-string
EX-STEXCEP-TRANSACTION-U.cbl	Exception character-string
EX-STEXCEP-TRANSIENT.cbl	Exception character-string
EX-STEXCEP-UNKNOWN.cbl	Exception character-string
EX-TYPECODE-BADKIND.cbl	Exception character-string
EX-TYPECODE-BOUNDS.cbl	Exception character-string
EXCDESCRIPTIONSEQ.cbl	ExceptionDef structure sequence
EXCEPTION-TYPE.cbl	ExceptionDef type
EXCEPTIONDEF.cbl	ExceptionDef object reference
EXCEPTIONDEFSEQ.cbl	ExceptionDef object reference sequence
EXCEPTIONDESCRIPTION.cbl	ExceptionDef structure

FREE-MEM.cbl	Exception
IMP-LIMIT.cbl	Exception
INITIALIZE.cbl	Exception
INTERNAL.cbl	Exception
INTF-REPOS.cbl	Exception
INV-FLAGS.cbl	Exception
INV-IDENT.cbl	Exception
INV-OBJREF.cbl	Exception
NO-IMPLEMENT.cbl	Exception
NO-MEMORY.cbl	Exception
NO-PERMISSION.cbl	Exception
NO-RESOURCES.cbl	Exception
NO-RESPONSE.cbl	Exception
NOT-EXIST.cbl	Exception
OBJ-ADAPTER.cbl	Exception
PARSIST-STORE.cbl	Exception
REBIND.cbl	Exception
TIMEOUT.cbl	Exception
TRANSACTION-MODE.cbl	Exception
TRANSACTION-UNAVAILABLE.cbl	Exception
TRANSIENT.cbl	Exception
UNKNOWN.cbl	Exception

NamingService Declarations

COSNAMING-BINDING-LIST.cbl	Naming context binding list structure
COSNAMING-BINDING.cbl	Naming context binding
COSNAMING-BINDINGITERATOR.cbl	Naming context binding iterator
COSNAMING-BINDINGTYPE.cbl	Naming context binding type
COSNAMING-ISTRING.cbl	Naming context string type
COSNAMING-NAME.cbl	Naming context name
COSNAMING-NAMECOMPONENT.cbl	Name component
COSNAMING-NAMINGCONTEXT.cbl	Naming context object reference
COSNAMING-NAMINGCONTEXT-CANNOT.cbl	Naming context CannotProceed exception
COSNAMING-NAMINGCONTEXTEXT.cbl	Extended naming context object reference
COSNAMING-NAMINGCONTEXTEXT-ADD.cbl	Extended naming context address type
COSNAMING-NAMINGCONTEXTEXT-STR.cbl	Extended naming context string-type naming context name
COSNAMING-NAMINGCONTEXTEXT-URL.cbl	Extended naming context URL address type
COSNAMING-NAMINGCONTEXT-NOT001.cbl	Naming context NotFound exception
COSNAMING-NAMINGCONTEXT-NOTFOU.cbl	Reason for naming context NotFound exception

Load Balance Option Declarations

This is not valid for Linux (64 bit).

CONST-LBO.cbl	Load balance option object ID
ISOD-LBG.cbl	Load balance object group's object reference
ISOD-LBG-LOADBALANCETYPE.cbl	Load balance type
ISOD-LBG-OBJECTLIST.cbl	Object reference list structure
ISOD-LBO.cbl	Load balance object reference
ISOD-LBO-BINDINGLBG.cbl	Load balance object group binding
ISOD-LBO-DOWNSERVERLIST.cbl	Down-server list structure
ISOD-LBO-DUPLICATETYPE.cbl	Load balance option AlreadyExist exception detail
ISOD-LBO-LBGLIST.cbl	Load balance object group's binding list structure
ISOD-LBO-LOADBALANCETYPE.cbl	Load balance type
ISOD-LBO-NOTFOUNDREASON.cbl	Load balance option NotFound exception

InterfaceRepository Declarations

INTERFACEDEF-FULLINTERFA.cbl	Interface information class
INTERFACEDEF.cbl	InterfaceDef object reference
INTERFACEDEFSEQ.cbl	InterfaceDef object reference sequence
INTERFACEDESCRIPTION.cbl	InterfaceDef information structure
INTERFACEREP.cbl	InterfaceDef object reference
INTF-BOA.cbl	Interface repository ID
INTF-CONTEXT.cbl	Interface repository ID
INTF-IDLTYPE.cbl	Interface repository ID
INTF-IMPLEMENTATIONDEF.cbl	Interface repository ID
INTF-INTERFACEDEF.cbl	Interface repository ID
INTF-NVLIST.cbl	Interface repository ID
INTF-OBJECT.cbl	Interface repository ID
INTF-OPERATIONDEF.cbl	Interface repository ID
INTF-ORB.cbl	Interface repository ID
INTF-PRINCIPAL.cbl	Interface repository ID
INTF-REQUEST.cbl	Interface repository ID
INTF-SERVERREQUEST.cbl	Interface repository ID
INTF-TYPECODE.cbl	Interface repository ID
IROBJECT.cbl	IR object reference

TypeCode Related Declarations

TC-ALIAS.cbl	Object standard type code
TC-ANY.cbl	Object standard type code
TC-ARRAY.cbl	Object standard type code
TC-BOA.cbl	Object standard type code

TC-BOOLEAN.cbl	Object standard type code
TC-CHAR-SEQ.cbl	Object standard type code
TC-CHAR.cbl	Object standard type code
TC-COMPLETION-STATUS.cbl	Object standard type code
TC-CONTEXT.cbl	Object standard type code
TC-DOUBLE.cbl	Object standard type code
TC-ENUM.cbl	Object standard type code
TC-ENUMMEMBERSEQ.cbl	Object standard type code
TC-EXCEPT.cbl	Object standard type code
TC-EXCEPTION-TYPE.cbl	Object standard type code
TC-FLAGS.cbl	Object standard type code
TC-FLOAT.cbl	Object standard type code
TC-IDENTIFIER.cbl	Object standard type code
TC-IDLTYPE.cbl	Object standard type code
TC-IMPLEMENTATIONDEF.cbl	Object standard type code
TC-INTERFACEDEF.cbl	Object standard type code
TC-LONG.cbl	Object standard type code
TC-NAMEDVALUE.cbl	Object standard type code
TC-NULL.cbl	Object standard type code
TC-NVLIST.cbl	Object standard type code
TC-OBJECT.cbl	Object standard type code
TC-OCTET-SEQ.cbl	Object standard type code
TC-OCTET.cbl	Object standard type code
TC-OPERATIONDEF.cbl	Object standard type code
TC-ORB-INVALIDNAME.cbl	Object standard type code
TC-ORB-OBJECTID.cbl	Object standard type code
TC-ORB-OBJECTIDLIST.cbl	Object standard type code
TC-ORB.cbl	Object standard type code
TC-ORBSTATUS.cbl	Object standard type code
TC-PRINCIPAL.cbl	Object standard type code
TC-REFERENCEDATA.cbl	Object standard type code
TC-REPOSITORYID-SEQ.cbl	Object standard type code
TC-REPOSITORYID.cbl	Object standard type code
TC-REQUEST-SEQ.cbl	Object standard type code
TC-REQUEST.cbl	Object standard type code
TC-SEQUENCE-CHAR.cbl	Object standard type code
TC-SEQUENCE-OCTET.cbl	Object standard type code
TC-SEQUENCE-REQUEST.cbl	Object standard type code
TC-SEQUENCE-STRING.cbl	Object standard type code
TC-SEQUENCE-STRUCTMEMBER.cbl	Object standard type code

TC-SEQUENCE-UNIONMEMBER.cbl	Object standard type code
TC-SEQUENCE.cbl	Object standard type code
TC-SERVERREQUEST.cbl	Object standard type code
TC-SHORT.cbl	Object standard type code
TC-STEXCEP-BAD-INV-ORDER.cbl	Object standard type code
TC-STEXCEP-BAD-OPERATION.cbl	Object standard type code
TC-STEXCEP-BAD-PARAM.cbl	Object standard type code
TC-STEXCEP-BAD-TYPECODE.cbl	Object standard type code
TC-STEXCEP-CODESET-INCOM.cbl	Object standard type code
TC-STEXCEP-COMM-FAILURE.cbl	Object standard type code
TC-STEXCEP-CONTEXT.cbl	Object standard type code
TC-STEXCEP-DATA-CONVERSION.cbl	Object standard type code
TC-STEXCEP-FREE-MEM.cbl	Object standard type code
TC-STEXCEP-IMP-LIMIT.cbl	Object standard type code
TC-STEXCEP-INITIALIZE.cbl	Object standard type code
TC-STEXCEP-INTERNAL.cbl	Object standard type code
TC-STEXCEP-INTF-REPOS.cbl	Object standard type code
TC-STEXCEP-INV-FLAG.cbl	Object standard type code
TC-STEXCEP-INV-IDENT.cbl	Object standard type code
TC-STEXCEP-INV-OBJREF.cbl	Object standard type code
TC-STEXCEP-INV-POLICY.cbl	Object standard type code
TC-STEXCEP-MARSHAL.cbl	Object standard type code
TC-STEXCEP-NO-IMPLEMENT.cbl	Object standard type code
TC-STEXCEP-NO-MEMORY.cbl	Object standard type code
TC-STEXCEP-NO-PERMISSION.cbl	Object standard type code
TC-STEXCEP-NO-RESOURCES.cbl	Object standard type code
TC-STEXCEP-NO-RESPONSE.cbl	Object standard type code
TC-STEXCEP-OBJ-ADAPTER.cbl	Object standard type code
TC-STEXCEP-PERSIST-STORE.cbl	Object standard type code
TC-STEXCEP-REBIND.cbl	Object standard type code
TC-STEXCEP-TIMEOUT.cbl	Object standard type code
TC-STEXCEP-TRANSACTION-M.cbl	Object standard type code
TC-STEXCEP-TRANSACTION-U.cbl	Object standard type code
TC-STEXCEP-TRANSIENT.cbl	Object standard type code
TC-STEXCEP-UNKNOWN.cbl	Object standard type code
TC-STRING.cbl	Object standard type code
TC-STRUCT.cbl	Object standard type code
TC-STRUCTMEMBER.cbl	Object standard type code
TC-STRUCTMEMBERSEQ.cbl	Object standard type code
TC-TCKIND.cbl	Object standard type code

TC-TYPECODE-BADKIND.cbl	Object standard type code
TC-TYPECODE-BOUNDS.cbl	Object standard type code
TC-TYPECODE.cbl	Object standard type code
TC-ULONG.cbl	Object standard type code
TC-UNION.cbl	Object standard type code
TC-UNIONMEMBER.cbl	Object standard type code
TC-UNIONMEMBERSEQ.cbl	Object standard type code
TC-USHORT.cbl	Object standard type code
TC-VOID.cbl	Object standard type code

Others

ALIASDEF.cbl	AliasDef object reference
ARRAYDEF.cbl	AliasDef object reference
ATTRDESCRIPTIONSEQ.cbl	AttributeDef structure sequence
ATTRIBUTEDEF.cbl	AttributeDef object reference
ATTRIBUTEDESCRIPTION.cbl	AttributeDef information structure
ATTRIBUTEMODE.cbl	AttributeDef information structure attribute
BOA.cbl	BOA object reference
CDR.cbl	CDR information structure
CHAR-SEQ.cbl	Char type sequence
CONSTANTDEF.cbl	ConstantDef object reference
CONSTANTDESCRIPTION.cbl	ConstantDef information structure
CONTAINED-DESCRIPTION.cbl	Contained object information structure
CONTAINED.cbl	Contained object
CONTAINEDSEQ.cbl	Contained object sequence
CONTAINER-DESCRIPTION.cbl	Contained object information structure
CONTAINER-DESCRIPTIONSEQ.cbl	Contained object information structure sequence
CONTAINER.cbl	Contained object reference
CONTEXT.cbl	Context object reference
CONTEXTIDENTIFIER.cbl	Context ID
CONTEXTIDSEQ.cbl	Context ID sequence
DEFINITIONKIND.cbl	Object type Enum
ENUMDEF.cbl	EnumDef object reference
ENUMMEMBERSEQ.cbl	Enum member sequence
FIXEDDEF.cbl	FixedDef object reference
IDENTIFIER.cbl	ID name
IDLTYPE.cbl	IDLType object reference
IMPLEMENTATIONDEF.cbl	ImplementationDef object reference
MARSHAL.cbl	Exception
MODULEDEF.cbl	ModuleDef object reference

MODULEDESCRIPTION.cbl	ModuleDef information structure
NAMEDVALUE.cbl	NamedValue structure
NVLIST.cbl	NVList type
OBJECTID.cbl	ID indicating an object
OBJECTIDLIST.cbl	List of IDs indicating an object
OBJECT-NIL.cbl	CORBA-OBJECT-NIL object
OCTET-SEQ.cbl	CORBA_Octet type sequence
OPDESCRIPTIONSEQ.cbl	Operation information structure sequence
OPERATIONDEF.cbl	Operation object reference
OPERATIONDESCRIPTION.cbl	Operation information structure
OPERATIONMODE.cbl	Operation attribute type
ORB.cbl	ORB object reference
PARAMETERDESCRIPTION.cbl	Parameter information structure
PARAMETERMODE.cbl	Parameter attribute type
PARDESCRIPTIONSEQ.cbl	Parameter information structure sequence
PRIMITIVEDEF.cbl	PrimitiveDef object reference
PRIMITIVEKIND.cbl	Primitive object reference type information
PRINCIPAL.cbl	Principal object reference
REFERENCEDATA.cbl	Reference data
REPLACE.cbl	Used at inheritance
REPOSITORY.cbl	Repository object reference
REPOSITORYID.cbl	ID indicating a repository object
REPOSITORYIDSEQ.cbl	Sequence of ID indicating a repository object
REQUEST-SEQ.cbl	Request object sequence
REQUEST.cbl	Request object reference
SCOPEDNAME.cbl	Scope name
SEQUENCE.cbl	CORBA-sequence type
SEQUENCEDEF.cbl	SequenceDef object reference
SERVERREQUEST.cbl	Server request object reference
STRINGDEF.cbl	StringDef object reference
STRUCTDEF.cbl	StructDef object reference
STRUCTMEMBER.cbl	Struct member information structure
STRUCTMEMBERSEQ.cbl	Struct member information structure sequence
TCKIND.cbl	Type code type enum
TYPEDEFDEF.cbl	TypeDef object reference
TYPEDESCRIPTION.cbl	TypeDef information structure
UNIONDEF.cbl	UnionDef object reference
UNIONMEMBER.cbl	Union object member information structure
UNIONMEMBERSEQ.cbl	Union object member information structure sequence
VERSIONSPEC.cbl	Version information

WSTRINGDEF.cbl	WStringDef object reference

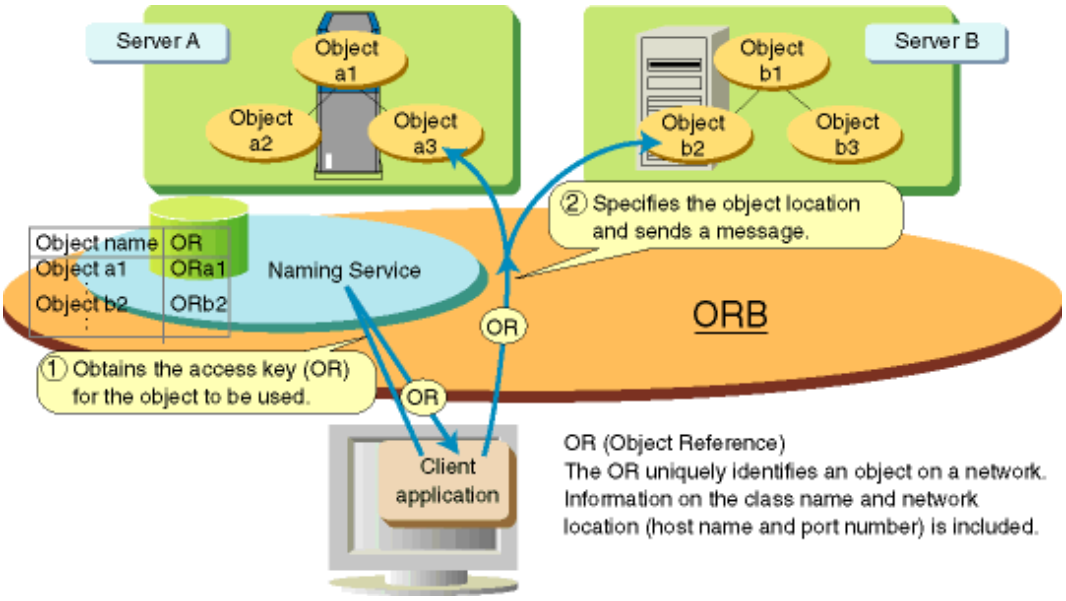
Chapter 8 Naming Service Programming

This chapter explains the API (Application Programming Interface) and the programming that Naming Service provides.

8.1 Naming Service Overview

The Naming Service manages object references by associating them with logical names called binding names. Client applications do not need to include the contents of server application object references in programs, thus enabling the latest server object references to be obtained using logical names. Refer to the following figure for an overview of the Naming Service.

Figure 8.1 Naming Service Structure



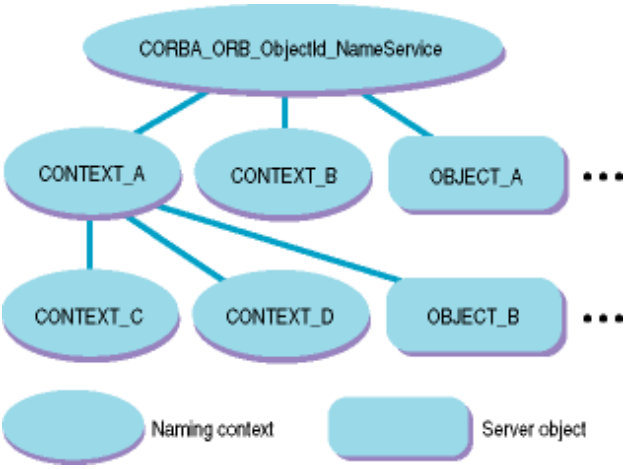
The Naming Service manages the following three types of object references as shown in the following figure:

- Ordinary object references indicating server applications
- Naming context

  A container (equivalent to a file system directory) for managing object references or other naming contexts
- Load balance object groups (*1)

1. *1 This is not valid for Linux (64 bit).

Figure 8.2 Naming Service Object References



8.1.1 Naming Service Interfaces

The Naming Service offers the interfaces shown in the following code sample.

```
module CosNaming {
  typedef string   Istring;
  struct   NameComponent{
      Istring   id;
      Istring   kind;
   };

   typedef   sequence <NameComponent> Name;

   enum   BindingType{
      nobject,
      ncontext
   };

   struct Binding{
      Name      binding_name;
      BindingType   binding_type;
   };

   typedef   sequence <Binding> BindingList;

   interface BindingIterator;

   interface NamingContext{

      enum NotFoundReason {
         missing_node, not_context,
         not_object
      };

      exception NotFound{
         NotFoundReason    why;
         Name       rest_of_name;
      };

      exception CannotProceed{
         NamingContext   cxt;
```

```
      Name      rest_of_name;
   };
   exception InvalidName{};
   exception AlreadyBound{};
   exception NotEmpty{};

   void bind(    in    Name   n,
         in   Object   obj )
      raises( NotFound, CannotProceed, InvalidName,
         AlreadyBound );

   void   rebind(   in    Name   n,
         in   Object   obj )
      raises( NotFound, CannotProceed, InvalidName );

   void   bind_context(
         in    Name      n,
         in   NamingContext   nc )
      raises( NotFound, CannotProceed, InvalidName,
         AlreadyBound );

   void   rebind_context(
         in    Name      n,
         in   NamingContext   nc )
      raises( NotFound, CannotProceed, InvalidName );

   Object   resolve(
         in   Name   n )
      raises( NotFound, CannotProceed, InvalidName );

   void   unbind(   in   Name   n )
      raises( NotFound, CannotProceed, InvalidName );

   NamingContext   new_context();

   NamingContext   bind_new_context(
         in   Name   n )
      raises( NotFound, CannotProceed, InvalidName,
         AlreadyBound );

   void   destroy()
      raises( NotEmpty );

   void   list(   in   unsigned long   how_many,
         out   BindingList   bl,
         out    BindingIterator   bi );
};

interface BindingIterator{
   boolean   next_one(
         out   Binding   b );

   boolean   next_n(   in   unsigned long   how_many,
         out   BindingList   bl );

   void      destroy();
};
  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 );

    };
};
```

The following table describes the interface functions provided by the Naming Service.

Table 8.1 Naming Service Functions

Interface Name	Method Name	Function Explanation
NamingContext	bind	Registers object references in the Naming Service using specified binding names. These relationships are called bindings.
	rebind	Registers object references in the Naming Service using specified binding names. If a binding with the same name already exists, the old binding is deleted and the new binding is registered.
	bind_context	Registers naming context object references in the Naming Service using specified binding names.
	rebind_context	Registers naming context object references in the Naming Service using specified binding names. If a binding with the same name already exists, the old binding is deleted and the new binding is registered.
	resolve	Obtains object references associated with the specified binding names. Specifies structure type binding names as the binding names.
	unbind	Deletes the bindings of specified binding names from the Naming Service.
	new_context	Creates new naming contexts. The created naming context object references are not registered in any naming context.
	bind_new_context	Creates new naming contexts and registers these naming context object references in the Naming Service using specified binding names. (Equivalent to new_context + bind_context)
	destroy	Deletes naming contexts.
	list	Obtains a list of bindings registered in a naming context.
BindingIterator	next_one	Retrieves the next binding within a naming context.

Interface Name	Method Name	Function Explanation
	next_n	Retrieves the next n number of bindings in a naming context.
	destroy	Destroys BindingIterator objects.
NamingContextExt	to_string	Converts structure type binding names to character string binding names.
	to_name	Converts from character string binding names to structure type binding names.
	to_url	Creates URL addresses from the addresses and character string binding names provided.
	resolve_str	Obtains object references associated with the specified binding names. Character string binding names are specified as the binding names.

8.2 Naming Context Interface

This section provides details of the data types and interfaces handled by the naming context interface.

8.2.1 Data Types Handled by the Naming Context Interface

This section provides information on data types handled by the Naming Context Interface.

Name Component

The IDL for the name component is shown below.

```
struct NameComponent {
  Istring id;
  Istring kind;
};
```

When an object reference is registered in a naming context, the name component is registered in a naming context so that it is associated with the object reference.

A name component consists of an id field and a kind field. Each field is represented by a character string and a length of 0 (" ") is also permitted. There are no special rules relating to the use of the id field and kind field. The user may use them without any restrictions.

Since the two name components are identical, the id field and the kind field must correspond.

Structure Type Binding Name

The IDL for a structure type binding name is shown below.

```
typedef sequence <NameComponent> Name;
```

A structure type binding name is a name component sequence type.

To request an operation that spans multiple naming contexts in a hierarchical structure to the Naming Service, specify the binding name to invoke the Naming Service operation.

A binding name consisting of only one name component is called a single name, and a binding name consisting of multiple name components is called a compound name.

In the example shown in the following figure, when a route naming context is requested, the structure type binding name representing object_C will be as shown in the following table.

Figure 8.3 Naming Service Contexts

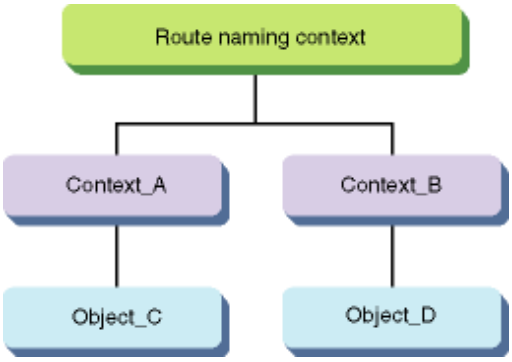


Table 8.2 Structure Type Binding Name

Index	Id	Kind	Remarks
1	"Context_A"	" "	Name component
2	"Object_C"	" "	Name component

Bindings and Binding Lists

Bindings registered in naming contexts are either context type or object type. A name component other than the last name component specified in a compound name must be a context type binding.

A binding type created by bind or rebind is of object type. The binding type created by bind_context, rebind_context, or bind_new_context will be a context type.

The IDL of a binding and binding list is shown below.

```
enum BindingType {nobject, ncontext};

  struct Binding {
    Name binding_name;
    BindingType binding_type;
  };

  typedef sequence <Binding> BindingList;
```

Bindings and binding lists are data types used for storing the binding information that is returned when a list of bindings registered in a naming context is obtained.

A binding consists of a binding name and a binding type, and one item of binding information is stored. A binding is returned by a BindingIterator::next_one operation.

A binding list is a data type used for storing multiple items of binding information, and is a binding sequence type. A binding list is returned by a NamingContext::list operation and a BindingIterator::next_n operation.

8.2.2 Exceptions Generated when the Naming Context Interface is Invoked

In addition to the standard system exceptions, the Naming Service generates the user exceptions shown below.

NotFound

```
  enum NotFoundReason{
      missing_node, not_context, not_object
  };

  exception NotFound {
      NotFoundReason why;
```

```
    Name rest_of_name;
};
```

This exception occurs when bindings corresponding to each name component specified in the binding name do not exist, or when a binding does exist but is different from the requested binding type.

The following values are set in why based on the exception that is generated.

missing_node

The requested binding is not registered in a naming context.

not_context

In processing where a context type binding is expected, an object type binding is registered.

not_object

In processing where an *object type* binding is expected, a *context type* binding is registered.

A residual binding name that has not been processed is set in the rest_of_name member. The initial name component of rest_of_name is the name component that contains a problem.

CannotProceed

```
exception CannotProceed {
    NamingContext cxt;
    Name rest_of_name;
};
```

This exception occurs when processing is terminated before completion as a result of a fault in the Naming Service.

A naming context for which processing has already been performed normally is set in the cxt member.

A residual binding name that has not been processed is set in the rest_of_name member.

Depending on the error content, processing can be re-executed using the returned ctx and rest_of_name.

InvalidName

```
exception InvalidName {};
```

This exception occurs when an invalid binding name is specified.

The same exception occurs when a binding name without a single name component and zero (0) length is specified.

AlreadyBound

```
exception AlreadyBound {};
```

This indicates that an object is already registered in a naming context with the specified binding name.

NotEmpty

```
exception NotEmpty {};
```

This exception occurs when one or more bindings exist in a naming context that is being deleted. A naming context must be empty before it can be deleted.

8.2.3 Creating Bindings

A binding operation associates an object with a binding name and registers it in the naming context. A binding associated with the binding name can be obtained using a resolve operation. The Naming Service supports the following five operations: bind, rebind, bind_context, rebind_context, and bind_new_context. For details of the bind_new_context operation, refer to "8.2.6 Creating Naming Contexts".

```
void bind(in Name n, in Object obj)
    raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
```

```
void rebind(in Name n, in Object obj)
    raises(NotFound, CannotProceed, InvalidName);
void bind_context(in Name n, in NamingContext nc)
    raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
void rebind_context(in Name n, in NamingContext nc)
    raises(NotFound, CannotProceed, InvalidName);
```

bind

Registers an object reference in the Naming Service using a specified binding name. The registered binding is an *object type* binding.

If a binding with the specified name exists, an AlreadyBound exception occurs.

rebind

Registers an object reference in the Naming Service using a specified binding name. If a binding with the specified name already exists, the old binding is deleted and the new binding is registered. The registered binding is an *object type* binding.

If this is already registered with the same binding name, the previous binding must be an *object type*. In other cases, a NotFound exception occurs and not_object is set in why.

bind_context

Registers a naming context object reference in the Naming Service using a specified binding name. The registered binding is a *context type* binding. If a nil reference is specified as the naming context to be registered, a BAD_PARAM system exception occurs.

If a binding with the specified name exists, an AlreadyBound exception occurs.

rebind_context

Registers a naming context object reference in the Naming Service using a specified binding name. If a binding with the specified name already exists, the old binding is deleted and the new binding is registered. The registered binding is a *context type* binding.

If a nil reference is specified as the naming context to be registered, a BAD_PARAM system exception occurs.

If a binding already exists, the previous binding must be a *context type*. In other cases, a NotFound exception occurs and not_context is set in why.

When rebind_context is used, the previously registered *naming context* binding is deleted. This *naming context* must therefore be registered in another *naming context*, or it will not be registered at all. If a *naming context* is no longer required, delete it using the *destroy* operation before executing rebind_context.

8.2.4 Retrieving Bindings

The resolve operation is a process for obtaining an object reference related to a binding name within a given naming context. This binding name must correspond to the binding name registered in the naming context. With the resolve operation, only an object reference is returned; the binding type is not returned.

Because the returned object reference is a CORBA::Object type, the client must perform narrow or cast processing to change the object reference to a suitable type.

```
Object resolve(in Name n )
    raises(NotFound, CannotProceed, InvalidName);
```

A binding name can have multiple components. (Refer to "8.2.1 Data Types Handled by the Naming Context Interface".) The processing of a binding name can span multiple naming contexts. These naming contexts can either be in the same Naming Service instance, or they can be linked between different Naming Service instances.

8.2.5 Deleting Bindings

An unbind operation deletes a specified binding from a naming context.

```
void unbind(in Name n)
    raises(NotFound, CannotProceed, InvalidName);
```

When deleting a context type binding, check that its naming context is registered in another naming context. To delete an unnecessary naming context, use the destroy operation to delete the naming context, then delete the binding using the unbind operation.

8.2.6 Creating Naming Contexts

The Naming Service supports two operations for creating new contexts: new_context and bind_new_context.

```
NamingContext new_context();
NamingContext bind_new_context(in Name n)
    raises(NotFound, AlreadyBound, CannotProceed, InvalidName);
```

new_context

Creates a new naming context. The naming context object reference that is created is not registered in any naming context.

bind_new_context

Creates a new naming context and registers the naming context object reference in the Naming Service using the specified binding name.

This operation is the same as the processing that combines a new_context operation and a bind_context operation. The registered binding is a *context type* binding.

8.2.7 Deleting Naming Contexts

The destroy operation deletes a naming context.

```
void destroy()
    raises(NotEmpty);
```

This operation deletes this naming context. If a binding exists in the context to be deleted, a NotEmpty exception occurs and the naming context is not deleted.

Even if a destroy operation is executed, the binding to which this Naming Service is registered is not deleted from the naming context. To delete an unnecessary naming context, also delete the binding using the unbind operation.

8.2.8 Obtaining Binding Lists

The list operation obtains a list of the bindings registered in a naming context.

```
void list(in unsigned long how_many,
      out BindingList bl, BindingIterator Bi);
```

The list operation returns the bindings contained in a naming context to binding list b1. Binding list b1 is a sequence of bindings showing single name components and containing binding names with a length of 1.

The Naming Service returns bindings requested by the client. The maximum number of bindings is specified in the how_many parameter. If the value specified in how_many is greater than the maximum number of bindings set in the b1_how_many parameter in the nsconfig file, the Naming Service returns a binding with the value specified in the b1_how_many parameter.

If "0" is set in how_many, the client returns the bi parameter to enable the binding to be accessed, and sequence b1 with a list length of zero (0).

If the bi parameter returns a reference other than nil, none of the bindings within the context are returned by invoking the list operation, thus indicating that the remaining bindings must be recovered using Iterator. If the bi parameter returns a nil reference, this means that all the bindings within the naming context are returned by the b1 parameter. The same applies, regardless of the value that is specified in the how_many parameter.

8.3 Binding Iterator Interface

The binding iterator interface is provided to enable the user to obtain a list of the bindings that have not been processed by the list operation.

```
interface BindingIterator {
  boolean next_one(out Binding b);
  boolean next_n(in unsigned long how_many,
        out BindingList bl);
```

```
    void destroy();
  }
```

next_one

The *next_one* operation returns the next binding that has not yet been returned by *list* or the previously invoked *next_n* or *next_one*. If a binding is returned by the operation, *true* is returned as the return value. In the *next_one* operation after all the bindings have been processed, *false* is returned. If *next_one* returns *false*, the binding value will be undefined and cannot be referenced.

If *next_one* is invoked after *false* is returned, an OBJECT_NOT_EXIST system exception occurs.

next_n

The *next_n* operation returns a binding that has not yet been returned by *list* or the previously invoked *next_n* or *next_one* to parameter *b1*. If *b1* is a sequence of non-zero length, *true* is returned. When all the bindings have been collected, *next_n* returns parameter *b1* with zero (0) length only once, and returns *false* as the return value. If *next_one* is invoked after *false* is returned, an OBJECT_NOT_EXIST system exception occurs.

The Naming Service returns bindings requested by the client. The maximum number of bindings is set in the how_many parameter. If the value specified in how_many is greater than the maximum number of bindings set in the b1_how_many parameter in the nsconfig file, the Naming Service returns a binding with the value specified in the b1_how_many parameter.

If 0 is set in how_many, a BAD_PARAM system exception occurs.

destroy

The *destroy* operation destroys this binding iterator. If the client invokes any operation after invoking *destroy*, the operation returns an OBJECT_NOT_EXIST system exception.

A client that creates the binding iterator using a list operation and does not invoke a *destroy* operation may inherit and use resources. A client that has received a returned binding iterator is obliged to delete this binding iterator using a *list* operation.

To prevent resources from being inherited and used for an undestroyed binding iterator, the standard rules of the Naming Service make it possible for the Naming Service to destroy an iterator object at any time without warning. When a highly portable client application is created, it must be assumed that the binding iterator will be destroyed and that an OBJECT_NOT_EXIST system exception will occur.

8.4 Character String Binding Names

A structure type binding name is a name component sequence. Setting and referencing these therefore complicates application processing. Functions to convert the syntax of character string binding names and to convert between character string binding names and structure type binding names are provided by the NamingContextExt interface. (Refer to "8.6 Conversion Between Binding Names, URLs, and IORs".)

In the syntax of character string binding names, the characters "/", ".", and "\" are treated as special characters. The slash ("/") is a name component delimiter, the period (".") is a delimiter for the id and kind fields, and the backslash ("\") is an escape character. (Refer to "8.4.2 Escape Function for Character String Binding Names".)

8.4.1 Basic Notation of Character String Binding Names

A character string binding name consists of name components delimited by slashes ("/"). For example, a binding name consisting of name components "a", "b", and "c" is represented as "a/b/c".

The period (".") is used to delimit the id and kind fields within a name component. For example, the character string binding name "a.b/c.d/." represents the structure type binding name shown in the following table.

Table 8.3 Structure Type Binding Name "a.b/c.d/."

index	id	kind
0	a	b
1	c	d
2	<empty>	<empty>

The name component "." represents a name component with an empty id field and an empty kind field.

If the name components within a character string binding name do not include a period ("."), the entire character string is processed as an id field, and the kind field is considered to be empty. For example, the structure type binding name corresponding to "a/./c.d/.e" is represented in the following table.

Table 8.4 Structure Type Binding Name "a/./c.d/.e"

index	id	kind
0	a	<empty>
1	<empty>	<empty>
2	c	d
3	<empty>	e

If the name component consists of an id field that is not empty and an empty kind field, only the id field needs to be coded in the character string. The last period (".") cannot be used.

8.4.2 Escape Function for Character String Binding Names

A backslash ("\") serves as an escape code for a slash ("/"), period ("."), and backslash ("\") that have reserved meanings within a character string binding name.

Name Component Delimiters

To include a slash ("/") in a name component, use a backslash ("\") as an escape character. For example, the character string binding name "a/x\/y\/z/b" represents a name consisting of components "a", "x/y/z", and "b".

id and *kind* Fields

The backslash escape function is also used to enable a period (".") to be included in the id and kind fields. The character string binding name "a\.b.c\.d/e.f" represents the structure type binding name shown in the following table.

Table 8.5 Structure Type Binding Name "a\.b.c\.d/e.f"

Index	id	kind
0	a.b	c.d
1	e	f

Escape Characters

Backslash ("\") can also be used to include the "\" escape character in a name component. The character string binding name "a/b\\/c" represents a binding name consisting of the components "a", "b\", and "c".

8.5 URL Schemas

This section describes the Uniform Resource Locator (URL) schemas used to code object references that are registered in the initial service or a naming context.

8.5.1 IOR URL Schema

The IOR URL is a CORBA::Object type character string code (IOR:<hex_octets>). This URL format is independent of the Naming Service.

8.5.2 corbaloc URL Schema

A URL schema in IOR format is difficult to identify because the length and the text are encoded. Like the ftp and http URLs, the corbaloc URL schema provides an easy-to-understand URL. This URL is also independent of the Naming Service.

For details of the corbaloc URL coding method, refer to "corbaloc URL Schemas" in the "Obtaining Naming Service Initial References" chapter.

8.5.3 corbaname URL Schema

The corbaname URL is the same as the corbaloc URL, except that a binding name is included in the naming context.

corbaname::nshost/NameService#NC represents a naming context object reference registered as binding name NC, in the naming context represented by the NameService initial service on the nshost.

As with corbaname:nshot, if an object key is not specified, NameService is assumed to be specified as the object key.

corbaname:rir:#NC represents a naming context object reference registered as character string binding name "NC" in the initial naming context setting with no override setting.

Like corbaloc:rir, corbaname:rir: and corbaname:rir:/NameService represent a naming context object reference initial setting in which the initial reference override is not set using an ORBInitRef argument or an ORBDefaultInitRef argument. For details of the ORBInitRef argument and the ORBDefaultInitRef argument, refer to the "Obtaining Naming Service Initial References" chapter.

The syntax of corbaname is as follows:

<corbaname>= "corbaname:"<corbaloc_obj>["#"<string_name>]

<corbaloc_obj>= <obj_addr_list> ["/"<key_string>]

<obj_addr_list>= Refer to "corbaloc URL Schemas" in the "Obtaining Naming Service Initial References" chapter.

<key_string>= Refer to "corbaloc URL Schemas" in the "Obtaining Naming Service Initial References" chapter.

<string_name>= stringfied Name | empty_string

corbaloc_obj:

Specifies a naming context using the same syntax as the corbaloc URL.

obj_addr_list:

Refer to "corbaloc URL Schemas" in the "Obtaining Naming Service Initial References" chapter.

key_string:

Refer to "corbaloc URL Schemas" in the "Obtaining Naming Service Initial References" chapter.

String_name:

This is the character string binding name with URL escape specified below.

corbaname Escape Characters

US-ASCII alphanumeric characters and characters other than those shown below are used as escape characters.

```
      ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" | "$" |
    "," | "-" | "_" | "." | "!" | "~" | "*" | "'" | "(" | ")"
```

corbaname Escape Mechanism

The percent character ("%") is used as an escape character. Characters requiring an escape mechanism within a name component are represented by two hexadecimal digits after "%". The first hexadecimal character represents the upper half-byte, and the second hexadecimal character represents the lower half-byte. If "%" is not followed by two hexadecimal digits, the syntax of the character string binding name will be invalid. Refer to the following table for details.

Table 8.6 Examples of the Escape Mechanism

Stringfied Name	After URL escape	Remarks
a.b/c.d	a.b/c.d	Same URL format
<a>.b/c.d	%3ca%3e.b/c.d	"<" and ">" undergo escape processing.
a.b/ c.d	a.b/%20%20c.d	Two nulls undergo escape processing.
a%b/c%d	a%25b/c%25d	Two % undergo escape processing.
a\\b/c.d	a%5c%5cb/c.d	"\" undergoes escape processing in the stringfied name.

8.6 Conversion Between Binding Names, URLs, and IORs

The NamingContextExt interface, successor to the NamingContext interface, provides operations for using URLs and character string binding names.

to_string

Converts a structure type binding name to a character string binding name.

If the specified structure type binding name is invalid, an InvalidName exception occurs.

to_name

Converts from a character string binding name to a structure type binding name.

If the specified character string binding name is invalid, an InvalidName exception occurs.

resolve_str

Obtains an object reference associated with the specified character string binding name.

to_url

Creates a URL schema from the address and character string binding name provided.

This operation processes all the escape codes required in the parameters from the corbaloc addresses, key parameters, and character string binding names defined in corbaloc URL Schemas in "Obtaining Naming Service Initial References", and returns fully formatted URL character strings. If there is an error in a corbaloc address, key parameter, or character string binding name, an exception occurs.

An empty character string binding name (" ") is valid, but an empty address causes an error. If an empty address is specified, an InvalidAddress exception occurs.

CORBA::ORB::string_to_object

This operation converts the URLs discussed in 8.5 URL Schemas, to object references. If the syntax of the URL schema is invalid, a BAD_PARAM system exception occurs.

8.7 Naming Service Programming Examples

This section contains Naming Service programming examples.

Windows64

OOCOBOL cannot be used.

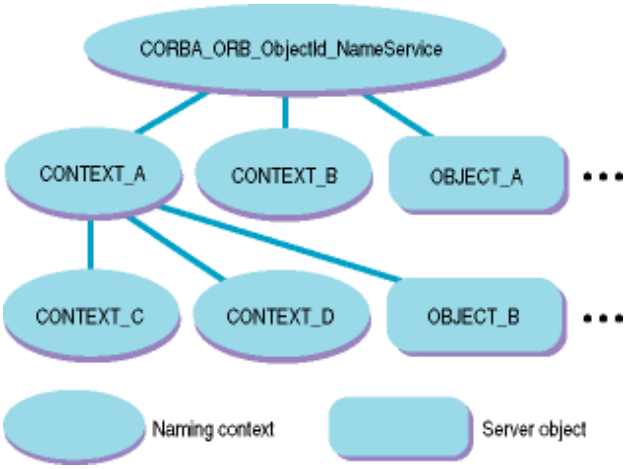
Linux64

OOCOBOL can only be used to create Windows(R) clients.

8.7.1 Retrieving Objects under Contexts

This section describes how the Naming Service retrieves object references when objects have been stored under naming contexts shown in the following figure.

Figure 8.4 Naming Service Example (Retrieval)



Examples of programs that retrieve OBJECT_B are given below.

C Programming Example

Use CORBA_ORB_resolve_initial_references(),which retrieves an ObjectReference, to retrieve Naming Service object references. Specify CORBA_ORB_ObjectId_NameService as a method parameter.

Next, set the pointer of the CosNaming_NameComponent array in the buffer pointer of sequence CosNaming_Name. Then, specify the name of the object reference to be retrieved. As OBJECT_B is placed under CONTEXT_A, specify CONTEXT_A for item 0 and OBJECT_B for item 1 of the name.

Finally, use the Naming Service CosNaming_NamingContext_resolve() method to retrieve the object reference from the Naming Service.

```
CosNaming_NamingContext   cos_naming;
CosNaming_Name            name;
CosNaming_NameComponent   name_component[2];
CORBA_ORB                 orb;
CORBA_Environment         env;
CORBA_Object              obj;
/* Obtain the root (CORBA_ORB_ObjectId_NameService) object. */
orb = CORBA_ORB_init(&current_argc, argv, FJ_OM_ORBid, &env );
cos_naming = CORBA_ORB_resolve_initial_references(orb,
CORBA_ORB_ObjectId_NameService, &env );

/* Obtain OBJECT_B from the root */
   name._length = name._maximum = 2;
   name._buffer = name_component;
   name._buffer[0].id = "CONTEXT_A";
   name._buffer[0].kind = "";
   name._buffer[1].id = "OBJECT_B";
   name._buffer[1].kind = "";

   obj = CosNaming_NamingContext_resolve(
      cos_naming,
      &name,
      &env );
```

C++ Programming Example

Use CORBA::ORB::resolve_initial_references(),which retrieves an ObjectReference, to retrieve Naming Service object references. Specify CORBA_ORB_ObjectId_NameService as a method parameter.

Next, use CosNaming::Name::allocbuf() to allocate location for the buffer pointer (CosNaming::NameComponent_var array) of sequence CosNaming::Name. Specify the name of the object reference to be retrieved. As OBJECT_B is placed under CONTEXT_A, specify the number of arrays to be retrieved as 2, CONTEXT_A for item 0 and OBJECT_B for item 1 of the name. Then, specify the above pointer in a parameter and use operator new to create the sequence.

Finally, use the Naming Service CosNaming::NamingContext::resolve() method to retrieve the object reference from the Naming Service.

```
CORBA::ORB_ptr                orb;
CosNaming::Name_ptr           name;
CosNaming::NameComponent_var  *name_component;
CORBA::Object_ptr             obj;
CORBA::Environment            env;

try {
      // Obtain the root (CORBA_ORB_ObjectId_NameService) object.
      orb = CORBA::ORB_init( argc, argv, FJ_OM_ORBid, env );
      obj = orb->resolve_initial_references(
            CORBA_ORB_ObjectId_NameService,env );
      CosNaming::NamingContext_ptr
         NamingContext_obj = CosNaming::NamingContext::_narrow( obj );

      CORBA::release( obj );

      // Obtain OBJECT_B from the root.
      name_component_var =    CosNaming::Name::allocbuf(2);
      name_component[0]->id = (const CORBA::Char *)"CONTEXT_A";
      name_component[0]->kind = (const CORBA::Char *)"";
      name_component[1]->id = (const CORBA::Char *)"OBJECT_B";
      name_component[1]->kind = (const CORBA::Char *)"";
      name = new CosNaming::Name(2,2,name_component_var,CORBA_TRUE);

      OPNAME("NamingContext::resolve");
      obj = NamingContext_obj->resolve( *name, env );
      CORBA::release( NamingContext_obj );
      delete name;
   }
   catch ( CORBA::Exception &e ){
      ... // Error processing
   }
```

Java Programming Example

Use org.omg.CORBA.ORB.resolve_initial_references(),which retrieves an ObjectReference, to retrieve Naming Service object references. Specify NameService as a method parameter.

Next, use new to allocate the location of the sequence org.omg.CosNaming.NameComponent[].

Specify the name in value. As OBJECT_B is placed under CONTEXT_A, specify CONTEXT_A for item 0, OBJECT_B for item 1, and null for item 2.

Finally, use the Naming Service org.omg.CosNaming.NamingContext.resolve() method to obtain the object reference from the Naming Service.

```
public class Resolve {
  public static void main(String args[]){
    try {
     int     i, j;

     org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(arg,null);

      //Obtain the root (NameService) object.
      org.omg.CORBA.Object obj =
            orb.resolve_initial_references("NameService");
      org.omg.CosNaming.NamingContextExt cos_naming =
            org.omg.CosNaming.NamingContextExtHelper.narrow(obj);

      // Obtain OBJECT_B from the root.
      org.omg.CosNaming.NameComponent [] name =
            new org.omg.CosNaming.NameComponent[2];
```

```
      name[0] = new org.omg.CosNaming.NameComponent( "CONTEXT_A", "");
      name[1] = new org.omg.CosNaming.NameComponent( "OBJECT_B", "");
      obj = cos_naming.resolve( name );

    }
    catch( Exception e ){
        ... // Error processing
    }
  }
}
```

COBOL Programming Example

Use CORBA-ORB-RESOLVE-INITIAL-REFERENCES, which retrieves an ObjectReference, to retrieve Naming Service object references. Specify CORBA-ORB-OBJECTID-NAMESERVICE as a method parameter.

Next, the name of the object reference which it wants to take out in COSNAMING-NAME and COSNAMING-NAMECOMPONENT is specified as the following program. As OBJECT_B is placed under CONTEXT_A, specify CONTEXT_A for item 0 and OBJECT_B for item 1 of the name.

Finally, use the Naming Service COSNAMING-NAMINGCONTEXT-RESOLVE method to retrieve the object reference from the Naming Service.

IDENTIFICATION DIVISION.

```
  PROGRAM-ID. "RESOLVE-TEST".
```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.

```
      ARGUMENT-NUMBER      IS  ARG-C
      ARGUMENT-VALUE       IS  ARG-V
      SYMBOLIC CONSTANT
      COPY SYMBOL-CONST IN CORBA.
      .
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
  COPY CONST  IN CORBA.
  01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.

  *########## NAMING SETTING PARAMETER #########
  01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING COSNAMING-NAMINGCONTEXT
                                                          BY COS-NAMING.
  01 COPY COSNAMING-NAME IN CORBA REPLACING COSNAMING-NAME BY NAME.
  01 NAME-A USAGE POINTER.
  01 COPY COSNAMING-NAMECOMPONENT IN CORBA REPLACING COSNAMING-NAMECOMPONENT
                                                    BY NAME-COMPONENT.
  01 NAME-COMPONENT-A USAGE POINTER.

  *########## ORB SETTING PARAMETER #########
  01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY CURRENT-ARG-C.
  01 CURRENT-ARG-V.
     02 FILLER OCCURS 6.
        03 CURRENT-ARG-V-VALUE USAGE POINTER.
  01 APLI-NAME PIC X(8) VALUE "simple_c".
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY ARG-COUNT.
  01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.

  01 TMP-STRING       USAGE POINTER.
```

```
01 TMP-STRING-BUF  PIC X(50).
01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY ELEMENT-NUMBER.
*
```

PROCEDURE DIVISION.

```
* ObjectDirector Initialization
   ACCEPT CURRENT-ARG-C FROM ARG-C.
    COMPUTE CURRENT-ARG-C = CURRENT-ARG-C + 1.
    PERFORM VARYING ARG-COUNT FROM 1 BY 1 UNTIL ARG-COUNT > CURRENT-ARG-C
       IF ARG-COUNT = 1
          MOVE APLI-NAME TO TMP-STRING-BUF
        ELSE
           ACCEPT TMP-STRING-BUF FROM ARG-V
        END-IF
        MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH
        CALL "CORBA-STRING-SET" USING
              CURRENT-ARG-V-VALUE (ARG-COUNT)
              STRING-LENGTH
              TMP-STRING-BUF
     END-PERFORM.
     SET CURRENT-ARG-V-VALUE (ARG-COUNT) TO NULL.

     MOVE FUNCTION LENG(FJ-OM-ORB-ID) TO STRING-LENGTH.
     CALL  "CORBA-STRING-SET" USING
           TMP-STRING
           STRING-LENGTH
           FJ-OM-ORB-ID.
     CALL "CORBA-ORB-INIT" USING
           CURRENT-ARG-C
           CURRENT-ARG-V
           TMP-STRING
           ENV
           ORB.
     CALL "CORBA-FREE" USING TMP-STRING.

* Obtain the root (CORBA-ORB-OBJECTID-NAMESERVICE) object
    MOVE FUNCTION LENG ( CORBA-ORB-OBJECTID-NAMESERVICE ) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TMP-STRING
           STRING-LENGTH
           CORBA-ORB-OBJECTID-NAMESERVICE.
     CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
           ORB
           TMP-STRING
           ENV
           COS-NAMING.
     CALL "CORBA-FREE" USING TMP-STRING.

* Obtain OBJECT_B from the root.
     MOVE 2 TO SEQ-LENGTH OF NAME.
     MOVE 2 TO SEQ-MAXIMUM OF NAME.
     CALL "CORBA-SEQUENCE-COSNAMING-NAMECOMPONENT-ALLOCBUF" USING
           SEQ-MAXIMUM OF NAME
           SEQ-BUFFER OF NAME.

     MOVE "CONTEXT_A" TO TMP-STRING-BUF.
     MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           IDL-ID OF NAME-COMPONENT
           STRING-LENGTH
           TMP-STRING-BUF.
```

```
      MOVE " " TO TMP-STRING-BUF.
      CALL "CORBA-STRING-SET" USING
            KIND OF NAME-COMPONENT
            STRING-LENGTH
            TMP-STRING-BUF.

      MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
      MOVE 1 TO ELEMENT-NUMBER.
      MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
      CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
            NAME-A
            ELEMENT-NUMBER
            NAME-COMPONENT-A.

      MOVE "OBJECT_B" TO TMP-STRING-BUF.
      MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH.
      CALL "CORBA-STRING-SET" USING
            IDL-ID OF NAME-COMPONENT
            STRING-LENGTH
            TMP-STRING-BUF.

      MOVE " " TO TMP-STRING-BUF.
      CALL "CORBA-STRING-SET" USING
            KIND OF NAME-COMPONENT
            STRING-LENGTH
            TMP-STRING-BUF.

      MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
      MOVE 2 TO ELEMENT-NUMBER.
      MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
      CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
            NAME-A
            ELEMENT-NUMBER
            NAME-COMPONENT-A.

      CALL "COSNAMING-NAMINGCONTEXT-RESOLVE" USING
            COS-NAMING
            NAME
            ENV
            OBJ.
      IF OBJ = NULL
            CALL "CORBA-STRING-GET" USING
                      IDL-ID OF ENV
                      STRING-LENGTH
                      TMP-STRING-BUF
            DISPLAY "ENV-ID : " TMP-STRING-BUF
      END-IF
      EXIT PROGRAM.
  END PROGRAM "RESOLVE-TEST".
```

OOCOBOL Programming Example

Use CORBA-ORB-RESOLVE_INITIAL_REFERENCES, which retrieves an ObjectReference, to retrieve Naming Service object references. Specify CORBA-ORB-OBJECTID_NAMESERVICE as a method parameter.

Next, the name of the object reference which it wants to take out in COSNAMING-NAME and COSNAMING-NAMECOMPONENT is specified as the following program. As OBJECT_B is placed under CONTEXT_A, specify CONTEXT_A for item 0 and OBJECT_B for item 1 of the name.

Finally, use the Naming Service COSNAMING-NAMINGCONTEXT-RESOLVE method to retrieve the object reference from the Naming Service.

IDENTIFICATION DIVISION.

```
 PROGRAM-ID.            "CLIENT-MAIN".
*
```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.

```
 REPOSITORY.
     COPY               CORBA--REP.
     COPY               COSNAMING--REP.
     .
```

SPECIAL-NAMES.

```
     SYMBOLIC CONSTANT
     COPY               CORBA--CONST.
     COPY               COSNAMING--CONST.
     .
*
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
 COPY                   CORBA--COPY.
 COPY                   COSNAMING--COPY.
*
 01 ORB                 USAGE OBJECT REFERENCE CORBA-ORB.
 01 BOA                 USAGE OBJECT REFERENCE CORBA-BOA.
 01 IMPL-REP            USAGE OBJECT REFERENCE FJ-IMPLEMENTATIONREP.
 01 IMPL                USAGE OBJECT REFERENCE CORBA-IMPLEMENTATIONDEF.
 01 INTF-REP            USAGE OBJECT REFERENCE CORBA-REPOSITORY.
 01 INTF                USAGE OBJECT REFERENCE CORBA-INTERFACEDEF.
 01 OBJ                 USAGE OBJECT REFERENCE CORBA-OBJECT.
 01 NAMING-CONTEXT      USAGE OBJECT REFERENCE COSNAMING-NAMINGCONTEXT.
 01 NAME                TYPE                   COSNAMING-NAME.
 01 NAME-COMPONENT      USAGE OBJECT REFERENCE COSNAMING-NAMECOMPONENT.
 01 NAME-ID             USAGE OBJECT REFERENCE CORBA-STRING.
 01 NAME-KIND           USAGE OBJECT REFERENCE CORBA-STRING.
*
 01 EXCEPTION-ID        USAGE OBJECT REFERENCE CORBA-STRING.
 01 EXCEPTION-ID-VALUE  PIC   X(50).
 01 API-NAME            PIC   X(50).
*
 01 I                   TYPE                   CORBA-UNSIGNED-LONG.
*
```

PROCEDURE DIVISION.

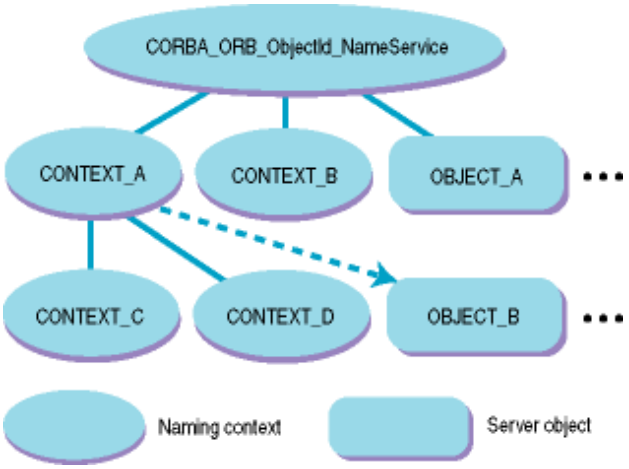
```
 DECLARATIVES.
*
 OTHER-ERROR SECTION.
     USE EXCEPTION CORBA-EXCEPTION.
     DISPLAY "CORBA::Exception: " API-NAME.
     SET EXCEPTION-ID TO IDL-ID OF EXCEPTION-OBJECT AS CORBA-EXCEPTION.
     INVOKE EXCEPTION-ID "GET-VALUE" RETURNING EXCEPTION-ID-VALUE.
     DISPLAY "   Exception-id: " EXCEPTION-ID-VALUE.
     EXIT PROGRAM.
 END-OTHER-ERR.
*
 END DECLARATIVES.
*
 MAIN SECTION.
*
     MOVE "CORBA::ORB_init" TO API-NAME.
     INVOKE CORBA "ORB_INIT"
```

```
                        USING     "Naming"
                                  FJ-OM_ORBID
                        RETURNING ORB.
*
     MOVE "CORBA::ORB::BOA_init" TO API-NAME.
     INVOKE ORB "BOA_INIT"
                        USING     "Naming"
                                  CORBA-BOA_OAID
                        RETURNING BOA.
*
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                        USING     CORBA-ORB-OBJECTID_NAMESERVICE
                        RETURNING OBJ.
*
     INVOKE COSNAMING-NAMINGCONTEXT "NARROW"
                        USING     OBJ
                        RETURNING NAMING-CONTEXT.
*
     INVOKE SEQUENCE-NAMECOMPONENT-001 "NEW-WITH-LENGTH"
                        USING 2 RETURNING NAME.
     INVOKE CORBA-STRING "NEW-WITH-VALUE" USING "CONTEXT_A" RETURNING
NAME-ID
     INVOKE CORBA-STRING "NEW-WITH-VALUE" USING " " RETURNING NAME-KIND
     INVOKE COSNAMING-NAMECOMPONENT "NEW" RETURNING NAME-COMPONENT
     SET IDL-ID OF NAME-COMPONENT TO NAME-ID
     SET KIND OF NAME-COMPONENT TO NAME-KIND
     INVOKE NAME "SET-VALUE" USING 1 NAME-COMPONENT
*
     INVOKE CORBA-STRING "NEW-WITH-VALUE" USING "OBJECT_B" RETURNING NAME-ID
     INVOKE CORBA-STRING "NEW-WITH-VALUE" USING " " RETURNING NAME-KIND
     INVOKE COSNAMING-NAMECOMPONENT "NEW" RETURNING NAME-COMPONENT
     SET IDL-ID OF NAME-COMPONENT TO NAME-ID
     SET KIND OF NAME-COMPONENT TO NAME-KIND
     INVOKE NAME "SET-VALUE" USING 2 NAME-COMPONENT
*
     MOVE "CosNaming::NamingContext::resolve" TO API-NAME.
     INVOKE NAMING-CONTEXT "RESOLVE"
                        USING     NAME
                        RETURNING OBJ.
*
     STOP RUN.
*
 END-MAIN.
*
 END PROGRAM "CLIENT-MAIN".
```

8.7.2 Registering Objects under Contexts

This section describes how the Naming Service registers object references when objects have been stored under naming contexts shown in the following figure.

Figure 8.5 Naming Service Example (Registering)



Examples of programs that register OBJECT_B are given below.

C Programming Example

Use CORBA_ORB_resolve_initial_references(),which retrieves an ObjectReference, to retrieve Naming Service object references. Specify CORBA_ORB_ObjectId_NameService as a method parameter.

Next, set the pointer of the CosNaming_NameComponent array in the buffer pointer of sequence CosNaming_Name. Then, specify the name of the object reference to be retrieved. As OBJECT_B is placed under CONTEXT_A, specify the number of arrays to be retrieved as 2 and CONTEXT_A for item 0 and OBJECT_B for item 1 of the name.

Finally, use the Naming Service CosNaming_NamingContext_bind() method to register the object reference in the Naming Service.

```
   CORBA_ORB            orb;
   CosNaming_NamingContext      cos_naming;
   CosNaming_Name               name;
   CosNaming_NameComponent      name_component   [2];
   CORBA_Environment            env;

   CORBA_Object                 obj;

   /* Obtain the root (CORBA_ORB_ObjectId_NameService) object. */
   orb = CORBA_ORB_init(&current_argc, argv, FJ_OM_ORBid, &env );
   cos_naming = CORBA_ORB_resolve_initial_references(
            orb,
            CORBA_ORB_ObjectId_NameService,
            &env );

   obj = ...; /* Use any method to create and obtain Object. */

/* Set up the sequence area of OBJECT_B. */
   name._length = name._maximum = 2;
   name._buffer = name_component;
   name_component[0].id = "CONTEXT_A";
   name_component[0].kind = "";
   name_component[1].id = "OBJECT_B";
   name_component[1].kind = "";

   /* Register the object reference in the Naming Service. */
   CosNaming_NamingContext_bind(
      cos_naming,
      &name,
      obj,
      &env );
```

C++ Programming Example

Use CORBA::ORB::resolve_initial_references(),which retrieves an ObjectReference, to retrieve Naming Service object references. Specify CORBA_ORB_ObjectId_NameService as a method parameter.

Next, use CosNaming::Name::allocbuf() to allocate the location of the buffer pointer (CosNaming::NameComponent_var array) of sequence CosNaming::Name. Specify the name of the object reference to be retrieved. AS OBJECT_B is placed under CONTEXT_A, specify 2 as the number of arrays to be retrieved and CONTEXT_A for item 0 and OBJECT_B as item 1 of the name. Then, specify the above pointer in the parameter and use operator new to create the sequence.

Finally, use the Naming Service CosNaming::NamingContext::bind() method to register the object reference in the Naming Service.

```
CORBA::ORB_ptr                orb;
CosNaming::Name_ptr           name;
CosNaming::NameComponent_var  *name_component;
CORBA::Environment            env;

CORBA::Object_ptr             obj;

try {
      //Obtain the root (CORBA_ORB_ObjectId_NameService) object.
      orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, env );

      OPNAME("resolve_initial_references");
      CORBA::Object_ptr
      obj = orb->resolve_initial_references(
            CORBA_ORB_ObjectId_NameService, env );

      CosNaming::NamingContext_ptr
      NamingContext_obj = CosNaming::NamingContext::_narrow( obj );
      CORBA::release( obj );

      obj = ...; //Use any method to create and obtain Object.

      //Set up the sequence area of OBJECT_B.
      name_component = CosNaming::Name::allocbuf(2);
      name_component[0]->id = (const CORBA::Char *)"CONTEXT_A";
      name_component[0]->kind = (const CORBA::Char *)"";
      name_component[1]->id = (const CORBA::Char *)"OBJECT_B";
      name_component[1]->kind = (const CORBA::Char *)"";
      name = new CosNaming::Name(2, 2, name_component,CORBA_TRUE);

      //Register the object reference in the Naming Service.
      OPNAME("NamingContext::bind");
      NamingContext_obj->bind( *name, obj, env );
      delete name;

CORBA::release( NamingContext_obj );
   }
   catch ( CORBA::Exception &e ){
      ... //Error processing
   }
```

Java Programming Example

Use org.omg.CORBA.ORB.resolve_initial_references(), which retrieves an ObjectReference, to retrieve Naming Service object references. Specify NameService as a method parameter.

Next, use new to retrieve the location of sequence org.omg.CosNaming.NameComponent[].

Specify the name in value. As OBJECT_B is placed under CONTEXT_A, specify CONTEXT_A for item 0, OBJECT_B for item 1, and null for item 2.

Finally, use the Naming Service org.omg.CosNaming.NamingContext.bind() method to register the object reference in the Naming Service.

```
public class Bind {
   public static void main(String args[]){
     try {
         //Obtain the root (NameService) object.
         org.omg.CORBA.ORB orb =  org.omg.CORBA.ORB.init(arg,null);

         org.omg.CORBA.Object obj =
                 orb.resolve_initial_references("NameService");
         org.omg.CosNaming.NamingContextExt cos_naming =
                 org.omg.CosNaming.NamingContextExtHelper.narrow(obj);

         obj = ...; //Use any method to create and obtain Object.

         //Set up the sequence area of OBJECT_B.
         org.omg.CosNaming.NameComponent [] name =
         new org.omg.CosNaming.NameComponent[2];
         name[0] = new org.omg.CosNaming.NameComponent("CONTEXT_A","");
         name[1] = new org.omg.CosNaming.NameComponent( "OBJECT_B", "");

         //Register the object reference in the Naming Service.
         cos_naming.bind( name, obj );

      }
      catch( Exception e ){
          ...//Error processing
      }
   }
}
```

COBOL Programming Example

Use CORBA-ORB-RESOLVE-INITIAL-REFERENCES,which retrieves an ObjectReference, to retrieve Naming Service object references. Specify CORBA-ORB-OBJECTID-NAMESERVICE as a method parameter.

Next, the name of the object reference which it wants to take out in COSNAMING-NAME and COSNAMING-NAMECOMPONENT is specified as the following program. As OBJECT_B is placed under CONTEXT_A, specify the number of arrays to be retrieved as 2 and CONTEXT_A for item 0 and OBJECT_B for item 1 of the name.

Finally, use the COSNAMING-NAMINGCONTEXT-BIND method to register the object reference in the Naming Service.

IDENTIFICATION DIVISION.

```
   PROGRAM-ID. "BIND-TEST".
```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.

```
        ARGUMENT-NUMBER      IS  ARG-C
        ARGUMENT-VALUE       IS  ARG-V
        SYMBOLIC CONSTANT
        COPY SYMBOL-CONST IN CORBA.
        .
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
   COPY CONST  IN CORBA.
   01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
   01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
   01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.

  *########## NAMING SETTING PARAMETER #########
   01 COPY COSNAMING-NAMINGCONTEXT IN CORBA
```

```
              REPLACING COSNAMING-NAMINGCONTEXT BY COS-NAMING.
 01 COPY COSNAMING-NAMINGCONTEXT IN CORBA
              REPLACING COSNAMING-NAMINGCONTEXT BY COS-NAMING2.
 01 COPY COSNAMING-NAME IN CORBA REPLACING COSNAMING-NAME BY NAME.
 01 NAME-A USAGE POINTER.
 01 COPY COSNAMING-NAMECOMPONENT IN CORBA
              REPLACING COSNAMING-NAMECOMPONENT BY NAME-COMPONENT.
 01 NAME-COMPONENT-A USAGE POINTER.

*########## ORB SETTING PARAMETER #########
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY CURRENT-ARG-C.
 01 CURRENT-ARG-V.
    02 FILLER OCCURS 6.
       03 CURRENT-ARG-V-VALUE USAGE POINTER.
 01 APLI-NAME PIC X(8) VALUE "simple_c".
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY ARG-COUNT.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.

 01 TMP-STRING       USAGE POINTER.
 01 TMP-STRING-BUF  PIC X(50).
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY ELEMENT-NUMBER.
*
```

PROCEDURE DIVISION.

```
 * ObjectDirector Initialization
      ACCEPT CURRENT-ARG-C FROM ARG-C.
      COMPUTE CURRENT-ARG-C = CURRENT-ARG-C + 1.
      PERFORM VARYING ARG-COUNT FROM 1 BY 1 UNTIL ARG-COUNT > CURRENT-ARG-C
         IF ARG-COUNT = 1
            MOVE APLI-NAME TO TMP-STRING-BUF
         ELSE
            ACCEPT TMP-STRING-BUF FROM ARG-V
         END-IF
         MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH
         CALL "CORBA-STRING-SET" USING
               CURRENT-ARG-V-VALUE (ARG-COUNT)
               STRING-LENGTH
               TMP-STRING-BUF
      END-PERFORM.
      SET CURRENT-ARG-V-VALUE (ARG-COUNT) TO NULL.

      MOVE FUNCTION LENG(FJ-OM-ORB-ID) TO STRING-LENGTH.
      CALL  "CORBA-STRING-SET" USING
            TMP-STRING
            STRING-LENGTH
            FJ-OM-ORB-ID.
      CALL "CORBA-ORB-INIT" USING
            CURRENT-ARG-C
            CURRENT-ARG-V
            TMP-STRING
            ENV
            ORB.
      CALL "CORBA-FREE" USING TMP-STRING.

 * Obtain the root (CORBA-ORB-OBJECTID-NAMESERVICE) object
      MOVE FUNCTION LENG ( CORBA-ORB-OBJECTID-NAMESERVICE ) TO STRING-LENGTH.
      CALL "CORBA-STRING-SET" USING
            TMP-STRING
            STRING-LENGTH
            CORBA-ORB-OBJECTID-NAMESERVICE.
      CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
```

```
            ORB
            TMP-STRING
            ENV
            COS-NAMING.
     CALL "CORBA-FREE" USING TMP-STRING.

* Use any method to create and obtain Object.
     MOVE  COS-NAMING TO OBJ.


* Set up the sequence area of OBJECT_B.
     MOVE 2 TO SEQ-LENGTH OF NAME.
     MOVE 2 TO SEQ-MAXIMUM OF NAME.
     CALL "CORBA-SEQUENCE-COSNAMING-NAMECOMPONENT-ALLOCBUF" USING
            SEQ-MAXIMUM OF NAME
            SEQ-BUFFER OF NAME.

     MOVE "CONTEXT_A" TO TMP-STRING-BUF.
     MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
            IDL-ID OF NAME-COMPONENT
            STRING-LENGTH
            TMP-STRING-BUF
            .
     MOVE " " TO TMP-STRING-BUF.
     CALL "CORBA-STRING-SET" USING
            KIND OF NAME-COMPONENT
            STRING-LENGTH
            TMP-STRING-BUF.
     MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
     MOVE 1 TO ELEMENT-NUMBER.
     MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
     CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
            NAME-A
            ELEMENT-NUMBER
            NAME-COMPONENT-A.

     MOVE "OBJECT_B" TO TMP-STRING-BUF.
     MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
            IDL-ID OF NAME-COMPONENT
            STRING-LENGTH
            TMP-STRING-BUF.

     MOVE " " TO TMP-STRING-BUF.
     CALL "CORBA-STRING-SET" USING
            KIND OF NAME-COMPONENT
            STRING-LENGTH
            TMP-STRING-BUF.

     MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
     MOVE 2 TO ELEMENT-NUMBER.
     MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
     CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
            NAME-A
            ELEMENT-NUMBER
            NAME-COMPONENT-A.
* Register the object reference in the Naming Service.
     CALL "COSNAMING-NAMINGCONTEXT-BIND" USING
            COS-NAMING
            NAME
            OBJ
            ENV.
```

```
      EXIT PROGRAM.
  END PROGRAM "BIND-TEST".
```

OOCOBOL Programming Example

Use CORBA-ORB-RESOLVE_INITIAL_REFERENCES, which retrieves an ObjectReference, to retrieve Naming Service object references. Specify CORBA-ORB-OBJECTID_NAMESERVICE as a method parameter.

Next, the name of the object reference which it wants to take out in COSNAMING-NAME and COSNAMING-NAMECOMPONENT is specified as the following program. As OBJECT_B is placed under CONTEXT_A, specify the number of arrays to be retrieved as 2 and CONTEXT_A for item 0 and OBJECT_B for item 1 of the name.

Finally, use the COSNAMING-NAMINGCONTEXT-BIND method to register the object reference in the Naming Service.

IDENTIFICATION DIVISION.

```
  PROGRAM-ID.            "CLIENT-MAIN".
```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.

```
  REPOSITORY.
      COPY               CORBA--REP.
      COPY               COSNAMING--REP.
      .
```

SPECIAL-NAMES.

```
     SYMBOLIC CONSTANT
      COPY               CORBA--CONST.
      COPY               COSNAMING--CONST.
      .
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
  COPY                   CORBA--COPY.
  COPY                   COSNAMING--COPY.
 *
  01 ORB                 USAGE OBJECT REFERENCE CORBA-ORB.
  01 BOA                 USAGE OBJECT REFERENCE CORBA-BOA.
  01 IMPL-REP            USAGE OBJECT REFERENCE FJ-IMPLEMENTATIONREP.
  01 IMPL                USAGE OBJECT REFERENCE CORBA-IMPLEMENTATIONDEF.
  01 INTF-REP            USAGE OBJECT REFERENCE CORBA-REPOSITORY.
  01 INTF                USAGE OBJECT REFERENCE CORBA-INTERFACEDEF.
  01 REF-ID              TYPE                   CORBA-REFERENCEDATA.
  01 OBJ                 USAGE OBJECT REFERENCE CORBA-OBJECT.
  01 NAMING-CONTEXT      USAGE OBJECT REFERENCE COSNAMING-NAMINGCONTEXT.
  01 NAME                TYPE                   COSNAMING-NAME.
  01 NAME-COMPONENT      USAGE OBJECT REFERENCE COSNAMING-NAMECOMPONENT.
  01 NAME-ID             USAGE OBJECT REFERENCE CORBA-STRING.
  01 NAME-KIND           USAGE OBJECT REFERENCE CORBA-STRING.
 *
  01 EXCEPTION-ID        USAGE OBJECT REFERENCE CORBA-STRING.
  01 EXCEPTION-ID-VALUE  PIC   X(50).
  01 API-NAME            PIC   X(50).
 *
  01 I                   TYPE                   CORBA-UNSIGNED-LONG.
```

PROCEDURE DIVISION.

```
 *
  DECLARATIVES.
 *
  OTHER-ERROR SECTION.
```

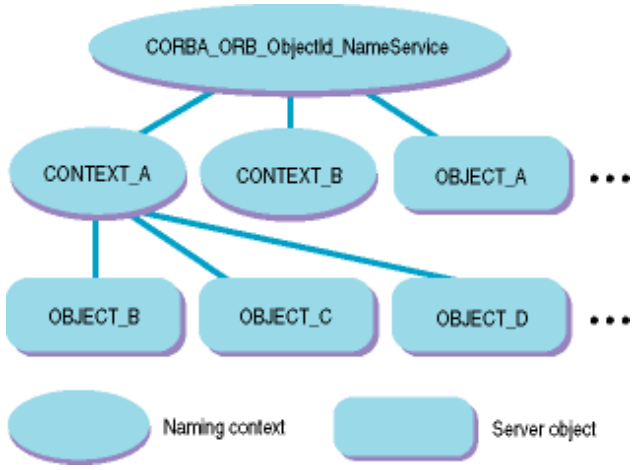
```
     USE EXCEPTION CORBA-EXCEPTION.
     DISPLAY "CORBA::Exception: " API-NAME.
     SET EXCEPTION-ID TO IDL-ID OF EXCEPTION-OBJECT AS CORBA-EXCEPTION.
     INVOKE EXCEPTION-ID "GET-VALUE" RETURNING EXCEPTION-ID-VALUE.
     DISPLAY "   Exception-id: " EXCEPTION-ID-VALUE.
     EXIT PROGRAM.
 END-OTHER-ERR.
*
 END DECLARATIVES.
*
 MAIN SECTION.
*
     MOVE "CORBA::ORB_init" TO API-NAME.
     INVOKE CORBA "ORB_INIT"
                       USING     "Naming"
                                 FJ-OM_ORBID
                       RETURNING ORB.
*
     MOVE "CORBA::ORB::BOA_init" TO API-NAME.
     INVOKE ORB "BOA_INIT"
                       USING     "Naming"
                                 CORBA-BOA_OAID
                       RETURNING BOA.
*
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                       USING     CORBA-ORB-OBJECTID_NAMESERVICE
                       RETURNING OBJ.
*
     INVOKE COSNAMING-NAMINGCONTEXT "NARROW"
                       USING     OBJ
                       RETURNING NAMING-CONTEXT.
*
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                       USING     CORBA-OBJECTID_LIGHTINTERF-001
                       RETURNING OBJ.
     INVOKE CORBA-REPOSITORY "NARROW" USING OBJ RETURNING INTF-REP.
*
     MOVE "CORBA::Repository::lookup_id" TO API-NAME.
     INVOKE INTF-REP "LOOKUP_ID" USING "IDL:OBJECT_B:1.0" RETURNING OBJ.
     INVOKE CORBA-INTERFACEDEF "NARROW" USING OBJ RETURNING INTF.
*
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                       USING     CORBA-OBJECTID_IMPLEMENTAT-001
                       RETURNING OBJ.
     INVOKE FJ-IMPLEMENTATIONREP "NARROW" USING OBJ RETURNING IMPL-REP.
*
     MOVE "FJ::ImplementationRep::lookup_id" TO API-NAME.
     INVOKE IMPL-REP "LOOKUP_ID" USING "IDL:OBJECT_B:1.0" RETURNING OBJ.
     INVOKE CORBA-IMPLEMENTATIONDEF "NARROW" USING OBJ RETURNING IMPL.
*
     MOVE "CORBA::BOA::create" TO API-NAME.
     INVOKE SEQUENCE-OCTET "NEW" RETURNING REF-ID.
     INVOKE BOA "IDL-CREATE" USING REF-ID INTF IMPL RETURNING OBJ.
*
     INVOKE SEQUENCE-NAMECOMPONENT-001 "NEW-WITH-LENGTH"
                       USING 2 RETURNING NAME.
     INVOKE CORBA-STRING "NEW-WITH-VALUE" USING "CONTEXT_A" RETURNING
NAME-ID
     INVOKE CORBA-STRING "NEW-WITH-VALUE" USING " " RETURNING NAME-KIND
     INVOKE COSNAMING-NAMECOMPONENT "NEW" RETURNING NAME-COMPONENT
```

```
      SET IDL-ID OF NAME-COMPONENT TO NAME-ID
      SET KIND OF NAME-COMPONENT TO NAME-KIND
      INVOKE NAME "SET-VALUE" USING 1 NAME-COMPONENT
*
      INVOKE CORBA-STRING "NEW-WITH-VALUE" USING "OBJECT_B" RETURNING NAME-ID
      INVOKE CORBA-STRING "NEW-WITH-VALUE" USING " " RETURNING NAME-KIND
      INVOKE COSNAMING-NAMECOMPONENT "NEW" RETURNING NAME-COMPONENT
      SET IDL-ID OF NAME-COMPONENT TO NAME-ID
      SET KIND OF NAME-COMPONENT TO NAME-KIND
      INVOKE NAME "SET-VALUE" USING 2 NAME-COMPONENT
*
      MOVE "CosNaming::NamingContext::bind" TO API-NAME.
      INVOKE NAMING-CONTEXT "BIND"
                         USING     NAME
                                   OBJ.
*
      STOP RUN.
*
 END-MAIN.
*
 END PROGRAM "CLIENT-MAIN".
```

8.7.3 Retrieving Object Lists under Contexts

This section describes how the Naming Service obtains lists of object references under naming contexts for objects stored under naming contexts shown in the following figure.

Figure 8.6 Naming Service Example (Obtaining Lists)



Examples of programs that obtain lists of CONTEXT_A are given below:

C Programming Example

First, use the same method for retrieving object references to obtain object references of the naming context for which a list is desired.

Next, use CosNaming_NamingContext_list() to obtain the list. The results are set in the third parameter CosNaming_BindingList for the number specified in the second parameter. If the number of included lists exceeds the number specified in the second parameter, the object references are set in object CosNaming_BindingIterator to retrieve the remaining lists. If the number of included lists is less than the number specified in the second parameter, CORBA_OBJECT_NIL is set in CosNaming_BindingIterator and the data of the list is set in CosNaming_BindingList.

Use CosNaming_BindingIterator_next_one() or CosNaming_BindingIterator_next_n() to obtain the remaining lists using CosNaming_BindingIterator. In the program below, CosNaming_BindingIterator_next_n() is used first to retrieve the first 40 object references. Then, CosNaming_BindingIterator_next_one() is used to retrieve the list of remaining object references until a sequence of _length 0 is returned.

Once retrieval is complete, CosNaming_BindingIterator_destroy() must be invoked to release the location on the server side. Use CORBA_Object_release() to release the location on the client side.

```
CosNaming_NamingContext     cos_naming, context_obj;
CosNaming_Name              name;
CosNaming_NameComponent     name_component;
CORBA_ORB                   orb;
CORBA_Environment           env;
CORBA_Object                obj;
CosNaming_BindingList       *bl, *bl2;
CosNaming_BindingIterator   bi;
CORBA_long                  how_many;

/* Obtain the root (CORBA_ORB_ObjectId_NameService) object. */
cos_naming = CORBA_ORB_resolve_initial_references(
         orb,
         CORBA_ORB_ObjectId_NameService,
         &env );

/* Fetch the naming context of CONTEXT_A. */
name._length = name._maximum = 1;
name._buffer = &name_component;
name._buffer[0].id = "CONTEXT_A";
name._buffer[0].kind = "";

context_obj = CosNaming_NamingContext_resolve(
         cos_naming,
         &name,
         &env );

/* Fetch the list of object references under CONTEXT_A. */
name._length = name._maximum = 1;
name._buffer = &name_component;
name._buffer[0].id = "CONTEXT_A";
name._buffer[0].kind = "";

how_many = 40;
CosNaming_NamingContext_list(
         cos_naming,
         how_many,
         &bl,
         &bi,
         &env );

/* Fetch the remaining object references under CONTEXT_A. */
if( bi != CORBA_OBJECT_NIL ){
   CORBA_free(bl);
   CosNaming_BindingIterator_next_n( bi, how_many, &bl2, &env );

   do {
      CosNaming_BindingIterator_next_one(
         bi, &b, &env );
      ... /* Process the results. */
      CORBA_free(b);
   }
   while( env._major != CORBA_NO_EXCEPTION &&
      b->binding_name._length != 0 );
   CosNaming_BindingIterator_destroy( bi, &env );
}
```

C++ Programming Example

First, use the same method for retrieving object references to obtain the object references of the naming context whose list is to be obtained.

Next, use CosNaming::NamingContext::list() to obtain the list. The results are set in the second parameter CosNaming::BindingList for the number specified in the first parameter. If the number of included lists exceeds the number specified in the first parameter, the object references are set in object CosNaming::BindingIterator to retrieve the remaining lists. If the number of included lists is less than the number specified in the second parameter, the NIL object is set in CosNaming::BindingIterator and the data of the list is set in CosNaming::BindingList.

Use CosNaming::BindingIterator::next_one() or CosNaming::BindingIterator::next_n() to obtain the remaining lists using CosNaming::BindingIterator. In the program below, CosNaming::BindingIterator::next_n() is used initially to retrieve the first 40 object references. Then, CosNaming::BindingIterator::next_one() is used to retrieve the list of remaining object references until a sequence of _length 0 is returned.

After retrieval is complete, CosNaming::BindingIterator::destroy() must be invoked to release the location on the server side. Use CORBA::release() to release the location on the client side.

```
CORBA::ORB_ptr                orb;
CosNaming::NamingContext_ptr  cos_naming;
CosNaming::Name_ptr           name;
CosNaming::NameComponent_var  *name_component;
CosNaming::BindingList_ptr    bl;
CosNaming::BindingIterator    bi;
CosNaming::Binding_ptr        b;
CORBA::Object_ptr             obj;
CosNaming::NamingContext_ptr  cos_naming, new_context;
CORBA::Long                   how_many;
CORBA::Environment            env;
CORBA::Long                   len;

try {
   //Obtain the root (CORBA_ORB_ObjectId_NameService) object.
   orb = CORBA::ORB_init( argc, argv, FJ_OM_ORBid, env );
   obj = orb->resolve_initial_references(
      CORBA_ORB_ObjectId_NameService,
      env );
   cos_naming = CosNaming::NamingContext::_narrow(obj);
   CORBA::release(obj);

   //Obtain CONTEXT_A from the root.
   CosNaming::NameComponent_var *name_component_var =
         CosNaming::Name::allocbuf(1);
   name_component[0]->id = (const CORBA::Char *)"CONTEXT_A";
   name_component[0]->kind = (const CORBA::Char *)"";
   name = new CosNaming::Name(1,1,name_component_var,CORBA_TRUE);
   obj = cos_naming->resolve( *name, env );
   delete name;
   new_context = CosNaming::NamingContext::_narrow(obj);
   CORBA::release(obj);

   //Fetch the list of object references under CONTEXT_A.
   how_many = 40;
   cos_naming->list( how_many, bl, bi, env );
   ...   //Process the results.
   delete bl;

   //Fetch the remaining object references under CONTEXT_A.
   if( bi->is_nil(env) != CORBA_TRUE ){
      bl->next_n( how_many, bl, env );
      ...   //Process the results.

      do {
      bl->next_one(b,env);
      ...   //Process the results.
      len = b->binding_name->length();
         delete b;
```

```
      }
      while( len != 0 );
   }
   bi->destroy(env);
   CORBA::release(bi);
}
catch ( CORBA::Exception &e ){
   ...      //Error processing
}
```

Java Programming Example

First, use the same method for retrieving object references to obtain the object references of the naming context whose list is to be obtained.

Next, use org.omg.CosNaming.NamingContext.list() to obtain the list. The results are set in the second parameter org.omg.CosNaming.BindingListHolder.value for the number of org.omg.CosNaming.Binding array specified in the first parameter. If the number of included lists exceeds the number specified in the first parameter, the object references of org.omg.CosNaming.BindingIterator are set in object org.omg.CosNaming.BindingIteratorHolder.value to retrieve the remaining lists. If the number of included lists is less than the number specified in the second parameter, null is set in org.omg.CosNaming.BindingIterator.value and data for the number of lists is set in org.omg.CosNaming.BindingListHolder.value.

Use org.omg.CosNaming.BindingIterator.next_one() or org.omg.CosNaming.BindingIterator.next_n() to obtain the remaining lists using org.omg.CosNaming.BindingIterator. In the program below, org.omg.CosNaming.BindingIterator.next_n() is used initially to retrieve the first 40 object references. Then, org.omg.CosNaming.BindingIterator.next_one() is used to retrieve the list of remaining object references until a sequence of _length 0 is returned.

```
public class List {
        public static void main(String args[]){
   try {
      int     i, j;

      //Obtain the root (NameService) object.
      org.omg.CORBA.ORB orb =  org.omg.CORBA.ORB.init(arg,null);
      org.omg.CORBA.Object obj = orb.resolve_initial_references("NameService");
      org.omg.CosNaming.NamingContextExt con_obj =
         org.omg.CosNaming.NamingContextExtHelper.narrow(obj);

      //Obtain CONTEXT_A from the root.
      org.omg.CosNaming.NameComponent [] name =
         new org.omg.CosNaming.NameComponent[1];
      name[0] = new org.omg.CosNaming.NameComponent("CONTEXT_A", "");

      org.omg.CORBA.Object tmp_obj =  con_obj.resolve( name );
      org.omg.CosNaming.NamingContextExt new_context =
         org.omg.CosNaming.NamingContextExtHelper.narrow(tmp_obj);

      //Fetch the list of object references under CONTEXT_A.
      int how_many = 40;
      org.omg.CosNaming.BindingListHolder bl_var =
         new org.omg.CosNaming.BindingListHolder();
      org.omg.CosNaming.BindingIteratorHolder bi_var =
         new org.omg.CosNaming.BindingIteratorHolder();
      new_context.list( how_many, bl_var, bi_var );
      //Process the results.

      //Fetch the remaining object references under CONTEXT_A.
      if( bi_var.value != null ) {
         org.omg.CosNaming.BindingIterator bi_obj = bi_var.value;
         bi_obj.next_n( how_many, bl_var );
         ...   //Process the results.

         org.omg.CosNaming.BindingHolder b_var =
            new org.omg.CosNaming.BindingHolder();
```

```
            do {
               bi_obj.next_one( b_var );
               ...  //Process the results.
            }
            while( b_var.value.binding_name.value.length != 0 );
         }
      }
      catch( Exception e ){
         ...  //Error processing
      }
   }
}
```

COBOL Programming Example

First, use the same method for retrieving object references to obtain object references of the naming context for which a list is desired.

Next, use COSNAMING-NAMINGCONTEXT-LIST to obtain the list. The results are set in the COSNAMING-BINDINGLIST specified in the BL for the number specified in HOW-MANY. If the number of included lists exceeds the number specified in HOW-MANY, the object references are set in object COSNAMING-BINDINGITERATOR to retrieve the remaining lists.

Use COSNAMING-BINDINGITERATOR-NEXT-ONE or COSNAMING-BINDINGITERATOR-NEXT-N to obtain the remaining lists using COSNAMING-BINDINGITERATOR. In the program below, COSNAMING-BINDINGITERATOR-NEXT-N is used first to retrieve the first 40 object references. Then, COSNAMING-BINDINGITERATOR-NEXT-ONE is used to retrieve the list of remaining object references.

IDENTIFICATION DIVISION.

```
  PROGRAM-ID. "LIST-TEST".
```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.

```
       ARGUMENT-NUMBER      IS  ARG-C
       ARGUMENT-VALUE       IS  ARG-V
       SYMBOLIC CONSTANT
       COPY SYMBOL-CONST IN CORBA.
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
  COPY CONST  IN CORBA.
  01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.
  01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.
  01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.

 *########## NAMING SETTING PARAMETER #########
  01 COPY COSNAMING-NAMINGCONTEXT IN CORBA
            REPLACING COSNAMING-NAMINGCONTEXT BY COS-NAMING.
  01 COPY COSNAMING-NAMINGCONTEXT IN CORBA
            REPLACING COSNAMING-NAMINGCONTEXT BY CONTEXT-OBJ.
  01 COPY COSNAMING-NAME IN CORBA REPLACING COSNAMING-NAME BY NAME.
  01 NAME-A USAGE POINTER.
  01 COPY COSNAMING-NAMECOMPONENT IN CORBA
            REPLACING COSNAMING-NAMECOMPONENT BY NAME-COMPONENT.
  01 NAME-COMPONENT-A USAGE POINTER.
  01 BL     USAGE POINTER.
  01 BL2    USAGE POINTER.
  01 COPY COSNAMING-BINDINGITERATOR IN CORBA
            REPLACING COSNAMING-BINDINGITERATOR BY BI.
  01 B       USAGE POINTER.

 *########## ORB SETTING PARAMETER #########
```

```
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY CURRENT-ARG-C.
 01 CURRENT-ARG-V.
   02 FILLER OCCURS 6.
      03 CURRENT-ARG-V-VALUE USAGE POINTER.
 01 APLI-NAME PIC X(8) VALUE "simple_c".
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY ARG-COUNT.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.

 01 TMP-STRING       USAGE POINTER.
 01 TMP-STRING-BUF  PIC X(20).
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY ELEMENT-NUMBER.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY HOW-MANY.
 01 COPY BOOLEAN IN CORBA REPLACING CORBA-BOOLEAN BY ROOP-FLAG.
 01 COPY BOOLEAN IN CORBA REPLACING CORBA-BOOLEAN BY RET.
```

LINKAGE SECTION.

```
 01 COPY COSNAMING-BINDING IN CORBA
          REPLACING COSNAMING-BINDING BY B-VALUE.
```

PROCEDURE DIVISION.

```

* ObjectDirector Initialization
     ACCEPT CURRENT-ARG-C FROM ARG-C.
     COMPUTE CURRENT-ARG-C = CURRENT-ARG-C + 1.
     PERFORM VARYING ARG-COUNT FROM 1 BY 1 UNTIL ARG-COUNT > CURRENT-ARG-C
        IF ARG-COUNT = 1
           MOVE APLI-NAME TO TMP-STRING-BUF
        ELSE
           ACCEPT TMP-STRING-BUF FROM ARG-V
        END-IF
        MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH
        CALL "CORBA-STRING-SET" USING
              CURRENT-ARG-V-VALUE (ARG-COUNT)
              STRING-LENGTH
              TMP-STRING-BUF
     END-PERFORM.
     SET CURRENT-ARG-V-VALUE (ARG-COUNT) TO NULL.

     MOVE FUNCTION LENG(FJ-OM-ORB-ID) TO STRING-LENGTH.
     CALL  "CORBA-STRING-SET" USING
           TMP-STRING
           STRING-LENGTH
           FJ-OM-ORB-ID.
     CALL "CORBA-ORB-INIT" USING
           CURRENT-ARG-C
           CURRENT-ARG-V
           TMP-STRING
           ENV
           ORB.
     CALL "CORBA-FREE" USING TMP-STRING.

* Obtain the root (CORBA-ORB-OBJECTID-NAMESERVICE) object.
     MOVE FUNCTION LENG ( CORBA-ORB-OBJECTID-NAMESERVICE ) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TMP-STRING
           STRING-LENGTH
           CORBA-ORB-OBJECTID-NAMESERVICE.
     CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
           ORB
           TMP-STRING
           ENV
```

```
           COS-NAMING.
     CALL "CORBA-FREE" USING TMP-STRING.

* Fetch the Naming Context under CONTEXT_A.
     MOVE 1 TO SEQ-LENGTH OF NAME.
     MOVE 1 TO SEQ-MAXIMUM OF NAME.
     CALL "CORBA-SEQUENCE-COSNAMING-NAMECOMPONENT-ALLOCBUF" USING
           SEQ-MAXIMUM OF NAME
           SEQ-BUFFER OF NAME.

     MOVE "CONTEXT_A" TO TMP-STRING-BUF.
     MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           IDL-ID OF NAME-COMPONENT
           STRING-LENGTH
           TMP-STRING-BUF.

     MOVE " " TO TMP-STRING-BUF.
     CALL "CORBA-STRING-SET" USING
           KIND OF NAME-COMPONENT
           STRING-LENGTH
           TMP-STRING-BUF.

     MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
     MOVE 1 TO ELEMENT-NUMBER.
     MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
     CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
           NAME-A
           ELEMENT-NUMBER
           NAME-COMPONENT-A.

     CALL "COSNAMING-NAMINGCONTEXT-RESOLVE" USING
           COS-NAMING
           NAME
           ENV
           CONTEXT-OBJ.

* Fetch the list of object references under CONTEXT_A.
     MOVE  40 TO  HOW-MANY.
     CALL "COSNAMING-NAMINGCONTEXT-LIST" USING
           CONTEXT-OBJ
           HOW-MANY
           BL
           BI
           ENV.
* Fetch the remaining object references under CONTEXT_A.
     IF NOT BI = CORBA-OBJECT-NIL THEN
            CALL "CORBA-FREE" USING  BL

            CALL "COSNAMING-BINDINGITERATOR-NEXT-N" USING
                   BI
                   HOW-MANY
                   BL2
                   ENV
                   RET

            MOVE CORBA-TRUE TO ROOP-FLAG

            PERFORM UNTIL ROOP-FLAG = CORBA-FALSE
                   CALL "COSNAMING-BINDINGITERATOR-NEXT-ONE" USING
                          BI
                          B
                          ENV
```

```
                        RET

* Process the results.

                    EVALUATE TRUE
                    WHEN CORBA-NO-EXCEPTION OF ENV
                           CONTINUE
                    WHEN CORBA-USER-EXCEPTION OF ENV
                    WHEN CORBA-USER-EXCEPTION OF ENV
                           EXIT PROGRAM
                    END-EVALUATE

                    SET ADDRESS OF B-VALUE TO B
                    IF SEQ-LENGTH OF B-VALUE = 0
                           MOVE CORBA-FALSE TO ROOP-FLAG
                    ELSE
                           CALL "CORBA-FREE" USING B
                    END-IF
             END-PERFORM

             CALL "COSNAMING-BINDINGITERATOR-DESTROY" USING
                    BI
                    ENV
                    RET
             CALL "CORBA-OBJECT-RELEASE" USING
                    BI
                    ENV
       END-IF.
       EXIT PROGRAM.
 END PROGRAM "LIST-TEST".
```

OOCOBOL Programming Example

First, use the same method for retrieving object references to obtain object references of the naming context for which a list is desired.

Next, use COSNAMING-NAMINGCONTEXT-LIST to obtain the list. The results are set in the COSNAMING-BINDINGLIST specified in the BL for the number specified in HOW-MANY. If the number of included lists exceeds the number specified in HOW-MANY, the object references are set in object COSNAMING-BINDINGITERATOR to retrieve the remaining lists.

Use COSNAMING-BINDINGITERATOR-NEXT_ONE or COSNAMING-BINDINGITERATOR-NEXT_N to obtain the remaining lists using COSNAMING-BINDINGITERATOR. In the program below, COSNAMING-BINDINGITERATOR-NEXT_N is used first to retrieve the first 40 object references. Then, COSNAMING-BINDINGITERATOR-NEXT_ONE is used to retrieve the list of remaining object references.

IDENTIFICATION DIVISION.

```
  PROGRAM-ID.            "CLIENT-MAIN".
```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.

```
  REPOSITORY.
      COPY               CORBA--REP.
      COPY               COSNAMING--REP.
```

SPECIAL-NAMES.

```
      SYMBOLIC CONSTANT
      COPY               CORBA--CONST.
      COPY               COSNAMING--CONST.
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
 COPY                   CORBA--COPY.
 COPY                   COSNAMING--COPY.
*
 01 ORB                 USAGE OBJECT REFERENCE CORBA-ORB.
 01 BOA                 USAGE OBJECT REFERENCE CORBA-BOA.
 01 OBJ                 USAGE OBJECT REFERENCE CORBA-OBJECT.
 01 NAMING-CONTEXT      USAGE OBJECT REFERENCE COSNAMING-NAMINGCONTEXT.
 01 NAME                TYPE                   COSNAMING-NAME.
 01 NAME-COMPONENT      USAGE OBJECT REFERENCE COSNAMING-NAMECOMPONENT.
 01 NEW-CONTEXT         USAGE OBJECT REFERENCE COSNAMING-NAMINGCONTEXT.
 01 BL                  TYPE                   COSNAMING-BINDINGLIST.
 01 BI                  USAGE OBJECT REFERENCE COSNAMING-BINDINGITERATOR.
 01 NAME-ID             USAGE OBJECT REFERENCE CORBA-STRING.
 01 NAME-KIND           USAGE OBJECT REFERENCE CORBA-STRING.
 01 HOW-MANY            TYPE                   CORBA-UNSIGNED-LONG.
 01 B                   USAGE OBJECT REFERENCE COSNAMING-BINDING.
 01 LEN                 TYPE                   CORBA-UNSIGNED-LONG.
*
 01 EXCEPTION-ID        USAGE OBJECT REFERENCE CORBA-STRING.
 01 EXCEPTION-ID-VALUE  PIC   X(50).
 01 API-NAME            PIC   X(50).
*
 01 I                   TYPE                   CORBA-UNSIGNED-LONG.
```

PROCEDURE DIVISION.

```
 DECLARATIVES.
*
 OTHER-ERROR SECTION.
     USE EXCEPTION CORBA-EXCEPTION.
     DISPLAY "CORBA::Exception: " API-NAME.
     SET EXCEPTION-ID TO IDL-ID OF EXCEPTION-OBJECT AS CORBA-EXCEPTION.
     INVOKE EXCEPTION-ID "GET-VALUE" RETURNING EXCEPTION-ID-VALUE.
     DISPLAY "   Exception-id: " EXCEPTION-ID-VALUE.
     EXIT PROGRAM.
 END-OTHER-ERR.
*
 END DECLARATIVES.
*
 MAIN SECTION.
*
     MOVE "CORBA::ORB_init" TO API-NAME.
     INVOKE CORBA "ORB_INIT"
                        USING     "Naming"
                                  FJ-OM_ORBID
                        RETURNING ORB.
*
     MOVE "CORBA::ORB::BOA_init" TO API-NAME.
     INVOKE ORB "BOA_INIT"
                        USING     "Naming"
                                  CORBA-BOA_OAID
                        RETURNING BOA.
*
     MOVE "CORBA::ORB::resolve_initial_references" TO API-NAME.
     INVOKE ORB "RESOLVE_INITIAL_REFERENCES"
                        USING     CORBA-ORB-OBJECTID_NAMESERVICE
                        RETURNING OBJ.
*
     INVOKE COSNAMING-NAMINGCONTEXT "NARROW"
                        USING     OBJ
                        RETURNING NAMING-CONTEXT.
*
*
```

```
     INVOKE SEQUENCE-NAMECOMPONENT-001 "NEW-WITH-LENGTH"
                       USING 1 RETURNING NAME.
     INVOKE CORBA-STRING "NEW-WITH-VALUE" USING "CONTEXT_A" RETURNING
NAME-ID
     INVOKE CORBA-STRING "NEW-WITH-VALUE" USING " " RETURNING NAME-KIND
     INVOKE COSNAMING-NAMECOMPONENT "NEW" RETURNING NAME-COMPONENT
     SET IDL-ID OF NAME-COMPONENT TO NAME-ID
     SET KIND OF NAME-COMPONENT TO NAME-KIND
     INVOKE NAME "SET-VALUE" USING 1 NAME-COMPONENT.
*
     MOVE "CosNaming::NamingContext::resolve" TO API-NAME.
     INVOKE NAMING-CONTEXT "RESOLVE"
                       USING     NAME
                        RETURNING OBJ.
     INVOKE COSNAMING-NAMINGCONTEXT "NARROW" USING OBJ RETURNING
NEW-CONTEXT.
*
     MOVE 40 TO HOW-MANY.
     MOVE "CosNaming::NamingContext::list" TO API-NAME.
     INVOKE NEW-CONTEXT "LIST" USING HOW-MANY BL BI.
*
     IF BI NOT = NULL
       MOVE "CosNaming::BindingIterator::next_n" TO API-NAME
       INVOKE BI "NEXT_N" USING HOW-MANY BL
       MOVE "CosNaming::BindingIterator::next_one" TO API-NAME
       SET NAME TO BINDING_NAME OF B
       MOVE SEQ-LENGTH OF NAME TO LEN
       PERFORM UNTIL LEN = 0
         MOVE "CosNaming::BindingIterator::next_one" TO API-NAME
         SET NAME TO BINDING_NAME OF B
         MOVE SEQ-LENGTH OF NAME TO LEN
       END-PERFORM
       MOVE "CosNaming::BindingIterator::destroy" TO API-NAME
       INVOKE BI "DESTROY"
     END-IF.
*
     STOP RUN.
*
 END-MAIN.
*
 END PROGRAM "CLIENT-MAIN".
```

Chapter 9 Interface Repository Service Programming

This chapter explains the API (Application Programming Interface) and the programming provided by the Interface Repository Service.

9.1 Types of Objects Managed by the Interface Repository Service

The Interface Repository service administers and stores within it objects such as ModuleDef, InterfaceDef, OperationDef, ConstantDef, and AliasDef. These correspond to the following declarations as defined in IDL: module declaration, interface declaration, operation declaration, constant declaration, and type declaration. The following table lists IDL declarations and their corresponding objects.

In addition to objects corresponding to IDL declarations, the repository object also exists as an Interface Repository service root object and a PrimitiveDef object to describe primitive types (e.g., long, short).

Table 9.1 Object Types

IDL Declaration	Object
-	Repository object
-	PrimitiveDef object
Module declaration	ModuleDef object
Interface declaration	InterfaceDef object
Operation declaration	OperationDef object
Variable declaration	AttributeDef object
Constant declaration	ConstantDef object
TypeDef declaration	AliasDef object
String type declaration	StringDef object
Wide string type declaration	WstringDef object
Fixed point type declaration (See Note)	FixedDef object
Enumerated type declaration	EnumDef object
Sequence type declaration	SequenceDef object
Structure declaration	StructDef object
Union declaration	UnionDef object
Array type declaration	ArrayDef object
Exception declaration	ExceptionDef object

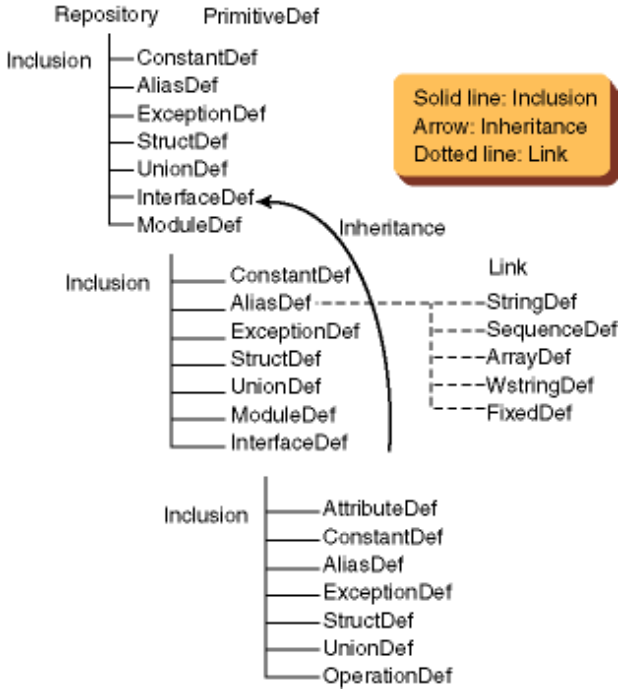
Note

The fixed-point type declaration can be used only in COBOL mapping.

9.2 Interface Repository Object Relationships (Inclusion/ Inheritance)

The relationship (inclusion) between objects managed in the Interface Repository service is constructed in line with IDL definitions. Repository, ModuleDef, and InterfaceDef objects can include objects from other repositories. An InterfaceDef object can inherit other InterfaceDef objects. The inclusion relationship is constructed to allow StringDef, ArrayDef, SequenceDef, WstringDef, and FixedDef objects to be included by being linked with an AliasDef object. Figure 9-1 shows this relationship.

Figure 9.1 Object Inclusive Relationships



9.3 Interface Repository Service Interface

The Interface Repository service provides the interfaces listed below.

9.3.1 Interfaces Provided by Interface Repository Service

```
module CORBA{
   interface IRObject;
   interface Contained    : IRObject{};
   interface Container    : IRObject{};
   interface IDLType      : IRObject{};
   interface Repository   : Container{};
   interface ModuleDef    : Container, Contained{};
   interface ConstantDef  : Contained{};
   interface TypedefDef   : Contained, IDLType{};
   interface StructDef    : TypedefDef{};
   interface UnionDef     : TypedefDef{};
   interface EnumDef      : TypedefDef{};
   interface AliasDef     : TypedefDef{};
   interface PrimitiveDef : IDLType{};
   interface StringDef    : IDLType{};
   interface WstringDef    : IDLType{};
   interface SequenceDef  : IDLType{};
   interface ArrayDef     : IDLType{};
   interface FixedDef     : IDLType{};
   interface ExceptionDef : Contained{};
   interface AttributeDef : Contained{};
   interface OperationDef : Contained{};
   interface InterfaceDef : Contained, Container, IDLType{};
// interface TypeCode;

   typedef string    Identifier;
   typedef string    ScopedName;
   typedef string    RepositoryId;

   enum DefinitionKind{
```

```
      dk_none,dk_all,dk_Attribute,dk_Constant,dk_Exception,
      dk_Interface,dk_Module,dk_Operation,dk_Typedef,
      dk_Alias,dk_Struct,dk_Union,dk_Enum,dk_Primitive,
      dk_String,dk_Sequence,dk_Array,dk_Repository,
      dk_Wstring,dk_Fixed
   };

   interface IRObject{
      // read interface
      readonly attribute DefinitionKind   def_kind;

      //write interface
      void destroy();
   };

   typedef string    VersionSpec;

 interface Contained;
   interface Repository;
   interface Container;
// interface TypeCode;

   interface Contained:IRObject{
      // read/write interface
               attribute RepositoryId      id;
               attribute Identifier        name;
               attribute VersionSpec       version;

      //read interface
      readonly attribute Container         defined_in;
      readonly attribute ScopedName        absolute_name;
      readonly attribute Repository        containing_repository;

      struct Description{
          DefinitionKind    kind;
          any               value;
      };

      Description describe();

      //write interface
      void move(
         in Container       new_container,
         in Identifier      new_name,
         in VersionSpec     new_version
      );
   };

 interface ModuleDef;
 interface ConstantDef;
 interface IDLType;
 interface StructDef;
 interface UnionDef;
 interface EnumDef;
 interface AliasDef;
 interface InterfaceDef;
 interface ExceptionDef;

 typedef sequence<InterfaceDef>InterfaceDefSeq;

   typedef sequence<Contained>ContainedSeq;

   struct StructMember{
```

```
   Identifier name;
   TypeCode   type;
   IDLType    type_def;
};

typedef sequence<StructMember>StructMemberSeq;

struct UnionMember{
   Identifier name;
   any        label;
   TypeCode   type;
   IDLType    type_def;
};

typedef sequence<UnionMember>UnionMemberSeq;

typedef sequence<Identifier>EnumMemberSeq;

interface Container:IRObject{
   //read interface

   Contained lookup(in ScopedName search_name);

   ContainedSeq contents(
      in DefinitionKind   limit_type,
      in boolean          exclude_inherited
   );

   ContainedSeq lookup_name(
      in Identifier       search_name,
      in long             levels_to_search,
      in DefinitionKind   limit_type,
      in boolean          exclude_inherited
   );

   struct Description{
      Contained           contained_object;
      DefinitionKind      kind;
      any                 value;
   };

   typedef sequence<Description>DescriptionSeq;

   DescriptionSeq describe_contents(
      in DefinitionKind   limit_type,
      in boolean          exclude_inherited,
      in long             max_returned_objs
   );

   //write interface

   ModuleDef create_module(
      in RepositoryId     id,
      in Identifier       name,
      in VersionSpec      version
   );

   ConstantDef create_constant(
      in RepositoryId     id,
      in Identifier       name,
      in VersionSpec      version,
      in IDLType          type,
      in any              value
```

```
      );

      ExceptionDef create_exception(
         in RepositoryId     id,
         in Identifier       name,
         in VersionSpec      version,
         in StructMemberSeq  members
      );

      StructDef create_struct(
         in RepositoryId     id,
         in Identifier       name,
         in VersionSpec      version,
         in StructMemberSeq  members
      );

      UnionDef create_union(
         in RepositoryId     id,
         in Identifier       name,
         in VersionSpec      version,
         in IDLType          discriminator_type,
         in UnionMemberSeq   members
      );

      EnumDef create_enum(
         in RepositoryId     id,
         in Identifier       name,
         in VersionSpec      version,
         in EnumMemberSeq    members
      );

      AliasDef create_alias(
         in RepositoryId     id,
         in Identifier       name,
         in VersionSpec      version,
         in IDLType          original_type
      );

      InterfaceDef create_interface(
         in RepositoryId     id,
         in Identifier       name,
         in VersionSpec      version,
         in InterfaceDefSeq  base_interfaces
      );
};

   interface IDLType:IRObject{
      readonly attribute TypeCode type;
   };

   enum PrimitiveKind{
      pk_null,pk_void,pk_short,pk_long,pk_ushort,pk_ulong,
      pk_float,pk_double,pk_boolean,pk_char,pk_octet,
      pk_any,pk_TypeCode,pk_Principal,pk_string,pk_objref,
      pk_longlong,pk_ulonglong,pk_longdouble,pk_wchar,pk_wstring
   };

   interface PrimitiveDef;
   interface StringDef;
   interface WstringDef;
   interface SequenceDef;
   interface ArrayDef;
   interface FixedDef;
```

```
interface Repository:Container{

   //read interface
   Contained lookup_id(in RepositoryId search_id);
   PrimitiveDef get_primitive(in PrimitiveKind kind);

   //write interface
   StringDef create_string(in unsigned long bound);

   WstringDef create_wstring(in unsigned long bound);

   SequenceDef create_sequence(
      in unsigned long bound,
      in IDLType element_type
   );

   ArrayDef create_array(
      in unsigned long length,
      in IDLType element_type
   );

  FixedDef create fixed(
      in unsigned short digits,
      in short scale
  );

   void load();
   void destroy_rep();
};

interface ModuleDef:Container,Contained{
};

struct ModuleDescription{
   Identifier         name;
   RepositoryId       id;
   RepositoryId       defined_in;
   VersionSpec        version;
};

interface ConstantDef:Contained{
   readonly attribute TypeCode type;
            attribute IDLType  type_def;
            attribute any      value;
};

struct ConstantDescription{
   Identifier         name;
   RepositoryId       id;
   RepositoryId       defined_in;
   VersionSpec        version;
   TypeCode           type;
   any                value;
};

interface TypedefDef:Contained,IDLType{
};

struct TypeDescription{
   Identifier         name;
   RepositoryId       id;
   RepositoryId       defined_in;
   VersionSpec        version;
```

```
   TypeCode           type;
};

interface StructDef:TypedefDef{
            attribute StructMemberSeq members;
};

interface UnionDef:TypedefDef{
   readonly attribute TypeCode discriminator_type;
            attribute IDLType  discriminator_type_def;
            attribute UnionMemberSeq members;
};

interface EnumDef:TypedefDef{
            attribute EnumMemberSeq members;
};

interface AliasDef:TypedefDef{
            attribute IDLType original_type_def;
};

interface PrimitiveDef:IDLType{
   readonly attribute PrimitiveKind kind;
};

interface StringDef:IDLType{
            attribute unsigned long bound;
};

interface WstringDef:IDLType{
            attribute unsigned long bound;
};

interface SequenceDef:IDLType{
            attribute unsigned long bound;
   readonly attribute TypeCode element_type;
            attribute IDLType  element_type_def;
};

interface ArrayDef:IDLType{
            attribute unsigned long length;
   readonly attribute TypeCode element_type;
            attribute IDLType  element_type_def;
};

interface ExceptionDef:Contained{
   readonly attribute TypeCode type;
            attribute StructMemberSeq members;
};

struct ExceptionDescription{
   Identifier      name;
   RepositoryId    id;
   RepositoryId    defined_in;
   VersionSpec     version;
   TypeCode        type;
};

enum AttributeMode{ATTR_NORMAL,ATTR_READONLY};

interface AttributeDef:Contained{
   readonly attribute TypeCode type;
            attribute IDLType  type_def;
```

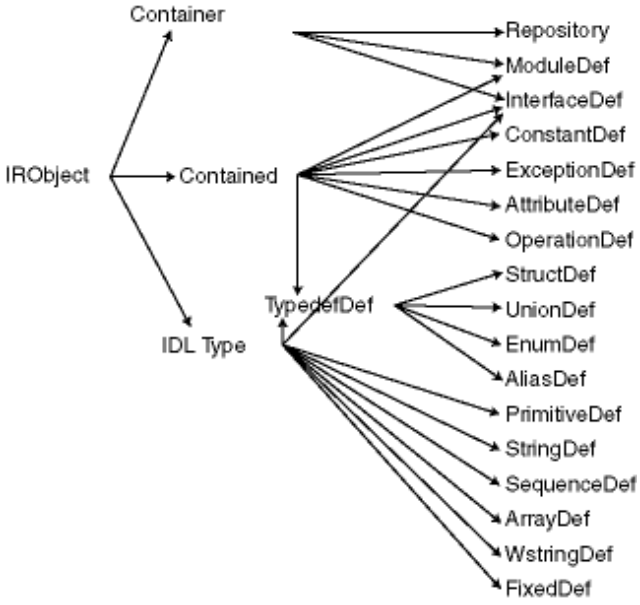
```
            attribute AttributeMode mode;
  };

  struct AttributeDescription{
     Identifier      name;
     RepositoryId    id;
     RepositoryId    defined_in;
     VersionSpec     version;
     TypeCode        type;
     AttributeMode   mode;
  };

  enum OperationMode{OP_NORMAL,OP_ONEWAY};

  enum ParameterMode{PARAM_IN,PARAM_OUT,PARAM_INOUT};

  struct ParameterDescription{
     Identifier      name;
     TypeCode        type;
     IDLType         type_def;
     ParameterMode   mode;
  };

  typedef sequence<ParameterDescription>ParDescriptionSeq;

  typedef Identifier ContextIdentifier;
  typedef sequence<ContextIdentifier>ContextIdSeq;

  typedef sequence<ExceptionDef>ExceptionDefSeq;
  typedef sequence<ExceptionDescription>ExcDescriptionSeq;

  interface OperationDef:Contained{
     readonly attribute TypeCode result;
              attribute IDLType  result_def;
              attribute ParDescriptionSeq params;
              attribute OperationMode mode;
              attribute ContextIdSeq contexts;
              attribute ExceptionDefSeq exceptions;
   };

  struct OperationDescription{
     Identifier         name;
     RepositoryId       id;
     RepositoryId       defined_in;
     VersionSpec        version;
     TypeCode           result;
     OperationMode      mode;
     ContextIdSeq       contexts;
     ParDescriptionSeq  parameters;
     ExcDescriptionSeq  exceptions;
  };

typedef sequence<RepositoryId>RepositoryIdSeq;
typedef sequence<OperationDescription>OpDescriptionSeq;
typedef sequence<AttributeDescription>AttrDescriptionSeq;

interface InterfaceDef:Container,Contained,IDLType{
     //read/write interface

   attribute InterfaceDefSeq base_interfaces;

     //read interface
```

```
    boolean is_a(in RepositoryId interface_id);

    struct FullInterfaceDescription{
       Identifier           name;
       RepositoryId         id;
       RepositoryId         defined_in;
       VersionSpec          version;
       OpDescriptionSeq     operations;
       AttrDescriptionSeq   attributes;
       RepositoryIdSeq      base_interfaces;
       TypeCode             type;
    };
    FullInterfaceDescription describe_interface();

      //write interface

    AttributeDef create_attribute(
       in RepositoryId id,
       in Identifier name,
       in VersionSpec version,
       in IDLType type,
       in AttributeMode mode
    );

    OperationDef create_operation(
       in RepositoryId id,
       in Identifier name,
       in VersionSpec version,
       in IDLType result,
       in OperationMode mode,
       in ParDescriptionSeq params,
       in ExceptionDefSeq exceptions,
       in ContextIdSeq contexts
    );
 };

 struct InterfaceDescription{
    Identifier        name;
    RepositoryId      id;
    RepositoryId      defined_in;
    VersionSpec       version;
    RepositoryIdSeq   base_interfaces;
 };

 interface InterfaceRep{
    //write interface
    Repository   create_repository();
    boolean      create_primitives();
 };
};
```

9.3.2 Inheritance Relationships between Interfaces

The following figure shows the inheritance relationship between interfaces.

Figure 9.2 Interface Inheritance Relationships



The following table lists the functions of interfaces provided by the Interface Repository service.

Table 9.2 Interface Repository Service Interfaces

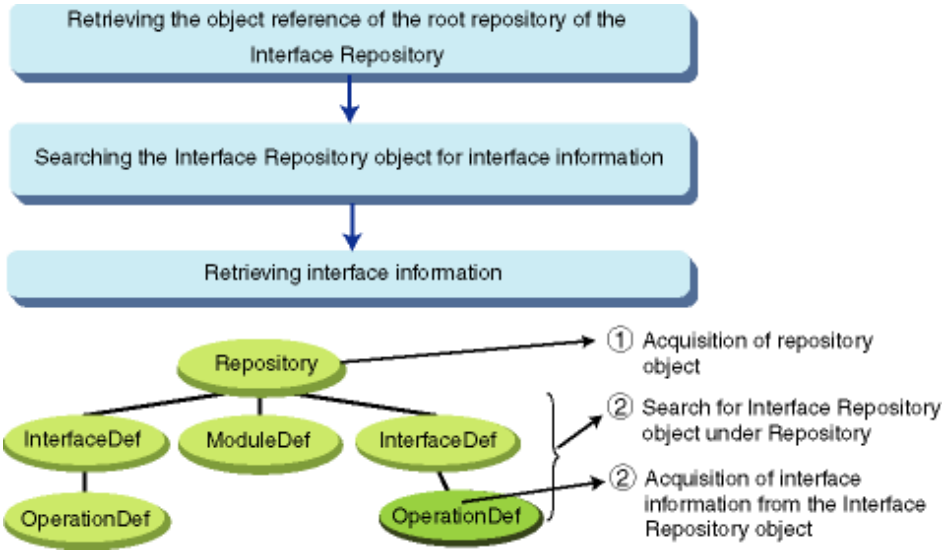
Interface	Object	Function
IRObject	def_kind	Posts the repository object type.
Contained	id	Posts the repository identifier (ID) of a repository object.
	name	Report the simple name of a repository object.
	version	Posts the version of a repository object.
	defined_in	Posts a repository object included in another repository object.
	absolute_name	Posts the scope name of a repository object.
	containing_repository	Posts a Repository object.
	describe	Posts interface information of a repository object.
Container	lookup	Searches for a repository object using the scope name.
	contents	Posts included repository objects in a list structure.
	lookup_name	Searches for a repository object using the simple name.
	describe_contents	Posts information on included repository objects in a list structure.
IDLType	type	Posts the type code of an IDLType object.
Repository	lookup_id	Searches for a repository object using the repository ID.
	get_primitive	Posts the object reference of a PrimitiveDef object exposing a primitive data type (e.g. long, short).
InterfaceDef	base_interfaces	Posts a list of InterfaceDef objects inheriting InterfaceDef objects.
	describe_interface	Posts detailed information about an InterfaceDef object. Posts operation/attribute information along with information posted by the describe operation.
	is_a	Judges whether an InterfaceDef object is inheriting another InterfaceDef object or not.
ConstantDef	type	Posts the type code of a ConstantDef object.
	type_def	Posts an object exposing the type of a ConstantDef object.
	value	Posts the constant value of a ConstantDef object.

Interface	Object	Function
ExceptionDef	type	Posts the type code of the exception definition of an ExceptionDef object.
	members	Posts a list of structure members in the exception definition of an ExceptionDef object.
AttributeDef	type	Posts the type code of an AttributeDef object.
	type_def	Object exposing the type of an AttributeDef object.
	mode	Posts whether the attribute of an AttributeDef object is read-only.
OperationDef	result	Posts the type code of the return value of an OperationDef object.
	result_def	Posts an object exposing the return value of an OperationDef object.
	params	Posts the parameter list of an OperationDef object.
	mode	Displays whether an OperationDef object is in one-way mode or not.
	contexts	Posts an OperationDef object context property name list.
	exceptions	Posts the ExceptionDef object list of an OperationDef object.
IDLType	type	Posts the type code of an IDLType object.
StructDef	members	Posts the member list of a StructDef object.
UnionDef	discriminator_type	Posts the type code of the discriminator of a UnionDef object.
	discriminator_type_def	Posts an object expressing the type of discriminator of a UnionDef object.
	members	Posts the member list of a UnionDef object.
EnumDef	members	Posts the member list of an EnumDef object.
AliasDef	original_type_def	Object of the actual type defined by an AliasDef object TypeDef.
PrimitiveDef	kind	Object of the actual type defined by the PrimitiveDef object type.
StringDef	bound	Posts the maximum number of characters in a StringDef object.
SequenceDef	bound	Posts the maximum number of characters in a SequenceDef object.
	type	Posts the type code of a SequenceDef object element type.
	type_def	Posts the object of a SequenceDef object element type.
ArrayDef	bound	Reports the number of elements in an ArrayDef object array.
	type	Reports the type code of an array element of an ArrayDef object.
	type_def	Reports the object of an ArrayDef object array element type.
WstringDef	bound	Posts the maximum number of characters in a WstringDef object.
FixedDef	digits	Posts the number of beams in a FixedDef object.
	scale	Posts the scale in a FixedDef object.

9.4 Interface Repository Service Programming

The following figure shows the flow of programming for the retrieval of the server application supplied function parameters and the return value related interface information using the Interface Repository service.

Figure 9.3 Interface Repository Service Programming Flow



9.4.1 Retrieving a Root Repository Object Reference

Retrieve the root repository object reference of the Interface Repository by using the system-provided service object reference fetching method; resolve_initial_references().

9.4.2 Searching for an Interface Repository Object

Search for the Interface Repository object relating to the interface information to be retrieved. Use one of the methods listed in the following table.

Table 9.3 Retrieving Object Methods

Interface Name	Method Name	Key	Function
Repository	lookup_id	Repository ID	Posts the object reference of an object specified with a repository ID.
Container	lookup	Scope name	Posts the object reference of an object specified with a scope name.
	lookup_name	Simple name	Posts the object references of objects specified with a simple name in a list structure.
	contents	_	Posts the object references of included objects in a list structure.

9.4.3 Retrieving Interface Information

In this step, interface information about an Interface Repository object is acquired using one of the methods listed in the following table.

Table 9.4 Methods for Acquiring Interface Information

Interface Name	Method Name	Function
Contained	describe	Posts interface information about a specified object.
Container	describe_contents	Posts interface information about objects included in a specified object in a list structure.

Interface information is acquired by specifying the object reference acquired in an Interface Repository object search (as described in 9.4.2 Searching for an Interface Repository Object).

9.4.4 Examples of Interface Repository Service Programming

This section provides two examples of Interface Repository Service Programming:

- Example 1: Retrieving OperationDef object information

- Example 2: Retrieving information on StructDef and AliasDef objects

Windows64

OOCOBOL cannot be used.

Linux64

OOCOBOL can only be used to create Windows(R) clients.

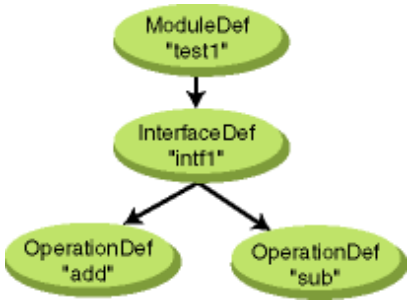
9.4.4.1 Example 1: Retrieving OperationDef Object Information

This example shows the programming necessary to acquire parameter information about the OperationDef object in the IDL definition shown below.

```
module test1 {
   interface intf1  {
      long add ( in long a,  in long b);
      long sub ( in long c,  in long d) ;
   };
};
```

Information about the objects shown in the following figure is stored in the Interface Repository service.

Figure 9.4 Acquiring OperationDef Object Information



Retrieving an Interface Repository Object Reference

Retrieve the object reference of the root repository using the provided service retrieval method; resolve_initial_references().

C

```
CORBA_Repository   intf_rep;
/*interface repository object reference acquisition*/
intf_rep = CORBA_ORB_resolve_initial_references(
      orb,CORBA_ORB_ObjectId_InterfaceRepository, &env);
```

C++

```
CORBA::Object_ptr   obj;
CORBA::Repository_ptr   intf_rep;

// Interface repository object reference acquisition
obj = orb->resolve_initial_references(
   CORBA_ORB_ObjectId_InterfaceRepository, *env );

// Conversion to Repository class
intf_rep = CORBA::Repository::_narrow( obj );
```

COBOL Windows32/64 Solaris32/64

```
*########## Interface repository object reference acquisition ###########
     MOVE FUNCTION LENG (CORBA-ORB-OBJECTID-INTFREP) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
```

```
                TEMP-BUF
                STRING-LENGTH
                CORBA-ORB-OBJECTID-INTFREP.
    CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
                ORB
                TEMP-BUF
                ENV
                OBJ.
```

Java

```
org.omg.CORBA.Object   Obj;
org.omg.CORBA.Repository   Rep;
// Interface repository object reference acquisition
Obj = Orb.resolve_initial_references( "InterfaceRepository" );

// Conversion to Repository_var class
Rep = org.omg.CORBA.RepositoryHelper.narrow( Obj );
```

Retrieving an Object Reference for an OperationDef Object

Retrieve an object reference for an OperationDef object by specifying the object repository ID of the target OperationDef "IDL:test1/intf1/add:1.0"and the object reference of the root repository, issue lookup_id().

C

```
CORBA_Contained intfobj;
/* object search */
intfobj = CORBA_Repository_lookup_id(intf_rep, "IDL:test1/intf1/add:1.0", &env);
```

C++

```
CORBA::Object_ptr intfobj;
// object search
intfobj = intf_rep->lookup_id( "IDL:test1/intf1/add:1.0", *env );
```

COBOL Windows32/64 Solaris32/64

```
*########## Interface repository object search ##########
       MOVE FUNCTION LENG ("IDL:test1/intf1/add:1.0") TO STRING-LENGTH
       MOVE "IDL:test1/intf1/add:1.0" TO IN-BUF
       CALL "CORBA-STRING-SET" USING
                   REPOSITORYID
                   STRING-LENGTH
                   IN-BUF.
       CALL "CORBA-REPOSITORY-LOOKUP-ID" USING
                   OBJ
                   REPOSITORYID
                   ENV
                   INTF-INTF.
```

Java

```
org.omg.CORBA.Object      obj;

//object search
obj = Rep.lookup_id( "IDL:test1/intf1/add:1.0" );
```

Retrieving Parameter Information

Search the Interface Repository for information about server application methods by specifying the object reference of the OperationDef object and activating the described method. As a result, the parameter information (parameter name, number of parameters, parameter type) is posted.

C

```
CORBA_Contained_Description       *ope_des;
CORBA_OperationDescription        *o;
CORBA_TCKind  kind;
CORBA_ParDescriptionSeq           *parameters;
CORBA_ParameterDescription        *prmp;

/*OperationDef object interface information acquisition*/
ope_des = CORBA_OperationDef_describe(intfobj,&env);
/*fetching of OperationDef object specific information structure*/
o = (CORBA_OperationDescription *)ope_des->value._value;

parameters = &(o->parameters);   /*Reference for parameter sequence information*/
/*reference for the number of parameters*/
printf("Parameter number = %d\n", parameters->_length );
for(k=0,prmp=parameters->_buffer; k<parameters->_length; k++){
   /*reference for parameter name*/
   printf("Parameter Identifier(%d) = %s\n", k, prmp[k].name);
   /*reference for parameter type*/
   kind = CORBA_TypeCode_kind(prmp[k].type,&env);
   printf("TypeCode_kind(%d) = %d\n",k,kind);
   /*parameter mode (IN/OUT/INOUT)*/
   printf("Parameter_mode(%d) = %d\n",k, prmp[k].mode);
}
```

C++

```
CORBA::OperationDef_ptr        opef;
CORBA::Contained::Description  *desc;
CORBA::OperationDescription    *c;

//OperationDef object interface information acquisition
opef = CORBA::OperationDef::_narrow(intfobj);
desc   = opef->describe(*env);
//fetching of OperationDef object specific information structure
CORBA::Any *opeany = (CORBA::Any*)(desc->value);
c = (CORBA::OperationDescription *)opeany->value();
//reference for parameter sequence information
CORBA::ParDescriptionSeq *parseq = (CORBA::ParDescriptionSeq*)c->parameters;
//reference for the number of parameters
printf("Parameter number = %d\n", parseq->length() );
for( int n = 0;n < parseq->length(); n++) {
   CORBA::ParameterDescription *parades =
         (CORBA::ParameterDescription*)(*parseq)[m];
   //reference for parameter name
   printf("para name = %s \n", (char*)parades->name);
   //reference for parameter type
   CORBA::TypeCode
   *typecode2 = (CORBA::TypeCode *)(parades->type);
   kind = typecode2->kind(*env);
   printf("TypeCode_kind = %d\n",kind);

   //parameter mode (IN/OUT/INOUT)
   printf("para mode = %d \n", parades->mode);
}
```

COBOL Windows32/64 Solaris32/64

```
*########## OperationDef object interface information acquisition ##########
     CALL "CORBA-OPERATIONDEF-DESCRIBE" USING
                    OBJ
                    ENV
                    LINK-BUF.
```

```
*--- Fetching of OperationDef object specific information structure ---
     SET ADDRESS OF LINK-CONDES TO LINK-BUF.
     MOVE IDL-VALUE OF LINK-CONDES TO TMP-ANY.
     MOVE ANY-VALUE OF TMP-ANY TO TEMP-BUF.
     SET ADDRESS OF TMP-OPR TO TEMP-BUF.
     MOVE FUNCTION ADDR(PARAMETERS OF TMP-OPR) TO TEMP-BUF.
     SET ADDRESS OF PARDESSEQ TO TEMP-BUF.

*---- Reference for parameter sequence information ---
*---- Reference for the number of parameters ---
     IF SEQ-LENGTH OF PARDESSEQ = 0
          DISPLAY "PARA-LENGTH = 0"
     END-IF.
     DISPLAY "PARAMETER NUMBER = "  SEQ-LENGTH OF PARDESSEQ.
     PERFORM TEST BEFORE
             VARYING CNT FROM 1 BY 1
                 UNTIL CNT > SEQ-LENGTH OF PARDESSEQ
             CALL "CORBA-SEQUENCE-ELEMENT-GET" USING
                   TEMP-BUF
                   CNT
                   ELEMENT-TYPE
             SET  ADDRESS OF TMP-PARA TO ELEMENT-TYPE
*----  Reference for parameter name   ----
             MOVE FUNCTION LENG (DSP-BUF) TO STRING-LENGTH
             CALL "CORBA-STRING-GET" USING
                 NAME OF TMP-PARA
                 STRING-LENGTH
                 DSP-BUF
             DISPLAY "NAME=" DSP-BUF
*----  Reference for parameter type   ----
             CALL "CORBA-TYPECODE-KIND" USING
                  IDL-TYPE OF TMP-PARA
                  ENV
                  TMP-KIND
             DISPLAY "TYPECODE KIND=" TMP-KIND
*----  Parameter mode (IN/OUT/INOUT)   ----
             DISPLAY "PARA MODE=" IDL-MODE OF TMP-PARA
     END-PERFORM.
```

Java

```
org.omg.CORBA.OperationDef      opef;
org.omg.CORBA.ContainedPackage.Description     desc;
org.omg.CORBA.OperationDescription   c;
org.omg.CORBA.ParameterDescription   paraseq[];

//OperationDef object interface information acquisition
opef = org.omg.CORBA.OperationDefHelper.narrow( obj );
desc = opef.describe();
//fetching of OperationDef object specific information structure
org.omg.CORBA.Any opeany = desc.value;
c = ( org.omg.CORBA.OperationDescription )
   org.omg.CORBA.OperationDescriptionHelper.extract( opeany );
//reference for parameter sequence information
paraseq = c.parameters;
//Reference for the number of parameters
inoutcc.output( "Parameter number = " + paraseq.value.length );
for( i = 0; i < paraseq.length; i++ ) {
   parades = paraseq[i];
   //reference for parameter name
   inoutcc.output( "para name = " + parades.name );
   //reference for parameter type
```

```
   typecode = parades.type;
   kind = typecode.kind().value();
   inoutcc.output( "TypeCode_kind = " + kind );

   //parameter mode
   inoutcc.output( "para mode = " + parades.mode.value() );
}
```

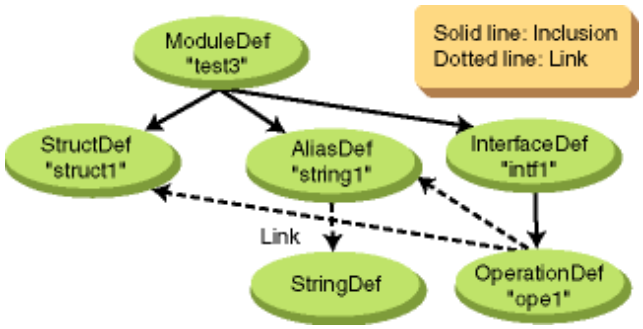
9.4.4.2 Example 2: Retrieving Information on StructDef and AliasDef Objects

This example shows the programming necessary to acquire information about a StructDef object (structure definition) and AliasDef object (TypeDef definition information) for the IDL definition shown below.

```
module test3 {
   struct  struct1{
      long  para1;
      short para2;
   };
   typedef  string<50>  string1;
   interface intf1  {
      long  ope1(in  struct1  struct1_data, In string1 string1_data);
      };
};
```

Information about the objects shown in the following figure is stored in the Interface Repository service.

Figure 9.5 Acquiring Information on StructDef and AliasDef Objects



Retrieving Object References

Retrieve the object references for the StructDef and AliasDef objects using the OperationDef object parameter information.

C

```
CORBA_Repository              intf_rep;
CORBA_Contained               intfobj;
CORBA_Contained_Description   *ope_des;
CORBA_OperationDescription    *o;
CORBA_TCKind                  kind;
CORBA_ParDescriptionSeq       *parameters;
CORBA_ParameterDescription    *prmp;
CORBA_StructDef               StructObj;
CORBA_AliasDef                AliasObj;

/*interface repository object reference acquisition*/
intf_rep = CORBA_ORB_resolve_initial_references(
      orb,CORBA_ORB_ObjectId_InterfaceRepository, &env);

/*object search*/
intfobj = CORBA_Repository_lookup_id(intf_rep, "IDL:test1/intf1/ope1:1.0", &env);

/*OperationDef object interface information acquisition*/
```

```
ope_des = CORBA_OperationDef_describe(intfobj,&env);
exception_check("CORBA_OperationDef_describe");

/*fetching of OperationDef object specific information structure*/
o = (CORBA_OperationDescription *)ope_des->value._value;

parameters = &(o->parameters);   /*reference for parameter sequence information*/
for(k=0,prmp=parameters->_buffer; k<parameters->_length; k++){
   /*reference for parameter type*/
   kind = CORBA_IRObject__get_def_kind(prmp[k].type_def,&env);
   if(kind == CORBA_dk_Struct)
      /*setting of StructDef object reference in Struct0bj*/
      StructObj = (CORBA_StructDef)prmp[k].type_def;
   else if(kind == CORBA_dk_Alias)
      /*setting of AliasDef object reference in Alias0bj*/
      AliasObj  = (CORBA_AliasDef)prmp[k].type_def;
   }
```

C++

```
CORBA::Object_ptr              obj;
CORBA::Repository_ptr          intf_rep;
CORBA::Contained_ptr           intfobj;
CORBA::OperationDef_ptr        opef;
CORBA::Contained::Description  *desc;
CORBA::OperationDescription    *c;
CORBA::IRObject_ptr            irobj;
CORBA::StructDef_ptr           StructObj;
CORBA::AliasDef_ptr            AliasObj;
CORBA::IDLType_ptr             idltypep;

   //interface repository object reference acquisition
   obj = orb->resolve_initial_references(
      CORBA_ORB_ObjectId_InterfaceRepository, *env );
   //conversion to CORBA::Repository class
   intf_rep = CORBA::Repository::_narrow( obj );

   //object search
   intfobj = intf_rep->lookup_id( "IDL:test3/intf1/ope1:1.0", *env );
   if(intfobj == CORBA_OBJECT_NIL){
      //The object is absent.
      printf("Not Found Object\n");
      exit(1);
      }

   //OperationDef object interface information acquisition
   opef = CORBA::OperationDef::_narrow(intfobj);
   desc   = opef->describe(*env);

   //fetching of OperationDef object specific information structure
   CORBA::Any *opeany = (CORBA::Any*)(desc->value);
   c = (CORBA::OperationDescription *)opeany->value();

   //reference for parameter sequence information
   CORBA::ParDescriptionSeq *parseq = (CORBA::ParDescriptionSeq*)c->parameters;
   for( int n = 0;n < parseq->length(); n++){
      CORBA::ParameterDescription *parades =
 (CORBA::ParameterDescription*)(*parseq)[n];
   //reference for parameter type
      objvar = (CORBA::Object_var*)&(parades->type_def);
      idlobj = (CORBA::Object_ptr)(*objvar);
      idltypep = CORBA::IDLType::_narrow( idlobj );
      kind  = idltypep->def_kind(*env);
```

```
     if(kind == CORBA::dk_Struct)
        //setting of StructDef object reference in Struct0bj
        StructObj = CORBA::StructDef::_narrow(idltypep);
      else if(kind == CORBA::dk_Alias)
        //setting of AliasDef object reference in Alias0bj
        AliasObj  = CORBA::AliasDef::_narrow(idltypep);
   }
```

COBOL Windows32/64 Solaris32/64

```
*########## Interface repository object search ##########
       MOVE FUNCTION LENG ("IDL:test3/intf1/ope1:1.0") TO STRING-LENGTH.
       MOVE "IDL:test3/intf1/ope1:1.0" TO REP-BUF.
       CALL "CORBA-STRING-SET" USING
                  REPOSITORYID
                  STRING-LENGTH
                  REP-BUF.
       CALL "CORBA-REPOSITORY-LOOKUP-ID" USING
                  OBJ
                  REPOSITORYID
                  ENV
                  INTF-INTF.
       CALL "CORBA-FREE" USING REPOSITORYID.
*--- Checking whether object is present ---
       CALL "CORBA-OBJECT-IS-NIL" USING
                  INTF-INTF
                  ENV
                  TMP-RESULT.
       IF TMP-RESULT = CORBA-TRUE
*--- The object is absent. ---
            DISPLAY "Null Object Reference"
            GO TO MAIN-END
       END-IF.

*########## OperationDef object interface information acquisition ##########
     MOVE INTF-INTF TO OBJ.
     CALL "CORBA-OPERATIONDEF-DESCRIBE" USING
                  OBJ
                  ENV
                  LINK-BUF.
     MOVE "CORBA-OPERATIONDEF--DESCRIBE" TO MESS.
     PERFORM ENV-CHECK.
*--- Fetching of OperationDef object specific information structure ---
     SET ADDRESS OF LINK-CONDES TO LINK-BUF.
     MOVE IDL-VALUE OF LINK-CONDES TO TMP-ANY.
     MOVE ANY-VALUE OF TMP-ANY TO TEMP-BUF.
     SET ADDRESS OF TMP-OPR TO TEMP-BUF.
     MOVE FUNCTION ADDR(PARAMETERS OF TMP-OPR) TO TEMP-BUF.
     SET ADDRESS OF PARDESSEQ TO TEMP-BUF.
*---- Reference for parameter sequence information ---
     IF SEQ-LENGTH OF PARDESSEQ = 0
*---- No parameter is present. ---
          DISPLAY "PARA-LENGTH = 0"
     END-IF.
     PERFORM TEST BEFORE
             VARYING CNT FROM 1 BY 1
                 UNTIL CNT > SEQ-LENGTH OF PARDESSEQ
             CALL "CORBA-SEQUENCE-ELEMENT-GET" USING
                   TEMP-BUF
                   CNT
                   ELEMENT-TYPE
             SET  ADDRESS OF TMP-PARA TO ELEMENT-TYPE
```

```
             MOVE FUNCTION LENG (DSP-BUF) TO STRING-LENGTH
*--- Reference for parameter type ---
             CALL "CORBA-TYPECODE-KIND" USING
                  IDL-TYPE OF TMP-PARA
                  ENV
                  TMP-KIND
             MOVE TMP-KIND TO TYPE-KIND
             IF  CORBA-TK-STRUCT OF  TYPE-KIND
*--- Setting of StructDef object reference in Struct0bj ---
                  MOVE TYPE-DEF OF TMP-PARA TO STRUCT-OBJ
             ELSE
                  IF  CORBA-TK-ALIAS OF TYPE-KIND
*--- Setting of AliasDef object reference in Alias0bj ---
                        MOVE TYPE-DEF OF TMP-PARA TO  ALIAS-OBJ
                  END-IF
             END-IF
     END-PERFORM.
```

Java

```
org.omg.CORBA.Object                        Obj;
org.omg.CORBA.Repository                    Rep;
org.omg.CORBA.Object                        obj;
org.omg.CORBA.OperationDef                  opef;
org.omg.CORBA.ContainedPackage.Description  desc;
org.omg.CORBA.OperationDescription          c;
org.omg.CORBA.ParameterDescription          paraseq[];
org.omg.CORBA.ParameterDescription          parades;
int                                         kind;
org.omg.CORBA.IDLType                       idltype;
org.omg.CORBA.AliasDef                      AliasObj;
org.omg.CORBA.StructDef                     StructObj;

//interface repository object reference acquisition
Obj = Orb.resolve_initial_references( "InterfaceRepository" );
//conversion to RepositoryHelper class
Rep = org.omg.CORBA.RepositoryHelper.narrow( Obj );
//object search
obj = Rep.lookup_id( "IDL:test3/intf1/ope1:1.0" );
//OperationDef object interface information acquisition
opef = org.omg.CORBA.OperationDefHelper.narrow( obj );
desc = opef.describe();

//fetching of OperationDef object specific information structure
org.omg.CORBA.Any opeany = desc.value;
c = ( org.omg.CORBA.OperationDescription )
   org.omg.CORBA.OperationDescriptionHelper.extract( opeany );
//reference for parameter sequence information
paraseq = c.parameters;
for( i = 0; i < paraseq.length; i++ ) {
   //reference for parameter type
   parades = paraseq[i];
   idltype = org.omg.CORBA.IDLTypeHelper.narrow( parades.type_def );
   kind = idltype.def_kind();
   if( kind == org.omg.CORBA.DefinitionKind.dk_Struct.value() ) {
      //setting of StructDef object reference in Struct0bj
      StructObj = org.omg.CORBA.StructDefHelper.narrow( obj );
   } else if( kind == org.omg.CORBA.DefinitionKind.dk_Alias.value() ) {
      //setting of AliasDef object reference in Alias0bj
      AliasObj = org.omg.CORBA.AliasDefHelper.narrow( stcf );
   }
}
```

Retrieving StructDef Object Member Information

Structure member information (identifying name, type code) is posted in a list structure by specifying the object reference of the StructDef object and issuing the members method.

C

```
CORBA_StructMemberSeq      *members;
CORBA_StructMember         *str;

/*report of structure member information in list format*/
members = CORBA_StructDef__get_members(StructObj,&env);

/*structure member analysis*/
for(i=0,str=members->_buffer; i<members->_length;i++){
   /*reference for identification name*/
   printf("Identifier   : %s\n",str[i].name);
   /*reference for type code*/
   printf("TypeCode_kind   : %d\n",CORBA_TypeCode_kind(str[i].type, &env));
}
```

C++

```
CORBA::StructDef_ptr     strobj;
CORBA::StructMemberSeq   *structseq;
CORBA::StructMember      *structmem;

//report of structure member information in list format
strobj = CORBA::StructDef::_narrow(StructObj);
structseq = strobj->members(*env);

for(int y=0; y < structseq->length(); y++) {
   //reference for identification name
   structmem = (CORBA::StructMember*)(*structseq)[y];
   printf( "name = %s\n", (char *)structmem->name );
   //reference for type code
   CORBA::TypeCode
   *typecode2 = (CORBA::TypeCode *)(structmem->type);
   kind = typecode2->kind(*env);
   printf("TypeCode_kind = %d\n",kind);
}
```

COBOL Windows32/64 Solaris32/64

```
*##########  Report of structure member information in list format ##########
   CALL "CORBA-STRUCTDEF--GET-MEMBERS" USING
                STRUCT-OBJ
                ENV
                LINK-BUF.
   SET ADDRESS OF STRMEMSEQ TO LINK-BUF.
   IF SEQ-LENGTH OF STRMEMSEQ = 0
        DISPLAY "STRMEM-LENGTH = 0"
   END-IF.
   PERFORM TEST BEFORE
           VARYING CNT FROM 1 BY 1
               UNTIL CNT > SEQ-LENGTH OF STRMEMSEQ
           CALL "CORBA-SEQUENCE-ELEMENT-GET" USING
                 LINK-BUF
                 CNT
                 ELEMENT-TYPE

           SET  ADDRESS OF STRMEM TO ELEMENT-TYPE
*---        Reference for identification name ---
           MOVE FUNCTION LENG (DSP-BUF) TO STRING-LENGTH
```

```
           CALL "CORBA-STRING-GET" USING
               NAME OF STRMEM
               STRING-LENGTH
               DSP-BUF
           DISPLAY "NAME=" DSP-BUF
*---        Reference for type code ---
           CALL "CORBA-TYPECODE-KIND" USING
                IDL-TYPE OF STRMEM
                ENV
                TMP-KIND
           DISPLAY "TYPECODE KIND=" TMP-KIND
   END-PERFORM.
```

Java

```
org.omg.CORBA.StructDef        strobj;
org.omg.CORBA.StructMember     structseq[];
org.omg.CORBA.StructMember     structmem;
org.omg.CORBA.TypeCode         typecode;
int tkind;

//report of structure member information in list format
strobj = org.omg.CORBA.StructDefHelper.narrow( StructObj );
structseq = strobj.members();
for( i = 0; i < structseq.length; i++ ) {
   //reference for identification name
   structmem = structseq[i];
   inoutcc.output( "name = " + structmem.name );
   //reference for type code
   typecode = structmem.type;
   tkind = typecode.kind().value();
   inoutcc.output( "TypeCode_kind = " + tkind );
}
```

Retrieving the Object Reference of a StringDef Object

The object reference of a StringDef object is posted by specifying the object reference of an AliasDef object, and issuing the original_type_def method.

C

```
CORBA_IDLType  originalobj;
/* StringDef object of object reference acquisition */
originalobj = CORBA_AliasDef__get_original_type_def(AliasObj,&env);
```

C++

```
CORBA::IDLType_ptr  originalobj;
//StringDef object of object reference acquisition
originalobj = AliasObj->original_type_def(*env);
```

COBOL Windows32/64 Solaris32/64

```
*########## StringDef object of object reference acquisition ##########
     CALL "CORBA-ALIASDEF--GET-ORIGINAL-TYPE-DEF" USING
                  ALIAS-OBJ
                  ENV
                  IDLTYPE.
```

Java

```
CORBA.IDLType    originalobj;
// StringDef object of object reference acquisition
originalobj = AliasObj.original_type_def();
```

Retrieving StringDef Object Information

Maximum string length is posted by specifying the object reference of the StringDef object and issuing the bound method.

C

```
CORBA_unsigned_long bound;
/* acquisition of the maximum string length of StringDef object */
bound = CORBA_StringDef__get_bound(originalobj,&env);
```

C++

```
CORBA::ULong      len;
CORBA::StringDef_ptr   stringobj;
//acquisition of the maximum string length of StringDef object
stringobj = CORBA::StringDef::_narrow(originalobj);
len = stringobj->bound(*env);
```

COBOL Windows32/64 Solaris32/64

```
*--- acquisition of the maximum string length of StringDef object ---
     MOVE IDLTYPE TO TMP-OBJ.
     CALL "CORBA-STRINGDEF--GET-BOUND" USING
                  TMP-OBJ
                  ENV
                  BOUND.
```

Java

```
org.omg.CORBA.StringDef  stringobj;
int  len;
//acquisition of the maximum string length of StringDef object
stringobj = org.omg.CORBA.StringDefHelper.narrow( originalobj );
len = stringobj.bound();
```

Chapter 10 CORBA Programming

This chapter explains the technique of developing the CORBA application.

10.1 Factory

A user program of server application implements a Factory interface.

Add the following interface definition in the IDL file. It is available only for C++ mapping.

IDL Definition

```
interface factory {
      Object create_obj();   /* Define a method which creates an ObjectReference */
};
```

In this section, following the IDL definition below, the implementation of Factory is described.

```
module ODsample {
        interface intf1 {
                attribute long      x;
                attribute string    y;
                void destroy();
        };
        interface factory {
                intf1 create_obj();
        };
  };
```

10.1.1 Server Application Programming

Creating an ObjectReference using Factory interface is described here.

Initialization/Activation of a Server Application

Get interface information and implementation information. Create an ObjectReference.

1. Initialization

   Initialize a server application by calling CORBA::ORB_init(), and CORBA::ORB::BOA_init() function.

2. Getting Implementation Information

   Get implementation information of the Factory interface. Call FJ::ImplementationRep::lookup_id() function specifying the ImplementationRepositoryID (_IMPL_"module name"_factory) of the Factory interface.

3. Getting Interface Information

   Get the interface information other than the Factory.

4. Activating a Server Application

   Call CORBA::BOA::impl_is_ready() or CORBA::BOA::obj_is_ready() function specifying implementation information got in 2. All of the interfaces defined in the IDL file are activated.

   ```
       boa->impl_is_ready( impl, env );
   ```

An example of main() function getting implementation and interface information is shown here.

```
  static  CORBA::ORB_ptr               orb;
  static  CORBA::BOA_ptr               boa;
  static  CORBA::Repository_ptr        intf_rep;
  static  FJ::ImplementationRep_ptr    impl_rep;
  static  CORBA::InterfaceDef_ptr      intf1;
  static  CORBA::ImplementationDef_ptr impl;
```

```
static  CORBA::Object_ptr            o;
static  CORBA::Environment           env;
CORBA::ReferenceData                 id;

int
main( int argc, char *argv[] )
{
      int        current_argc = argc;
      char       buf[128];

        // Initialization
        orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, env );
        boa = orb->BOA_init( current_argc, argv, CORBA_BOA_OAid, env );

        // Getting implementation information
        o = orb->resolve_initial_references(
                   CORBA_ORB_ObjectId_ImplementationRepository, env );
        impl_rep = FJ::ImplementationRep::_narrow( o );
        o = impl_rep->lookup_id( _IMPL_ODsample_factory, env );
        impl = CORBA::ImplementationDef::_narrow( o );

        // Getting interface information
        o = orb->resolve_initial_references(
                   CORBA_ORB_ObjectId_LightInterfaceRepository, env );
        intf_rep = CORBA::Repository::_narrow( o );
                o = intf_rep->lookup_id( _INTF_ODsample_intf1, env );
                intf1 = CORBA::InterfaceDef::_narrow( o );
        // Activating the server
        boa->impl_is_ready( impl, env );
}
```

Implementation Function of Factory Interface

Call CORBA::BOA::create() function specifying implementation and interface information got in (1). It creates an ObjectReference.

```
    // create_obj method implementation
    ODsample::intf1_ptr
      ODsample_factory_impl::create_obj( CORBA::Environment &env )
         throw( CORBA::Exception )
      {
            // Calling BOA::create and creating an ObjectReference
         CORBA::Object_ptr new_obj = boa->create( id, intf1,  impl, env );
            ODsample::intf1_ptr fac = ODsample::intf1::_narrow( new_obj );
            CORBA::release( new_obj );
            return( fac );
      }
```

Registering to NamingService

Execute *OD_or_adm* command to register the ObjectReference of the Factory interface to NamingService. Specify the InterfaceRepositoryID of the Factory interface with -c option.

```
OD_or_adm -c IDL:ODsample/factory:1.0 -n ODsample::factory
```

10.1.2 Client Application Programming

This section provides information on client application programming.

Getting the ObjectReference of the Factory Interface

Get the ObjectReference of the Factory interface from NamingService.

```
    CosNaming::Name_ptr           name;
    CosNaming::NameComponent_var  *name_component;

    CORBA::Object_ptr
           obj = orb->resolve_initial_references( CORBA_ORB_ObjectId_NameService,env );

    CosNaming::NamingContext_ptr
                  NamingContext_obj = CosNaming::NamingContext::_narrow( obj );
       CORBA::release( obj );

    name_component = CosNaming::Name::allocbuf(1);
    name_component[0]->id = (const CORBA::Char *)"ODsample::factory";
    name_component[0]->kind = (const CORBA::Char *)"";
    name = new CosNaming::Name(1, 1, name_component,CORBA_TRUE);
    obj = NamingContext_obj->resolve( *name, env );
    CORBA::release( NamingContext_obj );
    delete name;
```

Calling a Method Implemented in the Factory Interface

Call create_obj() method specifying the ObjectReference got from NamingService. When registering a server application to ImplementationRepository by calling *OD_impl_inst* command, specify "iswitch=ON" in a definition file ("*OD_impl_inst* -ax"). When a method is called for the first time, an instance data is created for each client in the server application.

```
     // Getting inferface "intf1" ObjectReference
     ODsample::factory_ptr ap = ODsample::factory::_narrow( obj );
     ODsample::factory_var av = ap;

     ODsample::intf1_var target = av->create_obj( env );
```

10.1.3 Implementing Private Area

A programming example using private area is shown below. When attribute is defined in IDL, it is mapped to private data area in implementation class. The server application controls the private area. For details of attribute, refer to "Mapping Attribute Declaration (Attribute)" in the "C++ Programming Guide".

Processing in a Server Application

An implementation example of a server application which sets and gets a private area is shown below.

```
// Setting attribute x (fixed length data)
void
  ODsample_intf1_impl::x( CORBA::Long l, CORBA::Environment &env )
        throw( CORBA::Exception )
  {
        __x = l;
  }

// Getting attribute x (fixed length data)
CORBA::Long
  ODsample_intf1_impl::x( CORBA::Environment &env )
        throw( CORBA::Exception )
  {
        return( __x );
  }

// Setting attribute y (variable length data)
void
  ODsample_intf1_impl::y( const CORBA::Char* s, CORBA::Environment &env )
        throw( CORBA::Exception )
  {
        cout << " s = [" << s << "]" << endl;
```

```
        __y = (const CORBA::Char *)s;
  }

// Getting attribute y (variable length data)
CORBA::Char *
  ODsample_intf1_impl::y( CORBA::Environment &env )
        throw( CORBA::Exception )
  {
        CORBA::Char *tmp;

        tmp = CORBA::string_alloc(strlen(__y));
        strcpy( tmp, __y );
        cout << tmp << endl;
        return( tmp );
  }
```

Also private data can be set directly in header files generated by IDL compiler. When doing so, if you set variable length data, like string and etc., you need to release allocated area. Refer to "10.1.5 Programming for Termination Processing".

```
class intf1: public virtual CORBA::Object
{
public:
     static intf1_ptr          _duplicate( intf1_ptr );
     static intf1_ptr          _narrow( CORBA::Object_ptr );
     static intf1_ptr          _nil();

     // Attribute Method ( for _get)
     virtual CORBA::Long x(
     ....

private:
     intf1 ( const intf1&);
     void operator=( const intf1& );

     // User defined area
     long user_var;
     char *data;
}
```

Processing in a Client Application

An example using attributes is shown below.

```
// Setting attribute x value (fixed length data)
intf1_obj->x( 3, env );

// Getting attribute x value (fixed length data)
CORBA::Long result = intf1_obj->x( env );

// Setting attribute y value (variable length data)
CORBA::Char *dummy = CORBA::string_alloc(7);
strcpy( dummy, "testabc" );
target->y( dummy, env );

// Getting attribute y value (variable length data)
CORBA::Char *data = target->y( env );
cout << "data = " << data << endl;
```

10.1.4 Initializing Private Data for Each Client

An example is shown below in which how values for initializing private data are set for each client when generating a Factory object.

IDL Definition

Add a parameter to be used as the identifier to the create_obj method that generates objects of the Factory interface. In this example, a string parameter is used and a user name using a client application is passed.

```
module ODsample {
    interface intf1 {
        attribute long      x;
        attribute string    y;
        void destroy();
    };
    interface factory {
        intf1 create_obj( in string user_name );
    };
};
```

(1) Server Application Processing

Factory Interface Implementation Function

Set the parameter "user_name" shown in the above IDL definition as identification information (ReferenceData) of an object.

```
// Implement the create_obj method
ODsample::intf1_ptr
    ODsample_factory_impl::create_obj(
        CORBA::Char *user_name, CORBA::Environment &env )
throw( CORBA::Exception ) {
    CORBA::ReferenceData  *id;
    CORBA::Octet          *data;
    CORBA::ULong          len;

    len = strlen( user_name ) + 1;
    data = CORBA::ReferenceData::allocbuf( len );
    strcpy( data, user_name );
    id = new CORBA::ReferenceData( len, len, data, CORBA_FALSE );
    CORBA::Object_ptr new_obj = boa->create( id, intf1,  impl, env );
    ODsample::intf1_ptr fac = ODsample::intf1::_narrow( new_obj );
    CORBA::release( new_obj );
    delete id;
    delete data;
    return( fac );
}
```

Skeleton Change

The constructor is generated on the skeleton generated by the IDL compiler, in the following way. Add processing to initialize user_var, data set directly in the header file, as shown below.

```
ODsample_intf1_impl::ODsample_intf1_impl()
{
    // Initialize private data
    user_var = 0;
    data = NULL;
}
```

Processing Example of the Method Implementation Function

If data in private data is NULL when each method is invoked, initialization is carried out, assuming that initialization of private data for each client is not carried out. The following example shows an implementation function of x reference method in the IDL definition.

```
// attribute x reference (fixed length data)  CORBA::Long
ODsample_intf1_impl::x( CORBA::Environment &env )
    throw( CORBA::Exception )
{
    if (this->data == NULL)
```

```
        intf1_initialize( this );
    return( __x );
}
```

Extract the identification information (ReferenceData) of an object and set an initial value depending on the user name in the following way:

```
// Initialize private data for each client
void
intf1_initialize( ODsample_intf1_impl obj )
{
    CORBA::Environment     env;
    CORBA::ReferenceData   *id;
    CORBA::ULong           len, i;
    CORBA::Char            *name;

    id = obj->get_id( &env );
    len = id->length();
    name = CORBA::string_alloc( len );
    for (i = 0; i < len; i++)
        name[i] = id[i];
        if (strcmp(name, ...) == 0) {
            // Initialization depending on the user name
            obj->x = ...;
            obj->y = ...;
            obj->user_var = ...;
            obj->data = name;
        }
           ...
        delete id;
}
```

(2) Client Application Processing

Refer to "10.1.2 Client Application Programming".

10.1.5 Programming for Termination Processing

When dispose is called in a method, the destructor of the implementation class implemented in the skeleton is called. To implement termination processing for an application, modify the skeleton (xxxx_C++_skel.c) generated by IDL compiler.

Processing in a Server Application

Issue CORBA::BOA::dispose() function to the pointer. An example of termination processing method is shown below.

```
// Termination processing method
void
ODsample_intf1_impl::destroy( CORBA::Environment &env )
     throw( CORBA::Exception )
{
     try {
          boa->dispose( this, env );
     }
     catch( CORBA::SystemException &se) {
          throw( se );
     }
}
```

Modifying the Skeleton

A destructor is generated in a skeleton as follows. Add termination processing as needed. If you use a variable length private data, add releasing processing.

```
ODsample_intf1_impl::~ODsample_intf1_impl()
{

// Releasing private data
     if (data)
          delete data;

// Termination processing
     ....
}
```

10.1.6 Registering the Processing Function Used when Disconnecting from a Client

If connection is cut off due, for example, to a client error, the destructor of the implementation class implemented in the skeleton is automatically invoked. If any operation should be performed before the destructor is executed, a processing function to carry out the desired operation can be registered.

The function registered here is executed just before the destructor is invoked.

(1) Processing Function Format

The format of the processing function is shown below. Its argument is a void pointer and this pointer is passed when the destructor is invoked. There is no return value.

```
  void exit_func( void *this_pointer );
```

(2) Processing Function Registration

Register the processing function using CORBA::ORB::reg_exit_function(). Carry out the registration between CORBA::ORB_init() and CORBA::BOA::impl_is_ready(). A registration example is shown below:

```
  void exit_func( void *this_pointer )
  {
      // Processing contents. An example is shown later.
  }

  int
  main( int argc, char *argv[] )
  {
      int    current_argc = argc;
      .....

      // Initialize
      orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, env );

      ....

      // Register the processing function
      orb->reg_exit_function( orb, _INTF_ODsample_intf1, exit_func, env );

      ....

      // Activate the server
      boa->impl_is_ready( impl, env );
  }
```

(3) Contents Example of the Processing Function

A simple warning is output along with private data (However, it is necessary to make the exit_func function a friend function of the ODsample_intf1_impl class, to reference the private members).

```
void exit_func( void *this_pointer )
{
    ODsample_intf1_impl *ptr = (ODsample_intf1_impl *)this_pointer;
}
```

(4) Note

If connection with the client is cut off after CORBA::BOA::dispose() is issued, the registered processing function will not be invoked. Also in this case, execute the processing function before issuing CORBA::BOA::dispose() so that the processing function is executed.

Example

```
void
ODsample_intf1_impl::destroy( CORBA::Environment &env )
    throw( CORBA::Exception )
{
    try {
        exit_func( this ); // Invoke explicitly the registered function
        boa->dispose( this, env );
    }
    catch( CORBA::SystemException &se ) {
        throw( se );
    }
}
```

10.2 Object to Process Bind Function

In server applications in which process concurrency is two or more, allocation of client requests to processes is unpredictable. The Object to Process Bind function allows assignment of requests to specific server process.

Only C++ server applications can use Object to Process Bind. The client application can be written in any language.

For examples of programming using process bind, refer to "Appendix I Example of Session Management using the Object to Process Bind Function".

10.2.1 Definition Information

To use Object to Process Bind, set the 'iswitch' parameter to "object" when registering the implementation repository. If Session Timeout is used, the 'ssn_timeout' parameter must also be set. For details, refer to "OD_impl_inst" in the "CORBA Service Operation Commands" chapter of the Reference Manual (Command Edition).

The number of process to object bind relationships that can be registered in a system is set in the 'max_bind_instances' parameter of the CORBA Service operating environment file (config). For details, refer to "config" in the "CORBA Service Environment Definition" appendix of the Tuning Guide.

10.2.2 API Used by Object to Process Bind Function

The following API is used by the Object to Process Bind Function:

- CORBA::ORB::bind_object()

  This registers the object to process bind relationship.

- CORBA::ORB::unbind_object()

  This cancels the object to process bind relationship.

- CORBA::ORB::set_unbinded_object_rejecting()

  This registers the interface that notifies exceptions for unbound objects.

- CORBA::Object::check_ssn_timeout()

  This checks if a session timeout has occurred.

For details on these API functions, refer to the Reference Manual (API Edition).

10.2.3 Object and Instance Relationship

The CORBA object and C++ instance relationship in the server process is 1:1. For this reason, if the same object is shared by more than one client, they use the same C++ instance on the server side.

The server side C++ instance is created immediately before the method call when the first access is made using the object. It is destroyed when:

- CORBA::ORB::unbind_object() has completed successfully and the bind relationship is cancelled
- A session timeout occurs

When CORBA::ORB::unbind_object() is called, the instance is destroyed only when:

1. The skeleton has returned, and
2. All client access to that instance has been completed

For example, if several clients are accessing the instance when CORBA::ORB::unbind_object() is called, the instance is released after the skeleton returns and the final client access is complete.

Note

If accessed by the client using the session continuation object after releasing the session continuation object instance, behavior depends on whether the object interface was registered using CORBA::ORB::set_unbinded_object_rejecting() as follows:

- If the object interface was not registered

  A new C++ instance is created and the server function is called

- If the object interface was registered

  A system exception is notified to the client

10.2.4 Request Distribution Method

For objects for which a bind relationship has been registered (bound objects) and the 'iswitch' parameter in the server application implementation has been set to "object",, requests are distributed to the process to which the object is bound.

If an unbound object is used and the number of bind relationships registered for each server application process (bind number) is equal, requests are distributed using normal CORBA server application logic.

If an unbound object is used and the bind number is not equal, requests are distributed to the process having the lowest bind number. Requests are queued until a request processing thread becomes available in the process to which it was distributed, even if there are idle threads in other processes.

10.2.5 Session Timeout Function

When the CORBA server application information definition "ssn_timeout" parameter is specified, if there is no access from the client during the specified time, the object to process bind relationship is cancelled.

If the instance object has already been created when session timeout occurs, the destructor is run to destroy the instance. Call CORBA::Object::check_ssn_timeout() in the destructor to check if a session timeout has occurred.

The session timeout value is independent of other timeouts (such as period_receive_timeout).

10.2.6 Ending the Process

Process Ends Abnormally

When an object to process bind relationship has been registered and the process ends, the bind relationship information managed by the CORBA Service is automatically deleted.

WorkUnit is Stopped

During a user session, WorkUnit normal stop using the *isstopwu* command fails. The WorkUnit will stop if synchronous stop or forced stop is used. Bind relationship information registered in the table for session management is automatically deleted.

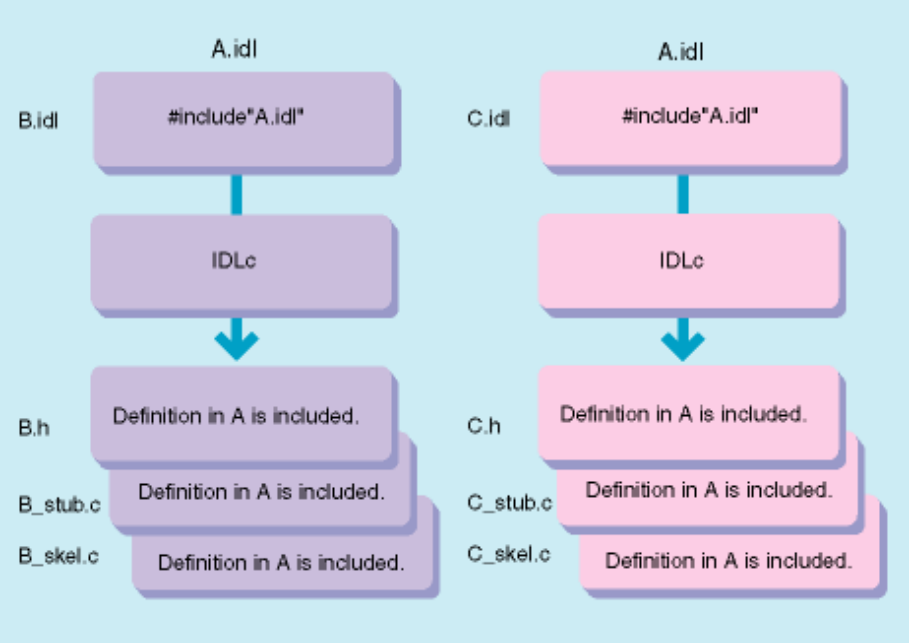
10.3 Compiling Multiple IDL Files

When operating multiple IDL files, a method in which #include is used in the IDL file is available. The following describes problems when using #include and the -noinclude option used as a measure against these problems.

10.3.1 #include Statement

If the same IDL is included in multiple IDLs and compiled with an IDL compiler as shown in the following figure, an include file definition is contained in the language division created using IDL. If a program that includes the same IDL is linked, a link error occurs.

Figure 10.1 Compiling IDL with an #include Statement



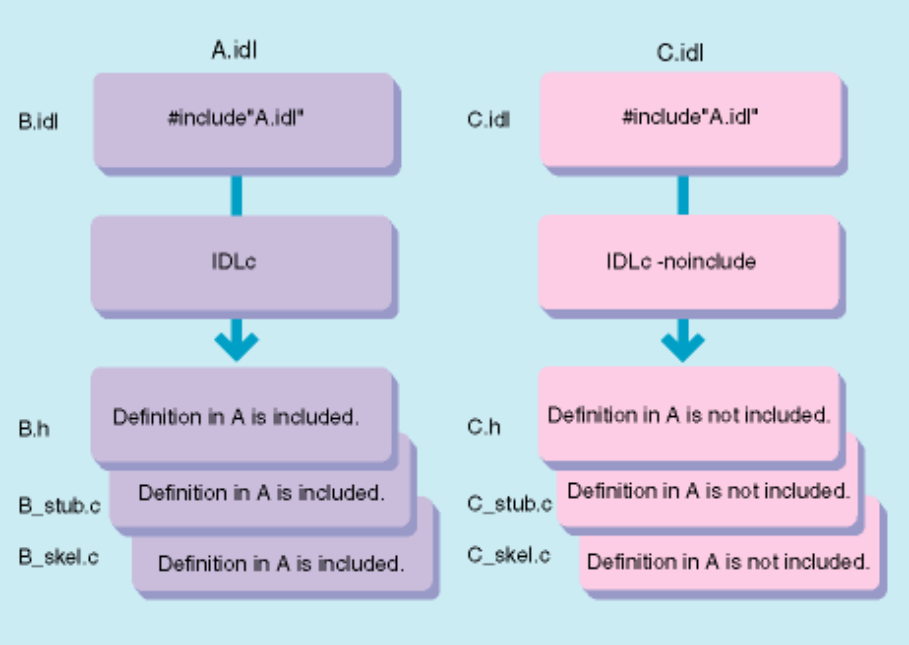
The -noinclude option of the IDL compiler avoids the expansion of the IDL file (hereinafter, referred to as the included file) defined by the #include statement.

When a static skeleton is created, only "#include XX.h" (IDL file named "XX.idl") is written in each header.

In the interface repository, only information on the IDL file (hereinafter, referred to as the IDL file) specified while the IDL compiler is running is included. Information on the included file is not registered in InterfaceRepository.

The following figure shows the mapping results with option -noinclude specified and unspecified.

Figure 10.2 Mapping Results with Option -noinclude Specified and Unspecified



10.3.2 When to Use the -noinclude Function

Specifying the -noinclude option in the following situations can prevent duplicate definitions of the same data:

Mapping

Included file has been compiled with the IDL compiler and a product from its included file is linked.

The same included file is defined in multiple IDL files and products that are linked.

Registering in the Interface Repository

Information from the included file has been registered in the Interface Repository and only information on the IDL file requires registering.

The same included file is defined in multiple IDL files and all IDL definitions are to be registered in the Interface Repository.

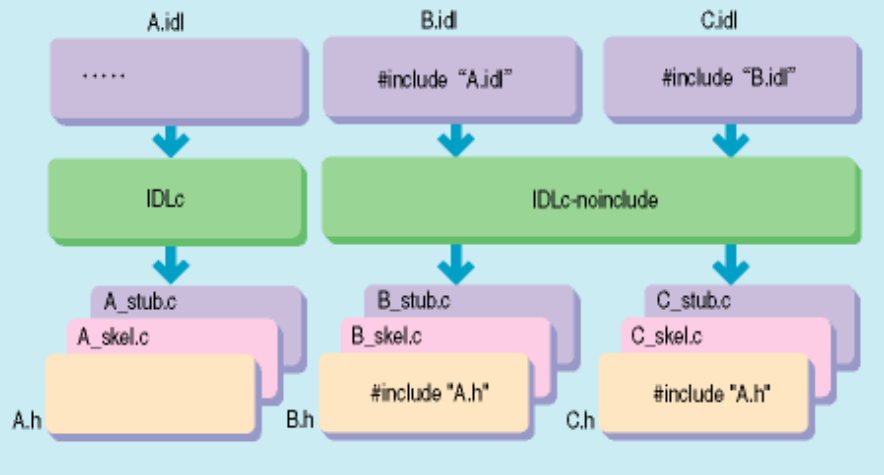
10.3.3 Using -noinclude

Compile IDL to the IDL file and included file separately using the IDL compiler as explained below:

- Compile IDL to the included file without specifying the -noinclude option.
- Compile IDL to the IDL file by specifying the -noinclude option.

The following figure maps the results obtained with the -noinclude option specified and unspecified.

Figure 10.3 Mapping Results with the -noinclude Option



10.3.4 Notes

When compiling an IDL file that includes an #include statement, note the following:

- The IDL compiler analyzes the syntax of an included file. It checks for both syntax errors and conflicts with descriptions in the IDL file. Therefore, correct any syntax errors in an included file before compiling it.
- If the -noinclude option is specified using -R or -a mode, inheritance and reference information in the Interface Repository must be held. Therefore, the IDL compiler checks whether an ID written in an included file has been registered in the Interface Repository. Information on the included file must be registered before compilation.
- The -noinclude option cannot be used with -update/-delete. When an IDL file is to be changed, use -R+-delete to delete information from the Interface Repository. After changing the information, use -R or **-a** to register it again. Definitions in the included file are also deleted from the Interface Repository. Therefore, use -R or -a to register the included file before beginning this processing.

10.3.5 Handling Problems

The following table lists the error messages displayed by the IDL compiler with -noinclude specified.

Table 10.1 -noinclude Error Messages

1	Symptom:	An "identifier redeclaration of ..." error occurs.
	Cause:	1. The definition of an included file conflicts with that of an IDL file. 2. An IDL file definition has already been registered when *-noinclude* is used with -a or -R.
	Handling:	1. Change data such as the ID name to resolve the conflicting definitions. 2. When information on an IDL file is to be changed, use -delete to delete information and restart the processing with IDL compile specified for the included file. This is because *-noinclude* cannot be used with -delete/-update.
2	Symptom:	An "XXX is not installed" error occurs when -R is specified.
	Cause:	The include file is not registered in InterfaceRepository.
	Handling:	Register the include file in the InterfaceRepository in advance.
3	Symptom:	A syntax error other than the above occurs.
	Cause:	The IDL syntax contains an error.
	Handling:	Correct the IDL definition corresponding to the message.

10.4 Multiple Interface Implementation to One Process

This section describes the multiple interface implementation to one process.

10.4.1 Invoking Different Object Methods within a Program

This is not valid for Linux (64 bit).

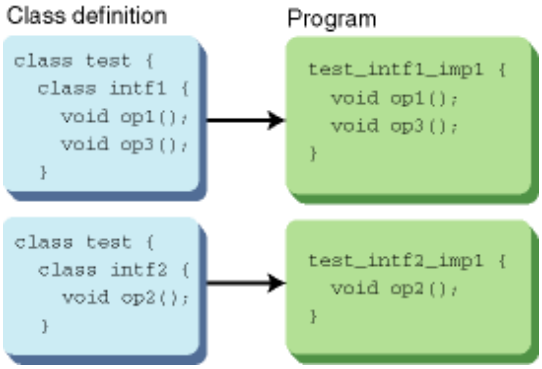
When multiple objects are implemented in the same program, the required programming for invoking methods for these objects is described below.

IDL Example

```
module test {
       interface intf1 {
          void op1();
          void op3();
       };
       interface intf2 {
          void op2();
       };
};
```

In C++, the server application division is mapped for each impl class member function as shown in the following figure.

Figure 10.4 Server Application Mapping



This section explains how to invoke method op3 in the same object from method op1 in test::intf1, and how to invoke method op2 for a different object in the same program.

- Write this->op3() to invoke method op3 in the same object.
- To invoke a method for a different object in the same program, use a new operator to create an impl class instance and invoke that instance. The instance must be deleted using a delete operator after processing is finished.

Example

```
test_intf1_impl::op1()
{
     this->op3();

        obj = new test_intf2_impl();
        obj->op2();
        delete obj;
}
```

10.4.2 Implementing One Interface in One Process

The IDL compiler outputs an implementation and interface repository ID into header files.

IDL Example

```
module mod1 {
     interface intf1 {
          ...
     };
```

```
};
// *.h
#define _IMPL_mod1_intf1     "IDL:mod1/intf1:1.0"
#define _INTF_mod1_intf1     "IDL:mod1/intf1:1.0"
```

Registering and Activating Implementation

Use an *OD_impl_inst* command for registration with the above implementation repository ID.

```
OD_impl_inst -a -r IDL:mod1/intf1:1.0 ...
```

Use the server program to activate the above implementation repository ID.

Example

```
impl = FJ_ImplementationRep_lookup_id( impl_rep,
       _IMPL_mod1_intf1,&env );
       CORBA_BOA_impl_is_ready(boa, impl, &env );
```

10.4.3 Implementing Multiple Interfaces in One Process

The IDL outputs implementations and interface repository IDs into header files.

IDL Example

```
module mod1 {
     interface intf1 {
          ...
     };
};
     module mod2 {
     interface intf2 {
          ...
     };
};
// *.h
#define _IMPL_mod1_intf1     "IDL:mod1/intf1:1.0"
#define _INTF_mod1_intf1     "IDL:mod1/intf1:1.0"
#define _IMPL_mod2_intf2     "IDL:mod2/intf2:1.0"
#define _INTF_mod2_intf2     "IDL:mod2/intf2:1.0"
```

If the above IDL is compiled, two implementation and repository IDs are output. When two interfaces are to be installed in one process, determine one implementation repository ID to expose the process. (A format other than IDL:*/*:1.0 can be used.)

In the following example, implementation repository ID IDL:multi-interfaces:1.0 is assumed to indicate a process where interfaces are to be implemented.

Registering/Activating Implementation

Using the command, display the above process and register using an implementation repository ID.

To install multiple interfaces, define all the interface repository IDs that are to be installed using the definition file with the -ax option.

```
OD_impl_inst -ax def
```

Contents of the def file

```
rep_id             = IDL:multi-interfaces:1.0
type               =..
IDL:mod1/intf1:1.0 =
IDL:mod/intf2:1.0
```

In the server program, activate the implementation repository ID that indicates this process.

```
impl = FJ_ImplementationRep_lookup_id (
     impl_rep,
     "IDL:multi-interfaces:1.0",
     &env );
CORBA_BOA_impl_is_ready(boa, impl, &env );
```

Registering in the Naming Service

Using the *OD_or_adm* command, create object references to be registered in the name service with the repository ID that was specified during registration of the implementation described above.

```
OD_or_adm -a IDL:multi_interfaces:1.0 -c IDL:mod1/intf1:1.0 -n mod1::intf1
OD_or_adm -a IDL:multi_interfaces:1.0 -c IDL:mod1/intf1:1.0 -n mod1::intf2
```

Alternatively, perform the same processing as for the *OD_or_adm* command indicated above but using a server program.

```
impl = FJ_ImplementationRep_lookup_id (
     impl_rep,
     "IDL:multi-interfaces:1.0",
     &env );

intf1 = CORBA_Repository_lookup_id(
     intf_rep,
     _INTF_mod1_intf1,
     &env );
intf2 = CORBA_Repository_lookup_id(
     intf_rep,
     _INTF_mod2_intf2,
     &env );
/*mod1/intf1 object creation*/
obj1 = CORBA_BOA_create( boa, &id, intf1, impl, &env );

/*mod1/intf1 registration in naming service*/
name1._length = name1._maximum = 1;
name1._buffer = &name1_component;
name_component1.id = "mod1::intf1";
name_component1.kind = "";

CosNaming_NamingContext_bind(
     cos_naming,
     &name1,
     obj1,
     &env);

/*mod2/intf2 object creation*/
obj2 = CORBA_BOA_create( boa, &id, intf2, impl, &env );

/*mod2/intf2 registration in naming service*/
name2._length = name2._maximum = 1;
name2._buffer = &name2_component;
name_component2.id = "mod2::intf2";
name_component2.kind = "";

CosNaming_NamingContext_bind(
     cos_naming,
     &name2,
     obj2,
     &env);
```

Accessing from a Client

To access interface mod1/intf1, use an object fetched from the naming service with name mod1::intf1. To access interface mod2/intf2, use an object fetched from the naming service with name mod2::intf2 to issue an access request.

10.5 Application Libraries

When creating an application, you can link objects into one load module or individual objects can be made into so libraries (DLL).

10.5.1 Creating Libraries

This section provides information about creating libraries.

Example Defined in IDL

In the following example, two interfaces are defined in one module.

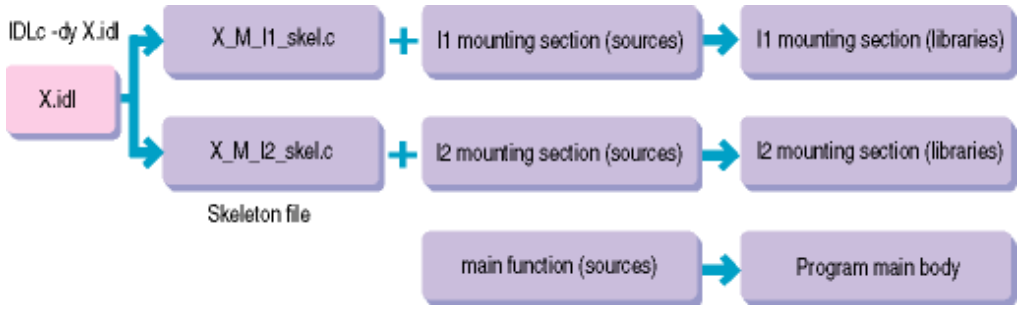
IDL Example

```
[ X.idl ]
    module M {
         interface I1 {
              ...
         };
         interface I2 {
              ...
         };
    };
```

C Mapping

The following figure shows an example of C mapping.

Figure 10.5 Creating Libraries (C Mapping)



(1) Specify the option to create a so application and compile IDL.

```
IDLc-dy idl-file-name
```

(2) Create individual interface implemented libraries (so) from skeleton sources (*_skel.c) generated for interfaces and from sources implemented by interface method functions. Code the sections to be implemented using the same methods as for objects that are not made into libraries (so).

(3) A library is created as follows:

Windows32/64

Visual Specify "DLL (the dynamic link library) preparation" by the project new preparation of C++.

Solaris32/64

```
cc -c *.c ....
cc -G -Kpic -o library-name  object-file(*.o) ...
```

Linux32/64

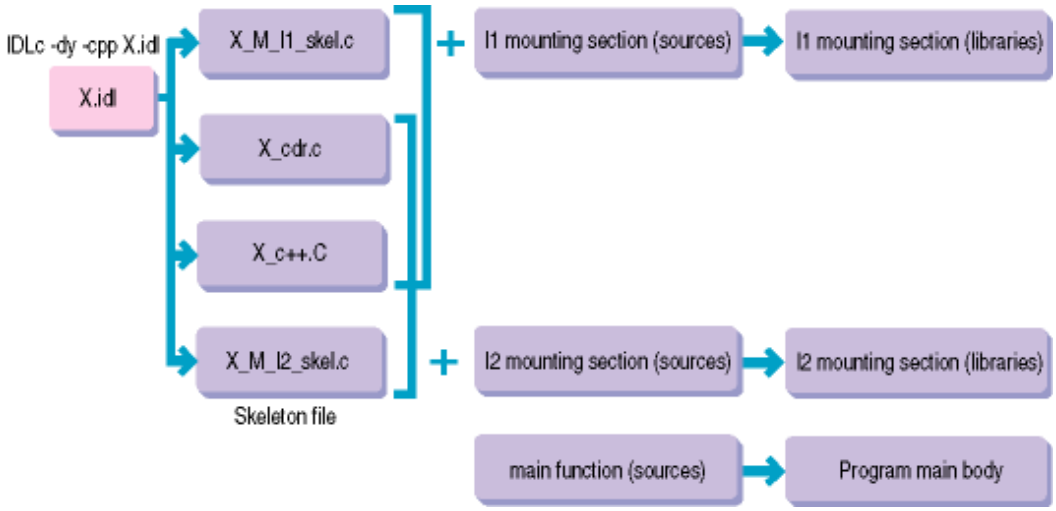
```
gcc -c *.c ....
gcc -shared -fPIC -o library-name  object-file(*.o) ...
```

C++ Mapping

This is not valid for Linux (64 bit).

The following figure shows an example of C++ mapping.

Figure 10.6 Creating Libraries (C++ Mapping in Solaris/Linux)



(1) Specify the option to create a so application and compile IDL.

Windows32/64

```
IDLc -dy -vcpp idl-file-name
```

Solaris32/64 Linux32/64

```
IDLc -dy -cpp idl-file-name
```

(2) Create libraries (*.dll) in which interfaces are mounted from skeleton sources (*_skel_c++.C) generated for interfaces, sources in which the method functions of the interfaces are mounted, CDR sources (*_cdr.c), and class sources (*_c++.C).

Code the sections to be implemented using the same methods as for objects that are not made into libraries .

(3) A library is created as follows:

Windows32/64

Visual Specify "DLL (the dynamic link library) preparation" by the project new preparation of C++.

Solaris32/64

```
cc -G -Kpic -o library-name  object-file(*.o) ...
```

Linux32/64

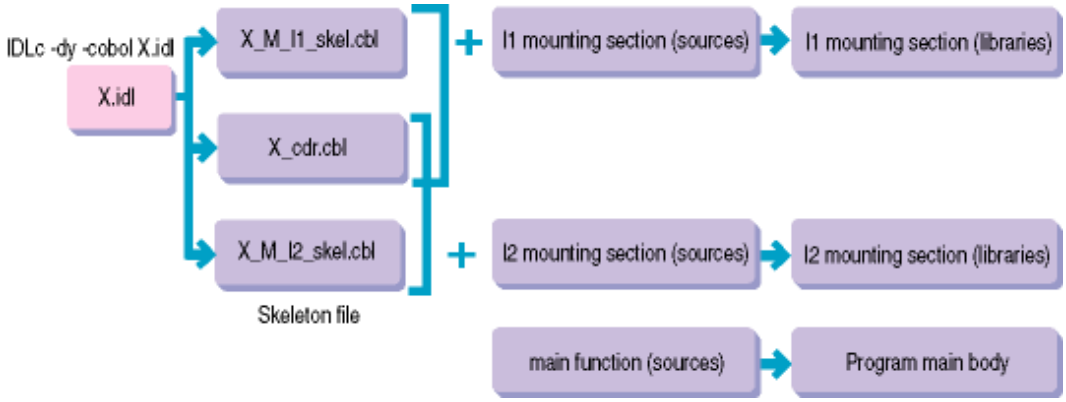
```
g++ -shared -fPIC -o library-name  object-file(*.o) ...
```

COBOL Mapping

This is not valid for Linux (64 bit).

The following figure shows an example of COBOL mapping.

Figure 10.7 Creating Libraries (COBOL Mapping)



(1) Compile IDL.

```
IDLc-cobol idl-file-name
```

(2) Create libraries (*.dll) for implementing individual interfaces from skeleton sources (*_skel.c) corresponding to interfaces, from sources that implement method functions, and CDR sources (*_cdr.cbl).

(3) A library is created as follows:

Windows32/64

Create a library with the Program Manager of the COBOL development environment.

Solaris32/64

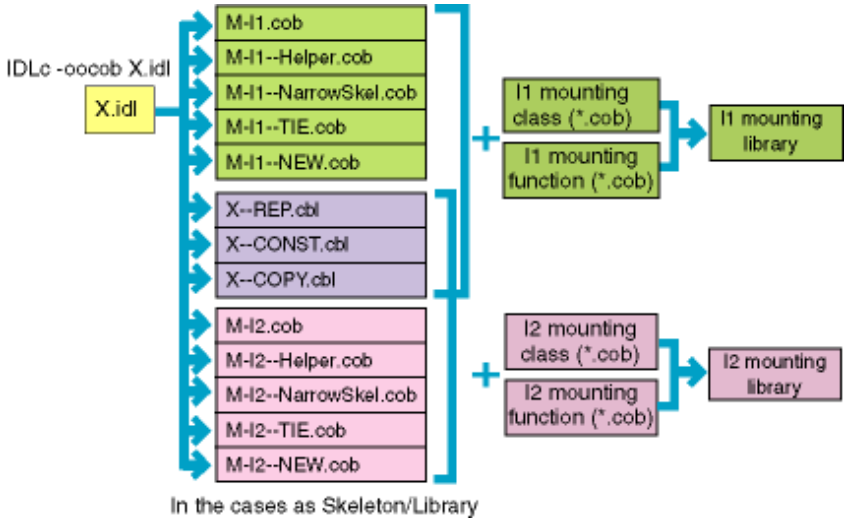
```
cobol -c *.cbl ....
cc -G -Kpic -o library-name  object-file(*.o) ...
```

OOCOBOL Mapping

This is not valid for Linux (64 bit).

The following figure shows an example of OOCOBOL mapping.

Figure 10.8 Creating Libraries (OOCOBOL Mapping)



(1) Compile IDL.

```
IDLc-oocobol idl-file-name
```

(2) Create libraries (*.dll) for implementing individual interfaces from skeleton class (interface name*.con) corresponding to interfaces, from sources that implement method functions from sources that implement method functions, and for the library that is formed from the IDL file (*.cbl).

Registering the Application

Execute the following command to register the so application in the Implementation Repository:

```
OD_impl_inst -ax definition-file
```

Insert items such as the so configuration for each application in the definition file as follows:

```
rep_id          = implementation-repository-id
type            = server-type
library         = program-main-body-path
IDL:M/I1:1.0    = i1-mounting-section-library-path     (*1)
IDL:M/I2:1.0    = i2-mounting-section-library-path
lang            = language-type
```

Note

*1 You must insert "interface-repository-id=library-path[,prefix]" once for each interface. You must specify the prefix specified by the -S option of the IDL compiler.

10.5.2 Inheritance and Libraries of the Interface

This section describes the relationship between a so application and IDL inheritance function.

Example Defined in IDL

In this example, interface B inherits interface A.

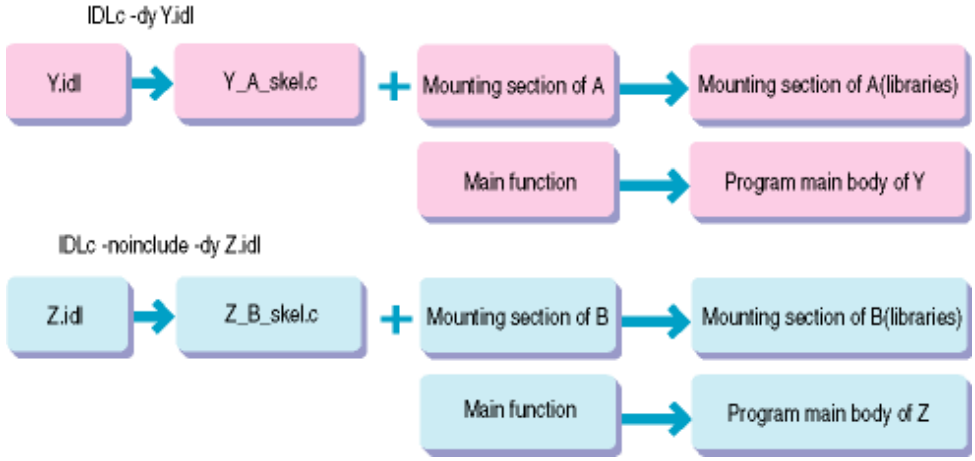
Example

```
 [ Y.idl ]
     interface A {
          ....
     };

[ Z.idl ]
     #include "Y.idl"
     interface B : A {
          ....
     };
```

Creating the Program

The following figure shows an example of creating libraries.

Figure 10.9 Creating Libraries (when an interface is inherited)



(1) The inheritance source IDL file must be compiled in the same way as a file that is not inherited.

```
IDLc -dy [-C|-cpp|-cobol|-oocob] idl-file
```

(2) Specify the -noinclude option in IDL to compile the inheritance destination IDL file:

```
IDLc -dy -noinclude [-C|-cpp|-cobol|-oocob] idl-file
```

(3) Create libraries (so) as implementation sections that correspond to interfaces. Create program bodies.

Registering in the Implementation Repository

Register inheritance source and destination applications as follows:

Inheritance Source

```
OD_impl_inst -ax y-definition-file
```

```
    [y-definition-file]
         rep_id = y-implementation-repository-id
         ...
         IDL:A:1.0=a-mounting-section-library-path
         ....
```

Inheritance Destination

```
OD_impl_inst -ax z-definition-file
```

```
    [z-definition-file]
         rep_id = z-implementation-repository-id
         ...
         IDL:A:1.0=a-mounting-section-library-path               (1)
         IDL:B:1.0=b-mounting-section-library-path,,IDL:A:1.0    (2)
         ...
```

(1) Insert the inheritance source interface definition using the following format:

```
    interface-repository-id = library-path,[prefix]
```

(2) Insert the inheritance destination interface definition using the following format:

```
    interface-repository-id =
         library-path,[prefix],inheritance-source-interface-repository-id
```

If there are multiple inheritance source interfaces, insert multiple interface repository IDs delimited by commas (,).

Example Windows32/64

```
IDL:a:1.0 = C:\server\libA.dll,,IDL:b:1.0,IDL:c:1.0
```

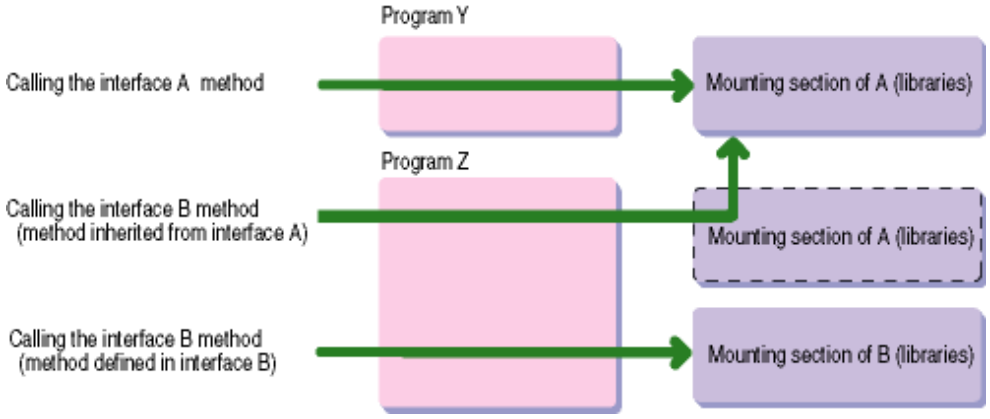
Example Solaris32/64 Linux32/64

```
IDL:a:1.0=/usr/prog/liba.so,,IDL:b:1.0,IDL:c:1.0
```

Method Invoking Image

The following figure shows an invocation method image for a so application created using the IDL inheritance function.

Figure 10.10 Creating an Invocation Method Image



10.5.3 Notes on Creating Libraries

This is not valid for Linux (64 bit).

This section contains notes on creating libraries.

When Interfaces are Interdependent (C++ Mapping)

You must take care when creating various libraries to be implemented across interfaces. In addition, when an interface parameter or data type member to be implemented in one library is used in another interface.

Example 1: IDL Definition

```
interface A {
    ...
};
interface B {
    long  op1(  in  A  param  );      /* Use interface A */
    ...
};
```

When creating the library of interface B, the library name of A must be defined in the -1 option.

Solaris32/64

```
CC -G -Kpic -o B-library-name object-file(*.o)...-1 A-library-name
```

Linux32/64

```
gCC -shared -fPIC -o B-library-name object-file(*.o)...-1 A-library-name
```

When executing the application, the A library must be declared in environment variable LD_LIBRARY_PATH.

Example 2: IDL-Specified Definition

```
interface C {
     long  op1( in D param );     /* Use interface D */
```

```
};
interface D {
     long  op2( in C param );     /* Use interface C */
};
```

To use interfaces alternately as parameters and as data type members, you must create the library in which interfaces C and D are implemented as a single library.

10.5.4 Example of the Server as a Library

The following subsections contain examples of the server as a library.

Example Using C and C++

When dividing the methods section of a server in the interface definition unit of an IDL file to make the server a .so library, note the following.

An example of defining one interface in one module is described below.

IDL Example

```
 [ X.idl ]
    module M {
         interface I1 {
              ...
         };
    };
```

You should specify the -dy option to generate a skeleton for each interface defined in the IDL file. You must create a .so library for each generated skeleton file.

Compiling the IDL File

When compiling the IDL file to generate a stub skeleton, you must specify the following options:

```
IDLc -dy X.idl          /* For the C    */
IDLc -dy -cpp X.idl     /* For the C++ */
```

You must specify the -dy option to generate a skeleton for each interface defined in the IDL file. You must then create a .so library for each generated skeleton file.

Procedures for Compiling an Application

You should create the *.o file by compiling the skeleton generated from the IDL file. You must then specify the C compiler .so library creation option (e.g., -G) to create the .so library.

Solaris32/64

```
cc -I$OD_HOME/include  -D_REENTRANT -DNeedFunctionPrototypes
   -lsocket -lnsl -lthread -L$OD_HOME/lib -lOM
   -o X_M_I1_skel.o  -c X_M_I1_skel.c

cc -G -Kpic -I$OD_HOME/include  -D_REENTRANT -DNeedFunctionPrototypes
   -lsocket -lnsl -lthread -L$OD_HOME/lib -lOM
   -o libM_I1.so  X_M_I1_skel.o
```

Linux32/64

```
gcc -I$OD_HOME/include  -D_REENTRANT -DNeedFunctionPrototypes
    -Wl -E -lnsl -lpthread -L$OD_HOME/lib -lOM
    -o X_M_I1_skel.o  -c X_M_I1_skel.c

gcc -shared -fPIC -I$OD_HOME/include  -D_REENTRANT -DNeedFunctionPrototypes
    -Wl -E -lnsl -lpthread -L$OD_HOME/lib -lOM
    -o libM_I1.so  X_M_I1_skel.o
```

Registering in OD_impl_inst

Specify the so file created from the IDL file in the definition file of the *OD_impl_inst* command.

```
OD_impl_inst -ax X_def    (X_def:  Definition file)
```

When requested by the client, /usr/libM_I1.so is read and the server method is invoked by inserting IDL:M/I1:1.0=/usr/libM_I1.so in the definition file X_def.

Using the Inheritance Function

You must specify the following parameters in the definition file of the *OD_impl_inst* command when using the inheritance function:

Example Solaris32/64 Linux32/64

When the module name is M, interface name is I1, and inheritance name is I2:

```
IDL:M/I1:1.0 = /usr/lib/libM_I1.so
IDL:M/I2:1.0 = /usr/lib/libM_I2.so,,IDL:M/I1:1.0
```

Example Using COBOL

This is not valid for Linux (64 bit).

When using COBOL, divide a method section of a server in units of interfaces defined in the IDL file to make the server a .so library, note the points below.

Example Windows32/64 Solaris32/64

The following is an example of interface definition in a module.

```
 [ X.idl ]
    module M {
         interface I1 {
              ...
         };
    };
```

Compiling the IDL File

When compiling the IDL file to generate a stub skeleton, you must specify the following options:

```
IDLc -cobol idl-file-name    /*For COBOL */
```

Procedures for Compiling an Application

You must compile the skeleton generated from the IDL file to create the *.o file. You must then specify the .so library creation option (e.g., -G) of the COBOL compiler to create the .so library.

```
cobol  -G  -o libX_cdr.so  X_cdr.cbl
cobol  -G  -o libX_skel.so  X_skel.cbl
cobol  -G  -o libX_M_I1_skel.so  X_M_I1_skel.cbl
cobol  -G  -lcobol -L$OD_HOME/lib -lOMcbl -o libM_I1.so -lX_cdr
          -lX_skel -lX_M_I1_skel M_I1.cbl
```

Notes

(1) The location of the COBOL run time libcobol.so and the current directory where the application was created must be specified in the environment variable LD_LIBRARY_PATH.

(2) Set the following parameters in the CORBA reserved environment variable to reference the library provided by ObjectDirector, and to facilitate COBOL source compilation.

```
CORBA = $OD_HOME/include/COBOL
```

(3) The COBOL server application is used as the .so library. The .so library name is as follows (the module name and interface name parts of the library file name are always in upper case, regardless of the IDL definition content):

```
lib-module-name_interface-name.so
```

The created .so library consists of the following modules (when the IDL file is X.idl):

```
libX_cdr.so
libX_skel.so
libX_MM_I1_skel.so
```

When invoking another function from a server implementation section created by the user, make the server a .so library and set it in the environment variable LD_LIBRARY_PATH. For details, refer to the COBOL manual.

(4) Do not specify the -dy option when compiling the COBOL source.

Registering in OD_impl_inst

Specify the so file created from the IDL file in the definition file of the *OD_impl_inst* command.

```
OD_impl_inst -ax  X_def    (X_def:  Definition file)
```

When requested by the client, /usr/lib/libM_I1.so is read and the server method is invoked by inserting IDL:M/I1:1.0=/usr/lib/libM_I1.so in the definition file(X_def).

Note

- Specify the library file created in 'Procedures for Compiling an Application' for the statement to insert in the definition file. This is case sensitive.

Using the Inheritance Function

When using the inheritance function, you must specify the following parameters in the definition file of the *OD_impl_inst* command:

Example Solaris32/64 Linux32/64

When the module name is M, interface name is I1, and inheritance name is I2:

```
IDL:M/I1:1.0 = /usr/lib/libM_I1.so
IDL:M/I2:1.0 = /usr/lib/libM_I2.so,,IDL:M/I1:1.0
```

Generating "IDL:BAD_OPERATION:1.0" when Using COBOL

This section contains review points for the server application when "IDL:BAD_OPERATION:1.0" system exception has been returned.

Application Error

A non-existent function is being used: If a non-existent function is invoked by mistake when creating the .so library, the .so library is created with undetermined symbols.

Usually, these undetermined symbols are solved by another .so library. However, if the undetermined symbols are not solved when the application operates, you will not be able to invoke g the server application method and a BAD-OPERATION will be posted.

COBOL Compilation Specification Error

Compilation option specification error: If "-dy" is specified when the COBOL source is compiled, you cannot use ObjectDirector service functions provided by libOMcbl.so or libOMircbl.so from the application created by specifying "-dy". In this case, the message below is displayed at COBOL run time. You will not be able to invoke the server application files , and a BAD-OPERATION will be posted.

Example

```
libXXXX.so is not found.  (XXXX:  ObjectDirector function name)
```

Error in creating the so file for the server method: If the library created from the COBOL sources described below is unlinked, a BAD-OPERATION will be posted when the server method so file in the definition file specified by the *OD_impl_inst* command is created. These COBOL sources are generated from the IDL compiler.

```
idl-file-name_interface-name_skel.so
idl-file-name_cdr.so
(idl-file-name_skel.so) Sources for XXX_Alloc function such as struct or union
```

Environment Setting Error

Implementation information registration error: If the definition file specified by the *OD_impl_inst* command contains one of the following errors, you will not be able to invoke the server application method and a BAD-OPERATION will be posted.

- The repository ID (rep_id) contained in the definition file does not match IDL.
- The environment variable LD_LIBRARY_PATH has not been set in the environment variable definition (env) of a variable other than persistent
- The retrieval path of the .so library of the COBOL run time or server application storage destination, server application, or method has not been defined.
- The definition of the .so library to be inherited is missing.
- A repository ID does not correspond correctly to a so file.

```
IDL:module/interface:1.0=so-file-name
```

If there is an error in setting the environment variable LD_LIBRARY_PATH at application activation then:

- The .so library of the server application storage destination, server application, or method has not been defined in the environment variable LD_LIBRARY_PATH.
- If the thread libOM.so is valid, you will not be able to run the server application or invoke the method. When using COBOL, you must specify the following path:

```
$OD_HOME/lib/nt
```

10.6 Thread Type and Process Type Applications

This section describes thread type and process type applications on Solaris and Linux systems.

10.6.1 Creating Thread Type and Process Type Applications

This section describes the creation of thread type and process type applications.

(1) For Thread Type

To create a thread type application, you need to link to the libraries described below. Use the *ldd* command to check the linked libraries.

Library (Default installation path) Solaris32/64

```
libthread.so
/opt/FSUNod/lib/libOM.so     (required)
```

Library (Default installation path) Linux32/64

```
libthread.so
/opt/FJSVod/lib/libOM.so     (required)
```

When compiling a C application, or a C++ application (all platforms excluding Linux 64), you must declare -D_REENTRANT_. For further information, refer to the System Thread Programming manual.

(2) For Process Type

To create process type client, you need to link to the library described below. Use the *Idd* command to check the linked library.

Library (Default installation path) Solaris32/64

```
/opt/FSUNod/lib/nt/libOM.so(required)
```

Library (Default installation path) Linux32/64

```
/opt/FJSVod/lib/nt/libOM.so(required)
```

Note

When running the created application, you must set LD_LIBRARY_PATH=$OD_HOME/lib/nt.

(3) Generating a Thread from a Thread Application

To generate a thread from a thread type server application, the following thread libraries are used.

Solaris32/64

Thread library	thr_create
Flag	THR_NEW_LWP (Forms a fixed link for a new thread to LWP.)
	THR_BOUND (Increases the number of concurrent jobs of an unlinked thread by 1.)

If these flags are not set, control returns with an exception UNKNOWN error from the *CORBA_ORB_init()* function.

Linux32/64

Thread library	pthread_create

(4) Generating a Process from a Process Application

To generate a process from a process type server application, use the *fork()* system call.

(5) Notes on Registering a Process Application

This section describes how to register a process type application as a shared/unshared/server-per-method type (other than persistent) in the Implementation Repository.

You must create the definition file in accordance with the application type, and specify the -ax option in the *OD_impl_inst* command for registration.

```
OD_impl_inst -ax definition-file
```

An example of a definition file is given below. The items in the example must be set. You may assign other items as required.

Example: Definition File Solaris32/64

```
rep_id = implementation-repository-id
type   = server-type
binary = application-path
env    = LD_LIBRARY_PATH=/opt/FSUNod/lib/nt
lang   = language-type
```

Example: Definition File Linux32/64

```
rep_id = implementation-repository-id
type   = server-type
binary = application-path
env    = LD_LIBRARY_PATH=/opt/FJSVod/lib/nt
lang   = language-type
```

10.6.2 Simultaneously Implementing Server and Client

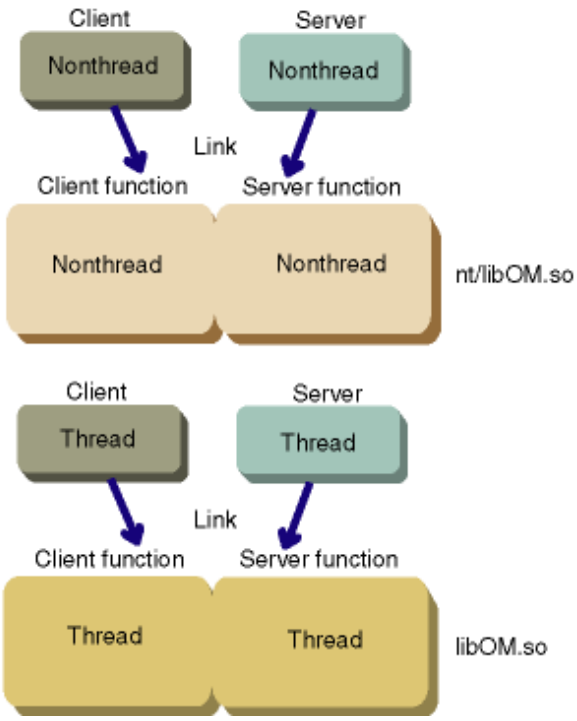
This section describes how to implement server and client functions simultaneously.

Application Types

If server and client functions are implemented simultaneously in an application, thread type and process type applications cannot be linked, so all must be one type or the other. The following figure shows the two combinations of application type.

Figure 10.11 Combinations of Libraries

Combinations of libraries



1. Thread type library:

   /opt/FSUNod/lib/libOM.so Solaris32/64

   /opt/FJSVod/lib/libOM.so Linux32/64

   Core library that uses thread libraries for the client and the server

2. Process type library:

   /opt/FSUNod/lib/nt/libOM.so Solaris32/64

   /opt/FJSVod/lib/nt/libOM.so Linux32/64

   Core library that does not use a thread library for the client or the server

Duplication of Function Names (Stub and Skeleton)

If server and client functions are implemented simultaneously in a single application, the same function name is sometimes generated in stub and skeleton libraries. Linking these stubs and skeletons results in duplicate definitions. Care should therefore be taken when creating IDL files (stub, skeleton).

10.7 Locating Server Applications in Multiple Hosts

This section explains the programming and environment setting needed to locate the same server application in multiple hosts, and how to access various applications from a client application.

10.7.1 Server Application Programming

The server application programming method is not dependent on whether the server application is located in one host or in multiple hosts. Therefore, location in multiple hosts needs no special consideration.

10.7.2 Registering the Server Application

This section explains how to register a created server application in ObjectDirector.

Registering in an Implementation Repository

The following explains how to register server application installation information in an implementation repository.

Use an *OD_impl_inst* command to register the installation information in each host where the server application is located.

Shared type example

```
    OD_impl_inst -a -r  IDL:mod/intf:1.0 -t S -f /home/prog/prog_s
                           (1)                       (2)
```

(1) Repository ID

Specify the same repository ID because it is the same server application.

(2) Specify server application paths for each host.

Registering in Naming Service

The following explains how to register a server application object reference in the naming service.

Use an *OD_or_adm* command to register the object reference in each host where the server application is located.

```
    OD_or_adm -c IDL:mod/intf:1.0 -n name
                  (1)             (2)
```

(1) Repository ID

Specify the same repository ID because it is the same server application.

(2) Object reference name

Specify the name of the server application in each host. The client application then recognizes the server application in each host.

10.7.3 Client Application Programming

The names appended to server application object references in each host are registered in the naming service. Client applications use these names to acquire object references and access the server application. An example of programming follows.

Example

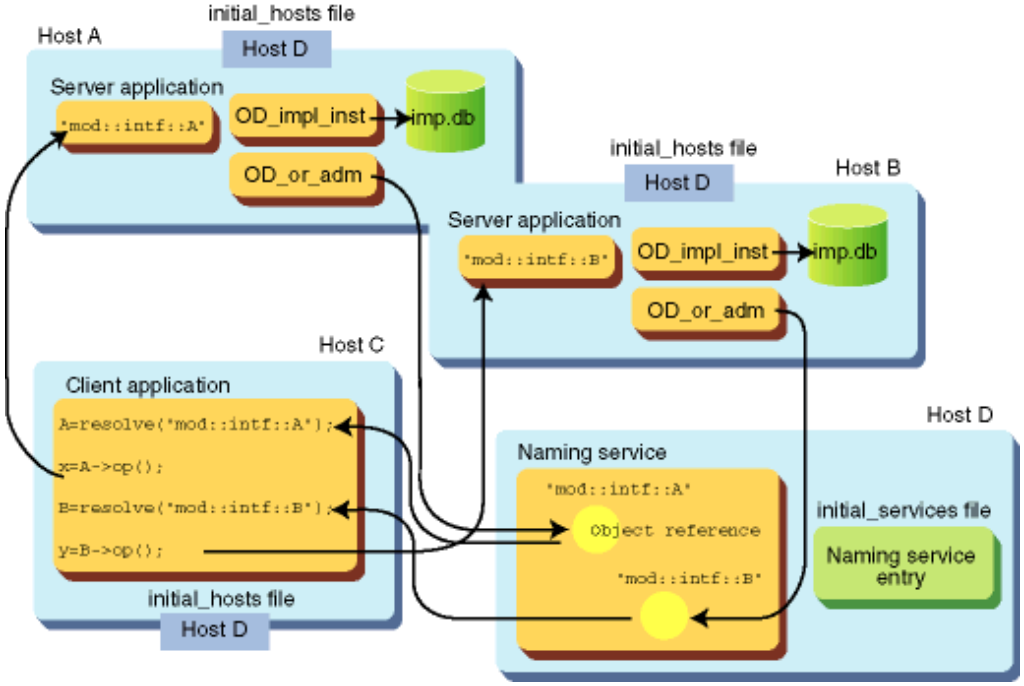
```
CosNaming_NamingContext     naming;

CosNaming_Name               nameA, nameB;
CosNaming_NameComponent     namecpA, namecpB;

/*acquisition of object reference "mod::intf::A"*/
nameA._length = nameA._maximum = 1;
nameA._buffer = &namecpA;
namecpA.id     = "mod::intf::A";
namecpA.kind   = "";

objA = CosNaming_NamingContext_resolve(naming, &nameA, &env );
/*acquisition of object reference "mod::intf::B"*/
nameB._length = nameB._maximum = 1;
nameB._buffer = &namecpB;
namecpB.id    = "mod::intf::B";
namecpB.kind  = "";
objB = CosNaming_NamingContext_resolve(naming, &nameB, &env );
```

10.7.4 Setting up the Environment

This section explains how to set up an environment in each host.

Set the name of the host (where the naming service is running) in the OD setup file initial_hosts in each host. The same naming service is then accessed during registration using the *OD_or_adm* command and the object references are retrieved using resolve() function as shown in the following figure. Accessing the same naming service ensures that information matches.

Figure 10.12 Environment Setting in Each Host



10.8 The exit Function

In the CORBA service, the exit function is provided in C and C++. Using this functionality, by registering a function in the CORBA server application in advance, the function can be executed after the server application sends a reply to the client (after interface implementation function processing is complete).

10.8.1 Registering the exit Function

Register the exit function by issuing an API for registration in the CORBA server application.

The APIs used for the registration of the exit function are shown below.

[Development language]	[API name]
C	CORBA_ORB_register_reply_interceptor()
C++	CORBA::ORB::register_reply_interceptor()

For details on these API functions, refer to the Reference Manual (API Edition).

10.8.2 Monitoring the Maximum Processing Time of the exit Function

A mechanism is provided to monitor the maximum processing time of the exit function. Using this mechanism, a timeout occurs if the exit function does not return when the maximum processing time is exceeded. The CORBA server application that issued the exit function is then stopped by force. At this time, the od10979 message is output.

The default maximum processing time of the exit function is 300 seconds. To specify another time, or to disable monitoring of the maximum processing time, set the reply_interceptor_timeout parameter in the implementation repository when the server application is registered. For details, refer to "OD_impl_inst" in the "Reference Manual (Command Edition)".

10.8.3 Notes about Using the exit Function

- The exit function is executed after interface implementation function processing is complete, even if the CORBA server application does not send a reply to the client, for example when 'oneway' is specified in the interface implementation function.
- The exit function format must be as follows:

```
void exit function name()
```

  If a member function is used as the exit function in C++, it must be a static member function in 'public'.

- One exit function can be registered per CORBA server application.
- If a timeout occurs while the maximum processing time of the exit function is being monitored, the CORBA server application is stopped by force. For this reason, it is recommended that the application is run on a WorkUnit that contains the automatic application restart functionality when using this function.

10.8.4 Exit Function Examples

In this section, programming examples for using the exit function and the output result are shown for C and C++.

C example

Programming example

An example of programming a server application using C is shown below.

```
/* exit function */
void
exitfunc()
{
    printf( "exitfunc is called.\n" );

    return;
}

/* interfaceimplementation function */
CORBA_long
ODsample_intf1_op1(
    ODsample_intf1      obj,
    CORBA_long          l,
    CORBA_Environment   *env )
{
    printf( "ODsample_intf1_op1 is called.\n" );

    return 0;
}

/* main */
int
main( int argc, char *argv[] )
{
    CORBA_ORB  orb;
    CORBA_BOA  boa;
    CORBA_ImplementationDef  impl;
    CORBA_Environment        env;

    orb = CORBA_ORB_init( &argc, argv, FJ_OM_ORBid, &env );

    CORBA_ORB_register_reply_interceptor( orb, exitfunc, &env );

    ...  /* Omitted */

    CORBA_BOA_impl_is_ready( boa, impl, &env );
```

```
    return 0;
}
```

Output result

The server application output result for when a request is sent to the above server application (interface implementation function) is as follows.

```
ODsample_intf1_op1 is called.
exitfunc is called.
```

C++ example

Programming example

An example of programming a server application using C++ (a member function is used as the exit function) is shown below.

```
/* exit function */
class MyClass
{
public:
    static void exitfunc()
    {
        printf( "MyClass::exitfunc is called.\n" );

        return;
    }
};

/* interfaceimplementation function */
CORBA::Long
ODsample_intf1_impl::op1(
    CORBA::Long         l,
    CORBA::Environment  &env )
    throw( CORBA::Exception )
{
    printf( "ODsample_intf1_impl::op1 is called.\n" );

    return 0;
}

/* main */
int
main( int argc, char *argv[] )
{
    CORBA::ORB_ptr  orb;
    CORBA::BOA_ptr  boa;
    CORBA::ImplementationDef_ptr  impl;
    CORBA::Environment_ptr        env = new CORBA::Environment;

    try {
        orb = CORBA::ORB_init( argc, argv, FJ_OM_ORBid, *env );

        orb->register_reply_interceptor( MyClass::exitfunc, *env );

        ...  /* Omitted */

        boa->impl_is_ready( impl, *env );
    }
    catch ( CORBA::SystemException &se ) {
        printf( "SystemException raised!\n" );
        return 1;
    }
```

```
    catch ( CORBA::Exception &e ) {
        printf( "Exception raised!\n" );
        return 1;
    }

    delete env;

    return 0;
}
```

Output result

The server application output result for when a request is sent to the above server application (interface implementation function) is as follows.

```
ODsample_intf1_impl::op1 is called.
Myclass::exitfunc is called.
```

10.9 Notes on Application Development

This section contains notes on application development.

Signal Processing

Do not write applications that receive signals.

Application Termination Processing Windows32/64

If the TerminateProcess function is used to terminate an application, the application will terminate without releasing system resources.

To prevent resource shortage, do not use the TerminateProcess function. Use the following method to terminate applications.

Terminating Server Applications

Use the following procedure to terminate server applications:

```
odcntlque -d -que Impl-ID
odcntlque -c Impl-ID
odcntlque -s Impl-ID
```

(Where Impl-ID is the server application Implementation Repository ID)

Note

Executing the above commands changes the application status as shown below according to the operation mode (specified in mode in the *OD_impl_inst* command in the definition file) selected when the server application is activated.

When mode=SYNC_END is specified:

The server application activation method such as CORBA_BOA_impl_is_ready() returns and a specified termination process is executed.

When mode=COMPATIBLE is specified:

No new requests are processed.

Terminating Client Applications

As is the case with server applications, do not use the TerminateProcess function to terminate client applications. Always terminate a client application after termination processing is completed on the server side.

Duplicate Function Names (Stub and Skeleton) for Server and Client Applications in Single Application Windows32/64

If server and client functions are implemented in a single application, the same function name is sometimes generated in stub and skeleton libraries.

Linking these stubs and skeletons results in duplicate definitions. Care should therefore be taken when creating IDL files (stub, skeleton).

Generating and Terminating Child-Processes Solaris32/64 Linux32/64

When a child process is created in the server application (by using fork( )), the parent process can be terminated by using exit( ) in the child process. Use _exit() if only the child-process is to be terminated.

Chapter 11 CORBA Interface

This chapter explains CORBA interface that is used when programming application to handle object dynamically.

11.1 TypeCode Object

This section explains the following aspects of the TypeCode Object:

- TypeCode Object
- TypeCode Interface

11.1.1 TypeCode Object

The TypeCode object is used when a data type between a client and server is not defined. This object can express all data types. The object is used:

- For any type descriptions
- When an interface repository (IR) return value and dynamic invocation interface are used.

The following table details the valid data types. The TypeCode object consists of a data ID indicating a type (e.g., integer type, string type) and a property that indicates additional information (e.g., maximum string length).

Table 11.1 TypeCode Object

TypeCode Object	Description	Properties Information	
		Type Identifier	Properties
TC_null	Undefined	tk_null	None
TC_void	Void type	tk_void	None
TC_ushort	Integer type	tk_ushort	None
TC_short		tk_short	
TC_ulong		tk_ulong	
TC_long		tk_long	
TC_longlong		tk_longlong	
TC_float	Floating-point type	tk_float	None
TC_double		tk_double	
TC_longdouble Windows32/64 Solaris32/64		tk_longdouble	
TC_char	Character type	tk_char	None
TC_octet	Octet type	tk_octet	None
TC_boolean	Boolean type	tk_boolean	None
TC_string	String type	tk_string	Maximum length
TC_enum(*)	Enumerator type	tk_enum	Data type repository ID, name, name of included data, number of included data items, data type
TC_any	Any type	tk_any	None
TC_sequence(*)	Sequence	tk_sequence	Sequence member data type, maximum length

TypeCode Object	Description	Properties Information	
		Type Identifier	Properties
TC_struct(*)	Structure	tk_struct	Data type repository ID, name, name of included data, number of included data items, data type
TC_union(*)	Union	tk_union	Data type repository ID, name, name of included data, number of included data items, data type, member label number, classification information data type
TC_fixed(*)	Fixed-point	tk_fixed	Position of digits and scale
TC_array(*)	Array	tk_array	Array member data type, maximum number of elements
TC_alias(*)	Data type defined with typedef	tk_alias	Data type repository ID, name
TC_TypeCode	TypeCode	tk_TypeCode	None
TC_Principal	Object for authentication	tk_Principal	None
TC_objref	Object reference type	tk_objref	Data type repository ID, name
TC_except(*)	Exception	tk_except	Data type repository ID, name, name of included data, number of included data items, data type

* The name of TypeCode object is determined based on the IDL definition.

11.1.2 TypeCode Interface

TypeCode provides the interfaces listed below.

```
module CORBA{
     enum TCKind{
          tk_null, tk_void,
          tk_short, tk_long, tk_ushort, tk_ulong,
          tk_float, tk_double, tk_boolean, tk_char,
          tk_octet, tk_any, tk_TypeCode, tk_Principal, tk_objref,
          tk_struct, tk_union, tk_enum, tk_string,
          tk_sequence, tk_array, tk_alias, tk_except
     };

     interface TypeCode{
          exception Bounds{};
          exception BadKind{};

          boolean     equal ( in TypeCode tc );

          TCKind          kind();

          RepositoryId     id()
               raises( BadKind );

          Identifier     name()
               raises( BadKind );

          unsigned long     member_count()
               raises( BadKind, Bounds );

          Identifier     member_name(
               in     unsigned long     index )
```

```
                raises( BadKind, Bounds );

            TypeCode     member_type(
                in     unsigned long     index )
                raises( BadKind, Bounds );

            any          member_label(
                in     unsigned long     index )
                raises( BadKind, Bounds );

            TypeCode     discriminator_type()
                raises( BadKind );

            long          default_index()
                raises( BadKind );

            unsigned long     length()
                raises( BadKind );

            TypeCode     content_type()
                raises( BadKind );

            long          param_count();

            any          parameter(     in     long     index )
                raises( Bounds );
        };
};
```

The following table details the functions of these interfaces.

Table 11.2 TypeCode-provided Functions

TypeCode Method	Function
equal	Checks the equality of specified TypeCode.
kind	Returns a numeric value with a data type indicated by TypeCode.
id	Returns a repository ID with a data type indicated by TypeCode.
name	Returns a name with a data type indicated by TypeCode.
member_count	Returns the number of members with a data type indicated by TypeCode.
member_name	Posts the name of a specified member.
member_type	Posts the TypeCode of a specified member.
member_label	Posts the label of a specified member.
discriminator_type	Posts the TypeCode in classification information definition.
default_index	Posts the index of default members.
length	Posts the length (number of elements) of a TypeCode object.
content_type	Posts the TypeCode of a member.

The following tables below list the valid TypeCode interfaces for each data type.

Table 11.3 TypeCode Interfaces Usable for each Data Type (1)

	Id	Name	Member_count	Member_name	Member_type
struct	o	o	o	o	o
union	o	o	o	o	o
ObjectRef	o	o			

	Id	Name	Member_count	Member_name	Member_type
enum	o	o	o	o	o
alias	o	o			
exception	o	o	o	o	o

Table 11.4 TypeCode Interfaces Usable for each Data Type (2)

	Member_label	Discriminator_type	Default_index	Length	Counter_type
struct	x	x			
union	o	o	o		
string				o	
sequence				o	o
array				o	o

11.2 NVList Object

This section explains the following aspects of the NVList Object:

- NVList Object
- NVList Interface

11.2.1 NVList Object

The NVList object is an interface used to pass parameters between a client and server. It is used for the following:

- Dynamic invocation interface
- Dynamic skeleton interface

11.2.2 NVList Interface

NVList provides the interfaces listed below.

```
module CORBA{
     typedef string          Identifier;
     enum ORBStatus { OK, FAILED };
     enum Flags {
          ARG_IN,
          ARG_OUT,
          ARG_INOUT,
          OUT_LIST_MEMORY,
          IN_COPY_VALUE,
          INV_NO_RESPONSE,
          INV_TERM_ON_ERR,
          RESP_NO_WAIT,
          DEPENDENT_LIST,
          CTX_RESTRICT_SCOPE,
          CTX_DELETE_DESCENDENTS
     };

     interface NVList{
          ORBStatus     add_item(
               in     Identifier     item_name,
               in     TypeCode       item_type,
               in     Object         value,     /* void */
               in     long           value_len,
               in     Flags          flags );
```

```
          ORBStatus   free();

          ORBStatus   free_memory();

          ORBStatus   get_count(
               out    long            count );
     };
};
```

The following table lists the functions of these interfaces.

Table 11.5 NVList Interfaces

NVList Method	Function
add_item	Adds data to the NVList.
free	Releases NVList and NVList locations.
free_memory	Releases NVList locations.
get_count	Reports the number of data items stored in NVList.

11.3 Context Object

This section explains the following aspects of the Context Object:

- Context Object
- Context Interface

11.3.1 Context Object

Information is generally transferred from a client application to the server application as an argument of a method. If information to be transferred to the server application is set in a context object in advance, the information is transferred unconditionally when the method is invoked.

Context object manages the following three types of information:

Context name

Name for grouping properties

Property name

Name for identifying information

Property value

Actual information

For example, if printer is the Context name, printer paper size and printer type correspond to the property name. Context objects can be classified into the two types listed in the following table.

Table 11.6 Context Object

Type	Explanation	Property Name
System default	Context previously defined in system	Vendor dependent
User definition (called ChildContext)	User-defined context managed as child of system default context	Specified in creation

CORBA system default property names include _USER(user), _GROUP(group), and _SYSTEM(system), but they are dependent on the vendor. CORBA Service does not define system default context. An environment variable search is done in the order of ChildContext and system default.

11.3.2 Context Interface

This section provides information on the context interface.

C

Application program interfaces (APIs) in C are provided as Context interfaces for IDL shown below.

```
module CORBA {
     typedef string          Identifier;
     enum ORBStatus { OK, FAILED };
     enum Flags {
          ARG_IN,
          ARG_OUT,
          ARG_INOUT,
          OUT_LIST_MEMORY,
          IN_COPY_VALUE,
          INV_NO_RESPONSE,
          INV_TERM_ON_ERR,
          RESP_NO_WAIT,
          DEPENDENT_LIST,
          CTX_RESTRICT_SCOPE,
          CTX_DELETE_DESCENDENTS
     };

     interface ORB {
          ...
          ORBStatus get_default_context( out Context );
          ...
     };
     interface Context {
          ORBStatus set_one_value(
                    in Identifier       prop_name,
                    in string           value );
          ORBStatus set_values(
                    in NVList           values );
          ORBStatus get_values(
                    in Identifier       start_scope,
                    in Flags            op_flags,
                    in Identifier       prop_name,
                    out NVList          values );
          ORBStatus delete_values(
                    in Identifier       prop_name );
          ORBStatus create_child(
                    in Identifier       ctx_name,
                    out Context         child_ctx );
          ORBStatus delete(
                    in Flags            del_flags );
     };
};
```

The following table lists the API functions.

Table 11.7 Functions Provided by Context Interface (C)

Interface Name	Method Name	Function
ORB	get_default_context	Retrieves the default context object.
Context	set_one_value	Relates a property name (string) and its data value (string) to a specified context.
	set_values	Relates a combination of two or more property names (strings) specified using NVList and their data values to contexts.

Interface Name	Method Name	Function
	get_values	Retrieves a list of information items related to specified parameters which are related to a context.
	delete_values	Deletes a list of property names and data values related to them.
	create_child	Creates a child context.
	delete	Deletes a context object.

C++ and Java

Linux64

C++ can only be used to create Windows(R) clients.

Application program interfaces (APIs) are provided as context interfaces based on IDL shown below.

```
module CORBA {
     typedef string          Identifier;
     enum ORBStatus { OK, FAILED };
     enum Flags {
          ARG_IN,
          ARG_OUT,
          ARG_INOUT,
          OUT_LIST_MEMORY,
          IN_COPY_VALUE,
          INV_NO_RESPONSE,
          INV_TERM_ON_ERR,
          RESP_NO_WAIT,
          DEPENDENT_LIST,
          CTX_RESTRICT_SCOPE,
          CTX_DELETE_DESCENDENTS
     };

     interface ORB {
          ...
          Status get_default_context( out Context );
          ...
     };
     interface Context {
          readonly attribute Identifier     context_name;
          readonly attribute context        parent;

          ORBStatus set_one_value(
               in Identifier propname,
               in any propvalue );
          ORBStatus set_values( in NVList values );
          ORBStatus get_values(
               in Identifier start_scope,
               in Flags op_flags,
               in Identifer pattern,
               out NVLIst values );
          ORBStatus delete_values(
               in Identifier start_scope,
               in Flags op_name,
               in Identifier pattern,
               out NVList values );
          ORBStatus create_child(
               in Identifier child_ctx_name,
               out Context child_ctx );
```

```
    };
};
```

The following table lists the API functions.

Table 11.8 Functions provided by Context Interface (C++ and Java)

Interface Name	Method Name	Function
ORB	get_default_context	Retrieves the default context object.
Context	context_name	Retrieves a context name.
	parent	Retrieves the parent context object.
	set_one_value	Relates a property name (string) and its data value (string) to a specified context.
	set_values	Relates a combination of two or more property names (strings) specified using NVList and their data values to contexts.
	get_values	Acquires a list of information items related to specified parameters that are related to a context.
	delete_values	Deletes a list of property names and related data values.
	create_child	Creates a child context.

11.3.3 Context Interface Examples

Program examples on how to set or acquire data values using the context interface are shown below. The following IDL is used for these examples.

```
module ODsample {
     interface contexttest {
          long     op( in long x )
               context( "DATA", "DATA1", "DATA2" );
     };
};
```

Note

If operating using Portable-ORB, the context interface cannot be used.

C

Sample C client program and server program examples follow.

Client Program

CORBA_Context_set_one_value() is issued to relate a property name and its data value to a context object acquired by CORBA_ORB_get_default_context().

```
CORBA_ORB            orb;
CORBA_Environment    env;
CORBA_Object         obj;
CORBA_Context_ptr    context;

     /*default context acquisition*/
     CORBA_ORBA_get_default_context( orb, &context, &env );

     /*data value setting*/
     CORBA_ORB_set_one_value( context, "DATA", "data", env );
     CORBA_ORB_set_one_value( context, "DATA1", "data1", env );
     CORBA_ORB_set_one_value( context, "DATA2", "data2", env );

     /*context transmission*/
     ODsample_contexttest_op( obj, 3, context, &env );
```

Server Program

A server can use CORBA_Context_get_values() to acquire a property name and its related data value. A wild card(*) may be used to search the property name.

```
void ODsample_contexttest_op(
     ODsample_contexttest     obj,
     CORBA_long               x,
     CORBA_Context            context,
     CORBA_Environment        *env )
{
     CORBA_NVList             nvlist;
     CORBA_long               len;

     /*acquisition of information related to context*/
     CORBA_Context_get_values( context, "", 0, "DATA*", &nvlist , env );

     /*nvlist analysis*/
     CORBA_NVList_get_count( nvlist, &len, &env );
     for ( i = 0; i < len; i++ ){
          printf( "NVList:[%s]\n",
                    *(CORBA_string *)nvlist[i].argument._value );
          /*display of data values DATA, DATA1, and DATA2*/
     }
     ...
}
```

C++

Sample C++ client program and server program examples follow.

Client Program

CORBA::Context::set_one_value() is issued to relate a property name and its data value to a context object acquired by CORBA::ORB::get_default_context().

```
CORBA::ORB_ptr          orb;
CORBA::Environment      env;
CORBA::Object_ptr       obj;
CORBA::Context_ptr      context;

try {
     /*default context acquisition*/
     orb->get_default_context( context, env );

     /*data value setting*/
     CORBA::Any     p1, p2, p3;
     p1 <<= (const CORBA::Char *)"data";
     context->set_one_value( "DATA", p1, env );
     p2 <<= (const CORBA::Char *)"data1";
     context->set_one_value("DATA1", p2, env );
     p3 <<= (const CORBA::Char *)"data2";
     context_set_one_value( "DATA2", p3, env );

     /*context transmission*/
     obj->op( 3, context, env );
}
catch( CORBA::Exception &e ){
     /* error processing */
}
```

Server Program

This is not valid for Linux (64 bit).

A server can use CORBA::Context::get_values() to obtain a property name and its related data value. A wild card(*) may be used to search the property name.

```
void ODsample_contexttest_impl::op(
     CORBA::Long               x,
     CORBA::Context_ptr        context,
     CORBA::Environment        &env )
     throw( CORBA::Exception )
{
     CORBA::NVList_ptr         nvlist;
     CORBA::NamedValue_ptr     nv;

     /*acquisition of information related to context*/
     context->get_values( NULL, 0, "DATA*", nvlist , env );

     /*nvlist analysis*/
     CORBA::Long     len = nvlist->count( env );
     for( I = 0; I < len; I++ ){
          nv = nvlist->item( I, env );
          CORBA::Any *data = nv->value( env );
          CORBA::Char *str = (CORBA::Char *)data->value();
     }
...
}
```

Java

Sample Java client program and server program examples follow.

Client Program

org.omg.CORBA.Context.set_one_value() is issued to relate a property name and its data value to a context object acquired by org.omg.CORBA.ORB.get_default_context().

```
  org.omg.CORBA.ORB   Orb;
  ODsample.contexttestOperations   obj;

  try {
                       :
                       :
     /*default context acquisition*/
        org.omg.CORBA.Context ctx = Orb.get_default_context();

     /*data value setting*/
        org.omg.CORBA.Any p1,p2,p3;
        p1 = Orb.create_any();
        p1.insert_string("data");
        ctx.set_one_value("DATA",p1);

        p2 = Orb.create_any();
        p2.insert_string("data1");
        ctx.set_one_value("DATA1",p2);

        p3 = Orb.create_any();
        p3.insert_string("data2");
        ctx.set_one_value("DATA2",p3);

     /*context transmission*/
        obj.op( 3, ctx );
  }
  catch ( Exception e ) {
        System.out.println( "ERROR" );
          /* error processing */
  }
```

Server Program

A server can use org.omg.CORBA.Context.get_values() to obtain a property name and its related data value. A wild card(*) may be used to search the property name.

```
/* implementation of operations in Servant */
public void op( int x, org.omg.CORBA.Context ctx )
{
      /* acquisition of information related to context */
      NVList nvlist = ctx.get_values("",0,"DATA*");

      /* nvlist analysis */
      int count = nvlist.count();
      org.omg.CORBA.NamedValue nv;
      String str;
      for(int i=0;i<count;i++){
          nv = nvlist.item(i);
          str = nv.value().extract_string();
          System.out.println( "result = " + str );
          /* display of data values DATA, DATA1, and DATA2 */
      }
      return;
}
```

Chapter 12 Obtaining Naming Service Initial References

This chapter explains how to obtain initial references for the Naming Service.

When initial service object references are obtained using CORBA::ORB::resolve_initial_references(), methods are provided to enable the different object references required for each client application to be obtained. In addition to the configuration using the initial settings, two CORBA::ORB_init arguments are provided to enable overrides to be set for the initial references.

12.1 ORBInitRef

The ORB initial reference argument "-ORBInitRef" is specified by the initial service for each service name required by the client. The format is as follows:

```
-ORBInitRef <ObjectID>=<ObjectURL>
```

Example

```
-ORBInitRef  NameService=IOR:00230021AB...
-ORBInitRef  NameService=corbaloc::nshost/NameService
-ORBInitRef  NameService=corbaname::nshost/NameService#nshost2
```

<ObjectID> specifies the service name of the initial service.

Any URL schema supported by CORBA::ORB::string_to_object can be specified in <ObjectURL>. If the URL is incorrect, a BAD_PARAM exception occurs in the CORBA::ORB_init operation.

Refer to the Reference Manual (API Edition) for the specification method when Java applets are used.

12.2 ORBDefaultInitRef

The ORB default initial reference argument "-ORBDefaultInitRef" helps to resolve an initial reference that is not explicitly specified in "-ORBInitRef".

When the CORBA::ORB::resolve_initial_references operation is executed, "-ORBDefaultInitRef" adds a slash ('/') and then obtains the target object reference by adding the object key specified as the service name.

Example

```
-ORBDefaultInitRef corbaloc::inithost
```

When CORBA::ORB::resolve_initial_references ("NotificationService") is executed after CORBA::ORB_init, a "corbaloc::inithost/NotificationService" object reference is obtained.

If CORBA::ORB::resolve_initial_references ("NameService") is invoked after "-ORBDefaultInitRef corbaname::inithost,:inithost2" is specified in the CORBA::ORB_init operation, a "corbaname::inithost/NameService" or "corbaname::inithost2/NameService" object reference is obtained.

Refer to the Reference Manual (API Edition) for the specification method when Java applets are used.

12.3 Initial Service Retrieval Order

The default order for invoking CORBA::ORB::resolve_initial_references using the <ObjectID> provided is as follows:

(1) Resolves with -ORBInitRef for <ObjectID>.

(2) Resolves with the -ORBDefaultInitRef entry. If <ObjectID> is one of the following, the -ORBDefaultInitRef entry is ignored:

```
ImplementationRepository
FJ_LightInterfaceRepository
FJ_ORB_admin
RootPOA
POACurrent
TransactionCurrent
```

```
SecurityCurrent
TransactionIdentification
TransactionServerInit
```

(3) Resolves with the initial_services file setting.

(4) Resolves with the initial_hosts setting in the environment setting file.

12.4 corbaloc URL Schemas

It is difficult to recognize a URL in IOR format because the length and text have been encoded. The corbaloc URL schemas provide easy-to-understand notation for character string object references like the ftp and http URLs.

12.4.1 corbaloc URL Schema

The corbaloc URL schema provides object references in character string notation.

```
corbaloc::inithost/NotificationService
corbaloc:iiop:1.0@inithost/NotificationService
corbaloc::inithost,:inithost2:8002/NameService
corbaloc:rir:/NotificationService
corbaloc:rir:/NameService
```

The complete syntax is as follows:

```
<corbaloc>= "corbaloc:"<obj_addr_list>["/"<key_string>]
<obj_addr_list>= [<obj_addr> ","]* <obj_addr>
<obj_addr>= <prot_addr>
<prot_addr>= <rir_prot_addr> | <iiop_prot_addr>

<rir_prot_addr>= <rir_prot_token>":"
<rir_prot_token>= "rir"

<iiop_prot_addr>= <iiop_id><iiop_addr>
<iiop_id>= ":" | <iiop_prot_token>":"
<iiop_prot_token>= "iiop"
<iiop_addr>= See this section.

<key_string>= <string> | empty_string
```

Details

obj_addr_list

A list in which the address information is delimited by commas (','). When obtaining object references, objects can be obtained from multiple addresses.

obj_addr

A protocol-specific address with a protocol ID and version tag. Commas (',') and slashes ('/') cannot be specified.

rir_prot_addr

The CORBA::ORB::resolve_initial_references protocol ID.

iiop_prot_addr

An address that includes an iiop protocol ID, version tag, and DNS format host name or IP address.

If an iiop protocol ID is omitted like "corbaloc::inithost/NotificationService", it has the same effect as "corbaloc:iiop:inithost/NotificationService" where "iiop" is specified as the iiop protocol ID.

key_string

An object key consisting of character strings.

Characters other than US-ASCII alphanumeric characters and the characters shown below undergo escape processing.

```
      ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" | "$" |
      "," | "-" | "_" | "." | "!" | "~" | "*" | "'" | "(" | ")"
```

12.4.2 corbaloc:rir URL

The corbaloc:rir URL is used to obtain ORB initial reference settings through a URL.

The corbaloc:rir URL is used to obtain the initial settings of initial references for which no overrides were set using the ORBInitRef argument or the ORBDefaultInitRef argument. For details of the ORBInitRef argument and the ORBDefaultInitRef argument, refer to 12.1 ORBInitRef and 12.2 ORBDefaultInitRef.

The protocol address syntax is as follows:

```
<rir_prot_addr>= "rir"
```

Details

rir_prot_addr

This is a CORBA::ORB::resolve_initial_references protocol ID. Specify "rir:".

12.4.3 corbaloc:iiop URL

The corbaloc:iiop URL is defined so that it can be used mainly in TCP/IP and DNS environments.

The complete protocol address syntax is as follows:

```
<iiop_prot_addr>= <iiop_id><iiop_addr>
<iiop_id>= <iiop_default> | <iiop_prot_token>":"
<iiop_default>= ":"
<iiop_prot_token>= "iiop"
<iiop_addr>= <version> <host> [":" <port>]
<host>= DNS-style Host Name | ip_address
<version>= <major> "." <minor> "@" | empty_string
<port>= number
<major>= number
<minor>= number
```

Details

iiop_prot_addr

An address that includes an iiop protocol ID, version tag, and DNS format host name or IP address.

iiop_id

An identified token used to show the corbaloc iiop protocol.

iiop_default

A default token (":") showing the iiop protocol.

iiop_prot_token

An iiop protocol token ("iiop").

iiop_addr

A single address.

host

A DNS format host name or an IP address.

version

The major and minor version numbers are divided by "." and followed by "@". The default is "1.0". Only version "1.0" is currently supported.

ip_address

The IP address.

port

The port number used for listening by the agent.

Appendix A IDL

IDL (Interface Definition Language) is a language used for defining object interfaces. Server application supported interfaces may be defined using IDL.

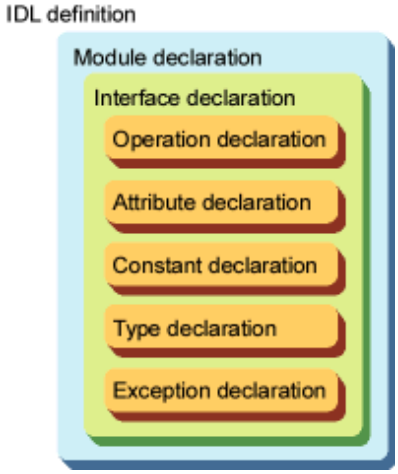
The IDL compiler uses interface defined IDL files to generate source codes for client and server applications mapped in specified languages. These source codes are called stubs and skeletons.

A client application can invoke an IDL defined server application interface via a stub or skeleton.

A.1 IDL Format

The following figure shows the format used to declare an object interface or a type for an object interface.

Figure A.1 IDL Definition



Each declaration must end with a semicolon (;).

The following is an example of an IDL interface definition:

```
module Module1 {                          // Module declaration
    interface Func1 {                     // Interface declaration
        typedef long otype;               // Type declaration
        exception  FuncException {        // Exception declaration
            string  reason;
        };
        otype  Open(in string name)       // Operation declaration
            raises(FuncException);
        readonly attribute long  data;    // Attribute declaration
        typedef Object  FuncObject;       // Type declaration
    };
};
```

A.1.1 Comments

Use one of the following two methods for writing comments in IDL:

- Insert the comment between "/*" and "*/"
- Insert the comment after "//" (The comment ends at the end of the line.)

In a comment, the characters "/*", "*/", and "//" do not have any special meaning and are treated as general characters. Alphanumeric characters, symbols, vertical tabs, horizontal tabs, forms feeds, line feeds, and blanks can be used in a comment.

A.1.2 Identifiers

An identifier is used to identify a declaration described in IDL. An identifier must start with an alphabetical character and may contain alphanumeric characters and underscores (_). There is no distinction made between upper case and lower case characters. If two identifiers contain the same character, one in upper case, the other lower case, both identifiers are considered the same and a compilation error occurs. Each identifier must be unique throughout all IDL definitions. For example, if the same identifier is used as both a constant and an interface even though it has already been specified in other IDL definitions, a compilation error occurs.

The identifiers listed below are keywords and cannot be used for other purposes.

any	context	fixed	longdouble	out	struct	union
attribute	default	float	longlong	raises	switch	void
boolean	double	in	module	readonly	TRUE	wchar
case	enum	inout	Object	sequence	TypeCode	wstring
char	exception	interface	octet	short	typedef	
const	FALSE	long	oneway	string	unsigned	

Each keyword is an identifier and must comply with rules governing identifiers. It must be written correctly as shown in the above list. For example, "interface" is correct, but "Interface" will cause a compilation error because the "I" is upper case.

The following are used as special characters:

```
;  {  }  :  ,  =  +  -  (  )  <  >  [  ]  '  "  \  |  ^  &  *  /  %  ~
```

Note

If you are developing Java language CORBA applications, the DOS device name cannot be specified in the identifier.

A.1.3 Constants

The following constants can be used in IDL.

- Integer constant
- Character constant
- Floating-point constant
- Character-string constant

(1) Integer Constant

The following three notation forms may be used for integer constants:

Octal notation	Numeric strings beginning with 0 (e.g., 0777, 04)
Decimal notation	Numbers other than 0 and numeric strings beginning with + or - (e.g., 128, -45)
Hexadecimal notation	Alphanumeric strings beginning with 0x or 0X (e.g., 0x00, 0Xeff9)

(2) Character Constant

A character constant must consist of a character enclosed in single quotation marks ('). To specify one of the following characters as a character constant, it must be combined with backslash character (\):

Line feed character	\n	Forms feed character	\f	Double quotation mark	\"
Horizontal tab character	\t	Alert character	\a	Octal number	\ooo
Vertical tab character	\v	Backslash	\\	Hexadecimal number	\xhh
Backspace	\b	Question mark	\?		

Return character	\r	Single quotation mark	\'		

- If a character other than the above is specified after '\', '\', the specified characters are treated as unrelated characters.
- \ooo is used to specify an octal number. "ooo" must be a number consisting of one to three digits.
- \xhh is used to specify a hexadecimal number. "hh" must be a number consisting of up to two digits.
- Constant wchar and wstring cannot be specified.

(3) Floating-point Constant

A floating-point constant may be specified using numbers from 0 to 9 and plus(+), minus(-), and period (.) characters. A floating-point constant can consist of two parts: integer and exponent.

The decimal point can be replaced with the character "e" or "E". A sign may be included in the exponent part. Examples of floating-point constants are given below.

```
1e5 = 1 * 10 5 = 100000
1.23E3 = 1.23 * 103 = 1230
-3E - 2 = -3 * 10 -2 = -0.03
```

(4) Character-string Constant

A character-string constant consists of a character string enclosed in double quotation marks ("). If a double quotation mark is included in the character string, the character '/' must be specified before the double quotation mark ("/""). A character-string constant must not include the character string '/0'.

A.1.4 Delimiters

The following characters can be used to separate adjacent identifiers or constants from each other:

- Blank character
- Horizontal tab character
- Vertical tab character
- Line feed character
- Forms feed character
- Comment character

A.1.5 Name and Scope

A scope defines the range of uniqueness of a name in IDL. You can use one of the following three methods to specify a scope:

(1) Scope Name Beginning with an Identifier (e.g., identifier::identifier etc.)

A scope name beginning with an identifier and concatenating each identifier with two colons (::) indicates a "full scope." A full scope specifies a list of identifiers from the highest- level module in the IDL file. In the following example, the type of constant "x" in module M3 corresponds to the one (short type) defined in scope name "M1::M2::T".

```
module M1 {
     typedef long  T ;
     module M2 {
          typedef short  T; -----------+ <---|--Corresponding definition
     };                                |
     module M3 {                       |
          const  M1::M2::T   x = 100;   -----+
     };
};
```

If an interface is inherited, the scope name specified for a particular interface can be specified using the definitions for the base interface. In the following example, the constant type "X" in interface I3 is the same as that (short type) defined by definition T in interface I1, which is the base interface for interface I2.

```
module M1 {
     interface  I1 {
          typedef short  T ;
     };
     interface  I2:I1 {
     };
     interface  I3 {
          const  M1::I2::T  x = 100 ;
     };
};
```

(2) Scope Name Beginning with (::)

A scope name beginning with (::) specifies a list of identifiers that includes the module specified in the scope name definition, and all lower-level modules in the hierarchy. In the following example, "T" in module M1 (which includes a constant defined by scope name "::T") is valid in interface I2, and constant x in interface I2 is of the long type.

```
module  M1 {
     typedef long  T ;
     interface  I1 {
          typedef short  T ;
     };
     interface  I2 {
          const ::T  x = 100 ;
     };
};
```

(3) Scope Name Consisting of an Identifier Only

In a scope, an identifier on its own can be used to reference another definition.

```
module  M {
     typedef long  L ;
     const  L x = 100 ;
};
```

If a specified name is not found in the same scope, a search is made for this name in higher-level scopes related to the one in question. For this reason, a name defined in a higher-level scope may be referenced by specifying the name only. In the following example, the name "L" used in module M2 is not defined within the scope of module M2, but its definition (long type) is valid because "L" is defined in module M1, a higher-level scope.

```
module  M1 {
     typedef long  L ;
     module  M2 {
          const  L x = 100 ;
};
```

If a non-specified name is used even once in a particular scope, that name cannot be redefined later in the same scope. Otherwise, a compilation error will occur.

Scopes are nested according to the following item definitions:

- Module
- Interface
- Structures
- Union
- Operation

- Exception

New scopes are formed according to the following item definitions:

- Type
- Constant
- Enumerated value
- Exception
- Interface
- Attribute
- Operation

The following is an example of how scopes are formed. In this example, three scopes are formed.

```
module  M {                          -+--
    interface  I {    -+--              |
          ...          | Scope 1       |
    };                -+--              | Scope 3
    struct  S {       -+--              |
          ...          | Scope 2       |
    };                -+--              |
    ...                                 |
};                                  -+--
```

A definition identifier must be unique within any one scope. In nested scopes, the same identifiers may be defined.

In identifiers, there is no distinction made between upper case and lower case characters. If two identifiers have the same characters and differ only in their use of upper case and lower case, both identifiers are considered identical.

A.1.6 Differences to C++

IDL syntax is based on C++ syntax, but includes the following restrictions which do not apply in C++:

- A type must always be specified for the return value from a function.
- A name must be defined for each parameter of an operation declaration.
- "void" must not be used in place of a null parameter list.
- A structure (structures, unions with discriminative information, or enumeration) requires a tag.
- An integer must be explicitly declared to be of the short type, long type or etc. It cannot be simply defined as "int" or "unsigned."
- The keyword "signed" or "unsigned" must not be attached to keyword "char."

A.1.7 Pre-processing

Pre-processing in IDL is based on pre-processing in ANSI C++, and has the following functions:

- Macro replacement (#define and #undef)
- Conditional compilation (#if, #ifdef, and #ifndef)
- Quotation of source file (#include)
- Line number control for diagnosis or symbolic debugging (#line)
- Generation of diagnostic messages with a given character string (#err)
- RepositoryID replacement (#pragma)

A line beginning with the character "#" specifies information to be transferred to the pre- process. The character "#" may be preceded by a space. This line is syntactically independent of other IDL definitions, and can be specified anywhere in IDL.

In a source file, a backslash character (\) before the line feed character at the end of a line indicates that the line continues to the next line. For this reason, a source file cannot end with a backslash character.

(1) #pragma

#pragma is the function used to change RepositoryID. The following three kinds of specifications are available for #pragma:

ID pragma

Format

#pragma ID <name> "<id>"

Function

The RepositoryID of the object specified by <name> is replaced by the one specified in <id>. <name> is the identifier of the target object, and may correspond to a scope name.

PREFIX pragma

Format

#pragma prefix "<string>"

Function

A declared PREFIX pragma sets the character string specified in <string> as the prefix of each generated RepositoryID until the boundary of the current scope is reached or another PREFIX pragma occurs. An arbitrary character string may be specified in <string>.

VERSION pragma

Format

#pragma version <name> <major>.<minor>

Function

The version of the RepositoryID specified by <name> is replaced by the version specified in <major>.<minor>. <name> is the identifier of the target object, and a scope name is used to specify the identifier. <major> and <minor> are positive numbers.

Example

The following example shows how to use #pragma to change the RepositoryID:

```
module  M1{
typedef long T1;
typedef long T2;
#pragma ID T2 "IDL:M1/T2_long:1.0"
};
#pragma prefix "P1"

module  M2{
module  M3{
  #pragma prefix "P2"
  typedef long T3;
};
typedef long T4;
#pragma version T4 2.4
};
```

The following example shows names included in scopes and their corresponding RepositoryIDs:

```
::M1::T1           IDL:M1/T1:1.0
::M1::T2           IDL:M1/T2_long:1.0
::M2::M3::T3       IDL:P2/T3:1.0
::M2::T4           IDL:P1/M2/T4:2.4
```

A.2 Module Declaration

Module declaration refers to the grouping of objects so that the method and type names in one IDL definition are not duplicated in any other IDL definition. Multiple modules can be grouped together and declared as one large module.

The following example shows the format used for module declaration:

```
module module-name {
     repetition-of-IDL-definition
};
```

The following is an example of a module declaration:

```
module  Module1 {                                       // Module declaration
    module  Module2 {                                   // Module declaration
        typedef Object  otype;                          // Type declaration
        interface  Func1 {                              // Interface declaration
            otype Open(in string name);                 // Operation declaration
        };
    };
    interface  Func1 {                                  // Interface declaration
         exception FuncException {
            string reason;
        };
        Module1::Module2::otype  Open(in string name)  // Operation declaration
            raises(FuncException);
    };
};
```

When interface name Func1 is specified both in and out of module Module2 in module Module1 as shown in the above example, interfaces to Func1 may be treated as different interfaces as shown below.

```
<1>  Module1::Module2::Func1
<2>  Module1::Func1
```

A.3 Interface Declaration

Interface declaration defines the entity of an interface. Like a class declaration in C++, an interface declaration can be inherited when making a new interface declaration. When an old interface declaration is inherited to a new interface declaration, the operation and type declarations of the old interface declaration may be treated as if declared for the new interface declaration.

When an old interface is inherited, only the differences between the old interface declaration and any other interface declaration may be defined. Multiple interface declarations can also be inherited. The inheritance of multiple interface declarations is called "multi-inheritance." All interface definitions inherit the definitions of the object interface even when inheritance is not specified explicitly. For this reason, object interface functions (e.g., the method "get_interfaceDef" to query the object itself on interface information) may be used.

The following shows the format used for an interface declaration (items enclosed by [ ] may be omitted):

```
interface interface-name ;                              // Forward declaration

interface interface-name [:interface-to-be-inherited] {// Header
    operation-declaration ;                             // Body
    attribute-declaration ;
    constant-declaration ;
    type-declaration ;
    exception-declaration ;
};
```

The first line (specifying an interface name) is a forward declaration defining an interface name prior to the definition of the interface entity. This forward declaration enables the interface definition to be cross-referenced. The entity of the interface defined by the forward declaration must be subsequently defined. The same interface name can be specified by multiple forward declarations in different locations.

In the above format, the second and subsequent lines declare the interface entity. When an interface is specified as "interface-to-be-inherited," that particular interface may be inherited. To specify an interface for inheritance, refer to A.3.1 Inheritance.

An interface declaration consists of one or more operation, attribute, constant, type, and exception declarations. It can be an open interface declaration (i.e. nothing declared). For further information on these declarations, refer to the following sections.

An example of an interface declaration is given below. This example shows how to define the inheritance of interface Func1 to interface Func2. In this case, the client can invoke the Open method for the object of interface Func2.

```
module Module1 {
    typedef Object otype;
    interface  Func1 {                          // Interface declaration
        exception  FuncException {              // Exception declaration
            string reason;
        };
        Module1::otype  Open(in string name)    // Operation declaration
            raises(FuncException);
        readonly attribute long  data;          // Attribute declaration
        typedef Object  FuncObject;             // Type declaration
    };

    interface  Func2 : Func1 {                  // Interface declaration
        Module1::otype  Reopen(in string name); // Operation declaration
        oneway void Close(in Module1::otype);   // Operation declaration
    };
};
```

A.3.1 Inheritance

Inheritance means transferring an operation defined for one interface to another interface. The original interface from which the operation is to be inherited is called the base interface. To specify the interface inheriting the operation, first specify the name of the inheriting interface, next a colon, then the base interface scope name.

An example of how to specify an inheritance is given below. In this example, interface A is the base interface. If you specify multiple base interfaces, separate them with comma (,).

```
interface A {
     long op1(in int a);
};

interface B:A {
     long op2(in int b);
};
```

New elements (constants, types, exceptions, attributes, and operations) can be also declared for the inheriting interface.

Where a type, constant, and exception have the same names as that defined for the base interface they may be redefined. It is not possible to inherit multiple interfaces where an identifier inherited from the same source already exists.

```
interface A {
     const long a = 10;
};

interface B:A {
     const long a = 100;
     int op(in long para);
};
```

An interface specified in "interface-to-be-inherited" is referred to as a "direct base interface." If the base interface inherits another interface, the third interface is called an "indirect base interface." In the example below, interface B is the direct base interface and interface A is an indirect base interface as viewed from interface C.

```
interface  A  { ... };
interface  B:A{ ... };
interface  C:B{ ... };
```

An interface can inherit multiple direct base interfaces. This inheritance is called "multi- inheritance." Even if the direct base interfaces are specified in a different order, the inheriting interface is always the same. The following is an example of multi-inheritance specification:

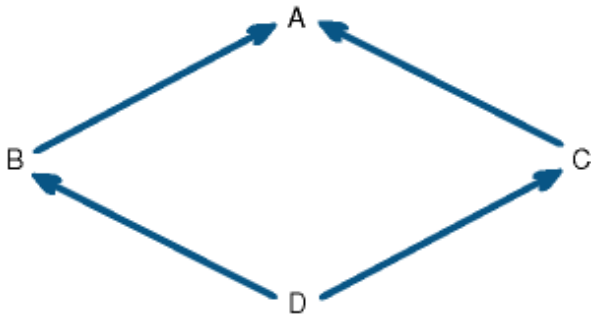
```
interface  A  { ... };
interface  B:A{ ... };
interface  C:A{ ... };
```

An interface may be specified as a direct base interface only once. However, it may be specified as an indirect base interface more than once. The following example shows how to specify an interface as an indirect base interface more than once:

```
interface  A  { ... };
interface  B:A{ ... };
interface  C:A{ ... };
interface  D:B,C{ ... };
```

The following figure shows the relationships between the interfaces in the above example.

Figure A.2 Interface Inheritance



References to the elements of base interfaces must not be ambiguous. Using the same name for a constant, type, or exception in the base interface is ambiguous. In the following example, type X in interface C is ambiguous.

```
interface A {
     typdef long X;
};

interface B {
     typdef short X;

};
interface C:A,B {
     const X = 100;
};
```

In multi-inheritance, the syntax of inherited operations and attributes remains the same, even if elements (constants, types, and exceptions) of the base interface are redefined. An example is given below.

```
module Module1 {
    const long L = 3;
    interface A {
        typedef float s[L];
        void f(in s para);          // S consists of three "float".
    };

    interface B {
        const long L = 4;
    };
```

```
    interface C:B,A { };            // What is the signature of f()?
};
```

Linkage of constant L to an instance of interface A is defined when interface A is defined. For this reason, the format of operation f is the same as that of interface A:

```
typedef float s[3];
void f(in s para);
```

According to this rule, even if constants, types, and exceptions are redefined in the interface that inherited the base interfaces, redefinition does not affect the operation and attributes inherited from the base interfaces.

A.4 Operation Declaration

Operation declaration includes the method name, return value type, parameter data type, exceptions returned to message calls, and context information (equivalent to the environment variable common to server and client applications).

The operation declaration format is shown below (items in [] can be omitted).

```
 [ oneway ] data-type-of-return-value method-name (      // Method name definition
  [ parameter-type  data-type  parameter-name[, ..] ]   // Parameter
)
  [ raises  ( exception-structure-name[, ..] ) ]
  [ context ( context-name[, ..] ) ];
```

The meanings of keywords and items to be specified are explained below.

Oneway

Specifies the operation attribute. When oneway is specified, the server does not notify the client of the execution result of the method called. The following restrictions apply:

- Where the output parameter cannot be provided, the return value type is void.
- The raises expression may not be included (initializing operations that include the raises expression may cause a standard exception).

The client can execute processing without waiting for a response to the method execution result. However, the client will not be notified if the method execution terminates abnormally. If oneway is not specified, the return value is notified when the method execution is successful. Any exception that occurs is also notified.

Return value data type

Specifies any type that can be defined in IDL. If no return value is specified, void is specified

Method name

Specifies the name of the operation to be defined.

Parameter

Specifies 0 or 1 or more parameters. To specify multiple parameters, use a comma (,) as the delimiter.

Parameter type

Specifies the direction in which a parameter is passed between client and server applications. The following three parameter types are available:

- in: A parameter is passed from a client to a method (server).
- out: A parameter is passed from a method (server) to a client.
- inout: A parameter is passed both ways between a client and server.

"in type" is used to pass a value for the invoke function. "out type" is used to receive multiple pieces of data for the method execution result. If only one piece of data is to be received, it can also be received as the method return value (described later). "inout type" is used for method processing that exchanges data in both directions. The in-type parameter must not be changed by the server. If method execution results in an exception, data is not then set in the out- and inout-type parameters and return value.

Parameter data type

The following parameter data types can be specified:

- Basic data types
- Sequences
- Strings
- Scope type

The scope name must have been defined as the above data type. A single identifier or array can be specified for a parameter name.

Raises

The raises expression defines the exception structure used to receive detailed information when a method terminates abnormally due to a user exception. Normally, a method is called with an environment structure (Environment) is added. If a system exception occurs, the error code is set in the environment structure. When the raises expression is defined, information about any method-specific error (user exception) is received if a method error occurs. The error information is passed within the environment structure.

Context

The context expression defines information shared by the client and server applications. For example, a server application cannot determine the client application from which a method is called. However, the server application can determine a client application by specifying the client application in the context expression.

Note

If operating using Portable-ORB, the context expression cannot be used.

Operation Declarations Example

```
module Module1 {
    typedef Object otype;
    interface A {
        exception  FuncException {                    // Exception declaration
            string        reason;
        };

        Module1::otype  Open(in string name)          // Operation declaration
            raises(FuncException);
        oneway void  Close(in Module1::otype obj);    // Operation declaration
    };
};
```

A.5 Attribute Declaration

You must declare the interface data. The attribute declaration format is shown below (items in [] may be omitted).

```
 [readonly] attribute  data-type  variable-name;
```

A single identifier or array can be specified for a variable name. When readonly is specified, a read-only variable can be defined. When readonly is omitted, a variable that can be read and written can be defined. An example of attribute declaration is shown below.

```
attribute string   Name;
readonly attribute long Id;
```

A.6 Constant Declaration

Constant declaration format is shown below. For details, refer to A.7 Data Types and Type Declaration.

```
const data-type constant-name = constant-expression;
```

The following data types may be used in constant declarations:

- Integer type
- Floating-point type
- Character type
- Boolean type
- Strings

A constant expression must correspond to the data type on the left side. Different data types (e.g., integer type, and floating-point type) cannot be combined in a constant expression. The following constants may be used in a constant expression:

- Integer constant
- Character constant
- Floating-point constant
- String constant

The operators shown in the following table can be used in a constant expression:

Table A.1 Operators

	Operation name	Operator	Format	Meaning
Monadic operator	Complement	~	~n	Generates ns complement.
	Plus sign	+	+n	Indicates a positive number.
	Minus sign	-	-n	Indicates a negative number.
Dyadic operator	Addition	+	n1 + n2	Adds n1 and n2.
	Subtraction	-	n1 - n2	Subtracts n2 from n1.
	Multiplication	*	n1 * n2	Multiplies n1 by n2.
	Division	/	n1 / n2	Divides n1 by n2.
	Remainder	%	n1 % n2	Remainder after dividing n1 by n2
	Left shift	<<	n1 << n2	Shifts n1 left n2 bit. n2 must be 0 to 31.
	Right shift	>>	n1 >> n2	Shifts n1 right n2 bit. n2 must be 0 to 31.
	AND	&	n1 & n2	Generates the AND of n1 and n2.
	OR	|	n1 | n2	Generates the OR of n1 and n2.
	XOR	^	n1 ^ n2	Generates the XOR of n1 and n2.

Any integer constant expression that includes a negative integer constant is handled as signed long. Any integer constant expression that does not include a minus sign or negative integer constant is handled as unsigned long. The calculation result value is converted to declared constant data type.

If the calculation result value exceeds the data type range or signed or unsigned long range during calculation, an error occurs. All of the above operators can be used in an integer constant expression.

A floating-point constant expression is converted to double-precision (double) and calculated at double precision. The calculation result value is converted to the declared data type. If this conversion fails or the value being calculated exceeds the double- precision range, an error occurs. The following operators may be used in a floating-point expression:

Monadic operator:	+ and /
Dyadic operator:	*, /, +, and -

An example of constant declaration is shown below.

```
const long l1  = 3;
const long l2  = (3 + 2);
const string S = "xxxxx";
```

```
typedef  long X;
const  X  l3 = 4;
```

A.7 Data Types and Type Declaration

This section explains IDL-supported data types and type declaration.

A.7.1 IDL-supported Data Types

The following table lists IDL-supported data types (data types that may be used for type declaration).

Table A.2 Data Types

Data type		Details	Remarks
Basic data types	Integer type (See Note)	short	Signed short:-2^{15} ... $2^{15}-1$
		unsigned short	Unsigned short: 0 ... $2^{16}-1$
		long	Signed long: -2^{31} ... $2^{31}-1$
		unsigned long	Unsigned long: 0 ... $2^{32}-1$
		long long	Signed long long:-2^{63} ... $2^{63}-1$ (See Note)
	Floating-point type	float	
		double	
		long double	
	Character type	char (8 bits)	When the coding scheme is system dependent, the character type is converted.
		wchar (16 bits)	
	Octet type	octet (8 bits)	Unlike the character type, data is not converted between systems.
	Boolean type	boolean	The boolean type only indicates true or false.
	Strings	string	Terminated by '\0'.
		wstring(wide character)	Terminated by '\0\0'.
	Enumerations	enum	
	any type	any	Data type used to combine data types. "any type" can be defined as a structure member. Only basic data type can be specified.
Sequences		sequence	
Structures		struct	
Unions		union	
Fixed		fixed	Only COBOL mapping and OOCOBOL mapping
Object reference		Object	
		interface	
Type code		TypeCode	

Note

The following values can be used in COBOL and OOCOBOL mapping. You must be careful when using these values because they are different from other languages.

- long: -999,999,999 - 999,999,999
- unsigned long: 0 - 999,999,999

- short: -9,999 - 9,999
- unsigned short: 0 - 9,999
- long long: -999,999,999,999,999,999 - 999,999,999,999,999,999

Linux64

COBOL, and OOCOBOL can only be used to create Windows(R) clients.

These data types can be classified into fixed-length and variable-length data as shown in the following table.

Table A.3 Fixed-length Data and Variable-length Data

Fixed-length data	Variable-length data
Integer type	Strings
Floating-point type	Sequences
Character type	Fixed
Octet type	Structures and unions, fixed including strings or sequences
Boolean type	Object reference
Enumeration type	
any type	
Structures and unions, fixed not including strings or sequences	

A.7.2 Basic Data Types

The data type declaration defines the data types to be used in the interface. Type declaration may also be used to add an alias to an existing data type. This section explains basic data type declaration.

"typedef" is used to declare all basic data types except the enumeration type. The type declaration format is shown below.

```
typedef long              data-type-name;
typedef short             data-type-name;
typedef unsigned long     data-type-name;
typedef unsigned short    data-type-name;
typedef long long         data-type-name;
typedef float             data-type-name;
typedef double            data-type-name;
typedef long double       data-type-name;
typedef char              data-type-name;
typedef wchar             data-type-name;
typedef octet             data-type-name;
typedef boolean           data-type-name;
typedef string <size>     data-type-name;
typedef string            data-type-name;
typedef wstring <size>    data-type-name;
typedef wstring           data-type-name;
typedef any               data-type-name;
```

String Type Declaration

The following shows a string type declaration format:

```
typedef string<size>  data-type-name;
typedef string        data-type-name;

typedef wstring<size> data-type-name;
typedef wstring       data-type-name;
```

There are two types of strings (string and wstring); bounded strings (above format) and unbounded strings (following format). The real size of an unbounded string is specified dynamically. Examples of string declarations are given below.

```
typedef string<100>  A;  (With size specification)
typedef string       A;  (Without size specification)
```

Note the following points when using the string type. Strings have an advantage in speed of char array types. When you use character strings, decide which to use as appropriate.

- The string type (string) has better performance than the char array type (char). When you use character strings, decide which to use as appropriate.

Enumerator Type Declaration

The following shows an enumeration type declaration format (typedef is not used):

```
enum  data-type-name { element, .. };
```

To specify multiple elements, use a comma (,) to delimit the elements. Up to 2^{32} elements may be specified in one enumeration type. An example of an enumeration type declaration is given below.

```
enum E {
 black, white, blue, red
};
```

A.7.3 Sequences

The format for a sequence declaration is shown below.

```
typedef  sequence<basic-data-type, size>      data-type-name;
typedef  sequence<basic-data-type>            data-type-name;
```

A sequence is a one-dimensional array of basic data type, sequence type, struct, union or array. Sequence type can be bounded sequences (above format) or unbounded sequences (following format). The unbounded sequence is specified during execution. An example of sequence type declaration is shown below.

```
sequence<long,10>  A;
sequence<long>     B;
struct sample {
           long    x;
           char    y;
};
typedef sequence<sample, 30>C;
typedef sequence<sample>D;
```

Sequences should be declared with typedef and its TypeCode object is TC_alias instead of TC_sequence. Refer to TypeCode Object in the "CORBA Interface" chapter for further details.

A.7.4 Structures

The format for a structure declaration is shown below.

```
struct data-type-name {
    structure-member-declaration
};
```

The format for a structure member declaration is shown below. ("typedef" is not used.)

```
basic-data-type          member-name;
sequence-type            member-name;
structure                member-name;
union                    member-name;
```

```
fixed                   member-name;
scope-name              member-name;
```

Specify a single identifier or array as a member name. One or more structure members are required and an empty structure is not valid. An example of structure description is shown below.

```
module A {
    typedef long B;
};
struct S {
    string        name;
    short         number;
    long          value;
    ::A::B        s;
};
```

Only recursive type declarations using sequence type are valid.

```
struct  aaa {
        long           l;
        sequence<aaa>  bbb;
};
```

"typedef" can also be used to define a structure name.

Note: Struct types in IDL is different from ones in C language and it means structure definition. Then write as follows to define struct which has struct members:

```
struct  A {
        char xx;
};
struct  B {
        A    yy;
};
```

A.7.5 Unions

The union declaration format is shown below.

```
union  data-type-name  switch (data-type) {
    case constant-expression ;
        element [, ..]
         :
    default;
        element [, ..]
         :
};
```

The element format is given below. ("typedef" is not used.)

```
basic-data-type         element-name;
sequence-type           element-name;
structure               element-name;
union                   element-name;
fixed                   element-name;
scope-name              element-name;
```

Specify a single identifier or array for an element name.

IDL union consists of a combination of C union and a switch statement. The IDL union element must be defined together with the case statement. The type determining the element to be used must also be specified for the type definition of the union switch statement.

The constant expression in the case statement must be consistent with the type definition of the switch statement. The constant expression may specify any value from 0 to 65,535. The case statement can only use default once. The scope name of the element must be of defined

integer, character, boolean, or enumeration type. The constant expression in the case statement must match the type specified in the type definition of the switch statement. Matching rules are as follows:

Long	All integer values in long range
Short	All integer values in short range
Unsigned long	All integer values in unsigned long range
Unsigned short	All integer values in unsigned short range
Long long	All integer values in long long range
Char	char
Boolean	True or false
Enum	All enum members

An element name must be unique within any one union. If the type specified in switch is enumeration type, that identifier is included in the union scope. The identifier must be different from the element name. All values applicable to the union type specification need not be specified in the case statement. The union value consists of the type definition value and one of the following:

- If the data type is explicitly specified in the case statement, the element value corresponds to the description in the case statement.
- If default is specified, the element value corresponds to the default description.
- Other values are not included.

An example of union description is shown below.

```
union U switch (long) {
    case 1:
                long a;
    case 2:
                short b;
    default:
                char c;
};
```

A.7.6 Fixed

The fixed format is shown below.

```
typedef fixed<effective number of digits, scale> data-type- name;
```

The fixed can be used in COBOL mapping and OOCOBOL mapping.

The length of digits (effective digits) of the number item (COBOL/OOCOBOL) is specified in the effective number of digits. Possible ranges are 1 - 18 as to the designation.

The position of a decimal point shows a position of a decimal point in the number item. The number of digits that a decimal point is moved from the right end of effective digits in the left direction is specified. (It decides to be moved in the right direction in case of negative value.) The ranges that they can be specified are "effective digits number -18" - 17.

The designation of effective digits and the scale. When the fixed is specified in the IDL language, it defines as the number item (PICTURE phrase) with COBOL/OOCOBOL. At this time, the definition form of the number item is decided by effective digits and the scale.

The expression of digits with COBOL/OOCOBL by effective digits and the scale and the definition form of the number item are shown in the following. (Effective digits is made |-|. The scale is made v. Virtual digits is made |0|.)

In case of fixed<5,2> (Effective digits:5, The scale: It is moved from the right end in the left direction 2 digits.)

```
|-|-|-|-|-|
      v
```

The definition format of the numerical item is:

```
    PIC  S9(3)V9(2)  PACKED-DECIMAL
```

In case of fixed<5,-2> (Effective digits:5, The scale: It is moved from the right end in the right direction 2 digits.)

Digits outside the range of effective digits are packed with zeros (0). Digits outside the range are called virtual digits.

```
|-|-|-|-|-|0|0|
              v
```

The definition format of the numerical item is:

```
    PIC  S9(5)P(2)V  PACKED-DECIMAL
```

In case of fixed<2,5> (Effective digits:2, The scale: It is moved from the right end in the left direction 5 digits.)

```
|0|0|0|-|-|
v
```

The definition format of the numerical item is:

```
    PIC  SVP(3)9(2)  PACKED-DECIMAL
```

Note that the total number of digits (effective digits + virtual digits) must not exceed 18 digits in COBOL and OOCOBOL.

Linux64

COBOL, and OOCOBOL can only be used to create Windows(R) clients.

A.7.7 Object Reference

Format of the object reference declaration is shown below.

```
    typedef Object       Name of data type;
```

A.7.8 TypeCode

Format of TypeCode declaration is shown below.

```
    typedef TypeCode      Name of data type;
```

A.7.9 Arrays

Array declaration defines a multidimensional fixed-length array. The array declaration format is sh.own below.

```
    data-type identifier[array-size], ..;
```

The array dimension is limited to five levels. The array size must be explicitly specified in each dimension. Specify the array size in a positive integer constant expression. The array size is fixed at compilation.

An example of an array declaration is shown below.

```
typedef long A[5][10];
```

An array type should be declared with typedef and its TypeCode object is TC_alias instead of TC_array. Refer to TypeCode Object in the "CORBA Interface" chapter for further details.

A.8 Exception Declaration

Exception declaration defines the identifier (or exception structure name) used to pass information when an exception occurs during method execution. To define the raises expression for operation declaration, specify this identifier. The exception declaration format is shown below.

```
exception exception-identifier      (exception-structure-name) {
    data-type member-name;      //  Structure member is declared.
        :
};
```

An exception identifier can also be defined as a structure (exception structure) by declaring the structure member. To define multiple structure members, use a comma (,) as the delimiter. The following data types can be defined as members:

- Basic data types
- Sequences
- Structures
- Union
- Fixed
- Scope-name

If an exception is returned as a method execution result, this exception identifier can be referenced. If the exception identifier includes a member declaration, the member can be accessed. If there is no member declaration included, only the exception identifier can be referenced. An example of exception declaration is given below.

```
exception  FuncException {           // Exception declaration
        string        reason;
};
```

Exceptions include system exceptions that notify of abnormal terminations of the system, and user exceptions that notify of abnormal termination of a server application. However, system exceptions cannot be used with the Component Transaction Service.

For information on use of exceptions, refer to Programming Server Application Exceptions of each language.

A.9 IDL Syntax

The IDL syntax is shown below. (::=) is defined as follows:

|	Or
<character >	Nonterminal character
"character"	Literal
*	A syntax element preceding the symbol is repeatable zero or more times.
+	A syntax element preceding the symbol is repeatable one or more times.
{}	The syntax element enclosed in { } is handled as one element.
[]	The syntax element enclosed in [ ] is an option that appears zero or more times.

IDL Specification

<specification>	::=<definition>+

IDL Definition

<definition>	::=<type_dcl>";" | <const_dcl>";" | <except_dcl>";" | <interface>";" | <module>";"

Module Declaration

<module>	::="module"<identifier>"{"<definition>"+}"

Interface Declaration

<interface>	::=<interface_dcl> | <forward_dcl>
<interface_dcl>	::=<interface_header>"{"<interface_body>"}"
<forward_dcl>	::="interface"<identifier>
<interface_header>	::="interface"<identifier>[<inheritance_spec>]
<interface_body>	::=<export>*
<export>	::=<type_dcl>";" | <const_dcl>";" | <except_dcl>";" | <attr_dcl>";" | <op_dcl>";"

Inheritance Specification

<inheritance_spec>	::=":"<scoped_name>{","<scoped_name>}*
<scoped_name>	::=<identifier> | "::"<identifier> | <scoped_name>"::"<identifier>

Constant Declaration

<const_dcl>	::="const"<const_type><identifier>"="<const_exp>
<const_type>	::=<integer_type> | <char_type> | <boolean_type> | <floating_pt_type> | <string_type> | <scoped_name>

Constant Expression

<const_exp>	::=<or_expr>
<or_expr>	::=<xor_expr> | <or_expr>"|"<xor_expr>
<xor_expr>	::=<and_expr> | <xor_expr>"^"<and_expr>
<and_expr>	::=<shift_expr> | <add_expr>"&"<shift_expr>

<shift_expr>	::=<add_expr> | <shift_expr>">>"<add_expr> | <shift_expr>"<<"<add_expr>
<add_expr>	::=<mult_expr> | <add_expr>"+"<mult_expr> | <add_expr>"-"<mult_expr>
<mult_expr>	::=<unary_expr> | <mult_expr>"*"<unary_expr> | <mult_expr>"/"<unary_expr> | <mult_expr>"%"<unary_expr>
<unary_expr>	::=<unary_operator><primary_expr> | <primary_expr>
<unary_operator>	::="-" | "+" | "~"
<primary_expr>	::=<scoped_name> | <literal> | "("<const_exp>")"
<literal>	::=<integer_literal> | <string_literal> | <character_literal> | <floating_literal> | <boolean_literal>
<boolean_literal>	::="TRUE" | "FALSE"
<positive_int_const>	::=<const_exp>

Type Declaration

<type_dcl>	::="typedef"<type_declarator> | <struct_type> | <union_type> | <enum_type>
<type_declarator>	::=<type_spec><declarators>
<type_spec>	::=<simple_type_spec> | <constr_type_spec>
<simple_type_spec>	::=<base_type_spec> | <template_type_spec> | <scoped_name>
<base_type_spec>	::=<floating_pt_type> | <integer_type> | <char_type> | <wide_char_type> | <boolean_type>

	| <octet_type> | <any_type> | <object_type>
<template_type_spec>	::=<sequence_type> | <string_type> | <wide_string_type> | <fixed_pt_type>
<constr_type_spec>	::=<struct_type> | <union_type> | <enum_type>
<declarators>	::=<declarator>{","<declarator>}*
<declarator>	::=<simple_declarator> | <complex_declarator>
<simple_declarator>	::=<identifier>
<complex_declarator>	::=<array_declarator>

Data Type

<floating_pt_type>	::="float" | "double" | "long" "double"
<integer_type>	::=<signed_int> | <unsigned_int>
<signed_int>	::=<signed_long_int> | <signed_short_int> | <signed_longlong_int>
<signed_long_int>	::="long"
<signed_short_int>	::="short"
<signed_longlong_int>	::="long" "long"
<unsigned_int>	::=<unsigned_long_int> | <unsigned_short_int>
<unsigned_long_int>	::="unsigned" "long"
<unsigned_short_int>	::="unsigned" "short"
<char_type>	::="char"
<wide_char_type>	::="wchar"
<boolean_type>	::="boolean"
<octet_type>	::="octet"
<any_type>	::="any"
<struct_type>	::="struct"<identifier>"{"<member_list>"}"
<member_list>	::=<member>+
<member>	::=<type_spec><declarator>";"

<union_type>	::="union"<identifier>"switch""(" <switch_type_spec>")""{"<switch_body>"}"
<switch_type_spec>	::=<integer_type> | <char_type> | <boolean_type> | <enum_type> | <scoped_name>
<switch_body>	::=<case>+
<case>	::=<case_label>+<element_spec>";"
<case_label>	::="case"<const_exp>";" | "default"";"
<element_spec>	::=<type_spec><declarators>
<enum_type>	::="enum"<identifier>"{"enumerator>{ ","<enumerator>}*"}"
<enumerator>	::=<identifier>
<sequence_type>	::="sequence""<"<simple_type_spec>"," <positive_int_const>">" | "sequence""<"<simple_type_spec>">"
<string_type>	::="string""<"<positive_int_const>">" | "string"
<wide_string_type>	::="wstring""<"<positive_int_const>">" | "wstring"
<fixed_pt_type>	::="fixed""<"<positive_int_const>","<integer_literal>">"
<array_declarator>	::=<identifier><fixed_array_size>+
<fixed_array_size>	::="["<positive_int_const>"]"
<attr_dcl>	::=["readonly"]"attribute"<simple_type_spec> <declarator>

Exception Declaration

<except_dcl>	::="exception"<identifier>"{"<member>*"}"

Operation Declaration

<op_dcl>	::=[<op_attribute>]<op_type_spec><identifier> <parameter_dcls>[<raises_expr>] [<context_expr>]
<op_attribute>	::="oneway"
<op_type_spec>	::=<param_type_spec> | "void"
<parameter_dcls>	::="("<param_dcl>{","<param_dcl>}*")" | "(")"
<param_dcl>	::=<param_attribute><param_type_spec> <simple_declarator>

<param_attribute>	::="in" | "out" | "inout"
<raises_expr>	::="raises""("<scoped_name>{","<scoped_name>}*")"
<context_expr>	::="context""("<string_literal>{","<string_literal>}*")"
<param_type_spec>	::=<base_type_spec> | <string_type> | <wide_string_type> | <fixed_pt_type> | <scoped_name>

A.10 IDL Usage in TD

This is not valid for Linux (64 bit).

TD conforms to IDL syntax, but the use ranges for TD applications are restricted as shown in the following table.

Table A.4 IDL Usage in TD

Data type			Support information	Comments
Basic Data Type	integer type (Note)	long long long short unsigned long unsigned short	YES	long long with sign :-263 to 263-1 long with sign :-231 to 231-1 short with sign :-215 to 215-1 long without sign : 0 to 232-1 short without sign : 0 to 216-1
	floating-point type	float double	YES*	
	character type	char	YES	
	octet type	octet	YES	
	boolean type	boolean	YES	
	enumeration type	enum	NO	
	any type	any	NO	
	String	string	YES	
Sequence Type		sequence	YES*	The data type of an element is basic data type.
Array Type		array	YES*	The data type of an array element is a structure containing a field with a basic data type. Array dimension is 1.
Structure Type		struct	YES*	The struct member data type is considered to be basic data type only.
Union Type		union	NO	
Object reference		Object	NO	
		interface	NO	
Type code		TypeCode	NO	

YES: supported

YES*: partly supported

NO: not supported

Note

The following values can be used in the COBOL mapping. You must be careful when using these values because they are different from other languages.

- long: -999,999,999 - 999,999,999
- unsigned long: 0 - 999,999,999
- short: -9,999 - 9,999
- unsigned short: 0 - 9,999
- long long: -999,999,999,999,999,999 - 999,999,999,999,999,999

Items not supported by the IDL syntax are as follows:

1. Interface without a module

   Not supported

   Example

```
module A {
interface B  {
}
}
interface C      -> Disabled
```

2. Nesting of module declarations in Web linkage

   Not supported

3. Attribute

   Not supported

4. Exception

   Not supported

5. Context

   Not supported

6. Object names consisting of more than 255 characters.

   Not supported

7. Operations that return a value with a data type other than long or oneway void

   Not supported

8. typedef declaration of struct

   Partly supported

   Invalid if the server application's language is COBOL

9. Interface scope specification not using inheritance

   Example

```
module1 M1 {
interface I1 {
typedef long L1;
long ope1;
}
interface I2 {
long ope2in M1::I1::L1 para1;      -> Disabled
```

```
    }
    }
```

10. Use of sequence type in COBOL applications

    Not supported

    The reserved word TD_RTNVAL has specific meaning in the TD. Therefore, it cannot be used as an identifier.

Appendix B Programs Provided

This appendix explains the programs that are provided by the CORBA Service.

Note

The storage directory describes the default installation directory.

B.1 Programs Provided by the CORBA Service

This section explains the programs that are provided by the CORBA Service.

B.1.1 Include Files

The include files required when each service is used by applications are shown below:

Storage directory Windows32/64

```
C:\Interstage\ODWIN\INCLUDE
```

Storage directory Solaris32/64

```
/opt/FSUNod/include
```

Storage directory Linux32/64

```
/opt/FJSVod/include
```

Include Files Windows32

Development language	Service	Include file
C language	ORB core (*1)	ORB.H
	Interface Repository	INTFREP.H
	Naming Service	COSNAMIN.H
	Load balance (*2)	OM_LBO.H
C++ language	ORB core (*1)	ORB_CPLUS.H
	Interface Repository	ORB_CPLUS.H (*1)
	Naming Service	COSNAMING_CPLUS.H
	Load Balance (*2)	OM_LBOCPP.H
COBOL (*3)	-	COBOL*
OOCOBOL	-	OOCOB*

*1 This is included in the ORB core include file.

*2 This can be used only by the Interstage Application Server Enterprise Edition.

*3 Refer to "COBOL Library" in the "COBOL Programming Guide" chapter.

Include Files Windows64

Development language	Service	Include file
C language	ORB core (*1)	ORB.H
	Interface Repository	INTFREP.H

Development language	Service	Include file
	Naming Service	COSNAMIN.H
C++ language	ORB core (*1)	ORB_CPLUS.H
	Interface Repository	ORB_CPLUS.H (*1)
	Naming Service	COSNAMING_CPLUS.H
COBOL (*2)	-	COBOL*

*1 This is included in the ORB core include file.

*2 Refer to "COBOL Library" in the "COBOL Programming Guide" chapter.

Include Files Solaris32/64 Linux32/64

Development language	Service	Include file
C language	ORB core (*1)	orb.h
	Interface Repository	InterfaceRep.h
	Naming Service	CosNaming.h
	Load balance (*2) (*4)	OM_LBO.h
C++ language	ORB core (*1)	orb_cplus.h
	Interface Repository	orb_cplus.h (*1)
	Naming Service	CosNaming_cplus.h
	Load Balance (*2) (*4)	OM_LBOcpp.h
COBOL (*3)	-	COBOL*
OOCOBOL (*4)	-	oocob*

*1 This is included in the ORB core include file.

*2 This can be used only by the Interstage Application Server Enterprise Edition.

*3 Refer to "COBOL Library" in the "COBOL Programming Guide" chapter.

*4 This is not valid for Linux (64 bit).

Include File Names in Windows®

Some of the include files installed in a Windows® system have different file names from the (Solaris system or Linux system) file names shown in the manuals. As an example, this is the case with the Reference Manual (API Edition). For this reason, care should be taken when creating applications.

File Name in Manual	Windows File Name
CosNaming.h	COSNAMIN.H
InterfaceRep.h	INTFREP.H
OM_impl_rep.h	OM_IMR.H
OM_intf_rep.h	OM_IFR.H
OM_ORB.h	OM_O.H
OM_stub_skel.h	OM_STSK.H

B.1.2 Libraries

This section describes the services and libraries provided by the CORBA Service (including clients). Specify the ORB core and libraries of the required services during application linkage (build).

B.1.2.1 Server Libraries

Tables Libraries (for Server) and Libraries (for Server) list the server libraries required if the application uses all services.

Storage directory (Other than Java) Windows32

```
C:\Interstage\ODWIN\LIB
```

Storage directory (Other than Java) Windows64

```
C:\Interstage\ODWIN\LIB\x64
```

Storage directory (Java) Windows32/64

```
C:\Interstage\ODWIN\ETC\CLASS
```

Remarks

Server libraries of both the thread type and the process type are installed in the above directories.

Storage directory (Other than Java) Solaris32/64

```
/opt/FSUNod/lib
```

Symbolic link files are created under /usr/lib.

Storage directory (Java) Solaris32/64

```
/opt/FSUNod/etc/class
```

Storage directory (Other than Java) Linux32/64

```
/opt/FJSVod/lib
```

Symbolic link files are created under /usr/lib.

Storage directory (Java) Linux32/64

```
/opt/FJSVod/etc/class
```

Remarks Solaris32/64 Linux32/64

However, the process type library libOM.so is stored under $OD_HOME/lib/nt. If the process type libOM.so library is used, LD_LIBRARY_PATH=$OD_HOME/lib/nt must be set.

Libraries (for Server) Windows32

Development language	Service	Library name
C	ORB core (*1)	ODSV.LIB
	Interface Repository	ODIFSV.LIB
	Naming Service	ODCNSV.LIB
	Load balance (*2)	ODLBOSV.LIB
C++	ORB core (*1)	ODSV.LIB ODSVCPP.LIB
	Interface Repository	(*3)
	Naming Service	ODCNSCPP.LIB
	Load balance (*2)	ODLBSCPP.LIB

Development language	Service	Library name
Java language (*4)		ODjava4.jar
COBOL (Thread Type)	ORB core (*1)	ODCOBCBLMTSV.LIB (other than UNICODE) ODCOBCBLSVUC.LIB (UNICODE) (*5)
	Interface Repository	LIBOMIRCBLMTSV.LIB (other than UNICODE) LIBOMIRCBLSVUC.LIB (UNICODE) (*5)
	Naming Service	(*3)
	Load balance (*2)	ODLBSCBLMT.LIB
COBOL (Process Type)	ORB core (*1)	ODCOBCBLSV.LIB (other than UNICODE) ODCOBCBLSVUC.LIB (UNICODE) (*5)
	Interface Repository	LIBOMIRCBLSV.LIB (other than UNICODE) LIBOMIRCBLSVUC.LIB (UNICODE) (*5)
	Naming Service	(*3)
	Load balance (*2)	ODLBSCBL.LIB
OOCOBOL	ORB core (*1)	ODOOCOBSV.LIB (other than UNICODE) ODOOCOBSVUC.LIB (UNICODE) (*5)
	Interface Repository	-
	Naming Service	ODCNSOOCOB.LIB (other than UNICODE) ODCNSOOCOBUC.LIB (UNICODE) (*5)
	Load balance (*2)	ODLBSOOCOB.LIB

*1 The ORB core library is also required when a service other than ORB core is used.

*2 This can be used only by the Interstage Application Server Enterprise Edition.

*3 This is included in the ORB core library.

*4 The file used differs according to the operation environment and mode. For details, refer to "Execution of CORBA Applications" in the "Java Programming Guide" chapter.

*5 Specify with applications that use UNICODE.

Libraries (for Server) Windows64

Development language	Service	Library name
C	ORB core (*1)	ODSV.LIB
	Interface Repository	ODIFSV.LIB
	Naming Service	ODCNSV.LIB
C++	ORB core (*1)	ODSV.LIB ODSVCPP.LIB
	Interface Repository	(*3)
	Naming Service	ODCNSCPP.LIB
Java language (*2)		ODjava4.jar
COBOL (Thread Type)	ORB core (*1)	ODCOBCBLMTSV.LIB (other than UNICODE) ODCOBCBLSVUC.LIB (UNICODE) (*5)
	Interface Repository	LIBOMIRCBLMTSV.LIB (other than UNICODE)

Development language	Service	Library name
		LIBOMIRCBLSVUC.LIB (UNICODE) (*5)
	Naming Service	(*3)
COBOL (Process Type)	ORB core (*1)	ODCOBCBLSV.LIB (other than UNICODE) ODCOBCBLSVUC.LIB (UNICODE) (*5)
	Interface Repository	LIBOMIRCBLSV.LIB (other than UNICODE) LIBOMIRCBLSVUC.LIB (UNICODE) (*5)
	Naming Service	(*3)

*1 The ORB core library is also required when a service other than ORB core is used.

*2 The file used differs according to the operation environment and mode. For details, refer to "Execution of CORBA Applications" in the "Java Programming Guide" chapter.

*3 This is included in the ORB core library.

Libraries (for Server) Solaris32/64 Linux32/64

Development language	Service	Library name
C	ORB core (*1)	libOM.so
	Interface Repository	libOMir.so
	Naming Service	libOMcn.so
	Load balance (*2) (*5)	libOMlbo.so
C++	ORB core (*1)	libOM.so libOMcpp.so
	Interface Repository	(*3)
	Naming Service	libOMcncpp.so
	Load balance (*2) (*5)	libOMlboccp.so
Solaris32/64 C++ (WorkShop 5.0)	ORB core (*1)	libOM.so libOMcpp50.so
	Interface Repository	(*3)
	Naming Service	libOMcncpp50.so
	Load balance (*2)	libOMlbocpp50.so
Java language (*4)		ODjava4.jar
COBOL (thread type)	ORB core (*1)	libOMcblMT.so
	Interface Repository	libOMircblMT.so
	Naming Service	(*3)
	Load balance (*2) (*5)	libOMlbocblMT.so
COBOL (process type)	ORB core (*1)	libOMcbl.so
	Interface Repository	libOMircbl.so
	Naming Service	(*3)
	Load balance (*2) (*5)	libOMlbocbl.so
OOCOBOL (*5)	ORB core (*1)	libOMoocob.so
	Interface Repository	-
	Naming Service	libOMcnoocob.so

Development language	Service	Library name
	Load balance (*2)	libOMlbooocob.so

*1 The ORB core library is also required when a service other than ORB core is used.

*2 This can be used only by the Interstage Application Server Enterprise Edition.

*3 This is included in the ORB core library.

*4 The file used differs according to the operation environment and mode. For details, refer to "Execution of CORBA Applications" in the "Java Programming Guide" chapter.

When the pre-installed type Java library is run, set the $OD_HOME/lib as the environment variable LD_LIBRARY_PATH.

*5 This is not valid for Linux (64 bit).

B.1.2.2 Client Libraries

The client libraries are installed in the following directory:

Storage directory (Other than Java)

```
C:\Interstage\ODWIN\LIB
```

Storage directory (Java)

```
C:\Interstage\ODWIN\ETC\CLASS
```

The following table lists the libraries required when each service is used for applications.

Libraries (for Clients)

Development language	Service	Library name
C language	ORB core (*1)	ODWIN.LIB
	Interface Repository	ODIF.LIB
	Naming Service	ODCN.LIB
C++ language	ORB core (*1)	ODWIN.LIB ODWINCPP.LIB
	Interface Repository	(*2)
	Naming Service	ODCNCPP.LIB
Java language (*3)		ODjava4.jar
COBOL (thread type)	ORB core (*1)	ODCOBCBLMT.LIB (other than UNICODE) ODCOBCBLUC.LIB (UNICODE) (*4)
	Interface Repository	LIBOMIRCBLMT.LIB (other than UNICODE) LIBOMIRCBLUC.LIB (UNICODE) (*4)
	Naming Service	(*2)
COBOL (process type)	ORB core (*1)	ODCOBCBL.LIB (other than UNICODE) ODCOBCBLUC.LIB (UNICODE) (*4)
	Interface Repository	LIBOMIRCBL.LIB (other than UNICODE) LIBOMIRCBLUC.LIB (UNICODE) (*4)
	Naming Service	(*2)
OOCOBOL	ORB core (*1)	ODOOCOB.LIB (other than UNICODE)

Development language	Service	Library name
		ODOOCOBUC.LIB (UNICODE) (*4)
	Interface Repository	-
	Naming Service	ODCNOOCOB.LIB (other than UNICODE) ODCNOOCOBUC.LIB (UNICODE) (*4)

*1 The ORB core library is also required when a service other than ORB core is used.

*2 This is included in the ORB core library.

*3 The file used differs according to the operation environment and mode. For details, refer to "Execution of CORBA Applications" in the "Java Programming Guide" chapter.

*4 Specify with applications that use UNICODE.

B.2 Programs Provided by Portable-ORB

The following table describes the libraries provided by Portable-ORB.

When operating Portable-ORB, the library used differs according to the operation environment and mode. For details, refer to "Execution of CORBA Applications" in the "Java Programming Guide" chapter.

Storage directory Windows32/64

```
C:\Interstage\PORB\LIB
```

Storage directory Solaris32/64 Linux32/64

```
/opt/FJSVporb/lib
```

Portable-ORB Library Files

Development language	Operation mode	Service	Library name
Java language	Java application	ORB core	ODporb4.jar
		Interface Repository	InterfaceRep4.jar
		Naming Service	CosNaming4.jar
	Java applet (JBK plug-in)	ORB core	ODporb4_plugin.jar
		Interface Repository	InterfaceRep4_plugin.jar
		Naming Service	CosNaming4_plugin.jar

Appendix C Importing and Exporting Interface Definition Information

When a distributed application developed in the CORBA service (ObjectDirector) is to use a Dynamic Invocation Interface, the interface definition information needs to be registered in the Interface Repository on the server where the application is to run. This appendix describes how to perform the registration.

C.1 Registering Interface Definition Information

When a distributed application developed with the CORBA service (ObjectDirector) is to use a Dynamic Invocation Interface, the interface definition information required for request assembly needs to be registered beforehand in the Interface Repository on the server where the application is to run.

The interface definition information can be registered either by using the IDL compiler, or by using the *import* and *export* commands.

This appendix describes how to register the interface definition information using the *import* and *export* commands.

C.2 The Flow of Procedure from Program Development to Operation

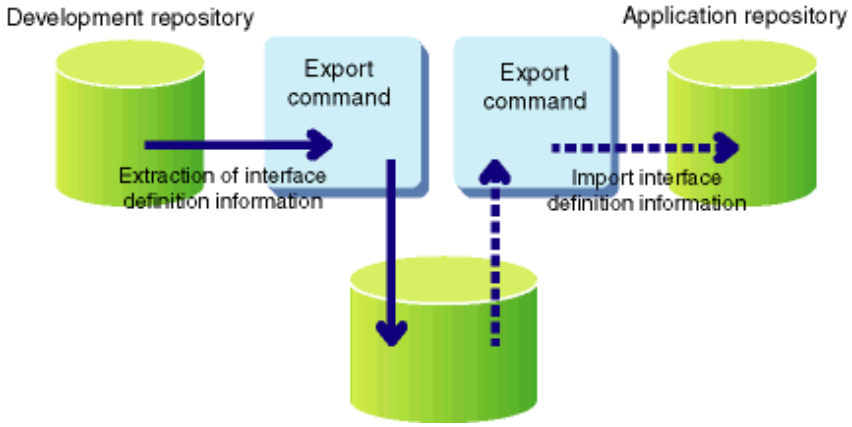
The *export* command is used to *export* the interface definition information required for registration from the development Interface Repository.

The *export* command *exports* the specified interface, and the interface definition information it includes, from the Interface Repository, and outputs it to a file (which will henceforth be known as the interface information file).

The interface information file is not system-specific, so it can be distributed in the form of an attachment to a distributed application.

On each server where the distributed application is implemented, the *import* command is used to register the interface definition information from the interface information file. The following figure shows the development to procedure operation flow.

Figure C.1 Development to Operation Procedure Flow



C.3 Execution Examples

This section contains execution examples.

C.3.1 Exporting the Interface Definition Information

The *odexportir* command is used to *export* the interface definition information.

Example

It *exports* the *mod1* module, and the interface definition information included in *mod1*, from the Interface Repository, and creates an interface information file called "intdef".

```
odexportir -r IDL:mod1:1.0 intdef
```

Windows32/64

Note

To execute the export command, the time-zone needs to be set to the TZ environment variable.

```
TZ = "JST-9"
```

If this setting is not made, the export time information of a created interface information file may not be set correctly.

C.3.2 Importing the Interface Definition Information

The *odimportir* command is used to register the interface definition information.

Example

It registers the content of the interface definition information file ("intdef") in the Interface Repository.

```
odimportir intdef
```

Note

In some cases, the existing interface definition information is overwritten when an *import* operation is carried out. As a precaution, we recommend that you always use the *odbackupsys* command to make a backup of the existing information before executing the *import* command.

Appendix D Collection of Maintenance Information

This appendix explains how to collect the maintenance information that is provided by the CORBA Service.

D.1 Trace Function

In the CORBA service, a working trace is collected for investigating problems occurring during the operation of the CORBA application.

Trace information is collected into memory during normal operation, and can be output to a file.

An outline of the CORBA service trace function and operation procedure is provided below.

D.1.1 Content and Collection Command of the Trace Information

The trace function of the CORBA service logs the input/output information of an application. The logged trace information is collected on the shared memory and can be saved in a file. The logged trace information is collected on shared memory while usual operations, and can be output to the file by the command operation. The collected trace information can be used to investigate the cause about the trouble when the application is operated.

Trace Information to be Collected

Information on the following operation is collected as the trace information of the CORBA application.

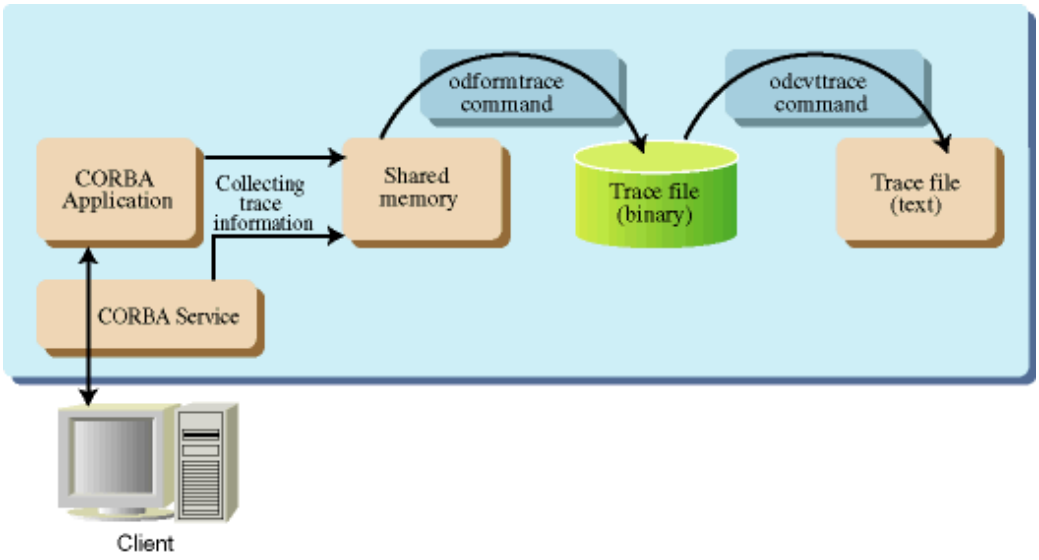
- When a server application is activated
- When a connection is established
- For all processes from the transmission of the request to the reply of the response
- When a connection is disconnected
- When a server application deactivates
- When an error occurs (error detailed information)

Trace Information Collection Command and Generated Product

The relationship between the command used to collect the trace information and the generated information is shown below.

The following figure shows the trace function.

Figure D.1 Trace Function



Trace Information Collection Commands

odformtrace

The *odformtrace* command outputs the collected trace information on the shared memory to the trace file (binary format). The trace information is output to separate files for each process. The suffix of older file name is changed from "log" to "old", and the files are saved.

odcvttrace

The *odcvttrace* command converts the trace file that is output by the *odformtrace* command to text file format so that the file is readable.

odprthdrtrace

The *odprthdrtrace* command outputs the process ID of trace information and the command execution form of applications in text format. If you only want to refer to the process ID and command execution form, you can use the *odprthdrtrace* command without using the *odcvttrace* command.

D.1.2 Event Type and Collection Timing

The operation and the event type of the application in which the trace information is collected are explained.

Application Operation and Collection Place

Operation and the collection place of the application are shown below.

Table D.1 Operation Collection Locations

Operation of application	Collection place	Note
Server application activation	Server side (library)	Issue of BOA_impl_is_ready method (etc.)
Connection establishment	Client side (library)	
All processes from the transmission of the request to the reply	Client side (library and stub)	
	Server side (CORBA service, library, and stub)	
Disconnection	Client side (library)	
Server application deactivation	Server side (library)	Issue of BOA_impl_is_ready method (etc.)
Other information	Client side and server side	

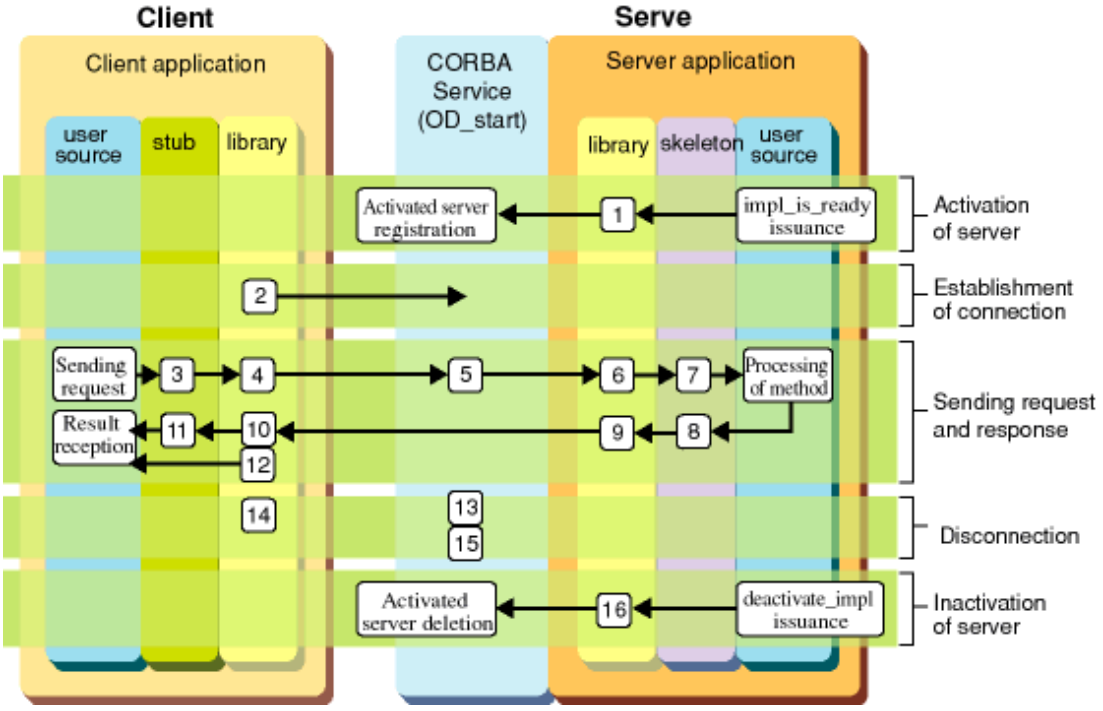
Note

It is necessary to use the static invocation interface or the static skeleton interface for programming so that the stub or the skeleton can collect the trace information.

Flow of Trace Collection

The event type of the trace is different, depending on the operation of the application and the collection locations. The relation between the event types and locations of collection is shown in the following figure. (Numbers in the illustration are described under "Event type" following the graphic.)

Figure D.2 Trace Collection Flow



Event Type

Server Application Activation

1. implementation is ready (implementation = %s1)

   [Variable information]

   %s1: Implementation depository ID

   [Meaning]

   The server application of implementation repository ID%s1 is activated.

Connection Establishment

2. connected to server (host = %s1, port = %s2)

   [Variable information]

   %s1: Name of server machine

   %s2: Port number

   [Meaning]

   Client application is connected to the port number %s2 of server machine%s1.

All Processes from the Transmission of the Request to the Reply

3. STUB: server method call start (request_id = %s1, intf_id = %s2, operation = %s3)

   [Variable information]

   %s1: Request ID

   %s2: Interface repository ID of server application

   %s3: Name of server method

   [Meaning]

   Invocation of the server method has begun.

Note

The trace is output by the application in the static invocation interface. The trace is not output in the Java application.

4. send request to server (request_id = %s1, intf_id = %s2, host = %s3, operation = %s4)

[Variable information]

%s1: Request ID

%s2: Interface repository ID of server application

%s3: Name of server machine

%s4: Name of server method

[Meaning]

Request has been sent.

4-1 client: The number of requests on a connection exceeded the maximum limit

[Meaning]

Number of requests that can be processed simultaneously in one connection of the client machine, has exceeded the number that is set by the environment setup. (Equivalent information to the message od10919)

4-2 The number of connections to a server exceeded the maximum limit

[Meaning]

Number of connections that are connected from client to server, has exceeded the number that is set by environment setup. (Equivalent information to the message od10917)

5. queue request (request_id = %s1, server pid = %s2, intf_id = %s3)

[Variable information]

%s1: Request ID

%s2: Process ID of server that is destination of request dispatch

%s3: Interface repository ID of the server that is destination of request dispatch

[Meaning]

Request of the request ID%s1, is accepted by the CORBA service (OD_start service) and queued.

5-1 server: The number of requests on a connection exceeded the maximum limit

[Meaning]

Number of requests that can be processed simultaneously in one connection of the server machine, has exceeded the number that is set by the environment setup. (Equivalent information to the message od10919)

5-2 The number of connections from clients exceeded the maximum limit

[Meaning]

Server has accepted requests that have exceeded the number of connections that can be received from client that is set by environment setup. (Equivalent information to the message od10918)

6. receive request from client (request_id = %s1, intf_id = %s2, operation = %s3)

[Variable information]

%s1: Request ID

%s2: Interface repository ID of server application

%s3: Name of server method

[Meaning]

Server application has accepted the request.

7. SKEL: server method invoke start (request_id = %s1, intf_id = %s2, operation = %s3)

[Variable information]

%s1: Request ID

%s2: Interface repository ID of server application

%s3: Name of server method

[Meaning]

The skeleton has begun to invoke the server method.

Note

The trace is output by the application in the static skeleton interface. The trace is not output in the Java application.

8. SKEL: server method invoke end (request_id = %s1, intf_id = %s2, operation = %s3)

[Variable information]

%s1: Request ID

%s2: Interface repository ID of server application

%s3: Name of server method

[Meaning]

The server method invoked by the skeleton has ended.

Note

The trace is output by the application in the static skeleton interface. The trace is not output in the Java application.

9. send reply to client (request_id = %s1)

[Variable information]

%s1: Request ID

[Meaning]

The Reply to the request is sent from the server application.

9-1 send standard exception to client (request_id = %s1, excep_id = %s2, minor = 0x%s3)

[Variable information]

%s1: Request ID

%s2: ID of standard exception

%s3: Minor code

[Meaning]

User-defined exception is sent from server application.

9-2 send user exception to client (request_id = %s1, excep_id = %s2)

[Variable information]

%s1: Request ID

%s2: ID of user exception

[Meaning]

User exception is sent out from server application.

9-3 fail to send reply to client (request_id = %s1, from = %s2, intf_id = %s3, operation = %s4)

[Variable information]

%s1: Request ID

%s2: IP address of client machine

%s3: Interface repository ID of server application

%s4: Name of server method

[Meaning]

Sending the reply has failed. (Equivalent information to the message od10605)

10. receive reply from server (request_id = %s1)

[Variable information][

%s1: Request ID

[Meaning]

The client application has received a reply from the server application.

10-1 receive standard exception from server (request_id = %s1, excep_id = %s2, minor = 0x%s3)

[Variable information]

%s1: Request ID

%s2: ID of standard exception

%s3: Minor code

[Meaning]

The client application has received the standard exception that is sent from server application.

10-2 receive user exception from server (request_id = %s1, excep_id = %s2)

[Variable information]

%s1: Request ID

%s2: ID of user exception

[Meaning]

The client application has received the user exception that is sent from server application.

11. STUB: server method call end (request_id = %s1, intf_id = %s2, operation = %s3)

[Variable information]

%s1: Request ID

%s2: Interface repository ID of server application

%s3: Name of server method

[Meaning]

Invocation of the server method has ended.

Note

The trace is output by the application in the static skeleton interface. The trace is not output in the Java application.

12. connection to server (%s1) time out

[Variable information]

%s1: Name of server machine

[Meaning]

The time-out occurred at the client because a reply had not been sent within the stand-by time (period_receive_timeout of the config file). (Equivalent information to the message od10925)

Disconnection

13. connection to client (%s1) was closed

[Variable information]

%s1: IP address of client machine

[Meaning]

Connection to client machine was closed.

14. connection to server (%s1) was closed

[Variable information]

%s1: Server hostname (Or, Internet Protocol address)

[Meaning]

Connection to server machine was closed.

15. connection to client (%s1) time out

[Variable information]

%s1: IP address of client machine

[Meaning]

The time-out occurred at the server because the request transmission from the client was not sent within the no-communication monitoring time (period_idle_con_timeout of the config file) was exceeded.

Server Application Deactivation

16. implementation is deactivated (implementation = %s1)

[Variable information]

%s1: Implementation depository ID

[Meaning]

The server application is deactivated.

Other Information

17. information : (%s1, %s2): %s3, %s4

[Variable information]

%s1:Filename

%s2:Line number

%s3:Details of error

%s4:Error number

[Meaning]

This is CORBA service (ObjectDirector) information message. (Equivalent information as the message od10924)

D.1.3 Preparation for Trace Information Collection

This section explains the preparation for trace information collection.

At Installation

It is necessary to install the CORBA service of "Interstage Server Function" to collect the trace information. Otherwise, trace function does not become effective even if you install "Interstage Client function" (CORBA Service client).

At Application Development

IDL Compilation

To build logging function into the stub or the skeleton generated from the IDL file, IDL is compiled.

Note

- The IDL compiler of V4.0 (or later) can use this function.
- The logging function is not built in the Java mapping.

- The logging function is not built in when the -nolog option is specified. It is recommended that you do not specify the -nolog option at IDL compiler execution time because it is used for diagnostics when a problem occurs.

Programming

The stub or the skeleton can also collect the trace information by programming using the static invocation interface or the static skeleton interface. When neither the static invocation interface nor the static skeleton interfaces are used; the trace information is collected only in the CORBA service and the library.

Linking Libraries

It is necessary to link the server libraries for the server of the CORBA service with the application to collect the trace information. The required server libraries are shown in the following tables.

Table D.2 Language Libraries Windows32/64

Development language	Library name
C, C++	ODSV.LIB
COBOL (Thread type)	ODCOBCBLMTSV.LIB or ODCOBCBLSVUC.LIB
COBOL (Process type)	ODCOBCBLSV.LIB or ODCOBCBLSVUC.LIB
OOCOBOL	ODOOCOBSV.LIB or ODOOCOBSVUC.LIB

Table D.3 Language Libraries Solaris32/64 Linux32/64

Development language	Library name
C, C++	libOM.so
COBOL (Thread type)	libOMcblMT.so or libOMcblUC.so
COBOL (Process type)	libOMcbl.so
OOCOBOL	libOMoocob.so or libOMoocobUC.so

When Application is Operated

config File Setup

trace_use

This parameter indicates whether or not trace information is to be collected. The initial value is "yes".

trace_size_per_process

The maximum memory size (Byte: per process) by which the trace information is collected is specified. The memory capacity for all processes should be satisfied because this value is the maximum size for every process. Refer to "Shared Memory Capacity" for details. The initial value is 10000 bytes.

trace_file_synch_level

This parameter determines output timing to the trace file. Available values are indicated below. Multiple parameter can be specified using delimiter "&".

none

Output to the trace file takes place only when the *odformtrace* command is used.

exit

When an application terminates normally, trace information of the terminated application is output to the trace file.

vanish

When an application terminates abnormally, trace information of the terminated application is output to the trace file.

stop

Trace information of all applications is output to the trace file when the CORBA service terminates.

loop

Output to the trace file takes place when the size of the trace information in memory exceeds the value set in trace_size_per_process.

Even if which value is set in this parameter, the trace information is output when the *odformtrace* command is executed.

Shared Memory Capacity

Because trace information is collected in the shared memory, the following free memory size must be satisfied.

When the size of the trace information reaches the memory size upper limit, old information in the memory is overwritten.

trace_size_per_process (maximum memory size) (see note) x max_processes (number of maximum processes) (see note) + 20 K bytes

Note: Parameter of the config file.

D.1.4 Trace Information Collection Procedure

The procedure for collecting trace information is shown below.

1. Confirm that "yes" is set in the trace_use parameter of the config file. If the config file is modified, restart the CORBA service.
2. Start up the CORBA application.
3. Trace information is output to the trace file at the following times.
   - The trace information of all of or the specified processes is output on startup of the *odformtrace* command.
   - If "trace_file_synch_level = exit" is specified, trace information of the applications that terminated normally is output.
   - If "trace_file_synch_level = vanish" is specified, trace information of the applications that terminated abnormally is output.
   - If "trace_file_synch_level = stop" is specified, trace information of all of the applications is output at the end of the CORBA service.
   - If "trace_file_synch_level = loop" is specified, trace information is output when the size of the trace information in memory exceeds trace_size_per_process.
4. To make trace files readable, the trace information is converted to text file format using the *odcvttrace* command.

D.1.5 Trace Information in Text Output

The output format of the trace file that is converted to a text file using the *odcvttrace* command is explained below.

Output Format

```
ProcessID   : 362
CommandLine : simple_s

thread   time          event
-------- ------------ ----------------------------------------
000000A0 16:27:08.651 implementation is ready (implementation =
                      IDL:test1/intf1:1.0)
```

ProcessID

Process ID of the process in which trace information is collected.

CommandLine

Command execution form of application (including argument). A character string of a maximum of 127 characters (including blanks) is displayed.

thread

Thread ID

time

Time when the trace information was collected. It is displayed in [hour: minute: second: millisecond] format.

event

Event recorded in trace information. (See "Event type", below).

Solaris32/64 Linux32/64

Note

If an error is found in the CORBA_ORB_init() (when C language is used), the command execution format (CommandLine) is not displayed correctly. Refer to the interface for each language in the Interstage Application Server Reference Manual (API Edition) for how to pass arguments to CORBA_ORB_init(). In the case of Java language however, the command execution format may not be displayed correctly regardless of arguments of org.omg.CORBA.ORB.init().Obtain the correct command execution format from the process ID displayed (ProcessID) using the *ps* command.

D.1.6 Trace Information Analysis

Here, trace information is explained when the client application, server application (Naming Service), and the OD_start service all exist in the same machine "hostABC. The IP address of the client machine is temporarily assumed to be [10.34.111.222].

Trace Information of Client Application

```
ProcessID   : 176
CommandLine : simple_c

 thread      time               event
-------- ------------ ------------------------------------------------------
00000140 16:27:10.684 connect to server (host = hostABC, port = 8002)
00000140 16:27:10.684 sensend request to serverd_request (request_id = 1,
                      intf_id = IDL:CosNaming/NamingContextExt:1.0, host = hostABC,
                      operation = resolve)
000000A5 16:27:10.694 receive reply from server (request_id = 1)
00000140 16:27:10.714 connection to server (hostABC) was closed
```

The ProcessID of the client application is 176 and the command line is "simple_c".

1. Port No. 8002 is connected to "hostABC" of the server machine at time [16:27:10.684].
2. The request of request ID1 was sent at time [16:27:10.684]. The Interface Repository ID of the server application is "IDL:CosNaming/Naming Context:1.0" and the name of the server method is "resolve".
3. The client application received a return for the request of Request ID1 from the server application at time [16:27:10.684].
4. The connection of the client application reached a deadlock status for "hostABC" of the server machine at time [16:27:10.714].

Trace Information of the OD_start Service Windows32/64

```
ProcessID   : 339
CommandLine : odstart.exe

 thread      time               event
-------- ------------ ------------------------------------------------
00000094 16:26:57.925 implementation is ready (implementation =
                      IDL:OM_ORB/admin:1.0)
0000008C 16:27:10.684 queue request (request_id = 1, server pid = 111,
                      intf_id = IDL:CosNaming/NamingContextExt:1.0)
00000097 16:27:12.947 connection to client (010.034.111.222) was closed
```

Trace Information of the OD_start Service Solaris32/64 Linux32/64

```
ProcessID   : 339
CommandLine : OD_start

 thread      time               event
-------- ------------ ------------------------------------------------
```

```
00000094 16:26:57.925 implementation is ready (implementation =
                      IDL:OM_ORB/admin:1.0)
0000008C 16:27:10.684 queue request (request_id = 1, server pid = 111,
                      intf_id = IDL:CosNaming/NamingContextExt:1.0)
00000097 16:27:12.947 connection to client (010.034.111.222) was closed
```

The ProcessID of the OD_start service is 339.

1. Activated by the Implementation Repository ID "IDL:OM_ORB/admin:1.0 at time [16:26:57.925].
2. The OD_start service is accepting the request (request ID is 1) issued from the client application and is queuing at time [16:27:10.684]. This request is dispatched to process ID1, Interface Repository ID "IDL:/CosNaming/NamingContextExt: 1.0"(Naming Service).
3. The connection to the client (10.34.111.222) has reached deadlock status.

Trace Information of Naming Service Windows32/64

```
ProcessID   : 111
CommandLine : D:\WINNT\system32\Naming.exe

 thread      time               event
-------- ------------ -------------------------------------------------
0000009A 16:26:59.277 implementation is ready (
                      implementation = IDL:CosNaming/NamingContext:1.0)
00000169 16:27:10.684 receive request from client (request_id = 1,
                      intf_id = IDL:CosNaming/NamingContextExt:1.0,
                      operation = resolve)
00000169 16:27:10.684 send reply to client (request_id = 1)
```

Trace Information of Naming Service Solaris32/64 Linux32/64

```
ProcessID   : 111
CommandLine : CosNaming_s

 thread      time               event
-------- ------------ -------------------------------------------------
0000009A 16:26:59.277 implementation is ready (
                      implementation = IDL:CosNaming/NamingContext:1.0)
00000169 16:27:10.684 receive request from client (request_id = 1,
                      intf_id = IDL:CosNaming/NamingContextExt:1.0,
                      operation = resolve)
00000169 16:27:10.684 send reply to client (request_id = 1)
```

The Naming Service process ID is 111.

1. Activated by Implementation Repository ID "IDL:CosNaming/NamingContext" at time [16:26:59.277].
2. The server application is accepting the request of request ID1 at time [16:27:10.684]. Method "resolve" of Interface Repository ID "IDL:CosNaming/NamingContextExt:1.0" is being called.
3. The server application is sending a return for the request ID to the client application at time [16:27:10.684].

D.2 Snapshot Function

This section explains the following topics about the Snapshot Function:

- Function Overview
- Environment Setup
- Operations

D.2.1 Function Overview

The snapshot function can be used to log input-output information for applications in process units or in Implementation Repository ID units. Logged snapshot information is stored in memory and can be output to a file.

Using this function enables the user to debug applications during initial development and to investigate the cause of problems during operations.

D.2.2 Environment Setup

This section describes the parameters to be set before starting snapshot operations.

Setting for Snapshot Collection

Use the snap_use parameter in the config file to specify whether snapshot information is to be collected. The default value is 'yes'. This setting is required only when you wish to change the default.

Setting for Snapshot Size

The size of snapshot information to be collected is specified in the snap_size parameter in the config file. This size is equivalent to the size of the memory used to store snapshot information. The default value is 40000 bytes.

This setting is required only when you wish to change the default. When the size of snapshot information collected has reached the upper limit of the memory size, old information in memory is overwritten in chronological order.

Note

When the client application sends a message to and receives a response from a server application, snap information (with the estimated sizes as listed below) is obtained by each Action, Refer to "D.2.3.2 Snapshot Information Output Format" for complete details. Consequently, you are advised to set the snapshot size with regard given to this point, as well as to the required data size and memory capacity of the machine.

request send

60+Implementation Repository ID length+Host name length+Operation name length (bytes)

request queue-in

40+Implementation Repository ID length (bytes)

request recv / reply send

60+Implementation Repository ID length+Operation name length (bytes)

reply recv

50+Implementation Repository ID length+Host name length (bytes)

D.2.3 Operations

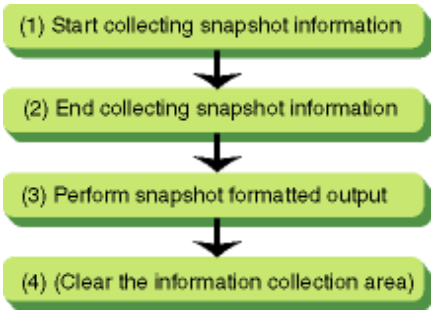
Note

To collect snapshot information, the server libraries must be linked to the applications. Refer to "Libraries" in the "Programs Provided" appendix for information on these libraries.

D.2.3.1 Collecting Snapshot Information

The following figure outlines the procedure used to collect snapshot information.

Figure D.3 Snapshot Collection Process



Procedure

(1) After checking that the server application is operating, use the *odstartsnap* command to start collecting snapshot information.

(2) When the target snapshot information has been collected, use the *odstopsnap* command to terminate the collection. If there is no need to terminate the collection, skip this step and proceed to (3).

(3) Use the *odformsnap* command to output the collected snapshot information to a file.

Send and receive information for applications can be collected using this technique.

The *odfreesnap* command can be used to clear snapshot information from memory if required.

Commands

The four commands listed below are used to collect snapshot information.

odstartsnap

Starts the collection of snapshot information.

The process that collects snapshot information must be active when this command is used.

odstopsnap

Terminates the collection of snapshot information.

odformsnap

*Out*puts the snapshot information stored in memory to a file. The directory created is the current directory when the command is used.

odfreesnap

Clears the snapshot information from memory.

D.2.3.2 Snapshot Information Output Format

The format of snapshot information output by the *odformsnap* command is as follows:

```
1            2         3              4        5         6         7                                    8          9
Time         ProcessID Action         Type     requestID QueueID   implID                               operation  host
-------------------------------------------------------------------------------------------------------------------
22:33:39.180 359       request send   client   1         -------   IDL:CosNaming/NamingContext:1.0      resolve    hostABC
22:33:39.191 202       request queue-in manager 1         1b3f664   IDL:CosNaming/NamingContext:1.0      ----       ---
22:33:39.191 248       request recv   server   1         1b3f664   IDL:CosNaming/NamingContext:1.0      resolve    ---
22:33:39.831 248       reply send     server   1         1b3f664   IDL:CosNaming/NamingContext:1.0      resolve    ---
22:33:39.831 359       reply recv     client   1         ------    IDL:CosNaming/NamingContext:1.0      ----       hostABC
```

Time	The time the snapshot information was collected, displayed as "Hours:Minutes:Seconds.Milliseconds". Since this is the time the snapshot information was registered, it does not agree with the time the information was actually sent or received.
ProcessID	Process ID of the process used to send and receive data.
Action	Shows the send and receive operation types. The five operation types are as follows: - request send Shows that a request has been sent.

	- request queue-in Shows that a request has been received by the *OD_start* service and that it has been queued. - request recv Shows that a request has been received by the server application. - reply send Shows that a reply to a request has been sent from the server application. - reply recv Shows that the client application has received a reply from the server application.
Type	Shows the process types used to send and receive data. The three operation types are as follows: - client Shows that the process is operating as a client application. - server Shows that the process is operating as a server application. - manager Shows that the process is the *OD_start* service.
requestID	Shows the ID of the request being sent or received.
QueueID	Shows the ID of the queue in which the request being sent or received is queued. This is displayed only when the operation type is *request queue-in*, *request recv*, or *reply send*.
implID	Shows the Implementation Repository ID of the destination server application. The meaning varies according to the operation type. - request send Shows the Implementation Repository ID of the destination server application. - request queue-in Shows the Implementation Repository ID of the server application to which the request received by the *OD_start* service is queued. - request recv Shows the Implementation Repository ID of the server application that received the request. - reply send Same meaning as *request recv*. - reply recv Same meaning as *request send*.
operation	Shows the operation name of the destination server method. This is displayed only when the operation type is *request send*, *request recv*, or *reply send*.
host	Shows the host name of the destination server machine. This is displayed only when the operation type is *request send* or *reply recv*.

D.2.3.3 Analyzing Snapshot Information

This section describes how snapshot information is analyzed using four examples.

Example 1

When the Server and Client Applications are Operating on the Same Machine

This example describes snapshot information where the client application, server application (Naming Service) and the OD_start service are all in the same machine ("hostABC")

```
Time          ProcessID  Action          Type     requestID  QueueID  implID                                operation  host
-------------------------------------------------------------------------------------------------------------------------------
22:33:39.180  359        request send    client   1          -------  IDL:CosNaming/NamingContext:1.0  resolve    hostABC
22:33:39.191  202        request queue-in manager  1          1b3f664  IDL:CosNaming/NamingContext:1.0  ---        ---
22:33:39.191  248        request recv    server   1          1b3f664  IDL:CosNaming/NamingContext:1.0  resolve    ---
22:33:39.831  248        reply send      server   1          1b3f664  IDL:CosNaming/NamingContext:1.0  resolve    ---
22:33:39.831  359        reply recv      client   1          -------  IDL:CosNaming/NamingContext:1.0  ---        hostABC
```

1. First, the client application sends request ID1 at 22:33:39.180. The Implementation Repository ID of the destination server application is IDL:CosNaming/NamingContextExt:1.0, and the destination server application is the Naming Service. It can be seen that the method name is resolve and the destination host is "hostABC".
2. At 22:33:39.191, the OD_start service receives the request (ID = 1) from the client application and queues it. The queue ID of the queued request is "1b3f664".
3. At 22:33.39.191, the server application retrieves the queue with queue ID "1b3f664" and receives request ID1. The resolve method is invoked.
4. At 22:33.39.831, the server application returns a reply to request ID1 to the client application. The queue ID corresponding to this request is "1b3f664", and the method name is resolve.
5. At 22:33.39.831, the client application receives the reply to request ID1 from the server application.

Note

Since the send and receive times and the time the snapshot information is written do not exactly match, the reply send and reply recv times may be reversed.

Example 2

When the Server and Client Applications are Operating on Different Machines

This example describes snapshot information where the server application (Naming Service) and the *OD_start* service are operating in "hostXYZ", and the client application is operating in a different machine.

Example of output on a client machine

```
Time          ProcessID  Action          Type     requestID  QueueID  implID                                operation  host
-------------------------------------------------------------------------------------------------------------------------------
22:35:39.150  407        request send    client   3          ------   IDL:CosNaming/NamingContext:1.0  resolve    hostXYZ
22:35:40.534  407        reply recv      client   3          ------   IDL:CosNaming/NamingContext:1.0  ---        hostXYZ
```

Example of output on a server machine

```
Time          ProcessID  Action          Type     requestID  QueueID  implID                                operation  host
-------------------------------------------------------------------------------------------------------------------------------
22:35:39.182  328        request queue-in manager  3          2b3f656  IDL:CosNaming/NamingContext:1.0  ---        ---
22:35:40.012  258        request recv    server   3          2b3f656  IDL:CosNaming/NamingContext:1.0  resolve    ---
22:35:40.442  258        reply send      server   3          2b3f656  IDL:CosNaming/NamingContext:1.0  resolve    ---
```

The operation types for the snapshot information collected on the client application side are request send and reply recv. The snapshot information on the client application side shows that a request with request ID3 was sent at 22:35:39.150, and that a reply was received at 22:35:40.534.

The operation types for the snapshot information collected on the server application side are request queue-in, request recv, and reply send. The snapshot information on the server application side shows that the OD_start service queued a request with request ID3 at 22:35:39.182, and that the server application sent a reply to the client application at 22:35:40.442.

Example 3

When the Server Application is in Abnormal Status

This example describes snapshot information where the server application (Naming Service) is in abnormal status.

```
Time          ProcessID  Action          Type     requestID  QueueID  implID                                operation  host
-------------------------------------------------------------------------------------------------------------------------------
22:41:59.730  237        request send    client   10         ------   IDL:CosNaming/NamingContext:1.0  resolve    hostABC
22:41:59.730  202        request recv    manager  10         3bff4b8  IDL:CosNaming/NamingContext:1.0  resolve    ---
22:41:59.730  202        reply send      manager  10         3bff4b8  IDL:CosNaming/NamingContext:1.0  resolve    ---
22:41:59.730  237        reply recv      client   10         ------   IDL:CosNaming/NamingContext:1.0  ---        hostABC
```

When the server is in an abnormal status, such as when the destination server process (which is sending the request) has not been activated, the *OD_start* service may receive the request instead of the server application, and exception information may be returned to the client application.

The exception information cannot be ascertained from the snapshot output. In addition, when the *OD_start* service is processing requests, no requests will be queued. There is therefore no snapshot information whose operation type is request queue-in, and queue ID "3bff4b8" has no meaning.

Example 4

When a Request is Issued to the OD_start Service

This example describes snapshot information where a request is issued to the *OD_start* service.

```
Time          ProcessID  Action         Type      requestID  QueueID  implID                        operation  host
-------------------------------------------------------------------------------------------------------------------
22:35:51.541  359        request send   client    6          -----    IDL:OM_ORB/admin:1.0          synch      hostABC
22:35:51.551  202        request recv   manager   6          0        IDL:OM_ORB/admin:1.0          synch      ---
22:35:51.581  202        reply send     manager   6          0        IDL:OM_ORB/admin:1.0          synch      ---
22:35:51.581  359        reply recv     client    6          ----     IDL:OM_ORB/admin:1.0          ---        hostABC
```

Management commands include a command for sending requests to the *OD_start* service (for example, *OD_impl_inst*).

Requests sent to the *OD_start* service are immediately processed by the *OD_start* service without being queued. When the process type corresponding to request recv is not server but manager, the *OD_start* service processes requests. In this case, because no requests are queued, there is no snapshot information whose operation type is request queue-in. Although the queue ID is '0', the queue ID corresponding to the request to be immediately processed by the *OD_start* service has no meaning.

D.2.4 CORBA Service Naming Service User Exception Log Collection

CORBA Service Naming Service log information is logged as Naming Service user configuration exception information when the Naming Service interface is used.

When a user configuration exception occurs in the Naming Service interface, it is possible to identify the API and defective parameter.

D.2.5 Log Data

Log File

The Naming Service user exception log is output in the following file:

File name

Windows32/64

```
<CORBA Service installation path>\var\cn_userexception_log, cn_userexception_log.old
```

Solaris32/64 Linux32/64

```
<CORBA Service installation path>/var/cn_userexception_log, cn_userexception_log.old
```

Files size

The disk space shown below is required to create the log file. To use log collection, make sure that enough disk space has been secured.

cn_userexception_log_size value (nsconfig file) * 2 (bytes)

For details about the nsconfig file parameters, refer to "D.2.6 Log Collection Environment (nsconfig File)".

Data Format

User exception log data is output in the following format:

```
<CORBA Service installation path>/var/cn_userexception_log, cn_userexception_log.old
```

Year-Month-Day

Date the log was collected

Hours:Minutes:Seconds.Millisecond

Time the log was collected

Process ID

Naming Service process ID

User exception

User exception

The following user exceptions are collection targets:

- NotFound: IDL:CosNaming/NamingContext/NotFound:1.0
- CannotProceed: IDL:CosNaming/NamingContext/CannotProceed:1.0
- InvalidName: IDL:CosNaming/NamingContext/InvalidName:1.0
- AlreadyBound: IDL:CosNaming/NamingContext/AlreadyBound:1.0
- NotEmpty: IDL:CosNaming/NamingContext/NotEmpty:1.0
- InvalidAddress: IDL:CosNaming/NamingContextExt/InvalidAddress:1.0

Specified name

Name of the Naming Service registration object specified in the Naming Service API

If the Naming Service context extension interface is used, the string notation binding name is output.

API exception

Naming Service API exception

Client IP address

Client Host IP address that called the Naming Service API

Output Example

An example of the user exception log output is shown below.

```
2003-05-07 02:04:06.123+09:00 1234 NotFound NAME_TEST
CosNaming_NamingContext_resolve() 10.1.1.1
2003-05-07 02:04:07.123+09:00 1234 AlreadyBound
  CONTEXT-A/CONTEXT-B/NAME_TEST1 CosNaming_NamingContest_bind() 10.1.1.1
2003-05-07 02:04:08.123+09:00 1234 NotFound CONTEXT-A/NAME_TEST2
 CosNaming_NamingContext_resolve() 10.1.1.1
```

Note

If the "specified name" is under a context that was created manually, the context is also output.

D.2.6 Log Collection Environment (nsconfig File)

The user exception log collection environment (whether log collection is used, and the log file size) is set in the Naming Service "nsconfig" file.

The parameter configurations for the "nsconfig" file used for log collection are described below.

Point

For details about the nsconfig file, refer to "nsconfig" in the "CORBA Service Environment Definition" appendix of the Tuning Guide.

cn_userexception_log_use

This indicates whether user exception log collection is used/not used.

The following values can be specified:

- yes: User exception log collection is used (this is the default).

- no: User exception log collection is not used.

cn_userexception_log_size

This indicates the maximum size of the user exception log file (in bytes). (Default: 2000000)

Note

The user exception log file is collected as two files:cn_userexception_log, and cn_userexception_log.old.

For this reason, the disk space must equal [cn_userexception_log_size * 2].

Appendix E Sample Programs (Windows®)

This appendix describes the sample programs that are provided.

E.1 Sample CORBA Service Programs

This section contains sample CORBA Service programs.

E.1.1 Types of Sample Programs

Tables Table E.1 Sample Programs and Table E.2 Sample Programs list the types of sample programs provided by CORBA Service (ObjectDirector). For building and executing sample programs, refer to each description (the Execution Procedure column in Tables Table E.1 Sample Programs and Table E.2 Sample Programs) in "E.1.2 Execution Procedure of Sample Programs".

Table E.1 Sample Programs Windows32

Sample Type		Storage Folder (*1)	Language	Execution Procedure
General data type		complex\samplelist.C\data* complex\samplelist.C++\data* complex\samplelist.Java\data* complex\samplelist.COBOL\data*	C C++ Java COBOL	(a) (b) -
Dynamic invocation interface		complex\samplelist.C\dii* complex\samplelist.C++\dii*	C C++	(c)
Naming service		complex\samplelist.C\naming* complex\samplelist.C++\naming*	C C++	(d)
Interface repository		irsample*	C, C++ COBOL	(e)
Four arithmetic operations	Static invocation	CalcSample\c CalcSample\c++ CalcSample\COBOL CalcSample\java	C C++ COBOL Java	(a) - (b)
	Dynamic invocation interface	CalcSample\c_dii CalcSample\c++_dii	C C++	(c)
POA interface		POA\java_1_4	Java	(b)
Process bind		complex\samplelist.C++\ssn*	C++	(a)
Other		simple_s, _c simple_cpp_s, _c java	C C++ Java	(a) (b)

Table E.2 Sample Programs Windows64

Sample Type	Storage Folder (*1)	Language	Execution Procedure
General data type	complex\samplelist.C\data* complex\samplelist.C++\data* complex\samplelist.Java\data*	C C++ Java	(f) (g) -

Sample Type		Storage Folder (*1)	Language	Execution Procedure
		complex\samplelist.COBOL\data*	COBOL	
Dynamic invocation interface		complex\samplelist.C\dii* complex\samplelist.C++\dii*	C C++	(h)
Naming service		complex\samplelist.C\naming* complex\samplelist.C++\naming*	C C++	(i)
Interface repository		irsample*	C, C++ COBOL	(j)
Four arithmetic operations	Static invocation	CalcSample\c CalcSample\c++ CalcSample\COBOL CalcSample\java	C C++ COBOL Java	(f) - (g)
	Dynamic invocation interface	CalcSample\c_dii CalcSample\c++_dii	C C++	(h)
POA interface		POA\java	Java	(g)
Process bind		complex\samplelist.C++\ssn*	C++	(f)

*1 All sample programs are under the folder as follows. This section describes sub-folder as the relative path to this folder.

C:\Interstage\ODWIN\src\sample

This section describes the details of the sample programs for each type of sample program listed above.

(1) General Data Type

General data type sample programs and their types are listed in the following table.

Table E.3 General Data Type

No.	Type	Outline	Remarks
1	any	Passes any type data.	
2	array	Passes array type data.	
3	attribute	Passes long and string data as attribute.	
4	context	Passes context type data.	
5	exception	Passes user-defined type exceptions.	C, C++, Java Only
6	sequence	Passes sequence type data.	
7	string	Passes string type data.	
8	struct	Passes structure type data having long and string types as elements.	
9	union	Passes union type data.	

(2) Dynamic Invocation Interface

Dynamic invocation interface sample programs and their types are listed in the following table.

Table E.4 Dynamic Invocation Interface

No.	Type	Outline	Remarks
1	dyn1	Passes long type data.	
2	dyn2	Passes structure type data having long and char types as elements.	
3	dyn3	Passes union type data.	
4	dii_long	Passes long type data.	C++ Only
5	dii_string	Passes string type data.	C++ Only
6	dii_struct	Passes structure type data.	C++ Only
7	dii_sequence	Passes sequence type data.	C++ Only

(3) Naming Service

Naming service sample programs are listed in the following table.

Table E.5 Naming Service

No.	Type	Outline	Remarks
1	bind	Executes the bind method of the naming service.	
2	resolve	Executes the resolve method of the naming service.	
3	destroy	Executes the unbind and destroy methods of the naming service.	
4	list	Executes the list method of the naming service.	

(4) Interface Repository

Interface Repository sample programs are listed in the follwing table.

Table E.6 Interface Repository

No.	Type	Outline	Remarks
1	csample1 cppsample1 cblsample1	Obtains OperationDef object information.	
2	csample2 cppsample2 cblsample2	Obtains StructDef and AliasDef object information.	

(5) Four Arithmetic Operations

Arithmetic operation sample programs and their types are listed in the following tables. Static invocation and static skeleton interface samples are in Table E.7 Four Arithmetic Operations. Dynamic invocation interface samples are in Table E.8 Four Arithmetic Operations (DII).

Table E.7 Four Arithmetic Operations

No.	Type	Outline	Remarks
1	c	Performs a zero division exception.	
2	c++	Performs a zero division exception.	
3	COBOL	Performs a zero division exception.	
4	java	Implements the four arithmetic operations on the value received from the Web browser, and displays the result.	

Table E.8 Four Arithmetic Operations (DII)

No.	Type	Outline	Remarks
1	c_dii	Implements the four arithmetic operations and displays the result.	
2	c++_dii	Implements the four arithmetic operations and displays the result.	

(6) Process bind

Process bind sample programs are listed in the following table.

Table E.9 Process bind

No.	Type	Outline	Remarks
1	ssn_sample	Implements session management using the object to process bind function.	C++ Only
2	ssn_sample2	Implements session management using the object to process bind function, a session timeout occurs.	C++ Only

(7) Other Samples

Windows32

About other sample programs, for thread version samples are in the following table.

Table E.10 Other Samples (Thread Version)

No.	Type	Outline	Remarks
1	simple	Adds and returns the result.	
2	simple_cpp	Returns the accumulated value of the values received from the client.	
3	java	Returns the result of addition on the value received from the Web browser.	

E.1.2 Execution Procedure of Sample Programs

This section explains the execution procedure of the sample programs.

Execute sample programs following execution of the operations described in E.1.3 Notes on Sample Programs.

Remarks

The examples used in this section assume that ObjectDirector is installed in the following folder:

C:\Interstage\ODWIN

E.1.2.1 (a) Sample Programs in C and C++

Windows32

This section describes execution procedures of sample programs written in C and C++.

Example Types	Sub Folder
General data type	complex\samplelist.C\data* complex\samplelist.C++\data*
Four arithmetic operations	CalcSample\c CalcSample\c++
Process bind	complex\samplelist.C++\ssn*
Other	simple_s, _c simple_ccp_s, _c

1) Samples for Basic Data Types, and Others

The following is an example execution of sample\complex\samplelist.C\data\any_s, _c using Microsoft® Visual C++® .NET Standard (*1) (*2):

Server

1. Change the current folder to the following folder:

```
$ cd  C:\Interstage\ODWIN\src\sample\complex\samplelist.C\data\any_s
```

2. Execute the IDL compiler from the command prompt to generate the skeletons.

```
$ IDLc simple.idl         (*3)
```

3. Double-click the solution file (.sln) from Windows Explorer and invoke Visual C++®.
4. Click Solution Explorer on Visual C++® and select the folder ("any_s" in the example). When files other than "simple_s.c" are indicated, delete the files except for "simple_s.c" by selecting Edit | Delete on the menu bar (simple_s.cpp when using C++ samples).
5. Select Project | Add Existing Item from the Visual C++® menu bar.
6. Select "All files" from file types in the dialog box.
7. Select the following files from the indicated files in the dialog box and click OK. (xxx.cpp when using C++ samples)
   - simple.h
   - simple_cdr.c
   - simple_cdr.h
   - simple_skel.c
   - simple_c++.cpp (only if C++ samples)
8. Select Build | Solution Build from the menu bar. The server applications are built. (*4)
9. Register the Implementation Repository ID and object reference required to execute the sample program.

```
$ register.bat
```

10. After the server application build process has terminated normally, select Debug | Start Without Debugging.

    <Console to execute server application is invoked.>

Client

1. Change the current folder to the client application sample program folder copied with the sample program under the folder C:\Interstage\ODWIN\src\sample\complex\samplelist.C\data\any_s:

```
$ cd ..\any_c
```

2. Execute the IDL compiler from the command prompt to generate the stubs.

```
$ IDLc simple.idl          (*3)
```

3. Double-click the solution file (.sln) from Windows Explorer and invoke Visual C++®.
4. Click Solution Explorer and select the folder ("any_c" in the example). When files other than simple_c.c are indicated, delete the files except for simple_c.c by selecting **Edit** | **Delete** on the menu bar (simple_s.cpp when using C++ samples).
5. Select Project | Add Existing Item from the Visual C++® menu bar.
6. Select "All files" from file types in the dialog box.
7. Select the following files from the indicated files in the dialog box and click **OK**. (xxx.cpp when using C++ samples)
   - simple.h

- simple_cdr.c
- simple_cdr.h
- simple_stub.c
- simple_c++.cpp (only if C++ samples)

8. Select Build | Solution Build from the menu bar. The client applications are built. (*4)
9. After the client application build process has terminated normally, select Debug | Start Without Debugging. (*5) (*6)

   <Console to execute client application is invoked, and the execution result is indicated.>
10. Terminate the server application.
11. Terminate the client application. (This step is not required if the client application was terminated after the result was indicated by the console.)
12. Change the current folder from C:\Interstage\ODWIN\src\sample\complex\samplelist.C\data\any_c to server application sample program folder.

```
$ cd ..\any_s
```

Server

Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
$ unregister.bat
```

Notes

*1 Folder whose name has suffix "_s" is for server applications, and folder whose name has suffix "_c" is for client applications.

*2 When client and server applications are used in remote system, after compiling the client applications on server system, the client applications must be copied to client system with using mediums or sharing facilities between client and server system.

*3 When IDL-compiling C++ samples, specify the -vcpp option, for example:

```
IDLc -vcpp simple.idl
```

When executing IDL-compiler, the suffix of specified IDL files must be lowercase.

*4 Warning messages may display during compilation of the sample programs, but they do not cause problems while the programs are in use.

*5 The following warning messages display during execution of exception sample programs. These are output from the sample programs and do not indicate errors on the system or ObjectDirector.

No	Sample Program Name Messages
1	complex\samplelist.C\data\exception ret = [10] id = [IDL:ODsample/exceptest/NOT_FOUND:1.0]:Detail [20] Count [3] env_check: invoke ODsample_getinfo fails
2	complex\samplelist.C++\data\exception 10 NOT_FOUND exception raised! Exception-id = IDL:ODsample/exceptest/NOTFOUND:1.0

*6 The following warning messages, which display during execution of sample programs, have zero-division exception. These are output from the sample programs and do not indicate errors on the system or ObjectDirector.

No	Sample Program Name Messages
1	CalcSample\c env_check: ODdemo_calculator_calculate [IDL:ODdemo/calculator/ZEROPARAM:1.0]fails
2	CalcSample\c++ exception-id = IDL:ODdemo/calculator/ZEROPARAM:1.0

2) Other (Data Input 1)

The following is an example execution of "sample\simple_s, _c" using Microsoft® Visual C++® .NET Standard (*1) (*2).

Server

1. Change the current folder to the following folder:

```
$ cd  C:\Interstage\ODWIN\odwin\src\sample\simple_s
```

2. Execute the IDL compiler on the command prompt to generate the skeletons.

```
$ IDLc  simple.idl
```

3. Double-click the solution file (.sln) from Windows Explorer and invoke Visual C++®.
4. Click Solution Explorer on Visual C++® and select the folder ("simple_s" in the example). When files other than "simple_s.c" are indicated, delete the files except for "simple_s.c" by selecting Edit | Delete on the menu bar.
5. Select Project | Add Existing Item from the Visual C++® menu bar.
6. Select "All files" from file types in the dialog box.
7. Select the following files from the indicated files in the dialog box and click OK.
   - simple.h
   - simple_cdr.c
   - simple_cdr.h
   - simple_skel.c
8. Select Build | Solution Build from the menu bar. The server applications are built. (*3)
9. Register the Implementation Repository ID and object reference required to execute the sample program.

```
$ register.bat
```

10. After the server application build process has terminated normally, select Debug | Start Without Debugging.

    < Console to execute server application is invoked. >

Client

1. Change the current folder to the client application sample program folder copied with the sample program under the folder C:\Interstage\ODWIN\src\sample\simple_s

```
$ cd ..\simple_c
```

2. Execute the IDL compiler from the command prompt to generate the stubs.

```
$ IDLc simple.idl
```

3. Double-click the solution file (.sln) from Windows Explorer and invoke Visual C++®.

4. Click Solution Explorer and select the folder ("simple_c" in the example). When files other than simple_c.c are indicated, delete the files except for simple_c.c by selecting Edit | Delete on the menu bar.
5. Select Project | Add Existing Item from the Visual C++® menu bar.
6. Select "All files" in file types in the dialog box.
7. Select the following files from the indicated files in the dialog box and click OK.
   - simple.h
   - simple_cdr.c
   - simple_cdr.h
   - simple_stub.c
   - Win.c
   - Win.def
   - Win.rc
   - Winfunc.c
   - Win.h (not necessary)
   - Winres.h (not necessary)
8. Select Build | Solution Build from the menu bar. The client applications are built. (*3)
9. After the client application build process has terminated normally, select Debug | Start Without Debugging.

   < When the message box is indicated to execute the client applications, select "init" of bar item "OD".

   Click OK without entering any value into the two fields.

   Confirm the OK messages, terminate the "init" process to click CANCEL. After then, select "test" of bar items "OD" and input the values to each field. The message box to confirm the input value of each field is indicated, then click OK to terminate. Click OK on the message box to confirm operation execution, the addition result of input values is returned. >
10. Terminate the server application.
11. Terminate the client application.
12. Change the current folder from the folder C:\Interstage\ODWIN\src\sample\simple_c to server application sample program folder.

```
$ cd ..\simple_s
```

Server

Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
$ unregister.bat
```

Notes

*1 Folder whose name has suffix "_s" is for server applications, and folder whose name has suffix "_c" is for client applications.

*2 When client and server applications are used in remote system, after compiling the client applications on server system, the client applications must be copied to client system with using mediums or sharing facilities between client and server system.

*3 Warning messages may display during compilation of the sample programs, but these do not cause problems while the programs are in use.

3) Other (Data Input 2)

The following is an example execution of sample\simple_cpp_s, _c using Microsoft® Visual C++® .NET Standard (*1) (*2) (*3):

Server

1. Change the current folder to the following folder:

```
$ cd  C:\Interstage\ODWIN\src\sample\simple_cpp_s
```

2. Execute the IDL compiler from the command prompt to generate the skeletons.

```
$ IDLc -vcpp intf1.
```

3. Double-click the solution file (.sln) from Windows Explorer and invoke Visual C++®.
4. Click Solution Explorer and select the folder ("simple_cpp_s" in the example). When files other than intfl_s.cpp are indicated, delete the files except for intfl_s.cpp by selecting Edit | Delete on the menu bar.
5. Select Project | Add Existing Item from the Visual C++® menu bar.
6. Select "All files" from file types in the dialog box.
7. Select the following files from the indicated files in the dialog box and click OK.
   - intf1.h
   - intf1_cdr.cpp
   - intf1_cdr.h
   - intf1_skel.cpp
   - intf1_c++.cpp
8. Select Build | Solution Build from the menu bar. The server applications are built. (*4)
9. Register the Implementation Repository ID and object reference required to execute the sample program.

```
$ register.bat
```

10. After the server application build process has terminated normally, select Debug | Start Without Debugging.

    < Console to execute server application is invoked. >

Client

1. Change the current folder to the client application sample program folder copied with the sample program under the folder C:\Interstage\ODWIN\src\sample\simple_cpp_s

```
$ cd ..\simple_cpp_c
```

2. Execute the IDL compiler from the command prompt to generate the stubs.

```
$ IDLc -vcpp intf1.idl
```

3. Double-click the solution file (.sln) from Windows Explorer and invoke Visual C++®.
4. Click Solution Explorer and select the folder ("intfl_c" in the example). When files other than intf1_c.cpp are indicated, delete the files except for intf_s.cpp by selecting Edit | Delete on the menu bar.
5. Select Project | Add Existing Item from the Visual C++® menu bar.
6. Select "All files" in file types in the dialog box.
7. Select the following files from the indicated files in the dialog box and click OK.
   - intf1.h
   - intf1_cdr.cpp
   - intf1_cdr.h
   - intf1_stub.cpp
   - intf1_c++.cpp

8. Select Build | Solution Build from the menu bar, and the client applications are built. (*4)
9. After the client application build process has terminated normally, select Debug | Start Without Debugging.

   < Console to execute client application is invoked and input the value for number prompt. >

   < After that, the accumulated value is indicated. >
10. Terminate the server application.
11. Terminate the client application.
12. Change the current folder from folder C:\Interstage\ODWIN\src\sample\simple_cpp_c to server application sample program folder.

```
$ cd ..\simple_cpp_s
```

Server

Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
$ unregister.bat
```

Notes

*1 Folder whose name has suffix "_s" is for server applications, and folder whose name has suffix "_c" is for client applications.

*2 Sample programs corresponding in the directories name except the suffix "_s" and "_c" must be used to combine.

*3 When client and server applications are used in remote system, after compiling the client applications on server system, the client applications must be copied to client system with using mediums or sharing facilities between client and server system.

*4 Warning messages may display during compilation of the sample programs, but these do not cause problems while the programs are in use.

4) Process bind

The following is an example execution of ssn_sample2_s, ssn_sample2_c using Microsoft® Visual C++® .NET Standard (*1) (*2):

Server

1. Change the current folder to the following folder:

```
$ cd C:\Interstage\ODWIN\src\sample\complex\samplelist.C++\ssn\ssn_sample2_s
```

2. Execute the IDL compiler from the command prompt to generate the skeletons.

```
$ IDLc -vcpp simple.idl
```

3. Double-click the solution file (.sln) from Windows Explorer and invoke Visual C++®.
4. Click Solution Explorer on Visual C++® and select the folder ("ssn_sample_s" in the example). Delete files listed other than "ssn_sample_s.cpp" by selecting Edit | Delete on the menu bar.
5. Select Project | Add Existing Item from the Visual C++® menu bar.
6. Select "All files" from file types in the dialog box.
7. Select the following files from the listed files in the dialog box and click OK.
   - simple.h
   - simple_cdr.cpp
   - simple_cdr.h
   - simple_skel.cpp
   - simple_c++.cpp
8. Select Build | Solution Build from the menu bar. The server applications are built. (*3)

9. Register the Implementation Repository ID and object reference required to execute the sample program.

```
$ register.bat
```

10. After the server application build process has terminated normally, select Debug | Start Without Debugging.

    <Console to execute server application is invoked.>

Client

1. Change the current folder to the following folder:

```
$ cd C:\Interstage\ODWIN\src\sample\complex\samplelist.C++\ssn\ssn_sample2_c
```

2. Execute the IDL compiler from the command prompt to generate the stubs.

```
$ IDLc -vcpp simple.idl
```

3. Double-click the solution file (.sln) from Windows Explorer and invoke Visual C++®.
4. Click Solution Explorer and select the folder ("ssn_sample_c" in the example). Delete files listed other than sample_c.cpp by selecting **Edit** | **Delete** on the menu bar.
5. Select Project | Add Existing Item from the Visual C++® menu bar.
6. Select "All files" from file types in the dialog box.
7. Select the following files from the listed files in the dialog box and click **OK**.
   - simple.h
   - simple_cdr.cpp
   - simple_cdr.h
   - simple_stub.cpp
   - simple_c++.cpp
8. Select Build | Solution Build from the menu bar. The client applications are built. (*3)
9. After the client application build process has terminated normally, select Debug | Start Without Debugging. (*4)

   < Console to execute client application is invoked, and the execution result is indicated.>
10. Terminate the server application.

```
$ odcntlque -s IMPL_SSNSAMPLE2
```

11. Terminate the client application.

    This step is not required if the client application terminated after the result was displayed in the console.
12. Change the current folder from:
    - C:\Interstage\ODWIN\src\sample\complex\samplelist.C++\ssn\ssn_sample2_c

    to the server application sample program folder.

```
$ cd ..\ssn_sample2_s
```

Server

Delete the Implementation Repository ID and the object reference for the sample program that is no longer required.

```
$ unregister.bat
```

Note

*1 Server applications folder names have "_s" suffix; client applications folders have "_c" suffix.

*2 When client and server applications are used on a remote system, after compiling the client applications on the server system, they must be copied to the client system.

*3 Warning messages may be displayed during compilation of sample programs, but they do not cause problems while the programs are in use.

*4 The following warning messages display during execution of exception sample programs. These are output from the sample programs and do not indicate system or ObjectDirector errors.

No	Sample Program Name Messages
1	complex/samplelist.C++/ssn/ssn_sample2 SystemException raised! exception-id = IDL:CORBA/StExcep/INV_OBJREF:1.0 minor = 0x464a0928

E.1.2.2 (b) Java Samples

Windows32

This section describes the execution procedure of the Java sample programs.

For details of the environment settings required to compile and execute CORBA applications in Java, refer to "Execution of CORBA Applications" in the "Java Programming Guide" chapter.

Sample type	Storage folder
General data type	complex\samplelist.Java\data*
POA interface	POA\java_1_4*
Four arithmetic operations	CalcSample\java
Other	java

1) General Data Type, POA Interface

The following is an example execution of complex\samplelist.Java\data\array as execution method of the sample program of General data type and POA interface:

The operation for running the application on the WorkUnit is also explained.

1. Set environment CLASSPATH. Setting example is as follows:

```
$ set CLASSPATH=.;C:\Interstage\ODWIN\etc\Class\ODjava4.jar;%CLASSPATH%;
```

Note

ODjava4.jar is set to a CLASSPATH variable from the JDK and JRE at the time of installation. Please also change the CLASSPATH variable when the JDK and JRE are changed.

2. Change the current folder to the following folder.

```
$ cd C:\Interstage\ODWIN\src\SAMPLE\COMPLEX\samplelist.Java\DATA\ARRAY
```

3. The server and client application are made.

```
$ make.bat
```

4. Register the Implementation Repository ID and object reference required to execute the sample program.

```
$ register.bat
```

5. The operations for running/not running the application on the WorkUnit are explained below separately.

   <General data type (when not running the application on the WorkUnit), POA interface>

   Execute the server application.

```
$ exec-SV.bat
```

   <General data type (when running the application on the WorkUnit)>

   1) Match the following simple.wu items to the installation environment:

```
 [Control Option]
  Path: Change the installation folder.
  Current Directory: Change this to the name of the folder used as the
  current folder.
[Application Program]
  CLASSPATH for Application: Change the installation folder.
```

   2) Register the WorkUnit definition required for starting the WorkUnit.

```
isaddwudef simple.wu
```

   3) Start the WorkUnit.

```
isstartwu odsample
```

6. Another DOS window is started. Execute the client application after the process of (1) and (2) is done.

```
$ exec-CL.bat
```

   < Result messages are indicated.>

7. Terminate server application.

   When running the application on the WorkUnit, perform the following operations:

   1) Stop the WorkUnit.

```
isstopwu odsample
```

   2) Delete sample program WorkUnit definitions that are no longer required.

```
isdelwudef odsample
```

8. Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
$ unregister.bat
```

2) Four Arithmetic Operations

The following is an example execution of CalcSample\java:

Install a JBK plug-in for Internet Explorer on the client PC.

Server

1. Change the current folder to the following folder.

```
$ cd C:\Interstage\ODWIN\src\sample\CalcSample\C_S (c++_s is acceptable, too.)
(*1)
```

2. Execute the IDL compiler on the command prompt, and a skeleton is made.

```
$ IDLc simple.idl (*2)
```

3. Double-click the solution file (.sln) from Windows Explorer and invoke Visual C++®.

4. Click Solution Explorer on Visual C++® and select the main project folder. When files other than 'simple_s.c' are indicated, delete the files except 'simple_s.c' using Edit | Remove from the menu bar (filename is simple_s.cpp in the case of C++ sample).
5. Select Project | Add Existing Item from the Visual C++® menu bar.
6. Select "All files" from file types in the dialog box.
7. Select the following files from the indicated files in the dialog box and click OK. (It is xxx.cpp in case as a C++ sample.)
   - simple.h
   - simple_cdr.c
   - simple_cdr.h
   - simple_skel.c
   - simple_c++.cpp (Only in case of a c++ sample)
8. Select Build | Build Solution from the menu bar, and the server applications are built. (*3)
9. Register the Implementation Repository ID and object reference required to execute the sample program.

```
$ register.bat
```

10. After the server application build process has terminated normally, select Debug | Start Without Debugging.

    < Console to execute server application is invoked, the messages which the initialization has completed is indicated. >

11. (Client's preparation) Change the current folder to the following folder.

```
$ cd C:\Interstage\ODWIN\src\sample\CalcSample\java
```

12. (Client's preparation) Create the stub for Java client.

```
$make.bat
```

Client

1. Copy all files under the folder C:\Interstage\ODWIN\src\sample\CalcSample\java on the server to working folder on client system.
2. Change the current folder to working folder.

```
$ cd C:\TEMP
```

3. Compile Java programs using batch file.

```
$ apl-compile.bat
```

4. Double-click CalcSample.html from Windows Explorer and invoke browser.
5. Input numbers or characters on input form, click OK.

Server

1. Terminate server application.
2. Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
$ unregister.bat
```

Notes

*1 Either following server application is used when this sample program is use.

- CalcSample\c_s
- CalcSample\c++_s

*2 When IDL-compiling C++ samples, specify the -vcpp option.

```
Example: IDLc -vcpp simple.idl
```

*3 Warning messages when compiling the sample programs may be indicated, but these do not cause problems in usage.

3) Other

The following is an example execution of java:

Install a JBK plug-in for Internet Explorer on the machine.

Server

1. Change the current folder to the following folder.

```
$ cd C:\Interstage\ODWIN\src\sample\Java\Server
```

2. The IDL compiler is carried out on the command prompt, and a skeleton is made.

```
$ IDLc sample.idl
```

3. Double-click the solution file (.sln) from Windows Explorer and invoke Visual C++®.
4. Click Solution Explorer on Visual C++® and select the main project folder. When files other than 'simple_s.c' are indicated, delete the files except 'simple_s.c' using Edit | Remove from the menu bar (filename is simple_s.cpp in the case of C++ sample).
5. Select Project | Add Existing Item from the Visual C++® menu bar.
6. Select "All files" from file types in the dialog box.
7. Select the following files from the indicated files in the dialog box and click OK (filename is xxx.cpp in the case of C++ sample).
   - sample.h
   - sample_cdr.c
   - sample_cdr.h
   - sample_skel.c
8. Select Build | Build Solution from the menu bar, and the server applications are built. (*1)
9. Register the Implementation Repository ID and object reference required to execute the sample program.

```
$ register.bat
```

10. After the server application build process has terminated normally, select Debug | Start Without Debugging.

    < Console to execute server application is invoked, the messages which the initialization has completed is indicated. >

11. (Client's preparation) Change the current folder to the following folder.

```
$ cd C:\Interstage\ODWIN\src\sample\Java\Client
```

12. (Client's preparation) Create the stub for Java client.

```
$make.bat
```

Client

1. Copy all files under the folder C:\Interstage\ODWIN\src\sample\Java\Client on the server to working folder on client system.
2. Copy all files under the folder C:\Interstage\ODWIN\src\sample\Java\Client on the client to working folder on client system.
3. Compile Java programs using batch file.

```
$ apl-compile.bat
```

4. Double-click AppSample.html from Windows Explorer and invoke browser.
5. Input numbers or characters on input form, click OK.

Server

1. Terminate server application.
2. Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
$ unregister.bat
```

Note

*1 Warning messages when compiling the sample programs may be indicated, but these do not cause problems in usage.

E.1.2.3 (c) Dynamic Invocation Interface

Windows32

This section describes the execution procedure of dynamic invocation interface sample programs.

Sample type	Storage folder
Dynamic invocation interface	complex\samplelist.C\dii* complex\samplelist.C++\dii*
Four arithmetic operations (Client Only)	CalcSample\c_dii CalcSample\c++_dii

1) Dynamic Invocation Interface

The following is an example execution of samples\complex\samplelist.C\dii\dyn1_s, _c using Microsoft® Visual C++® .NET Standard (*1) (*2):

Server

1. Change the current folder to the following folder.

```
$ cd  C:\Interstage\ODWIN\src\sample\complex\samplelist.C\dii\dyn1_s
```

2. Execute the IDL compiler from the command prompt to generate the skeletons.

```
$ IDLc simple.idl          (*3)
```

3. Double-click the solution file (.sln) from Windows Explorer and invoke Visual C++®.
4. Click Solution Explorer on Visual C++® and select the folder ("dyn1_s" in the example). When files other than "simple_server.c" are indicated, delete the files except for "simple_server.c" by selecting Edit | Delete on the menu bar. (*4)
5. Select Project | Add Existing Item from the Visual C++® menu bar.
6. Select "All files" from file types in the dialog box.
7. Select the following files from the indicated files in the dialog box and click OK. (xxx.cpp when using C++ samples)
   - simple.h
   - simple_cdr.c
   - simple_cdr.h
   - simple_skel.c
   - simple_c++.cpp (only if C++ samples)
8. Select Build | Solution Build from the menu bar. The server applications are built. (*5)
9. Register the Implementation Repository ID and object reference required to execute the sample program.

```
$ register.bat
```

10. After the server application build process has terminated normally, select Debug | Start Without Debugging.

    < Console to execute server application is invoked.>

Client

1. Change the current folder to the client application sample program folder copied with the sample program under the folder C:\Interstage\ODWIN\src\sample\complex\samplelist.C\dii\dyn1_s

```
$ cd ..\dyn1_c
```

2. Execute the IDL compiler from the command prompt to generate the stubs.

```
$ IDLc simple.idl         (*3)
```

3. Double-click the solution file (.sln) from Windows Explorer and invoke Visual C++®.
4. Click Solution Explorer and select the folder ("dyn_c" in the example). When files other than simple_client.c are indicated, delete the files except for simple_client.c by selecting Edit | Delete on the menu bar. (*6)
5. Select Project | Add Existing Item from the Visual C++® menu bar.
6. Select "All files" in file types in the dialog box.
7. Select the following files from the indicated files in the dialog box and click OK. (xxx.cpp when using C++ samples)
   - simple.h
   - simple_cdr.c
   - simple_cdr.h
   - simple_stub.c
   - simple_c++.cpp (only if C++ samples)
8. Select Build | Solution Build from the menu bar, and the client applications are built. (*5)
9. After the client application build process has terminated normally, select Debug | Start Without Debugging.

   <Console to execute client application is invoked, then the execution result is indicated.>
10. Terminate the server application.
11. Terminate the client application. (Not required if it is terminated after indicating the result.)
12. Change the current folder from the folder C:\Interstage\ODWIN\src\sample\complex\samplelist.C\dii\dyn1_c to server application sample program folder.

```
$ cd ..\dyn1_s
```

Server

Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
$ unregister.bat
```

Notes

*1 Folder whose name has suffix "_s" is for server applications, and folder whose name has suffix "_c" is for client applications.

*2 When client and server applications are used in remote system, after compiling the client applications on server system, the client applications must be copied to client system with using mediums or sharing facilities between client and server system.

*3 When IDL-compiling C++ samples, specify the -vcpp option.

```
IDLc -vcpp simple.idl
```

*4 The client application name of the C++ sample programs is "simple_c.cpp".

*5 Warning messages may display during compilation of the sample programs, but these do not cause problems while the programs are in use.

*6 The server application name of the C++ sample programs is "simple_s.cpp".

2) Four Arithmetic Operations

The following is an example execution of CalcSample\c_dii using Microsoft® Visual C++® .NET Standard (*1):

Client

1. Change the current folder to the following folder.

```
$ cd C:\Interstage\ODWIN\src\sample\CalcSample\c_dii
```

2. Execute the IDL compiler from the command prompt to generate the stubs.

```
$ IDLc simple.idl         (*2)
```

3. Double-click the solution file (.sln) from Windows Explorer and invoke Visual C++®.
4. Click Solution Explorer and select the folder ("c_dii" in the example). When files other than simple_c.c are indicated, delete the files except for simple_c.c by selecting Edit | Delete on the menu bar. (simple_c.cpp when using C++ samples)
5. Select Project | Add Existing Item from the Visual C++® menu bar.
6. Select "All files" in file types in the dialog box.
7. Select the following files from the indicated files in the dialog box and click OK. (xxx.cpp when using C++ samples)
   - simple.h
   - simple_cdr.c
   - simple_cdr.h
   - simple_stub.c
   - simple_c++.cpp (only if C++ samples)
8. Select Build | Solution Build from the menu bar. The client applications are built. (*3)
9. Register the Implementation Repository ID and object reference required to execute the sample program.

```
$ register.bat
```

Server

1. Change the current folder to the client application sample program folder copied with the sample program under the folder C:\Interstage\ODWIN\src\sample\CalcSample\c_dii

```
$ cd ..\c_s               (*4)
```

2. Execute the IDL compiler from the command prompt to generate the skeletons.

```
$ IDLc  simple.idl        (*2)
```

3. Double-click the solution file (.sln) from Windows Explorer and invoke Visual C++®.
4. Click Solution Explorer on Visual C++® and select the folder ("c_s" in the example). When files other than "simple_s.c" are indicated, delete the files except for "simple_s.c" by selecting Edit | Delete on the menu bar. (simple_s.cpp if C++ samples)
5. Select Project | Add Existing Item from the Visual C++® menu bar.
6. Select "All files" from file types in the dialog box.
7. Select the following files from the indicated files in the dialog box and click OK. (xxx.cpp when using C++ samples)
   - simple.h
   - simple_cdr.c
   - simple_cdr.h

- simple_skel.c
- simple_c++.cpp (only if C++ samples)

8. Select Build | Solution Build from the menu bar. The server applications are built. (*3)
9. After the server application build process has terminated normally, select Debug | Start Without Debugging.

   < Console to execute server application is invoked. >

Client

1. Change the current folder to the following folder.

```
$ cd ..\c_dii
```

2. After the client application build process has terminated normally, select Debug | Start Without Debugging.

   < Console to execute server application is invoked, then the result messages are indicated. >

3. Terminate the server application.
4. Terminate the client application.
5. Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
$ unregister.bat
```

Notes

*1 When client and server applications are used in remote system, after compiling the client applications on server system, the client applications must be copied to client system with using mediums or sharing facilities between client and server system.

*2 When IDL-compiling C++ samples, specify the -vcpp option.

```
IDLc -vcpp simple.idl
```

*3 Warning messages may display during compilation of the sample programs, but these do not cause problems while the programs are in use.

*4 Use the server application under the folder CalcSamplemple\c_s if using CalcSample\c_dii

Use the server application under the folder CalcSample\c++_s if using CalcSample\c++_dii

E.1.2.4 (d) Naming Service Samples

Windows32

This section describes the execution procedure of NamingService sample programs.

Sample type	Storage folder
Naming service	complex\samplelist.C\naming* complex\samplelist.C++\naming*

1) list Method

The following is an example execution of sample\complex\samplelist.C\naming\list using Microsoft® Visual C++® .NET Standard (*1) (*2) :

1. Change the current folder to the following folder.

```
$ cd  C:\Interstage\ODWIN\src\sample\complex\samplelist.C\naming\list
```

2. Double-click the solution file (.sln) from Solution Explorer and invoke Visual C++®.
3. Click FileView on Visual C++® and select the folder ('list' in the example). When files other than 'simple_c.c' are indicated, delete the files except for 'simple_c.c' by selecting Edit | Delete on the menu bar (filename is simple_c.cpp when using C++ samples).

4. Select Build | Solution Build from the menu bar. The client applications are built. (*1)
5. Register the Implementation Repository ID and object reference required to execute the sample program.

```
$ register.bat
```

6. After the server application build process has terminated normally, select Debug | Start Without Debugging.

   <Console to execute client application is invoked, then the execution result is indicated.>

7. Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
$ unregister.bat
```

Notes

*1 The warning messages may be displayed when selecting. They do not indicate problems with building or using the applications.

2) bind, resolve, destroy Methods

The following is an example execution of complex\samplelist.C\naming using Microsoft® Visual C++® .NET Standard:

1. Change the current folder to the following folder.

```
$ cd  C:\Interstage\ODWIN\src\sample\complex\samplelist.C\naming\bind
```

2. Double-click the solution file (.sln) from Windows Explorer and invoke Visual C++®.
3. Click FileView on Visual C++® and select the folder ('bind' in the example). When files other than 'simple_c.c' are indicated, delete the files except for 'simple_c.c' by selecting Edit | Delete on the menu bar (filename is simple_c.cpp when using C++ samples).
4. Select Build | Solution Build from the menu bar. The client applications are built. (*1)
5. Register the Implementation Repository ID and object reference required to execute the sample program.

```
$ register.bat
```

6. After the client application build process has terminated normally, select Debug | Start Without Debugging. (*2)

   <Console to execute client application is invoked, then the execution result is indicated.>

7. Change the current folder to the following folder.

```
$ cd  ..\resolve
```

8. Double-click the solution file (.sln) from Windows Explorer and invoke Visual C++®.
9. Click Solution Explorer on Visual C++® and select the folder ('resolve' in the example). When files other than 'simple_c.c' are indicated, delete the files except for 'simple_c.c' by selecting Edit | Delete on the menu bar (filename is simple_c.cpp when using C++ samples).
10. Select Build | Solution Build from the menu bar. The client applications are built. (*1)
11. After the client application build process has terminated normally, select Debug | Start Without Debugging. (*2)

    <Console to execute client application is invoked, then the execution result is indicated.>

12. Change the current folder to the following folder.

```
$ cd  ..\destroy
```

13. Double-click the solution file (.sln) from Windows Explorer and invoke Visual C++®.
14. Click Solution Explorer on Visual C++® and select the folder ('destroy' in the example). When files other than 'simple_c.c' are indicated, delete the files except for 'simple_c.c' by selecting Edit | Delete on the menu bar (filename is simple_c.cpp when using C++ samples).
15. Select Build | Solution Build from the menu bar. The client applications are built. (*2)

16. After the client application build process has terminated normally, select Debug | Start Without Debugging. (*2)

    <Console to execute client application is invoked, then the execution result is indicated.>

17. Change the current folder to the following folder.

```
$ cd  ..\bind
```

18. Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
$ unregister.bat
```

Notes

*1 The warning messages may be displayed when selecting. They do not indicate problems with building or using the applications.

*2 No output appears when sample programs of bind, resolve and destroy terminate normally.

E.1.2.5 (e) InterfaceRepository Samples

Windows32

This section explains the method of an executing sample of InterfaceRepository:

Sample type	Storage folder
InterfaceRepository	irsample*

The following execution examples are for irsample\csample1 using Microsoft® Visual C++® .NET Standard.

1. Change the current folder to the following folder.

```
$ cd C:\Interstage\ODWIN\src\sample\ irsample\csample1
```

2. Double-click the solution file (.sln) from Windows Explorer and invoke Visual C++®.
3. Click Solution Explorer on Visual C++® and select the folder ('csample1' in the example). When files other than 'irsample1.c' are indicated, delete the files except for 'irsample1.c' by selecting Edit | Delete from the menu bar.
4. Select Build | Solution Build from the menu bar. The server applications are built. (*1)
5. Register the Implementation Repository ID and object reference required to execute the sample program:

```
$ register.bat
```

6. After the client application build process has terminated normally, select Debug | Start Without Debugging.

    <The execution results of client applications are indicated on the message box. >

7. Delete the Implementation Repository ID and object reference of the sample program that is no longer required:

```
$ unregister.bat
```

Notes

*1 The warning messages may be displayed when selecting. They are not problems when building or using the applications.

E.1.2.6 (f) Sample Programs in C and C++

Windows64

This section details how to execute sample programs written in C.

Example Types	Sub Folder
General data type	complex\samplelist.C\data*

Example Types	Sub Folder
	complex\samplelist.C++\data*
Four arithmetic operations	CalcSample\c CalcSample\c++
Process bind	complex\samplelist.C++\ssn*

1) Samples for Basic Data Types, and Others

Using Microsoft® Visual Studio® 2005

The following is an example of executing sample\complex\samplelist.C\data\any_s, _c using Microsoft® Visual Studio® 2005 (*1) (*2):

Server

1. Change from the current folder to the following folder:

```
$ cd  C:\Interstage\ODWIN\src\sample\complex\samplelist.C\data\any_s
```

2. Execute the IDL compiler from the command prompt to generate the skeletons.

```
$ IDLc simple.idl         (*3)
```

3. Copy the any_s folder set to an environment in which Visual Studio® is installed.
4. Double-click the solution file (.sln) in Windows Explorer and invoke Visual Studio®.
5. Select [Build]-[Configuration Manager] from the menu bar of Visual Studio®, and then select "Release" in "Application Configuration". Additionally, select the applicable platform in "Active Solution Platform".
6. Click Solution Explorer on Visual Studio® and select the folder ("any_s" in the example). When files other than "simple_s.c" are indicated, delete the files except for "simple_s.c" by selecting Edit | Delete on the menu bar (simple_s.cpp when using C++ samples).
7. Select Project | Add Existing Item from the Visual Studio® menu bar.
8. Select "All files" from file types in the dialog box.
9. Select the following files from the files listed in the dialog box and click OK (xxx.cpp when using C++ samples):
   - simple.h
   - simple_cdr.c
   - simple_cdr.h
   - simple_skel.c
   - simple_c++.cpp (for C++ samples)
10. Select Build | Solution Build from the menu bar. The server applications are built. (*4)
11. Copy the any_s folder set to the environment in which Interstage is installed (shown below) to replace the existing folder set.

    C:\Interstage\ODWIN\src\sample\complex\samplelist.C\data\any_s
12. Register the Implementation Repository ID and object reference required to execute the sample program.

```
$ register.bat
```

13. In Windows Explorer, double-click any of the files below to execute it:

```
C:\Interstage\ODWIN\src\sample\complex\samplelist.C\data\any_s\x64\Release\any_s.exe
```

<Console to execute the server application is invoked.>

Client

1. Change from the current folder to the client application sample program folder that was copied with the sample program under the folder C:\Interstage\ODWIN\src\sample\complex\samplelist.C\data\any_s:

```
$ cd C:\Interstage\ODWIN\src\sample\complex\samplelist.C\data\any_c
```

2. Execute the IDL compiler from the command prompt to generate the stubs.

```
$ IDLc simple.idl          (*3)
```

3. Copy the any_c folder set to the environment in which Visual Studio® is installed.
4. Double-click the solution file (.sln) in Windows Explorer and invoke Visual Studio®.
5. Select [Build]-[Configuration Manager] from the menu bar of Visual Studio®, and then select "Release" in "Application Configuration". Additionally, select the applicable platform in "Active Solution Platform".
6. Click Solution Explorer and select the folder ("any_c" in the example). When files other than simple_c.c are indicated, delete the files except for simple_c.c by selecting **Edit** | **Delete** on the menu bar (simple_s.cpp when using C++ samples).
7. Select Project | Add Existing Item from the Visual Studio® menu bar.
8. Select "All files" from file types in the dialog box.
9. Select the following files from the files listed in the dialog box and click **OK** (xxx.cpp when using C++ samples):
   - simple.h
   - simple_cdr.c
   - simple_cdr.h
   - simple_stub.c
   - simple_c++.cpp (for C++ samples)
10. Select Build | Solution Build from the menu bar. The client applications are built. (*4)
11. Copy the any_c folder set to the environment in which Interstage is installed (shown below) to replace the existing folder set.

    C:\Interstage\ODWIN\src\sample\complex\samplelist.C\data\any_c
12. In Windows Explorer, double-click any of the files below: (*5) (*6)

    C:\Interstage\ODWIN\src\sample\complex\samplelist.C\data\any_s\x64\Release\any_c.exe

    <Console to execute the client application is invoked, and the execution result is displayed.>
13. Terminate the server application.
14. Terminate the client application. (This step is not required if the client application was terminated after the result was indicated by the console.)
15. Change from the current folder from C:\Interstage\ODWIN\src\sample\complex\samplelist.C\data\any_c to the server application sample program folder.

```
$ cd C:\Interstage\ODWIN\src\sample\complex\samplelist.C\data\any_s
```

Server

Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
$ unregister.bat
```

Notes

*1 Folders with the suffix "_s" are for server applications, and folders with the suffix "_c" are for client applications.

*2 When client and server applications are used in remote systems, after the client applications are compiled on the server system, they must be copied to the client system using media or file sharing between the client and server systems.

*3 When using an IDL compiler to compile C++ samples, specify the -vcpp option, for example:

```
IDLc -vcpp simple.idl
```

When executing the IDL compiler, the suffix of any specified IDL files must be lowercase.

*4 Warning messages may display during compilation of the sample programs, but they do not cause problems while the programs are in use.

*5 The following warning messages are displayed during execution of exception sample programs. These are output from the sample programs and do not indicate errors with the system or ObjectDirector.

No	Sample Program Name Messages
1	complex\samplelist.C\data\exception ret = [10] id = [IDL:ODsample/exceptest/NOT_FOUND:1.0]:Detail [20] Count [3] env_check: invoke ODsample_getinfo fails
2	complex\samplelist.C++\data\exception 10 NOT_FOUND exception raised! Exception-id = IDL:ODsample/exceptest/NOTFOUND:1.0

*6 The following warning messages are displayed during execution of sample programs with a zero-division exception. These are output from the sample programs and do not indicate errors with the system or ObjectDirector.

No	Sample Program Name Messages
1	CalcSample\c env_check: ODdemo_calculator_calculate [IDL:ODdemo/calculator/ZEROPARAM:1.0]fails
2	CalcSample\c++ exception-id = IDL:ODdemo/calculator/ZEROPARAM:1.0

2) Process Bind

Using Microsoft® Visual Studio® 2005

The following is an example execution of ssn_sample2_s, ssn_sample2_c using Microsoft® Visual Studio® 2005 (*1) (*2):

Server

1. Change the current folder to the following folder:

```
$ cd  C:\Interstage\ODWIN\src\sample\complex\samplelist.C++\ssn\ssn_sample2_s
```

2. Execute the IDL compiler from the command prompt to generate the skeletons.

```
$ IDLc -vcpp simple.idl
```

3. Copy the ssn_sample2_s folder set to an environment in which Visual Studio® is installed.
4. Double-click the solution file (.sln) in Windows Explorer and invoke Visual Studio®.
5. Select [Build]-[Configuration Manager] from the Visual Studio® menu bar, and then select "Release" in "Application Configuration". Additionally, select the applicable platform in "Active Solution Platform".

6. Click Solution Explorer on Visual Studio® and select the folder ("ssn_sample_s" in the example). Delete any listed files other than simple_s.c by selecting Edit | Delete on the menu bar.
7. Select Project | Add Existing Item from the Visual Studio® menu bar.
8. Select "All files" from file types in the dialog box.
9. Select the following files from the files listed in the dialog box and click OK
   - simple.h
   - simple_cdr.c
   - simple_cdr.h
   - simple_skel.c
   - simple_c++.cpp
10. Select Build | Solution Build from the menu bar.

    The server applications are built. (*3)
11. Copy the ssn_sample2_s folder set to the environment in which Interstage is installed (shown below) to replace the existing folder set.

    C:\Interstage\ODWIN\src\sample\complex\samplelist.C++\ssn\ssn_sample2_s
12. Register the Implementation Repository ID and object reference required to execute the sample program.

```
$ register.bat
```

13. In Windows Explorer, double-click any of the files below to execute it:

    C:\Interstage\ODWIN\src\sample\complex\samplelist.C++\ssn\ssn_sample2_s\x64\

    Release\ssn_sample_s.exe

    <Console to execute the server application is invoked.>

Client

1. Change the current folder to the following folder:

```
$ cd C:\Interstage\ODWIN\src\sample\complex\samplelist.C++\ssn\ssn_sample2_c
```

2. Execute the IDL compiler from the command prompt to generate the stubs.

```
$ IDLc -vcpp simple.idl
```

3. Copy the ssn_sample2_c folder set to the environment in which Visual Studio® is installed.
4. Double-click the solution file (.sln) in Windows Explorer and invoke Visual Studio®.
5. Select [Build]-[Configuration Manager] from the Visual Studio® menu bar, and then select "Release" in "Application Configuration". Additionally, select the applicable platform in "Active Solution Platform".
6. Click Solution Explorer and select the folder ("ssn_samole_c" in the example). Delete listed files other than simple_c.c by selecting **Edit** | **Delete** on the menu bar.
7. Select Project | Add Existing Item from the Visual Studio® menu bar.
8. Select "All files" from file types in the dialog box.
9. Select the following files from the files listed in the dialog box and click **OK**:
   - simple.h
   - simple_cdr.c
   - simple_cdr.h
   - simple_stub.c

- simple_c++.cpp

10. Select Build | Solution Build from the menu bar. The client applications are built. (*3)
11. Copy the ssn_sample2_c folder set to the environment in which Interstage is installed (shown below) to replace the existing folder set.

    C:\Interstage\ODWIN\src\sample\complex\samplelist.C++\ssn\ssn_sample2_c
12. In Windows Explorer double-click any of the files below: (*4)

    C:\Interstage\ODWIN\src\sample\complex\samplelist.C++\ssn\ssn_sample2_c\x64\Release

    \ssn_sample_c.exe

    <Console to execute the client application is invoked, and the execution result is displayed.>
13. Terminate the server application.

```
$ odcntlque -s IMPL_SSNSAMPLE2
```

14. Terminate the client application.

    This step is not required if the client application terminated after the result was displayed in the console.
15. Change the current folder from:

    - C:\Interstage\ODWIN\src\sample\complex\samplelist.C++\ssn\ssn_sample2_c

    to the server application sample program folder.

```
$ cd ..\ssn_sample2_s
```

Server

Delete the Implementation Repository ID and the object reference for the sample program that is no longer required.

```
$ unregister.bat
```

Note

*1 Server applications folder names are suffixed with "_s"; client applications folders are suffixed with "_c" suffix.

*2 When client and server applications are used on a remote system, the client applications must be compiled on the server system, then copied to the client system.

*3 Ignore warning messages displayed during compilation of sample programs, as they do result in program usage problems.

*4 The following warning messages display during execution of exception sample programs. These are output from the sample programs and do not indicate system or ObjectDirector errors.

No	Sample Program Name Messages
1	complex/samplelist.C++/ssn/ssn_sample2 SystemException raised! exception-id = IDL:CORBA/StExcep/INV_OBJREF:1.0 minor = 0x464a0928

E.1.2.7 (g) Java Samples

Windows64

This section describes the execution procedure of the Java sample programs.

For details of the environment settings required to compile and execute CORBA applications in Java, refer to "Execution of CORBA Applications" in the "Java Programming Guide" chapter.

Sample type	Storage folder
General data type	complex\samplelist.Java\data*
POA interface	POA\java_1_4*
Four arithmetic operations	CalcSample\java

1) General Data Type, POA Interface

The following is an example of using complex\samplelist.Java\data\array as the execution method of the General data type and POA interface sample program:

The operation for running the application on the WorkUnit is also explained.

1. Set the environment CLASSPATH as shown in the following example:

```
set CLASSPATH=.;C:\Interstage\ODWIN\etc\Class\ODjava4.jar;%CLASSPATH%;
```

Note

1. ODjava4.jar is set to a CLASSPATH variable from the JDK and JRE at the time of installation.
2. Change from the current folder to the following folder:

```
cd C:\Interstage\ODWIN\src\SAMPLE\COMPLEX\samplelist.Java\DATA\ARRAY
```

3. The server and client application are made.

```
make.bat
```

4. Register the Implementation Repository ID and object reference required to execute the sample program.

```
register.bat
```

5. The operations for running/not running the application on the WorkUnit are explained below separately.

   <General data type (when not running the application on the WorkUnit), POA interface>

   Execute the server application.

```
exec-SV.bat
```

   <General data type (when running the application on the WorkUnit)>

   1) Match the following simple.wu items to the installation environment:

```
 [Control Option]
  Path: Change the installation folder.
Current Directory: Change this to the name of the folder used as
the current folder.
[Application Program]
  CLASSPATH for Application: Change the installation folder.
```

   2) Register the WorkUnit definition required for starting the WorkUnit.

```
isaddwudef simple.wu
```

   3) Start the WorkUnit.

```
isstartwu odsample
```

6. Another DOS window is started. Execute the client application after the steps (1) and (2) are complete.

```
exec-CL.bat
```

   < Result messages are displayed.>

7. Terminate server application.

   When running the application on the WorkUnit, perform the following operations:

   1) Stop the WorkUnit.

```
isstopwu odsample
```

   2) Delete sample program WorkUnit definitions that are no longer required.

```
isdelwudef odsample
```

8. Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
unregister.bat
```

2) Four Arithmetic Operations

The following is an example execution of CalcSample\java:

Install a JBK plug-in for Internet Explorer on the client machine.

Using Microsoft® Visual Studio® 2005

The following is an example execution using Microsoft® Visual Studio® 2005:

Server

1. Change from the current folder to the following folder:

```
cd C:\Interstage\ODWIN\src\sample\CalcSample\C_S
```

2. Execute the IDL compiler on the command prompt. A skeleton is made.

```
IDLc simple.idl
```

3. Copy the C_S folder set to the environment in which Visual Studio® is installed.
4. Double-click the solution file (.sln) in Windows Explorer and invoke Visual Studio®.
5. Select [Build]-[Configuration Manager] from the menu bar of Visual Studio®, and then select "Release" in "Application Configuration". Additionally, select the applicable platform in "Active Solution Platform".
6. Click Solution Explorer on Visual Studio® and select the main project folder. When files other than 'simple_s.c' are indicated, delete the files except for 'simple_s.c' by selecting Edit | Remove on the menu bar.
7. Select Project | Add Existing Item from the Visual Studio® menu bar.
8. Select "All files" from file types in the dialog box.
9. Select the following files from the files listed in the dialog box and click OK:
   - simple.h
   - simple_cdr.c
   - simple_cdr.h
   - simple_skel.c
10. Select Build | Build Solution from the menu bar. The server applications are built. (*1)
11. Copy the C_S folder set to the environment in which Interstage is installed (shown below) to replace the existing folder set.

```
C:\Interstage\ODWIN\src\SAMPLE\CalcSample\C_S
```

12. Register the Implementation Repository ID and object reference required to execute the sample program.

```
register.bat
```

13. In Windows Explorer double-click any of the files below:

```
C:\Interstage\ODWIN\src\sample\CalcSample\C_S\x64\Release\C_S.exe
```

< Console to execute the server application is invoked, and messages display confirming that the initialization has completed. >

14. On the client, change from the current folder to the following folder:

```
$ cd C:\Interstage\ODWIN\src\sample\CalcSample\java
```

15. On the client, create the stub for the Java client.

```
make.bat
```

Client

1. Copy all files under the folder C:\Interstage\ODWIN\src\sample\CalcSample\java on the server to the working folder on the client system.
2. Change from the current folder to the working folder.

```
cd C:\TEMP
```

3. Compile Java programs using a batch file.

```
apl-compile.bat
```

4. Double-click CalcSample.html from Windows Explorer and invoke the browser.
5. Enter numbers or characters on the displayed form and click OK.

Server

1. Terminate the server application.
2. Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
unregister.bat
```

Notes

*1 Warning messages may display during compilation of the sample programs, but these do not cause problems while the programs are in use.

E.1.2.8 (h) Dynamic Invocation Interface

Windows64

This section describes the execution procedure of dynamic invocation interface sample programs.

Sample type	Storage folder
Dynamic invocation interface	complex\samplelist.C\dii* complex\samplelist.C++\dii*
Four arithmetic operations (Client Only)	CalcSample\c_dii CalcSample\c++_dii

1) Dynamic Invocation Interface

Using Microsoft® Visual Studio® 2005

The following is an example of executing samples\complex\samplelist.C\dii\dyn1_s, _c using Microsoft® Visual Studio® 2005 (*1) (*2):

Server

1. Change from the current folder to the following folder:

```
$ cd  C:\Interstage\ODWIN\src\sample\complex\samplelist.C\dii\dyn1_s
```

2. Execute the IDL compiler from the command prompt to generate the skeletons.

```
$ IDLc simple.idl          (*3)
```

3. Copy the dyn1_s folder set to the environment in which Visual Studio® is installed.
4. Double-click the solution file (.sln) in Windows Explorer and invoke Visual Studio®.
5. Select [Build]-[Configuration Manager] from the menu bar of Visual Studio®, and then select "Release" in "Application Configuration". Additionally, select the applicable platform in "Active Solution Platform".
6. Click Solution Explorer on Visual Studio® and select the folder ("dyn1_s" in the example). When files other than "simple_server.c" are indicated, delete the files except for "simple_server.c" by selecting Edit | Delete on the menu bar. (*4)
7. Select Project | Add Existing Item from the Visual Studio® menu bar.
8. Select "All files" from file types in the dialog box.
9. Select the following files from the files listed in the dialog box and click OK (xxx.cpp when using C++ samples):
   - simple.h
   - simple_cdr.c
   - simple_cdr.h
   - simple_skel.c
   - simple_c++.cpp (for C++ samples)
10. Select Build | Solution Build from the menu bar. The server applications are built. (*5)
11. Copy the dyn1_s folder set to the environment in which Interstage is installed (shown below) to replace the existing folder set.

    C:\Interstage\ODWIN\src\sample\complex\samplelist.C\dii\dyn1_s
12. Register the Implementation Repository ID and object reference required to execute the sample program.

```
$ register.bat
```

13. In Windows Explorer double-click any of the files below:

```
C:\Interstage\ODWIN\src\sample\complex\samplelist.C\dii\dyn1_s\x64\Release\dyn1_s.exe
```

    < Console to execute the server application is invoked.>

Client

1. Change from the current folder to the client application sample program folder that was copied with the sample program under the folder C:\Interstage\ODWIN\src\sample\complex\samplelist.C\dii\dyn1_s

```
$ cd C:\Interstage\ODWIN\src\sample\complex\samplelist.C\dii\dyn1_c
```

2. Execute the IDL compiler from the command prompt to generate the stubs.

```
$ IDLc simple.idl         (*3)
```

3. Copy the dyn1_c folder set to the environment in which Visual Studio® is installed.
4. Double-click the solution file (.sln) in Windows Explorer and invoke Visual Studio®.
5. Select [Build]-[Configuration Manager] from the menu bar of Visual Studio®, and then select "Release" in "Application Configuration". Additionally, select the applicable platform in "Active Solution Platform".

6. Click Solution Explorer and select the folder ("dyn_c" in the example). When files other than simple_client.c are indicated, delete the files except for simple_client.c by selecting Edit | Delete on the menu bar. (*6)
7. Select Project | Add Existing Item from the Visual Studio® menu bar.
8. Select "All files" in file types in the dialog box.
9. Select the following files from the files listed in the dialog box and click OK. (xxx.cpp when using C++ samples):
   - simple.h
   - simple_cdr.c
   - simple_cdr.h
   - simple_stub.c
   - simple_c++.cpp (for C++ samples)
10. Select Build | Solution Build from the menu bar. The client applications are built. (*5)
11. Copy the dyn1_c folder set to the environment in which Interstage is installed (shown below) to replace the existing folder set.

    C:\Interstage\ODWIN\src\sample\complex\samplelist.C\dii\dyn1_c
12. In Windows Explorer double-click any of the files below:

    ```
    C:\Interstage\ODWIN\src\sample\complex\samplelist.C\dii\dyn1_c\x64\Release\dyn1_c.exe
    ```

    <Console to execute the client application is invoked, and the execution result is displayed.>
13. Terminate the server application.
14. Terminate the client application. (Not required if it is terminated after indicating the result.)
15. Change from the current folder from the folder C:\Interstage\ODWIN\src\sample\complex\samplelist.C\dii\dyn1_c to the server application sample program folder.

    ```
    $ cd C:\Interstage\ODWIN\src\sample\complex\samplelist.C\dii\dyn1_s
    ```

Server

Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
$ unregister.bat
```

Notes

*1 Folders with the suffix "_s" are for server applications, and folders with the suffix "_c" are for client applications.

*2 When client and server applications are used in a remote system, after the client applications are compiled on the server system, they must be copied to the client system using media or file sharing between the client and server systems.

*3 When using an IDL compiler to compile C++ samples, specify the -vcpp option.

```
IDLc -vcpp simple.idl
```

*4 The server application name for C++ sample programs is "simple_s.cpp".

*5 Ignore warning messages displayed during sample program compilation, as these do not cause programs usage problems.

*6 The client application name for C++ sample programs is "simple_c.cpp".

2) Four Arithmetic Operations

Using Microsoft® Visual Studio® 2005

The following is an example of executing CalcSample\c_dii using Microsoft® Visual Studio® 2005 (*1):

Client

1. Change from the current folder to the following folder:

```
$ cd C:\Interstage\ODWIN\src\sample\CalcSample\c_dii
```

2. Execute the IDL compiler from the command prompt to generate the stubs.

```
$ IDLc simple.idl         (*2)
```

3. Copy the c_dii folder set to the environment in which Visual Studio® is installed.
4. Double-click the solution file (.sln) in Windows Explorer and invoke Visual Studio®.
5. Select [Build]-[Configuration Manager] from the menu bar of Visual Studio®, and then select "Release" in "Application Configuration". Additionally, select the applicable platform in "Active Solution Platform".
6. Click Solution Explorer and select the folder ("c_dii" in the example). When files other than simple_c.c are indicated, delete the files except for simple_c.c by selecting Edit | Delete on the menu bar (simple_c.cpp when using C++ samples).
7. Select Project | Add Existing Item from the Visual Studio® menu bar.
8. Select "All files" in file types in the dialog box.
9. Select the following files from the files listed in the dialog box and click OK (xxx.cpp when using C++ samples):
   - simple.h
   - simple_cdr.c
   - simple_cdr.h
   - simple_stub.c
   - simple_c++.cpp (for C++ samples)
10. Select Build | Solution Build from the menu bar. The client applications are built. (*3)
11. Copy the c_dii folder set to the environment in which Interstage is installed to replace the existing folder set (shown below).

```
C:\Interstage\ODWIN\src\sample\CalcSample\c_dii
```

12. Register the Implementation Repository ID and object reference required to execute the sample program.

```
$ register.bat
```

Server

1. Change from the current folder to the client application sample program folder that was copied with the sample program under the folder C:\Interstage\ODWIN\src\sample\CalcSample\c_dii

```
$ cd C:\Interstage\ODWIN\src\sample\CalcSample\c_s   (*4)
```

2. Execute the IDL compiler from the command prompt to generate the skeletons.

```
$ IDLc  simple.idl        (*2)
```

3. Copy the c_s folder set to the environment in which Visual Studio® is installed.
4. Double-click the solution file (.sln) in Windows Explorer and invoke Visual Studio®.
5. Select [Build]-[Configuration Manager] from the menu bar of Visual Studio®, and then select "Release" in "Application Configuration". Additionally, select the applicable platform in "Active Solution Platform".
6. Click Solution Explorer in Visual Studio® and select the folder ("c_s" in the example). When files other than "simple_s.c" are indicated, delete the files except for "simple_s.c" by selecting Edit | Delete on the menu bar (simple_s.cpp if C++ samples).
7. Select Project | Add Existing Item from the Visual Studio® menu bar.
8. Select "All files" from file types in the dialog box.

9. Select the following files from the files listed in the dialog box and click OK (xxx.cpp when using C++ samples):
   - simple.h
   - simple_cdr.c
   - simple_cdr.h
   - simple_skel.c
   - simple_c++.cpp (for C++ samples)
10. Select Build | Solution Build from the menu bar. The server applications are built. (*3)
11. Copy the c_s folder set to the environment in which Interstage is installed to replace the existing folder set (shown below).

    C:\Interstage\ODWIN\src\sample\CalcSample\c_s
12. In Windows Explorer double-click any of the files below:

```
C:\Interstage\ODWIN\src\sample\CalcSample\c_s\x64\Release\c_s.exe
```

    < Console to execute the server application is invoked. >

Client

1. In Windows Explorer double-click any of the files below:

```
C:\Interstage\ODWIN\src\sample\CalcSample\c_dii\x64\Release\c_dii.exe
```

   < Console to execute the server application is invoked, and the result messages are displayed. >
2. Terminate the server application.
3. Terminate the client application.
4. Change from the current folder to the following folder:

```
$ cd C:\Interstage\ODWIN\src\sample\CalcSample\c_dii
```

5. Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
$ unregister.bat
```

Notes

*1 When client and server applications are used in a remote system, after the client applications are compiled on server system, they must be copied to the client system using media or file sharing between the client and server systems.

*2 When using an IDL compiler to compile C++ samples, specify the -vcpp option.

```
IDLc -vcpp simple.idl
```

*3 Warning messages may display during compilation of the sample programs, but these do not cause problems while the programs are in use.

*4 Use the server application under the folder CalcSamplemple\c_s if using CalcSample\c_dii

Use the server application under the folder CalcSample\c++_s if using CalcSample\c++_dii

E.1.2.9 (i) Naming Service Samples

Windows64

This section describes the execution procedure of NamingService sample programs.

Sample type	Storage folder
Naming service	complex\samplelist.C\naming*

Sample type	Storage folder
	complex\samplelist.C++\naming*

1) list Method

Using Microsoft® Visual Studio® 2005

The following is an example execution using Microsoft® Visual Studio® 2005:

1. Change from the current folder to the following folder:

```
cd  C:\Interstage\ODWIN\src\sample\complex\samplelist.C\naming\list
```

2. Copy the list folder set to the environment in which Visual Studio® is installed.
3. Double-click the solution file (.sln) in Solution Explorer and invoke Visual Studio®.
4. Select [Build]-[Configuration Manager] from the menu bar of Visual Studio®, and then select "Release" in "Application Configuration". Additionally, select the applicable platform in "Active Solution Platform".
5. Select Build | Solution Build from the menu bar. The client applications are built. (*1)
6. Copy the list folder set to the environment in which Interstage is installed (shown below) to replace the existing folder set.

```
C:\Interstage\ODWIN\src\sample\complex\samplelist.C\naming\list
```

7. Register the Implementation Repository ID and object reference required to execute the sample program.

```
register.bat
```

8. In Windows Explorer double-click any of the files below:

```
C:\Interstage\ODWIN\src\sample\naming\list\x64\Release\LIST.exe
```

<Console to execute the client application is invoked, and the execution result is displayed.>

9. Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
unregister.bat
```

Notes

*1 Warning messages may be displayed. They do not indicate problems with building or using the applications.

2) bind, resolve, destroy Methods

Using Microsoft® Visual Studio® 2005

The following is an example execution using Microsoft® Visual Studio® 2005:

1. Change from the current folder to the following folder:

```
cd  C:\Interstage\ODWIN\src\sample\complex\samplelist.C\naming\bind
```

2. Copy the bind folder set to the environment in which Visual Studio® is installed.
3. Double-click the solution file (.sln) in Windows Explorer and invoke Visual Studio®.
4. Select [Build]-[Configuration Manager] from the menu bar of Visual Studio®, and then select "Release" in "Application Configuration". Additionally, select the applicable platform in "Active Solution Platform".
5. Select Build | Solution Build from the menu bar. The client applications are built. (*1)
6. Copy the bind folder set to the environment in which Interstage is installed (shown below) to replace the existing folder set.

```
C:\Interstage\ODWIN\src\sample\complex\samplelist.C\naming\bind
```

7. Register the Implementation Repository ID and object reference required to execute the sample program.

```
register.bat
```

8. In Windows Explorer double-click any of the files below:

```
C:\Interstage\ODWIN\src\sample\complex\samplelist.C\naming
\bind\x64\Release\BIND.exe
```

<Console to execute the client application is invoked, and the execution result is displayed.>

9. Change from the current folder to the following folder:

```
cd C:\Interstage\ODWIN\src\sample\complex\samplelist.C\naming\resolve
```

10. Copy the resolve folder set to the environment in which Visual Studio® is installed.
11. Double-click the solution file (.sln) in Windows Explorer and invoke Visual Studio®.
12. Select [Build]-[Configuration Manager] from the menu bar of Visual Studio®, and then select "Release" in "Application Configuration". Additionally, select the applicable platform in "Active Solution Platform".
13. Select Build | Solution Build from the menu bar. The client applications are built. (*1)
14. Copy the resolve folder set to the environment in which Interstage is installed (shown below) to replace the existing folder set.

```
C:\Interstage\ODWIN\src\sample\complex\samplelist.C\naming\resolve
```

15. In Windows Explorer double-click any of the files below: (*2)

```
C:\Interstage\ODWIN\src\sample\complex\samplelist.C\naming\resolve\
x64\Release\RESOLVE.exe
```

<Console to execute the client application is invoked, and the execution result is displayed.>

16. Change from the current folder to the following folder:

```
cd C:\Interstage\ODWIN\src\sample\complex\samplelist.C\naming\destroy
```

17. Copy the destroy folder set to the environment in which Visual Studio® is installed.
18. Double-click the solution file (.sln) in Windows Explorer and invoke Visual Studio®.
19. Select [Build]-[Configuration Manager] from the menu bar of Visual Studio®, and then select "Release" in "Application Configuration". Additionally, select the applicable platform in "Active Solution Platform".
20. Select Build | Solution Build from the menu bar. The client applications are built. (*1)
21. Copy the destroy folder set to the environment in which Interstage is installed (shown below) to replace the existing folder set.

```
C:\Interstage\ODWIN\src\sample\complex\samplelist.C\naming\destroy
```

22. In Windows Explorer double-click any of the files below: (*2)

```
C:\Interstage\ODWIN\src\sample\complex\samplelist.C\naming\destroy\
x64\Release\DESTROY.exe
```

<Console to execute the client application is invoked, and the execution result is displayed.>

23. Change from the current folder to the following folder:

```
cd C:\Interstage\ODWIN\src\sample\complex\samplelist.C\naming\bind
```

24. Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
unregister.bat
```

Notes

*1 Warning messages may be displayed. They do not indicate problems with building or using the applications.

*2 There is no output when sample bind, resolve and destroy programs terminate normally.

E.1.2.10 (j) InterfaceRepository Samples

Windows64

This section explains an executing sample of InterfaceRepository:

Sample type	Storage folder
InterfaceRepository	irsample*

Using Microsoft® Visual Studio® 2005

The following is an example execution using Microsoft® Visual Studio® 2005:

1. Change from the current folder to the following folder:

```
cd C:\Interstage\ODWIN\src\sample\ irsample\csample1
```

2. Copy the csample1 folder set to the environment in which Visual Studio® is installed.
3. Double-click the solution file (.sln) in Windows Explorer and invoke Visual Studio®.
4. Select [Build]-[Configuration Manager] from the menu bar of Visual Studio®, and then select "Release" in "Application Configuration". Additionally, select the applicable platform in "Active Solution Platform".
5. Select Build | Solution Build from the menu bar. The server applications are built. (*1)
6. Copy the csample1 folder set to the environment in which Interstage is installed (shown below) to replace the existing folder set.

```
C:\Interstage\ODWIN\src\sample\irsample\csample1
```

7. Register the Implementation Repository ID and object reference required to execute the sample program.

```
register.bat
```

8. In Windows Explorer double-click any of the files below:

```
C:\Interstage\ODWIN\src\sample\csample1\x64\Release\CSAMPLE1.exe
```

<The execution results of client applications are displayed in the message box. >

9. Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
unregister.bat
```

Notes

*1 The warning messages may be displayed. They do not indicate problems with building or using the applications.s

E.1.3 Notes on Sample Programs

- When using sample programs, the environment "PATH" must be set as follows.

```
set PATH=C:\Interstage Installation Folder\ODWIN\bin;
set PATH=%PATH%;JDK installation folder\bin;
          (only when using sample programs of the Java language)
```

(The default ObjectDirector-Installation-Folder is c:\Program Files.)

- Sample programs are made as persistent type. Naming service and interface repository service is the interface programming.

- Sample programs use the Naming Service and Interface Repository service. The Naming Service and Interface Repository service therefore need to be invoked in advance.
- When executing IDL-compiler, the suffix of specified IDL files must be lowercase.
- When using sample programs, the following development environment is needed:

  Windows32

  - Microsoft® Visual C++®
  - Microsoft® Visual Basic®

  Windows64

  - Microsoft® Visual Studio® 2005
  - Microsoft Platform SDK for Windows Server 2003 SP1

- Windows32/64

  The sample application has been developed using Microsoft(R) Visual C++(R) .NET Standard 2003. To open a solution (.sln) file using Microsoft(R) Visual Studio(R) 2005 or later, convert the solution (.sln) file using the Visual Studio Conversion Wizard, which is displayed when Microsoft(R) Visual Studio(R) is started.

E.2 Execution Procedure of Portable-ORB Sample Programs

This section describes the execution procedure of the Portable-ORB sample programs.

E.2.1 Execution Procedure of Sample Programs

This section describes the execution procedure of the Portable-ORB sample programs.

For details of the environment settings required to compile and execute CORBA applications in Java, refer to "Execution of CORBA Applications" in the "Java Programming Guide" chapter.

The sample program is stored in a subfolder of the following directories.

```
PORB_SAMPLES : %PORB_HOME%\src\samples
PORB_HOME: Portable-ORB installation folder
```

Note

Because context type data cannot be passed in Portable-ORB, a context type sample cannot be executed in Portable-ORB.

Windows32

Sample type	Storage folder
General data type	complex\samplelist.Java\data*
POA interface	POA\java_1_4*
Four arithmetic operations	%PORB_SAMPLES%\CalcSample\java

1) General Data Type, POA Interface

The following is an example execution of complex\samplelist.Java\data\array as execution method of the sample program of General data type and POA interface. ObjectDirector and Portable-ORB are installed in the same system. Server application is Java server application. The example client application uses a Portable-ORB.

1. Set the CLASSPATH environment variable, as in the following example:

   ```
   set CLASSPATH=.;C:\Interstage\ODWIN\etc\Class\ODjava.jar;%CLASSPATH%;
   ```

   ODjava4.jar is set to a CLASSPATH variable from the JDK and JRE at the time of installation.

2. Change the current folder to the following.

```
cd C:\Interstage\ODWIN\src\SAMPLE\COMPLEX\samplelist.Java\DATA\ARRAY
```

3. The server and client application are made.

```
make.bat
```

4. Register the Implementation Repository ID and object reference required to execute the sample program.

```
register.bat
```

5. Execute the server application.

```
exec-SV.bat
```

6. Another DOS window is started. Set environment CLASSPATH to work Portable-ORB.

```
CLASSPATH=.;%PORB_HOME%\lib\ODporb4.jar;%PORB_HOME%\lib\
CosNaming4.jar;%CLASSPATH%
```

7. Change the current folder to the following folder.

```
cd C:\Interstage\ODWIN\src\SAMPLE\COMPLEX\samplelist.Java\DATA\ARRAY
```

8. Execute the client application.

```
exec-CL.bat
```

< Result messages are indicated.>

9. Terminate server application.

10. Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
unregister.bat
```

2) Four Arithmetic Operations

The following is an example of CalcSample\java using Microsoft® Visual C++® .NET Standard.

ObjectDirector and Portable-ORB are installed in the same system. Server application is used c/c++ sample program. Java Applet is stored onto the Web server in the same system. The executive example when Java Applet and Portable-ORB are downloaded onto the PC and it is used is shown.

Install a JBK plug-in on the client machine.

Server

1. Change the current folder to the following folder.

```
cd C:\Interstage\ODWIN\src\sample\CalcSample\C_S (c++_s is
acceptable, too.) (*1)
```

2. Execute the IDL compiler on the command prompt, and a skeleton is made.

```
IDLc simple.idl (*2)
```

3. Double-click the solution file (.sln) from Windows Explorer and invoke Visual C++®.

4. Click Solution Explorer on Visual C++® and select the main project folder. When files other than 'simple_s.c' are indicated, delete the files except 'simple_s.c' using Edit | Delete from the menu bar(filename is simple_s.cpp in the case of C++ sample).

5. Select Project | Add Existing Item from the Visual C++® menu bar.

6. Select "All files" from file types in the dialog box.

7. Select the following files from the indicated files in the dialog box and click OK. (It is xxx.cpp in case as a C++ sample.)
   - simple.h
   - simple_cdr.c
   - simple_cdr.h
   - simple_skel.c
   - simple_c++.cpp (Only in case of a c++ sample)
8. Select Build | Solution Build from the menu bar, and the server applications are built. (*3)
9. Register the Implementation Repository ID and object reference required to execute the sample program.

```
register.bat
```

10. After the server application build process has terminated normally, select Debug | Start Without Debugging.

    < Console to execute server application is invoked, the messages which the initialization has completed is indicated. >

11. Change the current folder to the following folder.

```
cd %PORB_SAMPLES%\CalcSample\java
```

12. Create the client application.

```
set CLASSPATH=.;%PORB_HOME%\lib\ODporb.jar;
                %PORB_HOME%\lib\CosNaming.jar;%CLASSPATH%
IDLc -java simple.idl
javac -d .  *.java
```

13. Create the jar archive file (client2.jar) for the created client application.

```
jar cvf client2.jar *class ODdemo\*.class ODdemo\calculatorPackage\*.class
```

14. Apply digital signature to the jar archive file for the created client application and the library of Portable-ORB to use. Then make environment setting to enable digital signature on the Windows® client where the client application is to be executed.

Web Server

1. Copy made client application under the optional folder of the document route of the Web server.(APPLETDIR: The folder which an Applet is stored in)

```
mkdir %APPLETDIR%
copy %PORB_SAMPLES%\CalcSample\java\client2.jar %APPLETDIR%
copy %PORB_SAMPLES%\CalcSample\java\CalcSample4.html %APPLETDIR%
copy %PORB_SAMPLES%\CalcSample\java\CalcSample4.js %APPLETDIR%
```

2. Copy the necessary file to download Portable-ORB. (*4)

```
cd %APPLETDIR%
copy %PORB_HOME%\lib\ODporb4_plugin.jar  .
copy %PORB_HOME%\lib\CosNaming4_plugin.jar  .
copy %PORB_HOME%\lib\InterfaceRep4_plugin.jar  .
mkdir etc
copy %PORB_HOME%\etc\* etc
```

3. Set up the environment file of Portable-ORB.

   The host information of the movement environment file stored in %APPLETDIR%\etc is established by using the porbeditenv command.

Client

1. Invoke browser.

   CalcSample4.html stored in the Web server on the started browser is opened with a HTTP protocol. When it double-clicks on CalcSample4.html (use of a file protocol) is made the target of the security check, and it may not work normally.

2. Input numbers on input form, click **OK**.

Server

1. Terminate server application.
2. Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
unregister.bat
```

Notes

*1 Either following server application is used when this sample program is use.

- CalcSample\c_s
- CalcSample\c++_s

*2 When IDL-compiling C++ samples, specify the -vcpp option.

```
IDLc -vcpp simple.idl
```

*3. The warning messages may be displayed when selecting. They are not problems when building or using the applications.

*4 Change ARCHIVE in HTML so that it can download, and specify a position of storage of the movement environment file with PORB_HOME when it is used without copying it on the Applet and the same folder. Refer to "Portable-ORB Operation Environment File Settings" in the "Java Programming Guide" chapter for the way of specifying PORB_HOME.

Windows64

Sample type	Storage folder
General data type	complex\samplelist.Java\data*
POA interface	POA\java_1_4*
Four arithmetic operations	%PORB_SAMPLES%\CalcSample\java

1) General Data Type, POA Interface

The following is an example of using complex\samplelist.Java\data\array as the execution method of the General data type and POA interface sample program. ObjectDirector and Portable-ORB are installed in the same system. The Server application is the Java server application. The example client application uses a Portable-ORB.

1. Set environment CLASSPATH as follows:

   ```
   set CLASSPATH=.;C:\Interstage\ODWIN\etc\Class\ODjava4.jar;%CLASSPATH%;
   ```

   ODjava4.jar is set to a CLASSPATH variable from the JDK and JRE at the time of installation.

2. Change from the current folder to the following:

   ```
   cd C:\Interstage\ODWIN\src\SAMPLE\COMPLEX\samplelist.Java\DATA\ARRAY
   ```

3. The server and client application are made.

   ```
   make.bat
   ```

4. Register the Implementation Repository ID and object reference required to execute the sample program.

   ```
   register.bat
   ```

5. Execute the server application.

```
exec-SV.bat
```

6. Another DOS window is started. Set environment CLASSPATH to work Portable-ORB.

```
CLASSPATH=.;%PORB_HOME%\lib\ODporb4.jar;%PORB_HOME%\lib\
CosNaming4.jar;%CLASSPATH%
```

7. Change from the current folder to the following folder:

```
cd C:\Interstage\ODWIN\src\SAMPLE\COMPLEX\samplelist.Java\DATA\ARRAY
```

8. Execute the client application.

```
exec-CL.bat
```

< Result messages are displayed.>

9. Terminate the server application.
10. Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
unregister.bat
```

2) Four Arithmetic Operations

The example below describes execution under the following conditions:

- ObjectDirector and Portable-ORB are installed on the same system.
- The server application uses the sample C program.
- The java applet is stored on the Web server of the same system.
- The java applet and Portable-ORB are downloaded onto the machine.

Install a JBK plug-in on the client machine.

Using Microsoft® Visual Studio® 2005

The following is an example execution using Microsoft® Visual Studio® 2005:

Server

1. Change from the current folder to the following folder:

```
cd C:\Interstage\ODWIN\src\SAMPLE\CalcSample\C_S
```

2. Execute the IDL compiler on the command prompt. A skeleton is made.

```
IDLc simple.idl
```

3. Copy the C_S folder set to the environment in which Visual Studio® is installed.
4. Double-click the solution file (.sln) in Windows Explorer and invoke Visual Studio®.
5. Select [Build]-[Configuration Manager] from the menu bar of Visual Studio®, and then select "Release" in "Application Configuration". Additionally, select the applicable platform in "Active Solution Platform".
6. Click Solution Explorer on Visual Studio® and select the main project folder. When files other than 'simple_s.c' are indicated, delete the files except for 'simple_s.c' by selecting Edit | Delete on the menu bar.
7. Select Project | Add Existing Item from the Visual Studio® menu bar.
8. Select "All files" from file types in the dialog box.

9. Select the following files from the files listed in the dialog box and click OK:
   - simple.h
   - simple_cdr.c
   - simple_cdr.h
   - simple_skel.c
10. Select Build | Solution Build from the menu bar. The server applications are built. (*1)
11. Copy the C_S folder set to the environment in which Interstage is installed (shown below) to replace the existing folder set.

```
C:\Interstage\ODWIN\src\SAMPLE\CalcSample\C_S
```

12. Register the Implementation Repository ID and object reference required to execute the sample program.

```
register.bat
```

13. In Windows Explorer double-click any of the files below:

```
C:\Interstage\ODWIN\src\sample\CalcSample\C_S\x64\Release\C_S.exe
```

    < Console to execute the server application is invoked, and messages display confirming that initialization has completed. >

14. Change from the current folder to the following folder:

```
cd %PORB_SAMPLES%\CalcSample\java
```

15. Create the client application.

```
set CLASSPATH=.;%PORB_HOME%\lib\ODporb.jar;
                  %PORB_HOME%\lib\CosNaming.jar;%CLASSPATH%
IDLc -java simple.idl
javac -d .  *.java
```

16. Create the jar archive file (client2.jar) for the created client application.

```
jar cvf client2.jar *class ODdemo\*.class ODdemo\calculatorPackage\*.class
```

17. Apply the digital signature to the jar archive file for the created client application and the Portable-ORB library to use. Then configure the environment to enable digital signature on the Windows® client where the client application is to be executed.

Web Server

1. Copy the made client application under the optional folder of the document route of the Web server.(APPLETDIR: The folder which an Applet is stored in)

```
mkdir %APPLETDIR%
copy %PORB_SAMPLES%\CalcSample\java\client2.jar %APPLETDIR%
copy %PORB_SAMPLES%\CalcSample\java\CalcSample4.html %APPLETDIR%
copy %PORB_SAMPLES%\CalcSample\java\CalcSample4.js %APPLETDIR%
```

2. Copy the necessary file to download Portable-ORB. (*2)

```
cd %APPLETDIR%
copy %PORB_HOME%\lib\ODporb4_plugin.jar  .
copy %PORB_HOME%\lib\CosNaming4_plugin.jar  .
copy %PORB_HOME%\lib\InterfaceRep4_plugin.jar  .
mkdir etc
copy %PORB_HOME%\etc\* etc
```

3. Set up the Portable-ORB environment file.

   The host information of the movement environment file stored in %APPLETDIR%\etc is established by using the porbeditenv command.

Client

1. Invoke browser.

   A HTTP protocol is used to open the file CalcSample4.html, stored on the Web server where the browser is launched. Double-clicking CalcSample4.html (use of a file protocol), sets it as the security check target, so it may not run normally.

2. Enter numbers on the displayed form and click **OK**.

Server

1. Terminate the server application.
2. Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
unregister.bat
```

Web Server

1. Copy the made client application under the optional folder of the document route of the Web server.(APPLETDIR: The folder which an Applet is stored in)

```
mkdir %APPLETDIR%
copy %PORB_SAMPLES%\CalcSample\java\client2.jar %APPLETDIR%
copy %PORB_SAMPLES%\CalcSample\java\CalcSample4.html %APPLETDIR%
copy %PORB_SAMPLES%\CalcSample\java\CalcSample4.js %APPLETDIR%
```

2. Copy the necessary file to download Portable-ORB. (*2)

```
cd %APPLETDIR%
copy %PORB_HOME%\lib\ODporb4_plugin.jar  .
copy %PORB_HOME%\lib\CosNaming4_plugin.jar  .
copy %PORB_HOME%\lib\InterfaceRep4_plugin.jar  .
mkdir etc
copy %PORB_HOME%\etc\* etc
```

3. Set up the environment file of Portable-ORB.

   The host information of the movement environment file stored in %APPLETDIR%\etc is established by using the porbeditenv command.

Client

1. Invoke browser.

   A HTTP protocol is used to open the file CalcSample4.html stored on the Web server where the browser is launched. Double-clicking CalcSample4.html (use of a file protocol), sets it as the security check target, so it may not run normally.

2. Enter numbers on the displayed form and click **OK**.

Server

1. Terminate the server application.
2. Delete the Implementation Repository ID and object reference of the sample program that is no longer required.

```
unregister.bat
```

Notes

*1 Warning messages may be displayed. They do not indicate problems with building or using the applications.

*2 Change ARCHIVE to HTML so that it can be download. Specify a location to store the operating environment file with PORB_HOME when it is used without being copied to the same folder as the Applet. Refer to "Portable-ORB Operation Environment File Settings" in the "Java Programming Guide" chapter for details on how to specify PORB_HOME.

Appendix F Sample Programs (Solaris(TM) Operating System/Linux)

This appendix provides sample programs for use in a Solaris OS/Linux environment.

F.1 Sample Programs of CORBA Service

This section describes sample programs provided by CORBA Service.

F.1.1 Types of Sample Programs

The following table lists the types of sample programs provided by CORBA Service (ObjectDirector). For building and executing sample programs, refer to each description (Execution procedure column in the following table) in "F.1.2 Sample Programs Execution Procedure".

Linux64

C++ Dynamic Skeleton Interface sample programs cannot be used.

Table F.1 Sample Programs

Sample type		Storage directory (*1)	Language	Execution procedure
General data type	thread type	complex/samplelist.C/data complex/samplelist.C++/data complex/samplelist.Java/data complex/samplelist.C/data_win (*2)	C C++ Java C	(a) (b) (f)
	process type	complex/samplelist.C_nt/data complex/samplelist.C++_nt/data complex/samplelist.COBOL/data	C C++ COBOL	(a)
Dynamic invocation interface	thread type	complex/samplelist.C/dii complex/samplelist.C++/dii	C C++	(c)
	process type	complex/samplelist.C_nt/dii complex/samplelist.C++_nt/dii	C C++	(c)
Naming Service		complex/samplelist.C/naming complex/samplelist.C++/naming	C C++	(d)
Interface Repository		irsample	C, C++ COBOL	(e)
Four arithmetic operations	Static invocation	CalcSample/c CalcSample/c++ CalcSample/COBOL CalcSample/java (*2)	C C++ COBOL Java	(a) (b)
	DII	CalcSample/c_dii CalcSample/c++_dii	C C++	(c)
POA Interface	thread type	POA/java_1_4	Java	(b)
Process bind		complex/samplelist.C++/ssn	C++	(a)

Sample type		Storage directory (*1)	Language	Execution procedure
Other	thread type	simple simple_cpp java (*2)	C C++ Java	(a) (b)
	process type	complex/samplelist.C_nt/simple complex/samplelist.C++_nt/simple_cpp	C C++	(a)
	Making so library	simple_so simple_ccp_so	C C++	(a)

*1 All of the sample programs are located in the following directory. In this section, relative paths are used to specify sample program directories.

Solaris32/64

```
/opr/FSUNod/src/samples
```

Linux32/64

```
/opt/FJSVod/src/samples
```

*2 Sample programs require the Windows(R) client of CORBA service.

F.1.1.1 Sample Programs

This section outlines the sample programs for each type of sample program distributed.

(1) General Data Type

General data type sample programs and their types are listed in the following table. The same types are provided for thread and non-thread version.

Table F.2 General Data Type

No.	Type	Outline	Remarks
1	any	Passes any type data.	
2	array	Passes array type data.	
3	attribute	Passes long and string data as attribute.	
4	context	Passes context type data.	
5	exception	Passes user-defined type exceptions.	C, C++ only
6	sequence	Passes sequence type data.	
7	string	Passes string type data.	
8	struct	Passes structure type data having long and string types as elements.	
9	union	Passes union type data.	
10	other	Passes general type data.	COBOL only

(2) Dynamic Invocation Interface

Dynamic invocation interface sample programs and their types are listed in the following table. The same types are provided for thread and non-thread version.

Table F.3 Dynamic Invocation Interface

No.	Type	Outline	Remarks
1	dyn1	Passes long type data.	
2	dyn2	Passes structure type data having long and char types as elements.	
3	dyn3	Passes union type data.	
4	dii_long	Passes long type data.	C++ only
5	dii_string	Passes string type data.	C++ only
6	dii_struct	Passes struct type data.	C++ only
7	dii_sequence	Passes sequence type data.	C++ only

(3) NamingService

Naming Service sample programs are listed in the following table.

Table F.4 Naming Service

No.	Type	Outline	Remarks
1	bind	Executes the bind method of the Naming Service.	
2	resolve	Executes the resolve method of the Naming Service.	
3	destroy	Executes the unbind and destroy methods of the Naming Service.	
4	list	Executes the list method of the Naming Service.	

(4) InterfaceRepository

InterfaceRepository sample programs are listed in the following table.

Table F.5 Interface Repository

No.	Type	Outline	Language	Remarks
1	csample1 cppsample1 cblsample1	Obtains OperationDef object information.	C C++ COBOL	
2	csample2 cppsample2 cblsample2	Obtains StructDef and AliasDef object information.	C C++ COBOL	

(5) Four Arithmetic Operations

Arithmetic operation sample programs and their types are listed in the following tables. Static invocation and static skeleton interface samples are in Table F.6 Four Arithmetic Operations (Static Interface). Dynamic invocation interface samples are in Table F.14 Dynamic Invocation Interface Sample Program and Execution Procedures.

Table F.6 Four Arithmetic Operations (Static Interface)

No.	Type	Outline	Language	Remarks
1	c	Performs a zero division exception.	C	
2	c++	Performs a zero division exception.	C++	
3	COBOL	Performs a zero division exception.	COBOL	
4	java	Implements the four arithmetic operations on the value received from the Web browser, and displays the result.	Java	

Table F.7 Four Arithmetic Operations (DII)

No.	Type	Outline	Language	Remarks
1	c_dii	Implements the four arithmetic operations and displays the result.	C	
2	c++_dii	Implements the four arithmetic operations and displays the result.	C++	

(6) POA Interface

The following table lists the POA interface sample programs and their types.

Table F.8 POA Interface

No.	Type	Outline	Language	Remarks
1	DefaultServant	Uses the default interface. Each client uses the same Servant object.	Java	
2	Factory-1	Each client using a Factory object uses a different Servant object. The object reference and interface are created inside the Factory.	Java	
3	Factory-2	Each client using a Factory object uses a different Servant object. The object reference is created inside the Factory, and the interface is created inside the ServantObject when a request is received.	Java	
4	ServantLocator	Interface management is entrusted to a user-created ServantManager object.	Java	
5	AdapterActivator-1	POA objects are created when required. This type suits applications that implement multiple POA objects. The AdapterActivator runs when the find POA method is invoked.	Java	
6	AdapterActivator-2	A POA object is created when the first request is received. This type suits applications that implement multiple POA objects. The AdapterActivator runs when the first request is received.	Java	

(7) Process bind

The sample programs that use process bind and the types are shown in the table below.

No.	Type	Outline	Language	Remarks
1	ssn_sample	Performs process bind.	C++	
2	ssn_sample2	Performs process bind. The removal of the bind relationship occurs because of a session timeout.	C++	

(8) Other Samples

Thread version samples are in Table F.9 Other Samples (Thread Version), non-thread version samples are in Table F.10 Other Samples (Non-thread Version), and so library samples are in Table F.11 so Library Samples.

Table F.9 Other Samples (Thread Version)

No.	Type	Outline	Remarks
1	simple	Adds and returns the result.	
2	simple_cpp	Returns the accumulated value of the values received from the client.	

No.	Type	Outline	Remarks
3	java	Returns the result of addition on the value received from the Web browser.	

Table F.10 Other Samples (Non-thread Version)

No.	Type	Outline	Remarks
1	simple	Adds and returns the result.	
2	simple_cpp	Returns the accumulated value of the values received from the client.	

Table F.11 so Library Samples

No.	Type	Outline	Remarks
1	simple_so	Adds and returns the result.	
2	simple_cpp_so	Returns the accumulated value of the values received from the client.	

F.1.2 Sample Programs Execution Procedure

This section describes the execution procedure of the sample programs. Execute sample programs following execution of the operations described in Notes on Sample Programs.

(a) C, C++, and COBOL sample programs

(b) Java sample programs

(c) Dynamic launch interface

(d) Naming Service

(e) Interface Repository

(f) PC client

Remarks

In all examples, the following directory names and environment variables are used.

$OD_HOME:

The installation directory of the CORBA Service

(Solaris: /opt/FSUNod)

(Linux: /opt/FJSVod)

$SAMPLES:

The sample program directory

($OD_HOME/src/samples)

C:\Interstage\ODWIN:

The installation directory of CORBA Service Windows(R) Client

F.1.2.1 (a) Sample Programs in C, C++ and COBOL

Linux64

C++ and COBOL Sample Programs cannot be used.

This section describes execution procedures of sample programs written in C, C++ and COBOL. Table

The following table lists the sample types and execution procedures.

Table F.12 C, C++ and COBOL Sample Programs and Execution Procedures

Sample type	Sub directory
General data type	complex/samplelist.C/data/* complex/samplelist.C++/data/* complex/samplelist.C_nt/data/* complex/samplelist.C++_nt/data/* complex/samplelist.COBOL/data/*
Four arithmetic operations	CalcSample/c CalcSample/c++ CalcSample/COBOL
Other (display) (C only)	Simple complex/samplelist.C_nt/simple simple_so
Other (data input) (C++ only)	simple_cpp complex/samplelist.C++_nt/simple_cpp simple_ccp_so
Process bind (C++ only)	complex/samplelist.C++/ssn/*

1) Samples for Basic Data Types, Four Arithmetic Operations, and Others (Display)

As a sample for basic data types, four arithmetic operations, and others (display), the following sample procedure shows an example execution of simple:

1. Move the current directory to the following directory:

```
cd  $SAMPLES/simple
```

2. Create the server and client applications. (*1)

```
make
```

3. Register the implementation repository ID and object reference required to execute the sample program.

```
register
```

4. Execute the server application in the background. (*2)

```
simple_s &
```

5. Execute the client application. (*2) (*3)

```
simple_c &
```

<The execution result is displayed.>

6. Stop the server application.
7. Delete the implementation repository ID and object reference of the sample program that is no longer required.

```
unregister
```

Notes

*1 A warning message as shown below may be output when sample programs in C++ are compiled. It does not indicate an operational problem.

Solaris32/64

```
UX:make: WARNING: predecessor cycle (simple_stub_c.o)
"simple_stub_c++.C", line 330: warning:_result used before set
```

Linux32/64

```
xxx.C: In function 'void _dt_intf1 (void *)':
xxx.C:xx: warning: ...
In the any type, a warning message as shown below may be output.
simple_s.c: In function 'void print_any(const CORBA::Any &)':
simple_s.c:34: warning: choosing 'CORBA::String_var::operator char *()'
over 'CORBA::String_var::operator const char *() const'
simple_s.c:34: warning:   for conversion from 'CORBA::String_var' to 'const char *'
simple_s.c:34: warning:   because conversion sequence for the argument is better
```

*2 In a Solaris system, the server application name of each sample program under complex/samplelist.COBOL/data is XXX_s (XXX: three-digit numeric). The client application name is XXX_c (XXX: three-digit numeric).

*3 The messages shown in the table below list sample program messages output when exception sample programs are executed. They do not indicate system or CORBA Service failure.

No	Sample program name	Output message
1	complex/samplelist.C/data/ exception	ret = [10] id = [IDL:ODsample/exceptest/NOT_FOUND:1.0]:Detail [20] Count [3] env_check: invoke ODsample_getinfo fails
2	complex/samplelist.C++/ data/exception	10 NOT_FOUND exception raised! Exception-id = IDL:ODsample/exceptest/NOTFOUND:1.0
3	complex/samplelist.C_nt/ data/exception	ret = [10] id = [IDL:ODsample/exceptest/NOT_FOUND:1.0]:Detail [20] Count [3] env_check: invoke ODsample_getinfo fails
4	complex/samplelist.C++_nt/data/ exception	10 NOT_FOUND exception raised! Exception-id = IDL:ODsample/exceptest/NOTFOUND:1.0

The messages shown in the table below are output when sample programs performing zero division exception are executed. They do not indicate system or CORBA Service failure.

No	Sample program name	Output message
1	CalcSample/c	env_check: ODdemo_calculator_calculate [IDL:Oddemo/calculator/ZEROPARAM:1.0]fails
2	CalcSample/c++	exception-id = IDL:ODdemo/calculator/ZEROPARAM:1.0
3	Calcsample/COBOL	USER-EXCEPTION: ODdemo_calculator_calculate ID: IDL:ODdemo/calculator/ZEROPARAM:1.0

2) Other (Data Input)

The following sample program of data input shows an example execution of simple_cpp:

1. Move the current directory to the following directory:

```
cd $SAMPLES/simple_cpp
```

2. Create the server and client applications. (*1)

```
make
```

3. Register the implementation repository ID and object reference required to execute the sample program.

```
register
```

4. Execute the server application in the background.

```
Intfl_s &
```

5. Execute the client application.

```
intfl_c
input> <Enter the value.>
```

<The execution result is output.>

6. Stop the server application.

7. Delete the implementation repository ID and object reference of the sample program that is no longer required.

```
unregister
```

Notes

*1 A warning message as shown below may appear while compiling C++ sample programs. Ignore it. It does not affect program execution.

```
UX:make: WARNING: predecessor cycle (simple_stub_c++.o)
```

3) Process bind

The following example execution of ssn_sample2 uses the process bind sample program execution method:

1. Move the current directory to the following directory:

```
cd $SAMPLES/complex/samplelist.C++/ssn/ssn_sample2
```

2. Create the server and client applications.

```
make
```

3. Register the implementation repository ID and object reference required to execute the sample program.

```
register
```

4. Execute the server application in the background.

```
simple_s &
simple_s &
simple_s &
simple_s &
```

5. Execute the client application. (*1)

```
simple_c
```

<The execution result is output.>

6. Stop the server application.

```
odcntlque -s IMPL_SSNSAMPLE2
```

7. Delete the implementation repository ID and object reference of the sample program that is no longer required.

```
unregister
```

Notes

*1 The messages shown in the table below list sample program messages output when exception sample programs are executed. They do not indicate system or CORBA Service failure.

No	Sample program name	Output message
1	complex/samplelist.C++/ssn/ ssn_sample2	SystemException raised! exception-id = IDL:CORBA/StExcep/INV_OBJREF:1.0 minor = 0x464a0928

F.1.2.2 (b) Java Samples

This section describes the execution procedure of the Java sample programs. The following table lists the sample types and execution procedures.

For details of the environment settings required to compile and execute CORBA applications in Java, refer to "Execution of CORBA Applications" in the "Java Programming Guide" chapter.

Table F.13 Java Sample Program and Execution Procedures

Sample type	Storage directory
General data type	complex/samplelist.Java/data/*
POA interface	POA/java_1_4/*
Four arithmetic operations	CalcSample/java
Other	java

1) General Data Type and POA Interface

The following general data type and POA interface sample shows an example execution of complex/samplelist.Java/data/array.

1. Set the CLASSPATH environment variable.

```
CLASSPATH=.:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH
export  CLASSPATH
```

2. Set the LD_LIBRARY_PATH environment variable.

```
LD_LIBRARY_PATH=$OD_HOME/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

3. Set the PATH environment variable. Specify the directory of JDK/JRE used.

```
PATH=$JAVA_HOME/bin:$PATH
export PATH
```

4. Move the current directory to the following directory:

```
cd $SAMPLES/complex/samplelist.Java/data/array
```

5. Create server and client applications.

```
make
```

6. Register the Implementation Repository ID and object references required for execution of the sample program.

```
register
```

7. The operating procedures for both using and not using the WorkUnit are described below:

   <General data type (not using the WorkUnit), POA interface>

   Execute the server application.

```
exec-SV
```

   <General data type (using the WorkUnit)>

   1) Adjust the following simple.wu items according to the operating environment:

   [Control Option]

   Path: Specify the JDK/JRE directory that is used.

   [Application Program]

   CLASSPATH for Application (the second one) the Java library (ODjava4.jar) according to the JDK/JRE that is used.

   2) Register the WorkUnit definition required to start the WorkUnit.

```
isaddwudef simple.wu
```

   3) Start the WorkUnit.

```
isstartwu ODSAMPLE
```

8. Activate a separate terminal and perform Steps 1 and 2 above, then execute the client application.

```
exec-CL
```

   <The execution results are displayed.>

9. Close the server application.

   To use the WorkUnit for operation, perform the following steps:

   1) Stop the WorkUnit.

```
isstopwu ODSAMPLE
```

   2) Delete WorkUnit definitions for unnecessary sample programs.

```
isdelwudef ODSAMPLE
```

10. Delete the sample program Implementation Repository ID and object references that are no longer required.

```
unregister
```

2) Four Arithmetic Operations

The following procedure is an example execution of CalcSample/java. On a PC, install the JBK plug-in for Internet Explorer before executing the sample program.

<Solaris/Linux server>

1. Move the current directory to the following directory: (*1)

```
cd $SAMPLES/CalcSample/c
```

2. Create the server application. (*2)

```
make
```

3. Register the Implementation Repository ID and object reference required to execute the sample program.

```
register
```

4. Execute the server application in the background.

```
simple_s &
```

5. Move the current directory to the following directory:

```
cd ../java
```

6. Create the client application.

```
make
```

<Windows(R) client>

1. Transfer all files under the $SAMPLES/CalcSample/java directory of the Solaris/Linux server to the Windows(R) client. The following is an example of using ftp: (*3)

   First, create an *ODdemo* directory under the storage destination directory at the Windows(R) client, and create a *calculatorPackage* directory under that directory.

```
ftp <Solaris/Linux server name>
     :
cd $SAMPLES/CalcSample/java
mget *
lcd ODdemo
cd ODdemo
mget *
lcd calculatorPackage
cd calculatorPackage
mget *
```

2. Execute the batch file to compile the Java source.

```
apl-compile.bat
```

3. Create a jar archive file (client2.jar) for the created client application.

```
jar cvf client2.jar *.class ODdemo/*.class ODdemo/calculatorPackage/*.class
```

4. Set the authority for the Java library (ODjava4.jar, client2.jar).

5. Start the browser.

   Double-click CalcSample.html from the Explorer to start the browser.

6. Enter numerics and characters from the input screen and click OK.

<Solaris/Linux server>

1. Stop the server application.

2. Delete the implementation repository ID and object reference of the sample program that is no longer required.

```
unregister
```

Notes

*1 To execute this sample program, use one of the following server applications:

CalcSample/c

CalcSample/c++

*2 A warning message as shown below may be output when sample programs in C++ are compiled. It does not indicate an operational problem.

Solaris32/64

```
"simple_stub_c++.C", line 172: warning:_result used before set
```

Linux32/64

```
xxx.C: In function 'void _dt_intf1 (void *)':
xxx.C:xx: warning: ...
```

*3 Transfer the files to a personal computer as shown below:

All files: Transfer using network ASCII.

3) Other

The following procedure is an example execution of Java. On a PC, install the JBK plug-in for Internet Explorer before executing the sample program.

<Solaris/Linux server>

1. Move the current directory to the following directory:

```
cd $SAMPLES/java
```

2. Move the current directory to the directory on the server application side.

```
cd server
```

3. Create the server application.

```
make
```

4. Register the implementation repository ID and object reference required to execute the sample program.

```
register
```

5. Execute the server application in the background.

```
simple_s &
<A message indicating completed initialization is displayed.>
```

6. Move the current directory to the directory on the client application side.

```
cd ../client
```

7. Create the client application.

```
make
```

<Windows(R) client>

1. Transfer all files under the $SAMPLES/java/client directory of the Solaris/Linux server to the Windows(R) client. The following is an example of using ftp. (*1)

   First, create an *AppOpSample* directory under the storage destination directory at the Windows(R) client.

```
ftp <Solaris/Linux server name>
     :
cd $SAMPLES/java/client
mget *
lcd AppOpSample
cd AppOpSample
mget *
```

2. Copy all files under C:\Interstage\ODWIN\src\sample\Java\Client to the same directory as Step 1.

3. Execute the batch file to compile the Java source.

```
apl-compile.bat
```

4. Create a jar archive file (Client.jar) for the created client application.

```
jar cvf Client.jar *.class AppOpSample/*.class
```

5. Set the authority for the Java library (ODjava4.jar, client2.jar).
6. Start the browser.

   Double-click AppSample.html from the Explorer to start the browser.
7. Enter numerics and characters from the input screen and click OK.

<Solaris/Linux server>

1. Stop the server application.
2. Delete the implementation repository ID and object reference of the sample program that is no longer required.

```
unregister
```

Notes

*1 Transfer the files to a personal computer as shown below:

All files: Transfer using network ASCII.

F.1.2.3 (c) Dynamic Invocation Interface

This section describes the execution procedure of dynamic invocation interface sample programs. The following table lists the sample types and execution procedures.

Table F.14 Dynamic Invocation Interface Sample Program and Execution Procedures

Sample type	Storage directory
Dynamic invocation interface	complex/samplelist.C/dii/* complex/samplelist.C++/dii/* complex/samplelist.C_nt/dii/* complex/samplelist.C++_nt/dii/*
Four arithmetic operations	CalcSample/c_dii CalcSample/c++_dii

1) Dynamic Invocation Interface

The following procedure is an example execution of complex/samplelist.C/dii/dyn1. (*1)

1. Move the current directory to the following directory:

```
cd $SAMPLES/complex/samplelist.C/dii/dyn1
```

2. Create the server and client applications. (*2)

```
make
```

3. Register the implementation repository ID and object reference required to execute the sample program.

```
register
```

4. Execute the server application in the background. (*3)

```
simple_server &
```

5. Execute the client application. (*1) (*4)

```
simple_client
```

<The execution result is displayed.>

6. Stop the server application.

7. Delete the implementation repository ID and object reference of the sample program that is no longer required.

```
unregister
```

Notes

*1 A message like that shown below may appear when executing client applications of complex/samplelist.C++/dii/dyn3 and complex/samplelist.C++_nt/dii/dyn3. It depends on the execution timing. Ignore it. It does not affect application execution.

```
Error poll-response
```

*2 A warning message as shown below may be output when sample programs in C++ are compiled. It does not indicate an operational problem.

Solaris32/64

```
UX:make: WARNING: predecessor cycle (simple_stub_c++.o)
"simple_stub_c++.C", line 159: warning:_result used before set
```

Linux32/64

```
xxx.C: In function 'void _dt_intf1 (void *)':
xxx.C:xx: warning: ...
```

*3 The server application name of the C++ sample program is simple_s.

*4 The client application name of the C++ sample program is simple_c.

2) Four Arithmetic Operations

The following is an example execution of CalcSample/c_dii.

1. Move the current directory to the following directory:

```
cd $SAMPLE/CalcSample/c_dii
```

2. Create the client application.

```
make
```

3. Register the interface information of the server application to the interface repository. Register the implementation repository ID and object reference required to execute the sample program.

```
register
```

4. Move the current directory to the following directory:

```
cd ../c
```

5. Create the server application. (*1)

```
make
```

6. Execute the server application in the background. (*2)

```
simple_s &
```

7. Move the current directory to the following directory:

```
cd ../c_dii
```

8. Execute the client application. (*3)

```
simple_c
```

<The execution result is displayed.>

9. Stop the server application.

10. Delete the implementation repository ID and object reference of the sample program that is no longer required.

```
unregister
```

Notes

*1 In Linux systems, a warning message as shown below may be output when sample programs in C++ are compiled. It does not indicate an operational problem.

```
simple_c.c:77: warning: choosing 'CORBA::ParDescriptionSeq_var::operator
CORBA::ParDescriptionSeq *& ()' over
'CORBA::ParDescriptionSeq_var::operator CORBA::ParDescriptionSeq * () const'
simple_c.c:77: warning: for conversion from
'CORBA::ParDescriptionSeq_var' to 'CORBA::ParDescriptionSeq *'
simple_c.c:77: warning: because conversion sequence for the argument is better
```

*2 For CalcSample/c_dii, use the server application of CalcSample/c.

For CalcSample/c++_dii, use the server application of CalcSample/c++.

*3 A message like that shown below may appear when executing a client application of CalcSample/c++_dii. It depends on the execution timing. Ignore it. It does not affect application execution.

```
Error poll-response
```

F.1.2.4 (d) Naming Service Samples

This section describes the execution procedure of NamingService sample programs. The following table lists the sample types and execution procedures.

Table F.15 Naming Service Sample Program and Execution Procedures

Sample type	Storage directory
Naming Service	complex/samplelist.C/naming/* complex/samplelist.C++/naming/*

1) list Method

The following list method sample shows an example execution of complex/samplelist.C/naming/list.

1. Move the current directory to the following directory:

```
cd $SAMPLES/complex/samplelist.C/naming/list
```

2. Create the client application.

```
make
```

3. Register the implementation repository ID and object reference required to execute the sample program.

```
register
```

4. Execute the client application.

```
simple_c
```

5. Delete the implementation repository ID and object reference of the sample program that is no longer required.

```
unregister
```

2) bind, resolve, destroy Method

The following sample of bind, resolve, and destroy method shows an example execution of complex/samplelist.C/naming/bind.

1. Move the current directory to the following directory:

```
cd $SAMPLES/complex/samplelist.C/naming/bind
```

2. Create the client application.

```
make
```

3. Register the implementation repository ID and object reference required to execute the sample program.

```
register
```

4. Execute the client application. (*1)

```
simple_c
```

5. Move the current directory to the following directory:

```
cd ../resolve
```

6. Create the client application.

```
make
```

7. Execute the client application. (*1)

```
simple_c
```

8. Move the current directory to the following directory:

```
cd ../destroy
```

9. Create the client application.

```
make
```

10. Execute the client application. (*1)

```
simple_c
```

11. Move the current directory to the following directory:

```
cd ../bind
```

12. Delete the implementation repository ID and object reference of the sample program that is no longer required.

```
unregister
```

Notes

*1 Nothing is output if the bind, resolve, and destroy sample programs operate normally.

F.1.2.5 (e) InterfaceRepository Samples

This section explains the method of an executing sample of InterfaceRepository. The following table lists the sample types and execution procedures.

Table F.16 InterfaceRepository Sample Program and Execution Procedures

Sample type	Storage directory
InterfaceRepository	irsample/*

The following procedure is an example execution of irsample/csample1.

1. Move the current directory to the directory of the sample program.

```
cd $SAMPLE/irsample/csample1
```

2. Execute the *make.sh* command. Register the interface information of the sample program to the interface repository. Compile the sample program. (*1)

```
make.sh
```

3. Execute the *irsample1* command.

```
irsample1
```

<The execution result is displayed.>

Notes

*1 The message "od51006" is displayed at initial execution (interface information registration). It does not indicate an operational problem.

F.1.2.6 (f) Windows(R) Client Samples

This section explains the method of an executing sample of the Windows(R) client. The following table lists the sample types and execution procedures.

Table F.17 Windows(R) Client Sample Program and Execution Procedures

Sample type	Storage directory
General data type	complex/samplelist.C/data_win/*

The following procedure is an example execution of complex/samplelist.C/data_win/any.

<Solaris/Linux server>

1. Move the current directory to the following directory:

```
cd $SAMPLES/complex/samplelist.C/data_win/any
```

2. Execute IDL compilation to create the stub and skeleton.

```
IDLc simple.idl
```

3. Move the current directory to the sample directory of the server application corresponding to the sample program under the $SAMPLES/complex/samplelist.C/data_win directory. (*1)

```
cd ../../data/any
```

4. Create the server application.

```
make
```

5. Register the implementation repository ID and object reference required to execute the sample program.

```
register
```

6. Execute the server application.

```
simple_s
```

<Windows(R) client>

1. Transfer all files under the $SAMPLES/complex/samplelist.C/data_win/any directory of the Solaris/Linux server to the Windows(R) client. The following is an example of using ftp. (*2)

```
ftp <Solaris/Linux server name>
     :
get simple.h
     :
mget *.mak
bin
mget *.mdp
```

2. From the Explorer, double-click the transferred project work space (.mdp) file to start Visual C++.
3. Click FileView from the started Visual C++ screen. Click the main project folder (displayed as "any file" in the execution example). If files other than simple_c.c are displayed, select [Edit] - [Delete] from the menu bar to delete all files other than simple_c.c files.
4. Select [Insert] - [Add Project File] from the menu bar of the activated Visual C++.
5. Select "All files" from the file type in the dialog box.
6. Add the following files from the files displayed in the dialog box:
   - simple.h
   - simple_cdr.h
   - simple_cdr.c
   - simple_stub.c
7. Select [Build] - [Update All Dependent Relationships] from the menu bar of the activated Visual C++. Click OK. (*3)
8. Select [Build] - [Build] from the menu bar of the activated Visual C++ to create the client application.
9. When creation of the client application terminates, select [Build] - [Execute] from the menu bar. (*4)

   <The execution result is displayed in the message box.>

<Solaris/Linux server>

1. Stop the server application.
2. Delete the implementation repository ID and object reference of the sample program that is no longer required.

```
unregister
```

Notes

*1 For the server application, use the sample program whose directory name under the $SAMPLES/complex/samplelist.C/data_win directory and directory name under the $SAMPLES/complex/samplelist.C/data match.

*2 Transfer the files to the Windows(R) client as shown below:

Project work space file: Transfer using binary.

Other files: Transfer using network ASCII.

*3 Messages as shown below are displayed at initial selection. They do not indicate an operational problem in the sample program.

```
C:\Interstage\ODWIN\include\orb.h(525): File OM_stub_skel.h cannot be found.
            :
C:\Interstage\ODWIN\include\om_stsk.h(14): File OM_cdr_lib.h cannot be found.
```

*4 The messages shown in the table below are sample program messages output when exception sample programs are accepted. They do not indicate system or CORBA Service failure.

No	Sample program name	Output message
1	complex/samplelist.C/data_win/exception	id = [IDL:ODsample/exceptest/NOT_FOUND:1.0] IDL:Odsample/exceptest/NOT_FOUND:1.0

F.1.3 Notes on Sample Programs

- To use the sample programs, environment variable OD_HOME must be set up.
- The sample programs are created as persistent type sample programs. The Naming Service and Interface Repository service are interface programming.
- When using Java samples, set the environment variable PATH as follows:

  Example: (Using MS-DOS console)

  set PATH=%PATH%;JDK installation directory\bin;
- The sample programs use the Naming Service and Interface Repository. The Naming Service and Interface Repository cache server must be started beforehand.
- The pathname shown below must be added to the LD_LIBRARY_PATH environment variable when process type and COBOL sample programs (Solaris only) are used:

  Example: LD_LIBRARY_PATH=$OD_HOME/lib/nt
- To create an application using Linux for Intel64 (32-bit compatible) Interstage Application Server, the "-m32 -mtune=i386" option must be specified for the gcc/g++ command.

F.2 Portable-ORB Sample Programs

This section describes the Portable-ORB sample programs.

F.2.1 Execution Procedures

This section describes the execution procedure of the sample programs that can be used by the Portable-ORB Java linkage function.

For details of the environment settings required to compile and execute CORBA applications in Java, refer to "Execution of CORBA Applications" in the "Java Programming Guide" chapter.

Note

Because context type data cannot be passed in Portable-ORB, a context type sample cannot be executed in Portable-ORB.

The following table lists the sample types and execution procedures.

The sample programs are stored in the following directories:

Table F.18 Portable-ORB Sample Program and Execution Procedures

Sample type	Storage directory
General data type	$SAMPLES/complex/samplelist.Java/data/* (*1)
POA interface	$SAMPLES/POA/java_1_4/* (*1)
Four arithmetic operations	$PORB_SAMPLES/CalcSample/java

*1 Java sample programs can be used.

$SAMPLES:

The sample program directory of the CORBA Service

($OD_HOME/src/samples)

$OD_HOME:

The installation directory of the CORBA Service

(Solaris: /opt/FSUNod)

(Linux: /opt/FJSVod)

$PORB_SAMPLES :

The sample program directory of Portable-ORB

($PORB_HOME/src/samples)

$PORB_HOME:

The installation directory of Portable-ORB

(/opt/FJSVporb)

1) General Data Type and POA Interface

Solaris32/64

The following general data type and POA interface sample shows an example execution of complex/samplelist.Java/data/array. In this example, the CORBA Service and Portable-ORB are installed on the same system. The server application is a Java server application, and the client application uses Portable-ORB.

1. Set the CLASSPATH environment variable.

```
CLASSPATH=.:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH
export  CLASSPATH
```

2. Set the LD_LIBRARY_PATH environment variable.

```
LD_LIBRARY_PATH=$OD_HOME/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

3. Move the current directory to the following directory:

```
cd $SAMPLES/complex/samplelist.Java/data/array
```

4. Create server and client applications.

```
make
```

5. Register the Implementation Repository ID and object references required for execution of the sample program.

```
register
```

6. Execute the server application.

```
exec-SV
```

7. Activate a separate terminal and set the CLASSPATH environment variable such that Portable-ORB can run.

```
CLASSPATH=.:$PORB_HOME/lib/ODporb4.jar:$PORB_HOME/lib/CosNaming4.jar:$CLASSPATH
export  CLASSPATH
```

8. Move the current directory to the following directory:

```
cd $SAMPLES/complex/samplelist.Java/data/array
```

9. Execute the client application.

```
exec-CL
```

<The execution results are displayed.>

10. Stop the server application.
11. Delete the sample program Implementation Repository ID and object references that are no longer required.

```
unregister
```

Linux32/64

The following general data type and POA interface sample shows an example execution of complex/samplelist.Java/data/array.

In this example, the CORBA Service is installed on the server system and the Portable-ORB is installed on the Windows(R) client. The server application is a Java server application, and the client application uses Portable-ORB.

Install the Java Execution Environment on the Windows(R) client.

<Linux server>

1. Set the CLASSPATH environment variable.

```
CLASSPATH=.:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH
export  CLASSPATH
```

2. Set the LD_LIBRARY_PATH environment variable.

```
LD_LIBRARY_PATH=$OD_HOME/lib:$LD_LIBRARY_PATH
Export LD_LIBRARY_PATH
```

3. Move the current directory to the following directory:

```
cd $SAMPLES/complex/samplelist.Java/data/array
```

4. Create server and client applications.

```
make
```

5. Register the Implementation Repository ID and object references required for execution of the sample program.

```
register
```

6. Execute the server application.

```
exec-SV
```

<Windows(R) client>

1. Transfer in binary format all the class files (files with the extension "class") under the $SAMPLES/complex/samplelist.Java/data/array directory (including subdirectories) of the Linux server to an arbitrary folder of the Windows(R) client.

   In this practice, it is necessary to specify the same configuration and folder names (case sensitive) under subfolders of the Windows(R) client as those of the Solaris/Linux server. The following shows an example using ftp.

   Create in advance ODsample folder under the destination folder of the Windows(R) client and create in advance arraytestPackage folder under ODsample folder.

```
ftp <Linux server name>
 :
cd $SAMPLES/complex/samplelist.Java/data/array
bin
mget *.class
lcd ODsample
cd ODsample
```

```
mget *.class
lcd arraytestPackage
cd arraytestPackage
mget *.class
```

2. To operate the ORB class of Portable-ORB, specify the ORB class. For the specification method, refer to "ORB (Object Request Broker) Setup" in the "Java Programming Guide" chapter.

3. To operate the Portable-ORB, set the environment variable CLASSPATH.

```
set CLASSPATH=.;%PORB_HOME%\lib\ODporb4.jar;%PORB_HOME%\lib\CosNaming4.jar;%CLASSPATH%
```

4. Set the Operating environment file for Portable-ORB.

   Use the *porbediteny* command to set the host information of the operating environment file stored in %PORB_HOME%.

5. Execute the client application in an arbitrary folder where the class file has been transferred.

```
java simple_c
```

   <The execution result is displayed.>

<Linux server>

1. Stop the server application.

2. Delete the sample program Implementation Repository ID and object references that are no longer required.

```
unregister
```

2) Four Arithmetic Operations

The following procedure is an example execution of CalcSample/java.

In this example, the CORBA Service and Portable-ORB are installed on the same system. The server application is a Java server application, Java applets are stored under Web Server on the same system, and the Java applets and Portable-ORB are downloaded to the PC and used at the PC.

On the Windows(R) client, install the JBK plug-in before executing the sample program.

<Solaris/Linux server>

1. Move the current directory to the following directory: (*1)

```
cd $SAMPLES/CalcSample/c
```

2. Create the server application. (*2)

```
make
```

3. Register the implementation repository ID and object reference required to execute the sample program.

```
register
```

4. Execute the server application.

```
simple_s
```

5. Move the current directory to the following directory:

```
cd $PORB_SAMPLES/CalcSample/java
```

6. Create the client application.

```
CLASSPATH=.:$PORB_HOME/lib/ODporb4.jar:$PORB_HOME/lib/CosNaming4.jar:$CLASSPATH
export CLASSPATH
make
```

7. Create the jar archive file (client2.jar) for the created client application.

```
jar cvf client2.jar *.class ODdemo/*.class ODdemo/calculatorPackage/*.class
```

8. Apply digital signature to the jar archive file for the created client application and the library of Portable-ORB to use. Then make environment setting to enable digital signature on the Windows(R) client where the client application is to be executed.

<Web Server>

1. Copy the created client application to any directory under the Web Server document root. (APPLETDIR: Applet storage directory)

   client2.jar is a jar archive file to which digital signature is applied under <Solaris/Linux server> step 8.

```
mkdir $APPLETDIR
cp $PORB_SAMPLES/CalcSample/java/Calcsample4.html $APPLETDIR
cp $PORB_SAMPLES/CalcSample/java/Calcsample4.js $APPLETDIR
cp $PORB_SAMPLES/CalcSample/java/client2.jar $APPLETDIR
```

2. Use the link command to create a link to enable download of Portable-ORB.

   ODporb4_plugin.jar, CosNaming4_plugin.jar and InterfaceRep4_plugin.jar is a jar archive file to which digital signature is applied under <Solaris/Linux server> step 8.

```
ln -s $PORB_HOME/lib/ODporb4_plugin.jar ODporb4_plugin.jar
ln -s $PORB_HOME/lib/CosNaming4_plugin.jar CosNaming4_plugin.jar
ln -s $PORB_HOME/lib/InterfaceRep4_plugin.jar InterfaceRep4_plugin.jar
ln -s $PORB_HOME/etc etc
```

3. Enter settings in the Portable-ORB environment file. Use the *porbeditenv* command to set the host information in the operating environment file stored in $PORB_HOME.

<Windows(R) client>

1. Specify the ORB class to operate the ORB class of Portable-ORB.
2. Specify the URL that stores CalcSample4.html with the browser and start the applet.
3. Enter numerics and characters from the input screen and click OK.

<Solaris/Linux server>

1. Stop the server application.
2. Delete the implementation repository ID and object reference of the sample program that is no longer required.

```
unregister
```

Notes

*1 To execute this sample program, use one of the following server applications:

CalcSample/c

CalcSample/c++

*2 A warning message as shown below may be output when sample programs in C++ are compiled. It does not indicate an operational problem.

Solaris32/64

```
"simple_stub_c++.C", line 172: warning:_result used before set
```

Linux32/64

```
xxx.C: In function 'void _dt_intf1 (void *)':
xxx.C:xx: warning: ...
```

Appendix G Dynamic Skeleton Interface: DSI

G.1 C Programming

The DSI server application is made up of initialization and interface implementation components. The following figure outlines the initialization process.

Figure G.1 Server Application (DSI) Initialization Component

G.1.1 Initialization

Invoke the CORBA initialization method CORBA_ORB_init() to execute initialization. As a result, the ORB object reference is notified. This object reference is specified to use the ORB interface whenever it is invoked.

The basic object adapter is also initialized.

G.1.2 Gateway Registration

You may use the following methods to notify ORB of the server application gateway:

(1) Retrieving the Implementation Repository Object Reference

Use CORBA_ORB_resolve_initial_references() to retrieve the implementation repository object reference. Specify CORBA_ORB_ObjectId_ImplementationRepository as a parameter.

(2) Retrieving the ImplementationRep Object Reference

Use FJ_ImplementationRep_lookup_id() to obtain the object reference for the ImplementationRep object of the server application.

(3) Registering the Gateway

Use CORBA_BOA_set_impl_dsi() to register the gateway implemented by the server application.

```
CORBA_BOA_set_impl_dsi(
     boa, &env, (CORBA_DynamicImplementationRoutine)dsi );
```

G.1.3 Server Activation

When initialization is complete, the server application notifies ORB. When this instruction is issued, ORB sends a request from the client to the server application.

G.1.4 Gateway Processing

A gateway performs the following:

- Method analysis

- Parameter assembly
- Parameter analysis
- Process start
- Setting return values

(1) Method Analysis

To analyze a method, invoke CORBA_ServerRequest_op_name()

```
static void
dsi(
     CORBA_Object           obj,
     CORBA_ServerRequest    request,
     CORBA_Environment      *env )
{

     CORBA_RepositoryId     dsi_op_name;

     dsi_op_name = CORBA_ServerRequest_op_name(
                           request,
                           env );
     if( strcmp( dsi_op_name, "calculate" ) == 0 )
          method_calculate( request, env );
     CORBA_free( dsi_op_name );
     return;
}
```

(2) Parameter Assembly

This section provides information on parameter assembly.

Generating a List Object

Use CORBA_ORB_create_list() to generate a list object containing the location used to store parameters to be mapped to the server application. You must specify the number of parameters to be stored as a parameter. The object reference for the NVList object is reported.

```
CORBA_NVList arg_list;

CORBA_ORB_create_list( orb, 2, &arg_list, env );
```

Setting a Parameter List

Use CORBA_NVList_add_item() method to assign the parameters to be mapped to the server application in the list object. The CORBA_NVList object reference and server application parameter name, type, value, and length are specified in parameters.

```
CORBA_NVList_add_item(
     arg_list,
     "a", TC_long, NULL,
     sizeof( CORBA_long ),
     CORBA_ARG_IN,
     env );

CORBA_NVList_add_item(
     arg_list,
     "b", TC_long, NULL,
     sizeof( CORBA_long ),
     CORBA_ARG_IN,
     env );
```

(3) Parameter Analysis

To analyze parameters, use CORBA_ServerRequest_params().

```
CORBA_ServerRequest_params(
     dsi_request,
     arg_list,
     env );
```

(4) Setting Return Values

To set return values, invoke CORBA_ServerRequest_result().

```
CORBA_long                  a, b;
ODdemo_calculator_result    *res;
CORBA_any                   any_value;

     a = *(long *)(arg_list[0].argument._value );
     b = *(long *)(arg_list[1].argument._value );

     if( b == 0 ){
          CORBA_ServerRequest_exception(
                dsi_request,
                CORBA_USER_EXCEPTION,
                ex_ODdemo_calculator_ZEROPARAM,
                NULL,
                env );
          return;
     }
     res = ODdemo_calculator_result_alloc();
     res->add_result = a+b;
     res->subtract_result = a/b;
     res->multiple_result = a*b;
     res->divide_result = (CORBA_float)a/b;

     any_value._type = TC_ODdemo_calculator_result;
     any_value._value = res;
     CORBA_ServerRequest_result(
          dsi_request,
          any_value,
          env );
```

G.1.5 Deactivating the Server

If a server application receives a request to stop from the user, it will notify ORB that subsequent requests are not to be accepted from the client. When this notification is received, ORB will not send the processing request from the client to the server application and will return an exception to the client.

G.1.6 Allocating and Releasing Parameter Area Using Dynamic Interface

This section describes how to create parameters using Dynamic Skeleton Interface (DSI). Set parameters by CORBA_NVList_add_item() function.

IN Mode

When a server application receives in parameters, it does not need to allocate or release the parameter area in the server application. Specify CORBA_NVList_add_item() as follows.

```
CORBA_NVList_add_item(
     arg_list,
     name,           /* Specify the parameter name defined in IDL */
     type,           /* Specify the parameter's TypeCode */
```

```
    NULL,           /* Specify NULL */
    0,              /* Specify 0 */
    CORBA_ARG_IN,   /* Specify CORBA_ARG_IN */
    &env );
```

OUT Mode

To pass out parameters from a server application to a client application, allocate data area by data allocation function, like CORBA_long_alloc(), and specify the pointer to it in the 4th parameter of CORBA_NVList_add_item() as follows.

```
CORBA_NVList_add_item(
    arg_list,
    name,                   /* Specify the parameter name defined in IDL */
    type,                   /* Specify the parameter's TypeCode */
    &param,                 /* Specify the pointer to the parameter area */
    sizeof( CORBA_long ),   /* Specify the parameter size */
    CORBA_ARG_OUT,          /* Specify CORBA_ARG_OUT */
    &env );
```

Allocated area is released in CORBA Service after the request returns.

INOUT Mode

To receive *inout* parameters from a client application, it does not needed to allocate/release parameter area in a server application. Specify CORBA_NVList_add_item() as follows.

```
CORBA_NVList_add_item(
    arg_list,
    name,              /* Specify the parameter name defined in IDL */
    type,              /* Specify the parameter's TypeCode */
    NULL,              /* Specify NULL */
    0,                 /* Specify 0 */
    CORBA_ARG_INOUT,   /* Specify CORBA_ARG_INOUT */
    &env );
```

To pass *inout* parameters from the server application to the client application, do as follows. For fixed length data Set a value in the parameter extracted by CORBA_ServerRequest_params().

For Variable Length Data

Release the parameter extracted by CORBA_ServerRequest_params() using CORBA_free(), then allocate the area by allocating functions such as CORBA_string_alloc(), and set the pointer. Set a value in the allocated area. *Inout* parameter area is released in Skeleton after the request returns.

RETURN

To Return a value from a server application to a client application, allocate area by allocating function, like CORBA_long_alloc() and etc.

G.2 C++ Programming

This is not valid for Linux (64 bit).

A server application, using the dynamic skeleton interface (DSI), is composed of an initialization component and implementation interface components. The initialization component is used to execute the processes shown in the following figure.

Figure G.2 Server Application (DSI) Initialization Component

G.2.1 Initialization

To return an ORB object reference, invoke *CORBA::ORB_init()*. This object reference is specified whenever the ORB interface, described later, is invoked.

To initialize the Basic Object Adapter invoke *CORBA::BOA_init()*.

G.2.2 Registering a Gateway to ORB

Use the following methods to register the server application's gateway to ORB:

(1) Retrieving an Implementation Repository Object Reference

To retrieve an Implementation Repository object reference, invoke *CORBA::ORB::resolve_initial_references(),* specifying *CORBA_ORB_ObjectId_ImplementationRepository* as a parameter.

(2) Retrieving the ImplementationRep Object Reference

To retrieve an object reference for the ImplementationRep object of the server application, invoke *FJ::ImplementationRep_lookup_Id()*.

(3) Registering the Gateway

To register the server gateway, invoke *CORBA::BOA::set_impl_dsi()*:

```
boa->set_impl_dsi(
    boa,
    *env,
    (CORBA_DynamicImplementationRoutine_cpp)dsi );
```

G.2.3 Activating the Server

When initialization is complete, you must notify ORB. ORB will then begin to dispatch requests from its clients.

G.2.4 Gateway Processing

A gateway performs the following processes:

- Analyzing methods
- Constructing parameters
- Analyzing parameters
- Invoking call methods
- Setting return values

(1) Analyzing Methods

To analyze a method, invoke *CORBA::ServerRequest::op_name()*.

```
static void
dsi(
    CORBA::Object_ptr          obj,
    CORBA::ServerRequest_ptr   request,
    CORBA::Environment         &ev )
{
    CORBA::RepositoryId        dsi_op_name;

    try {
        dsi_op_name = request->op_name( ev );
        if( strcmp( dsi_op_name, "calculate" ) == 0 ) {
            method_calculate( request );
        }
        CORBA::string_free( dsi_op_name );
    }
    catch ( CORBA::Exception e ){
        CORBA::Any p;
        ev.exception( &e );
        request->exception( &p, ev );
    }
    return;
}
```

(2) Constructing Parameters

To construct parameters, invoke *CORBA::ORB::create_list()*. This will create a list object in which parameters passed to the server application will be stored. When invoking *create_list()*, you must specify the number of parameters which are to be passed to the server. An NVList object will be returned.

```
CORBA::NVList_ptr      arg_list;
orb->create_list( 2, arg_list, *env );
```

To assign values to the NVList object, you must invoke *CORBA::NVList::add_value()*. You must specify *CORBA::NVList* object, the parameter name, type, value, and length as parameters.

```
CORBA::Any p1, p2;
CORBA::Long l;

l = 0;
p1 <<= l;
arg_list->add_value(
            "a",
            p1,
            CORBA::ARG_IN,
            *env );
p2 <<= l;
arg_list->add_value(
            "b",
            p2,
            CORBA::ARG_IN,
            *env );
```

(3) Analyzing Parameters

To analyze parameters, invoke *CORBA::ServerRequest::params()*.

```
request->params(
         arg_list,
         *env );
```

(4) Setting Return Value

To set return values, invoke *CORBA::ServerRequest::result()*.

```
CORBA::NamedValue_ptr     nvp1, nvp2;

nvp1 = arg_list->item(0,*env);
nvp2 = arg_list->item(1,*env);

CORBA::Any *r1, *r2;
r1 = nvp1->value(*env);
r2 = nvp2->value(*env);

CORBA::Long a,b;
(*r1) >>= a;
(*r2) >>= b;
cout << " a = [" << a << "] b = [" << b << "]" << endl;

ODdemo::calculator::result *res =
    new ODdemo::calculator::result();

res->add_result = a+b;
res->subtract_result = a-b;
res->multiple_result = a*b;
res->divide_result = (CORBA::Float)a/b;

CORBA::Any *tmp_any = new CORBA::Any;
tmp_any->replace( _tc_ODdemo_calculator_result,
                  (void *)res, CORBA_TRUE );

request->result( tmp_any, *env );
```

G.2.5 Deactivating the Server

When a server application receives a request to stop, it will notify ORB that it has stopped responding to requests. Once ORB gets notified, it does not dispatch requests to the server but returns exceptions to clients.

G.2.6 Allocating and Releasing Parameter Area Using Dynamic Interface

This section describes how to create parameters using Dynamic Skeleton Interface (DSI). This section also explains how to fetch values from the created parameters.

IN Mode

When a server application receive in parameters, it does not need to allocate or release parameter area in the server application. Specify *add_value()* function as follows.

```
CORBA::Any               p1( ((*params)[0])->type, &i, CORBA_TRUE );

arg_list->add_value(
     name,               /* Specify the parameter name defined in IDL */
     p1,                 /* Specify Any type which stores in value */
     CORBA::ARG_IN,      /* Specify CORBA::ARG_IN */
     *env );
```

OUT Mode

To pass out parameters from a server application to a client application, allocate data area by data allocation function, like *CORBA::long_alloc()* function, and specify the pointer to it in the 2nd parameter of *add_value()* function as follows.

```
CORBA::Any     *p2 = new CORBA::Any( ((*params)[1])->type, &i, CORBA_TRUE );
```

```
arg_list->add_value(
     name,               /* Specify the parameter name defined in IDL */
     *p2,                /* Specify Any type which stores out value */
     CORBA::ARG_OUT,     /* Specify CORBA::ARG_OUT */
     *env );
```

Allocated area is released in CORBA Service after the request returns.

INOUT Mode

To receive *inout* parameters from a client application, it does not needed to allocate/release parameter area in a server application. Specify the *add_value()* function as follows.

```
CORBA::Any     *p3 = new CORBA::Any( ((*params)[2])->type, &i, CORBA_TRUE );

arg_list->add_value(
     name,               /* Specify the parameter name defined in IDL */
     *p3,                /* Specify Any type which stores inout value */
     CORBA::ARG_INOUT,   /* Specify CORBA::ARG_INOUT */
     *env );
```

To pass *inout* parameters from the server application to the client application, do as follows.

For fixed length data:

Set a value in the parameter extracted by *CORBA::ServerRequest::params()*.

For variable length data:

Use the *CORBA_free()* function to release the parameter extracted by *CORBA::ServerRequest::params()*, then allocate the area using allocation functions such as *CORBA::string_alloc()*, and set the pointer. Set a value in the allocated area.

Inout parameter area is released in skeleton after the request returns.

RETURN

To return a value from a server application to a client application, allocate the area using allocation functions such as *CORBA::long_alloc()*.

G.3 Java Programming

Dynamic Skeleton Interface (DSI) server applications are constructed of an initialization processing section and an interface implementation section. The process displayed in the following figure is run in the initialization processing section. In the following explanation, Servant objects are treated as the Default Servant.

Figure G.3 Server Application (DSI) Process Flow

G.3.1 Initialization

There are two initialization steps:

- Get RootPOA object references.
- Create child POA from RootPOA with policy using Default Servant.

G.3.2 Gateway Registration

Creates a Servant object from an implemented gateway, and registers a child POA as a Default Servant. Thus, the *invoke()* method described above is launched in Servant in response to requests from clients.

G.3.3 Server Activation

Runs the *activate()* method in response to POAManager instances related to the POA that set the Servant objects.

G.3.4 Gateway Processing

The following processes are run by a gateway:

- Method analysis
- Parameter construction
- Parameter analysis
- Setting of return information

An example of a gateway implementation is displayed below.

Gateway Implementation Example

```
class DsiServant
  extends org.omg.PortableServer.DynamicImplementation {

      public void
      invoke(org.omg.CORBA.ServerRequest req)
      {
          org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();

          String name = req.operation();   // method name acquisition
          if (name.equals("add"))          // method name analysis
          {

              // parameter list setting
              // parameter list creation
              org.omg.CORBA.NVList args = orb.create_list(2);
              //parameter information registration
              org.omg.CORBA.Any _a = orb.create_any();
              _a.type(orb.get_primitive_tc(
                  org.omg.CORBA.TCKind.tk_long));
              args.add_value("a", _a, org.omg.CORBA.ARG_IN.value);
              org.omg.CORBA.Any _b = orb.create_any();
              _b.type(orb.get_primitive_tc(
                  org.omg.CORBA.TCKind.tk_long));
              args.add_value("b", _b, org.omg.CORBA.ARG_IN.value);

              //parameter analysis
              req.arguments(args);
              int a = _a.extract_long();
              int b = _b.extract_long();

              //processor launch
              org.omg.CORBA.IntHolder ret = new org.omg.CORBA.IntHolder();
              UserMethod um = new UserMethod();
              ret.value = um.method1(a, b);
```

```
            // return information setting
            org.omg.CORBA.Any _ret = orb.create_any();
            _ret.type(orb.get_primitive_tc(
                org.omg.CORBA.TCKind.tk_long));
            _ret.insert_long(ret.value);
            req.set_result(_ret);
            return;
        }
        throw (new org.omg.CORBA.BAD_OPERATION(
            0, org.omg.CORBA.CompletionStatus.COMPLETED_MAYBE));
    }

    // implementation required method
    public java.lang.String[] _all_interfaces(
        org.omg.PortableServer.POA poa,
        byte[] objectId) {
        return ids;
    }
    private static java.lang.String[] ids = {"IDL:ODsample/intf:1.0"};

}

//processing
class UserMethod
{
    public int method1( int a, int b ){
        //describe necessary processing
        return( a + b );
    }
}
```

Servant objects that implement gateways are created as objects that acknowledge *org.omg.PortableServer.DynamicImplementation.* Also, they run *invoke()* as a method that is launched in response to requests from clients. A ServerRequest class object (req in the above example) is returned as the attribute at the time of launch.

Gateway Servant Required Method (1)

```
public void invoke(org.omg.CORBA.ServerRequest req)
```

(1) Method Acquisition and Analysis

An operational name registered in a returned ServerRequest class is acquired by the *org.omg.CORBA.ServerRequest.operation() method.*

(2) Parameter Assembly

Parameter assembly tasks are described below.

Parameter List Creation

You will create a list object to preserve the domain for storing parameters deliverable to the application by the *org.omg.CORBA.ORB.create_list()* method. Now, you will specify how many parameters to store as an attribute. As a result, the *org.omg.CORBA.NVList* object is returned (No. 2 in the above example).

Parameter Information Registration

Using the *org.omg.CORBA.add_value()* method, you will store parameter information in an *NVList* object. Specify as attributes the parameter name, an any type object where you will store the parameter values, and parameter type (input or output)

(3) Parameter Analysis

With the created *NVList* object as an attribute, invoke the *org.omg.ServerRequest.arguments()* method. As the result, a parameter value deliverable to the application is acquired.

(4) Return Information Settings

Runs the or*g.omg.CORBA.ServerRequest.set_result()* method and does return information settings.

(5) Other Methods Required for Implementation

A Servant that implements a gateway will need to implement the OMG *protocol_all_interfaces()* method. Like the above examples, it will implement with a format where a *org.omg.PortableServer.POA* class object and a byte-array object are taken as attributes and where a String-array object holding the character string ("IDL:xxx:1.0") corresponding to the Interface Repository ID in its initial element is taken as a return value. It is not particularly necessary to handle the attribute values within a method.

Gateway Servant Example

```
public java.lang.String[] _all_interfaces(
            org.omg.PortableServer.POA poa,
            byte[] objectId)
```

G.4 COBOL Programming

This is not valid for Linux (64 bit).

A server application, using the dynamic skeleton interface (DSI), is composed of an initialization component and implementation interface components. The initialization component is used to execute the following (refer to the following figure):

- To invoke the CORBA initialization method.
- To register a gateway to ORB.
- Server activation.

Figure G.4 Server Application (DSI) Process Flow

G.4.1 Initialization

To initialize ObjectDirector invoke *CORBA-ORB-INIT*. This function notifies an ORB object reference(). This object reference is specified whenever the ORB interface is invoked.

To initialize the Basic Object Adapter invoke *CORBA::BOA_INIT*.

G.4.2 Activating a Server

When initialization is complete, you must notify ORB. ORB will then begin to dispatch requests from its clients.

G.4.3 Gateway Processing

A gateway performs the following processes:

(1) Analyzing Methods

To analyze a method, you must invoke *CORBA-SERVERREQUEST-OP-NAME*.

```
LINKAGE SECTION.
01 COPY SERVERREQUEST IN CORBA REPLACING
        CORBA-SERVERREQUEST BY REQUEST.
```

DATA DIVISION.

WORKING-STORAGE SECTION.

```
         01 COPY REPOSITORYID IN CORBA
            REPLACING CORBA-REPOSITORYID BY DSI-OP-NAME.

INVOKE "CORBA-SERVERREQUEST-OP-NAME" USING
        REQUEST
        ENV
        DSI-OP-NAME.

EVALUATE DSI-OP-NAME
  WHEN "add"
    CALL "METHOD-ADD" USING
        REQUEST
        ENV
  WHEN "add2"
    CALL "METHOD-ADD2" USING
        REQUEST
        ENV
END-EVALUATE.
```

(2) Assembling Parameters

(1) To create a list object to reserve locations in which to store parameters sent to the server application, you must invoke the *CORBA-ORB-CREATE-LIST method*. You must specify the number of parameters to be stored as a parameter. An NVList object reference is then posted.

(2) To assign values to the NVList object, you must invoke *CORBA-NVLIST-ADD-ITEM*. You must specify a *CORBA-NVLIST* object reference and the name, type, value, and length of the server application as parameters.

(3) Analyzing Parameters

To analyze parameters invoke *CORBA-SERVERREQUEST-PARAMS*.

```
CALL "CORBA-SERVERREQUEST-PARAMS" USING
     DSI-REQUEST
     ARG-LIST
     ENV.
```

(4) Setting of Return Information

To set return information invoke *CORBA-SERVERREQUEST-RESULT*.

```
MOVE FUNCTION LENG (TC-LONG) TO STRING-LENGTH.
CALL "CORBA-STRING-SET" USING
      TMP-BUF
      STRING-LENGTH
      TC-LONG.
CALL "CORBA-ORB-TYPECODE-FROM-CGEN-TC" USING
      TMP-BUF
      TYPE OF ANY-VALUE.
MOVE FUNCTION ADDR (RET) TO VALUE OF ANY-VALUE.
CALL "CORBA-SERVERREQUEST-RESULT" USING
      DSI-REQUEST
      VALUE
      ENV.
```

G.4.4 Deactivating a Server

When the server application receives a stop request from a user, it notifies ORB that subsequent requests from clients will not be accepted.

Appendix H COM/CORBA Linkage Programming

This appendix describes how client applications written in Visual Basic use the OLE2 interface to link to server applications that conform to CORBA.

Note that if Visual Basic is used as the user interface, then the recommended calling method is to use DLLs created in C (C++).

Windows64

COM/CORBA linkage programming cannot be used.

H.1 OLE-CORBA Gateway

Client applications are referred to as OLE clients from here on. To implement the function, use an OLE- CORBA gateway function as the OLE server for the client. In response to an information request from an OLE client, the *OLE-CORBA* gateway function activates a CORBA server application and then posts the result back to the OLE client. The *OLE-CORBA* gateway function supports the following three functions:

- GetObject Function
- CreateType Function
- CreateTypeByld Function

GetObject Function

The *GetObjec*t function retrieves CORBA server application object references and posts the results to clients.

CreateType Function

Some data types are defined differently in CORBA and Visual Basic. Data types common to both Visual Basic and CORBA can be sent without modification. Otherwise, the *CreateType* or *CreateTypeById* functions from the *OLE-CORBA* gateway function can be used to retrieve data type object references and to access members within objects. The *CreateType* function retrieves data-type object references using the names of server application parameters.

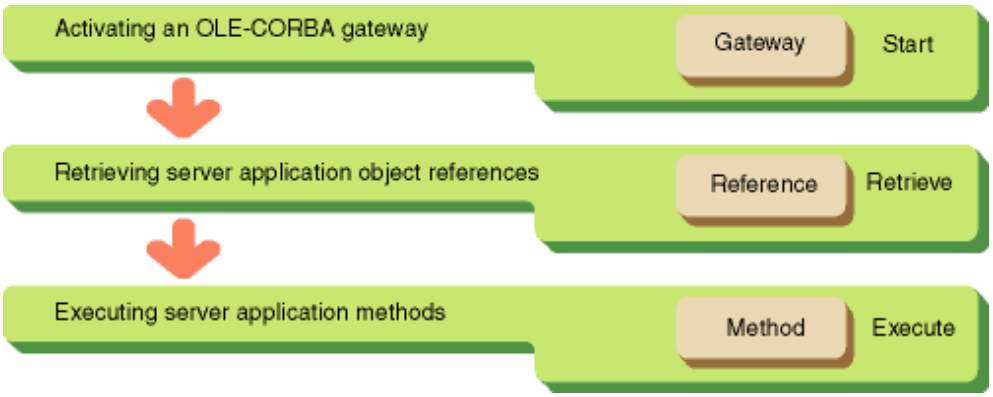
CreateTypeById Function

The *CreateTypeById* function retrieves data-type object references using the Repository IDs of server application parameters.

H.2 OLE Client Processing

The following figure shows OLE client processing.

Figure H.1 OLE Client Processing



H.2.1 Activating OLE-CORBA Gateway

The OLE-CORBA gateway is registered in the Registry of the client PC using the name *CORBA.Factory*. To activate the OLE-CORBA gateway, use the *CreateObject* function offered as standard in Visual Basic to specify *CORBA.Factory* as a parameter.

H.2.2 Retrieving Server Application Object References

To retrieve server application object references, use the *GetObject* function to specify the name of the server application as a parameter. After the server application specified in the parameter is retrieved from the naming service, the *GetObject* function posts the object reference to the client. To specify the application name of a server, specify the object name that is registered in the naming service. The server object IDL definition must be registered in the interface repository.

H.2.3 Executing Methods

This section provides information on executing methods.

Late Binding

The server application method may be invoked by activating the object method retrieved using the *GetObject* function.

An example of using Visual Basic to program the client follows.

Example

```
Dim GW As Object                                    // OLE-CORBA gateway instance
Dim calculator As Object                            // Server object reference
Dim a As Long                                       // Input variable
Dim b As Long                                       // Input variable
Dim exp As Object                                   // Error setting
Dim res As Variant                                  // Return value variable

Set GW = CreateObject("CORBA.Factory")              // Activate OLE-CORBA gateway
Set calculator = GW.GetObject("ODdemo::calculator")   // Create CORBA object
a = Text1.Text                                      // Assign data input on the screen
b = Text2.Text                                      // Assign data input on the screen
Set result = calculator.calculate(a, b, exp)        // Call method
```

Early Binding

Early binding for Visual Basic is enabled by declaring the CORBA.Factory object storage variable as a class type. Because early binding does not read a CORBA class during Visual Basic program execution, it provides better performance than late binding. To use early binding when developing a Visual Basic application, select Set Reference from the Project menu in Visual Basic, and select ODOLE 1.0 Type Library. A coding example of early binding is shown below.

```
Dim Factory as CORBAFactory

Set Factory = CreateObject("CORBA.Factory")
```

H.3 Sending Parameters to Server Applications

Some data types are defined differently in CORBA and Visual Basic. Data types common to both Visual Basic and CORBA can be passed without modification. Otherwise, the *CreateType* or *CreateTypeById* functions from the OLE server can be used to retrieve data type object references and access members within objects.

H.4 Specifying Basic Data Types

The following table lists the data types offered by CORBA and Visual Basic, and how to specify methods using Visual Basic.

Table H.1 Specifying Data Type Methods in Visual Basic

CORBA Data Type			Visual Basic Specification Method
Basic data type	Integer type	long	Long
		short	Integer
		unsigned long	Long
		unsigned short	Long
	Floating-point type	float,	Single
		double	Double
	Character type	char	Integer
	Octet type	octet	Integer
	Boolean type	boolean	Integer
	String type	string	refer to "H.4.1 String Type"
	Enumerator type	enum	refer to "H.4.2 Enumerator Type"
	Any type	any	refer to "H.4.3 Any Type"
Sequence type		sequence	refer to "H.5.1 Sequence Type"
Structure		struct	refer to "H.5.2 Structure"
Union		union	refer to "H.5.3 Union"
Array		array	refer to "H.5.4 Array"

Data Type Notes

- To invoke a server application of unsigned short type, specify within the unsigned short range. Negative values or values exceeding 65536 may not be specified.
- A negative value may not be specified when invoking a server application of unsigned long type.
- To invoke a server application of a char or octet type, specify within the char range. Negative values or values exceeding 256 may not be specified.
- To invoke a server application of Boolean type, only the values 0 or 1 may be specified.
- To invoke a server application any type, declare variables as Object type. Use the CreateType or CreateTypeById functions to create a data-type instance. Handle the value and type as properties of the instance.
- To use the objref type, typecode type, or array type as the parameter and include string, ojref, struct, union, sequence, or array in the member of any type, sequence type, structure type, or union, ensure that sufficient memory space is available.

Notes on Writing Visual Basic Applications

This section contains notes on writing Visual Basic applications.

Releasing Memory

Use the following method to release objects and out and inout parameters in Visual Basic.

a) Release CORBA objects (including prospective CreateType data) as follows:

```
    Set Ap = Nothing
```

Ap:

Object to be released

b) Declare an array as variable (dynamic), not as fixed, and release it using the Erase statement.

c) Release character strings by setting NULL characters.

```
    str = " "
```

Sending and Receiving Array (including Sequence Type) Parameters

Use the following procedure to send and receive arrays (including sequence type arrays) as parameters. The sequence of elements in an array declaration is different for Visual Basic and IDL.

Visual Basic

```
Dim a(2,3) As Long
```

IDL Definition

```
long a(4,3)
```

When the number of elements is changed in inout, use dynamic declaration and send the data by reassigning it in the client as:

```
ReDim abc(n1,n2,n3) As Long.
```

When the result is received from the server, type conversion is performed while retaining the contents by issuing:

```
ReDim Preserve abc(m1,m2m3) As Long and accessing the data.
```

When the value has not been set in the area specified in the in parameter, 0 is passed.

H.4.1 String Type

For fixed-length strings, use the string for which the length is specified. For variable-length strings use the string for which the length is not specified. To use a fixed-length character string, the value of "number of used characters + 1" must be specified in the IDL definition.

IDL definition and Visual Basic examples follow.

IDL Definition

```
   typedef string<11>   sten;     // Fixed-length (10 characters) string
   typedef string       sinf;     // Variable-length string
```

Visual Basic

```
   Dim sten As String * 11       // Fixed-length (10 characters) string
   Dim sinf As String            // Variable-length string
```

H.4.2 Enumerator Type

For enumerator type, declare each element as a constant. An example of using enumerator type in Visual Basic is given below.

IDL Definition

```
typedef enum EM { red,green,blue } Foo;
```

Visual Basic

```
Global Const red  = 0       // Declare each element as a constant.
Global Const green = 1      // Declare each element as a constant.
Global Const blue = 2       // Declare each element as a constant.
```

H.4.3 Any Type

To call a server application any type, you must declare variables as Object type. Use the *CreateType* function or the *CreateTypeById* function to retrieve data type object references. Value and type must be dealt with as properties of the object. To set a value in the TypeCode property of any-type data, prepare the operation for returning TypeCode to the server and set the returned value. Set the value in the value property according to the data type.

Setting Data in Any-Type Data

An example of using any type data in Visual Basic is given below.

IDL Definition

```
module mod {
  typedef  any  typeany;     // any type
  interface intf {
    TypeCode  GetTypeCode(); // Function for returning TypeCode set in TypeCode property
    any op( in any a );      // op function having a parameter or any type
                             // Return value of any type
   };
};
```

Visual Basic

```
Dim factory As Object                           // OLD-CORBA gateway instance
Dim obj As Object                               // Object reference of server
Set factory = CreateObject("CORBA.Factory")     // Activate OLE-CORBA gateway
Set obj = factory.GetObject("example")          // Retrieve object

Dim anydata As Object                  // Data-type reference (for passing parameters)
Dim anydata1 As Object                 // Data-type reference (for return values)
Dim obj3 As Object                     // Data-type reference (for TypeCode)

Set anydata =factory.CreateType(obj, "typeany") // Retrieve data-type references
Set obj3= obj.GetTypeCode()                     // Obtain TypeCode objects
anydata.TypeCode = obj3                         // Set up TypeCode
anydata.Value = &H80000000                      // Set up data

anydata1 =obj.op(anydata)                       // Call op function
```

Retrieving Data from Any-Type Data

To fetch values from any-type data, invoke the function related to the TypeCode member of the any-type data and retrieve the data type. Retrieve the value of the retrieved data type from the value member. An example of using any-type data in Visual Basic is given below.

Visual Basic Example

```
anydata = obj.op(anydata)         // Call op function

kind = anydata.TypeCode.kind      // Fetch TypeCode
Select Case kind
Case 2      'short                // Processing where data type is short
s = anydata.Value
Case 15      'struct              // Processing where data type is struct
l = anydata.Value.d1
str = anydata.Value.d2
Case 16   'union                  // Processing where data type is union
dic = anydata.Value.UNION_d
  Select Case dic
  Case 1
l = anydata.Value.d1
  Case 2
str = anydata.Value.d2
  End Select
End Select
```

H.5 Specifying Other Data Types

For other data types, either the *CreateType* or *CreateTypeById* function must be used to create an instance of a data type such as structure. Then access the member in the instance. For the CreateType function, you must specify the name of the data type to be accessed. For the *CreateTypeById* function, specify the repository ID of the data type.

H.5.1 Sequence Type

A sequence is handled as a one-dimensional array. The definition depends on the data type.

Sequence for Primitive Data Types (Other than Any Type)

If the sequence is fixed length, as many one-dimensional arrays as the number specified using IDL are obtained. An example of using Visual Basic where the sequence has been declared as the long type is given below.

IDL Definition

```
typedef sequence<long,10> vec10;   // 10 long types
```

Visual Basic

```
Dim vec10(10) As long              // Declare 10 long types
vec10(0) = 100                     // Assign data to each member
vec10(1) = 101                     // Assign data to each member
...
vec10(9) = 109                     // Assign data to each member
```

For Data Types where the Sequence is Structure, Union, Any, Object, or TypeCode

If the sequence is configured from structure, union, any type, object, or *TypeCode*, use the *CreateType* or *CreateTypeById* function to retrieve object references for the sequence members and store the results in Object-type arrays. An example of using Visual Basic where the sequence has been declared as the struct type is given below:

IDL Definition

```
typedef struct STR {                          // struct of a and b
      long a,b;
   } Foo;
typedef sequence<Foo,10> FooSeq;              // Sequence of 10 structs of a and b
```

Visual Basic

```
Dim FooSeq(10) As Object                      // Declare 10 Object types

FooSeq(0) = factory.CreateType( obj, "Foo" )   // Retrieve 10 data-type references
FooSeq(1) = factory.CreateType( obj, "Foo" )
...
```

H.5.2 Structure

To set values in a structure, use either the *CreateType* or *CreateTypeById* function to retrieve object references and store the results in an Object-type variable. Setting the values in the properties of the object can set values in the members of the structure. An example of using Visual Basic for structure type is given below.

IDL Definition

```
interface example5 {
   typedef struct STR {                    // struct of long-type a and b
      long a,b;
   }Foo;
   ...
};
```

Visual Basic

```
Dim str1 As Object                          // References of data-type struct

Set str1 = factory.CreateType(obj, "Foo")   // Retrieve data-type references
str1.a = 100                                // Assign data to a
str1.b = -50                                // Assign data to b
```

H.5.3 Union

To set values in a union, use either the *CreateType* or *CreateTypeById* function to retrieve object references and store the results in Object-type variables. Setting the values in object properties can set values in the members of the union. The *UNION_d* property provides identifying information for stored data. An example of union type in Visual Basic is given below.

IDL Definition

```
interface example6 {
   typedef union UNI switch(long) {      // Union of long, float, and char
      case 1 : long x;
      case 2 : float y;
      default: char z;
   } Foo;
   short set(in Foo udata);              // set function having union-type parameter,
                                         // Return value short type
   ...
} ;
```

Visual Basic

```
Dim uni1 As Object                   // References of data type
Dim discriminator As Variant         // Discrimination information storage variable
Dim ret As Integer                   // Return value storage variable
Dim ans As Variant                   // Union data storage variable

Set uni1 = factory.CreateType(obj, "Foo")  // Retrieve data-type references
uni1.x = 100                               // Assign data to union
ret = obj.set(uni1)                        // Call set function

discriminator = uni1.UNION_d         // Store discrimination information
Select Case discriminator            // Judge data based on discrimination information
Case 1                               // Processing of long type
   ans = uni1.x
Case 2
   ans = uni1.y
Case Else                            // Processing of char type
   ans = uni1.z
```

H.5.4 Array

Visual Basic arrays are used. If the array is structure, union, any type, object, or *TypeCode,* use either the *CreateType* or *CreateTypeById* function to create an instance and store the results in an Object-type variable. An example of using Visual Basic for an array is given below:

IDL Definition

```
typedef long a[4][5];
interface example8 {
   typedef struct STR {
      long a,b;
   } Foo;
    typedef Foo FooArr[3][4];
    ...
} ;
```

Visual Basic

```
Dim a(4,3) As long
Dim FooArr(3,2) As Object
Dim factory As Object
Dim obj As Object
a(0,0) = 100
a(1,0) = 101
...
a(4,3) = 119
Set factory = CreateObject("CORBA.Factory")
Set obj = factory.GetObject("example8")
FooArr(0,0) = factory.CreateType(obj, "Foo")
FooArr(1,0) = factory.CreateType(obj, "Foo")
...
```

H.6 Server Processing Results

The server processing results are posted as return values. If the type is structure, union, any, object, or *TypeCode,* use an Object-type variable to receive the return value. For other types, the return value is a value of the relevant type.

IDL Definition

```
interface example13 {
   typedef   struct STR {
      long   a,b;
   } Foo;
   long      ope1();
   string    ope2();
   Foo       ope3();
} ;
```

Visual Basic

```
Dim factory As Object
Dim ex13 As Object
Dim ret1 As long
Dim ret2 As string
Dim ret3 As Object
Dim sa As long

Set factory = CreateObject("CORBA.Factory")
Set ex13 = factory.GetObject("example13")
ret1 = ex13.op1
ret2 = ex13.op2
Set ret3 = ex13.op3
sa = ret3.a
```

H.7 Assigning and Referencing Attributes

To assign or reference an attribute value, assign or reference the value corresponding to the instance property created using the *GetObject* function. For the read only attribute, values can only be referenced.

IDL Definition

```
interface example1 {
   attribute long              att1;
   readonly attribute string   att2;
} ;
```

Visual Basic

```
Dim factory As Object
Dim obj As Object
Dim att1 As long
Dim att2 As string
Set factory = CreateObject( "CORBA.Factory" )
Set obj = factory.GetObject( "example1" )
obj.att1 = 100
att1 = obj.att1
att2 = obj.att2
```

H.8 Exceptions

To handle exceptions, declare a variable as an object type. Then, add the variable to the end of the parameters when the operation is being executed. When an exception occurs, the exception object reference holding the exception information is stored in the variable. The following values are set in the EX_major member of the object:

2:	System Exception
1:	User Exception
0	Normal Termination

When a system exception occurs, identifying information on the system exception is set in the object repository ID member as a character string. To identify the system exception, a list of system exceptions and character strings are compared. The following table lists the exception codes and their meanings.

Refer to the Reference Manual (API Edition) for the meaning of each exception.

Table H.2 Exception Codes

Exception Information	Exception Code
BAD_CONTEXT	IDL:CORBA/StExcep/BAD_CONTEXT:1.0
BAD_INV_ORDER	IDL:CORBA/StExcep/BAD_INV_ORDER:1.0
BAD_OPERATION	IDL:CORBA/StExcep/BAD_OPERATION:1.0
BAD_PARAM	IDL:CORBA/StExcep/BAD_PARAM:1.0
BAD_QOS	IDL:CORBA/StExcep/BAD_QOS:1.0
BAD_TYPECODE	IDL:CORBA/StExcep/BAD_TYPECODE:1.0
CODESET_INCOMPATIBLE	IDL:CORBA/StExcep/CODESET_INCOMPATIBLE:1.0
COMM_FAILURE	IDL:CORBA/StExcep/COMM_FAILURE:1.0
CONTEXT	IDL:CORBA/StExcep/CONTEXT:1.0
DATA_CONVERSION	IDL:CORBA/StExcep/DATA_CONVERSION:1.0
FREE_MEM	IDL:CORBA/StExcep/FREE_MEM:1.0
IMP_LIMIT	IDL:CORBA/StExcep/IMP_LIMIT:1.0
INITIALIZE	IDL:CORBA/StExcep/INITIALIZE:1.0
INTERNAL	IDL:CORBA/StExcep/INTERNAL:1.0
INTF_REPOS	IDL:CORBA/StExcep/INTF_REPOS:1.0
INV_FLAG	IDL:CORBA/StExcep/INV_FLAG:1.0
INV_IDENT	IDL:CORBA/StExcep/INV_IDENT:1.0
INV_OBJREF	IDL:CORBA/StExcep/INV_OBJREF:1.0
INV_POLICY	IDL:CORBA/StExcep/INV_POLICY:1.0

Exception Information	Exception Code
MARSHAL	IDL:CORBA/StExcep/MARSHAL:1.0
NO_IMPLEMENT	IDL:CORBA/StExcep/NO_IMPLEMENT:1.0
NO_MEMORY	IDL:CORBA/StExcep/NO_MEMORY:1.0
NO_PERMISSION	IDL:CORBA/StExcep/NO_PERMISSION:1.0
NO_RESOURCES	IDL:CORBA/StExcep/NO_RESOURCES:1.0
NO_RESPONSE	IDL:CORBA/StExcep/NO_RESPONSE:1.0
OBJ_ADAPTER	IDL:CORBA/StExcep/OBJ_ADAPTER:1.0
PERSIST_STORE	IDL:CORBA/StExcep/PERSIST_STORE:1.0
REBIND	IDL:CORBA/StExcep/REBIND:1.0
TIMEOUT	IDL:CORBA/StExcep/TIMEOUT:1.0
TRANSIENT	IDL:CORBA/StExcep/TRANSIENT:1.0
UNKNOWN	IDL:CORBA/StExcep/UNKNOWN:1.0
INVALID_TRANSACTION	IDL:CORBA/StExcep/INVALID_TRANSACTION:1.0
TRANSACTION_MODE	IDL:CORBA/StExcep/TRANSACTION_MODE:1.0
TRANSACTION_REQUIRED	IDL:CORBA/StExcep/TRANSACTION_REQUIRED:1.0
TRANSACTION_ROLLEDBACK	IDL:CORBA/StExcep/TRANSACTION_ROLLEDBACK:1.0
TRANSACTION_UNAVAILABLE	IDL:CORBA/StExcep/TRANSACTION_UNAVAILABLE:1.0

Minor codes are set in the EX_minor member when a system exception occurs. Refer to information on CORBA Service Minor Codes in the Reference Manual (API Edition) for details of minor code values.

When a user exception occurs, the repository ID of the exception information defined using IDL is set in the repository ID of the exception object. The repository ID format is shown below:

```
IDL:Exception-name:1.0
```

In addition, more detailed exception information can be accessed as an object member variable using the IDL defined name. An example of an exception using Visual Basic is given below:

IDL Definition

```
exception foo {         // Define user exception
   long dummy;
} ;
interface example9 {
   //ope function having long-type parameter, no return value
   void ope(in long arg) raises(foo);
} ;
```

Visual Basic

```
Global  Const  CORBA_NO_EXCEPTION   = 0      // Normal termination
Global  Const  CORBA_USER_EXCEPTION   = 1    // User exception
Global  Const  CORBA_SYSTEM_EXCEPTION   = 2  // System exception

Dim excp As Object         // Variable storing exception information
Dim dummy As long          // Variable storing user exception

obj.ope(1000, excp)        // Call ope function
                           // Judge system exception and execute processing
if excp.Ex_major = CORBA_SYSTEM_EXCEPTION Then
   if excp.Ex_RepositoryID = "IDL:CORBA/StExcep/UNKNOWN:1.0" Then
   'Error processing corresponding to UNKNOWN
```

```
   else if excp.Ex_RepositoryID = "IDL:CORBA/StExcep/BAD_PARAM:1.0" Then
   'Error processing corresponding to BAD_PARAM
   .
   .
                          // Judge user exception and execute processing
else if excp.Ex_major = CORBA_USER_EXCEPTION Then
   if excp.Ex_RepositoryID = "IDL:foo:1.0" Then
   dummy = excp.dummy
   'Processing corresponding to user exception
   endif
endif
```

Appendix I Example of Session Management using the Object to Process Bind Function

The following example implements session management for an IDL definition using the object to process bind function:

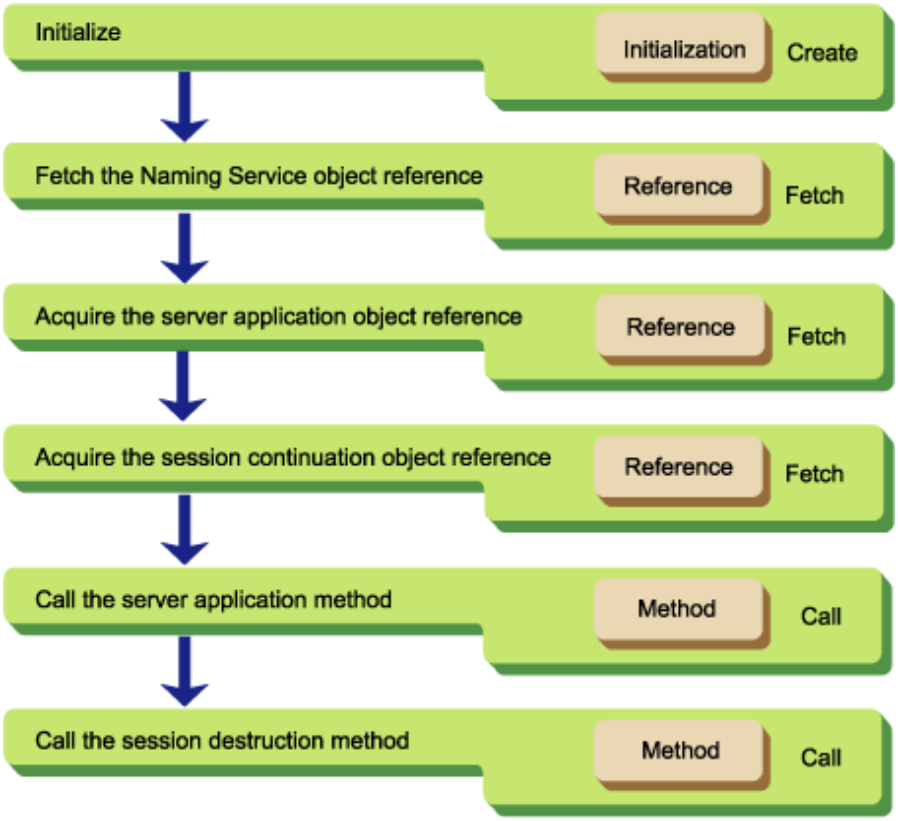
```
module ODsample
{
  interface ssnintf
  {
    void destroyssn();
    void invoke_ssn(in long a);
  };
  interface testintf
  {
    Object createssn();
  };
};
```

In this example, the *testintf* interface is used to get the object reference for session continuation, and the *ssnintf* interface is used to call the method that implements session continuation.

I.1 Client Application Programming

The client application process flow is shown below.

Figure I.1 Client Application Process Flow



I.1.1 Initialization

Call the CORBA initialization method 'CORBA::ORB_init()'. It returns the ORB object reference. This object reference is used when other ORB interfaces are called.

```
main( int argc, char *argv[] )
{
    CORBA::ORB_ptr          orb;                // ORB object reference
    CORBA::Environment_ptr  env;                // Exception information
    int                     current_argc = argc;

    env = new CORBA::Environment;               // Exception information
    orb = CORBA::ORB_init(current_argc, argv, FJ_OM_ORBid, *env);
```

I.1.2 Fetching the Naming Service Object Reference

The Naming Service object reference is required for searching the Naming Service for executed objects. It is fetched by the CORBA interface object reference fetch method, 'CORBA::ORB::resolve_initial_references()', with CORBA_ORB_ObjectId_NameService specified as a parameter.

```
// Acquire the Naming Service object reference
CORBA::Object_ptr
obj = orb->resolve_initial_references( CORBA_ORB_ObjectId_NameService, env );

// Convert to the NamingContext class
CosNaming::NamingContext_ptr
cos_naming = CosNaming::NamingContext::_narrow( obj );
```

I.1.3 Acquiring the Server Application Object Reference

The server application object reference used to obtain the session continuation object reference is fetched by the Naming Service 'CosNaming::NamingContext::resolve()' method. The object name to be searched for is specified as a parameter.

```
CosNaming::Name        name;    // CosNaming::Name interface
CORBA::Object_ptr      obj;     // Server application object reference

name.length(1);
// Object name
name[0]->id = (const CORBA::Char *)"ODsample::ssntest";
// Object type
name[0]->kind = (const CORBA::Char *)"";

// Acquire the server application object reference
obj = cos_naming->resolve( name, *env );

// Convert to the "ODsample::testintf" class
ODsample::testintf_ptr nossn_ap = ODsample::testintf::_narrow( obj );
```

I.1.4 Acquiring the Session Continuation Object Reference

References for server application objects to be executed later are obtained by calling the method used to obtain the session continuation object reference implemented in the server.

```
// Acquire the object reference for session continuation
obj = nossn_ap->createssn( *env );

// Convert to the "ODsample::ssnintf" class
ODsample::ssnintf_ptr ssn_ap = ODsample::ssnintf::_narrow( obj );
```

I.1.5 Calling the Server Application Method

The session continuation server application method is called. If there is an exception on the server application when the method is called, the "CORBA::Environment" structure is used to acquire the exception information.

```
CORBA::Long           iVal;  // Parameter

// Call the method
iVal = 10;
ssn_ap->invoke_ssn( iVal, *env );

iVal = 20;
ssn_ap->invoke_ssn( iVal, *env );
```

I.1.6 Calling the Session Destruction Method

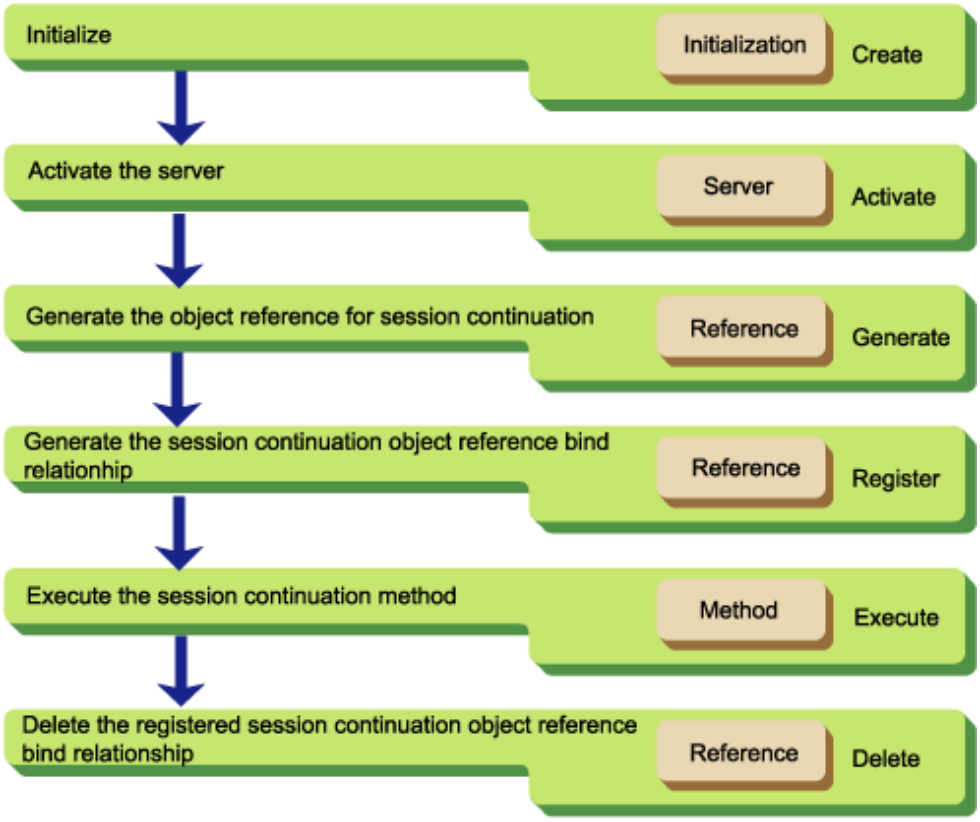
The session destruction method is called to delete the session implemented in the server application.

```
// Call the method for session destruction
ssn_ap->destroyssn( *env );
```

I.2 Server Application Programming

Server application configuration involves initialization and interface implementation. The process flow is shown below.

Figure I.2 Server Application Configuration Process Flow



I.2.1 Initialization

Call the CORBA initialization method 'CORBA::ORB_init()'. It returns an ORB object reference. This object reference is used when ORB class methods are called.

```
main ( int argc, char *argv[] )
{
    CORBA::ORB_ptr          orb;    // ORB object reference
    CORBA::Environment_ptr  env;    // Exception information
    int                     current_argc = argc;
```

```
    env = new CORBA::Environment;
    orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, *env );
```

Call "CORBA::ORB::BOA_init()" to initialize the Basic Object Adapter.

```
    CORBA::BOA_ptr  boa;                // Object reference for BOA
    boa = orb->BOA_init( current_argc, argv, CORBA_BOA_OAid, *env );
```

The session continuation interface object reference is acquired by calling 'CORBA::Repository::lookup_id()' to generate the session continuation object reference.

```
CORBA::InterfaceDef_ptr  intf;// Used to search for interface information
CORBA::Repository_ptr    intf_rep; // Repository object reference
CORBA::Object_ptr        o;        // Object reference pointer

// Search for the interface information object reference
o = orb->resolve_initial_references(
    CORBA_ORB_ObjectId_LightInterfaceRepository, *env );

// Convert to the CORBA::Repository class
intf_rep = CORBA::Repository::_narrow(o);

// Search for the session continuation interface object
o = intf_rep->lookup_id( _INTF_ODsample_ssnintf, *env );

// Convert to the CORBA::InterfaceDef class
intf = CORBA::InterfaceDef::_narrow(o);
```

This can also be used to perform server application initialization processing if required.

I.2.2 Activating the Server

The ORB is notified when server application activation is complete. When the ORB has been notified, it begins dispatch of client requests to the server application.

1. Retrieve the implementation repository object reference

   Use CORBA::ORB::resolve_initial_references() methodto retrieve the implementation repository object reference, specifying CORBA_ORB_ObjectId_ImplementationRepository as a parameter.

2. Search for the ImplementationDef object reference

   Call FJ::ImplementationRep::lookup_id() to get the ImplementationDef object reference object reference, specifying the server application ImplementationRep object reference as a parameter.

3. Server activation

   The server is activated using 'CORBA::BOA::impl_is_ready()'.

```
CORBA::ImplementationDef_ptr impl;     // For searching for implementation information
FJ::ImplementationRep_ptr    impl_rep; // ImplementationRep object reference

// Search for the implementation information object reference
o = orb->resolve_initial_references(
    CORBA_ORB_ObjectId_ImplementationRepository, *env );

// Convert to the ImplementationRep class
impl_rep = FJ::ImplementationRep::_narrow(o);

// Search for the server ImplementationRep object
o = impl_rep->lookup_id( "IMPL_SSNSAMPLE", *env );

// Convert to the ImplementationDef class
impl = CORBA::ImplementationDef::_narrow(o);
```

```
  // Activate the server
  boa->impl_is_ready( impl, *env );
```

I.2.3 Generating the Session Continuation Object Reference

When a client requests the method for obtaining a session continuation object reference, generate an object reference to send to the client.

```
CORBA::Object_ptr
ODsample_testintf_impl::createssn(
    CORBA::Environment  &arg_env )
    throw ( CORBA::Exception )
{
  CORBA::Object_ptr            obj;       // Object reference pointer
  CORBA::ReferenceData         ref_data;  // Object ID information
  CORBA::Environment           local_env; // Exception information

  // Generation of the object reference
  obj = boa->create( ref_data, intf, impl, local_env );
```

I.2.4 Registering the Session Continuation Object Reference Bind Relationship

After generation of the object reference, registration of the bind relationship with the local process is notified to the ORB. If a request that uses this object reference is received, the request is distributed to a process in the registered relationship.

```
  //Register the local process and object reference bind relationship
  orb->bind_object( obj, local_env );

  return obj;
}
```

I.2.5 Executing the Session Continuation Method

When a client requests the session continuation interface method, there is no special processing for session continuation in the server application.

```
void
ODsample_ssnintf_impl::invoke_ssn(
    CORBA::Long         a,
    CORBA::Environment  &arg_env )
    throw ( CORBA::Exception )
{
  // Perform any type of processing
  cout << "invoke_ssn" << a << endl;

  return;
}
```

I.2.6 Deleting the Registered Session Continuation Object Reference Bind Relationship

When the clients request the method for session destruction, issue the function to delete the registered object reference bind relationship.

```
void
ODsample_ssnintf_impl::destroyssn(
    CORBA::Environment  &arg_env )
    throw ( CORBA::Exception )
{
```

```
    CORBA::Environment       local_env; // Exception information

    // Delete the registered object reference bind relationship
    orb->unbind_object( this, local_env );

    return;
  }
```