

FUJITSU Software

Interstage Application Server

A decorative horizontal band with a dark red background. It features several glowing, overlapping white and light red curved lines and arcs, creating a sense of motion and depth.

OLTP Server User's Guide

Windows/Solaris/Linux

B1WS-1091-03ENZ0(00)
April 2014

Preface

Purpose of this Document

The Interstage Application Server OLTP Server User's Guide provides details of the OLTP Server.

Note

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

Intended Readers

This manual is aimed at developers of distributed applications.

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

- Basic knowledge of the OS used

Structure of This Document

The structure of this manual is as follows:

[Chapter 1 OLTP Server of Interstage Application Server](#)

This chapter explains the OLTP server.

[Chapter 2 Designing the OLTP Server](#)

This chapter explains the design method of an OLTP server.

[Chapter 3 Starting / Stopping / Surveillance of WorkUnits](#)

This chapter explains starting, stopping and surveillance of WorkUnits.

[Chapter 4 WorkUnit Operation of Each Application](#)

This chapter explains WorkUnit operation for each application.

[Chapter 5 Operating the Distributed Transaction Function](#)

This chapter explains operating the distributed transaction function.

[Appendix A WorkUnit Definition](#)

This appendix describes the WorkUnit definition.

[Appendix B Interstage Operation API Sample Programs](#)

This appendix contains Interstage operation API sample programs.

[Appendix C Notes on OLTP Server Operations](#)

This appendix contains notes on OLTP server operations.

[Appendix D WorkUnit Automatic Start Setting File](#)

This appendix explains the WorkUnit automatic start setting file that is created to automatically start the WorkUnit.

[Appendix E Procedure for CORBA WorkUnit Operation Using the Interstage Management Console](#)

This appendix explains the procedure for operating CORBA WorkUnits using the Interstage Management Console.










[Appendix F CORBA WorkUnit Activation Change](#)

This appendix explains the CORBA WorkUnit activation change 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.

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

Copyright 2002-2014 FUJITSU LIMITED

April 2014 Third Edition
November 2012 First Edition

Contents

Chapter 1 OLTP Server of Interstage Application Server.....	1
1.1 Application to Business Critical Systems.....	1
1.2 System Integration by Using Existing Applications.....	1
1.3 Client/Server Communication.....	2
1.3.1 Client/Server Communication.....	2
1.4 Improved Distributed System Management.....	4
1.5 Distributed Transactions	4
1.6 Functions for Reliable System Operation.....	6
1.6.1 Basic Operation Control.....	6
1.6.2 Performance Information Measurement.....	7
1.6.3 Realtime Monitoring.....	8
1.7 Performance Analysis.....	9
1.7.1 Automatic Centralized Monitoring.....	9
1.7.2 Automatic Operation for Built-in Batch Programs.....	9
Chapter 2 Designing the OLTP Server.....	12
2.1 Designing WorkUnits.....	12
2.1.1 Application Execution Environment Using WorkUnits.....	12
2.1.2 Application Process Concurrency.....	12
2.1.3 Automatic Application Restart.....	12
2.1.4 Server Application Timer Function.....	13
2.1.5 Current Directory.....	15
2.1.6 Environment Variables.....	17
2.1.7 Queue Control.....	19
2.1.8 Inhibiting and Permitting Queues.....	20
2.1.9 Maximum Queuing Function.....	20
2.1.10 Alarm Report Function for Stagnant Queues.....	21
2.1.11 Priority Control.....	22
2.1.12 Queued Message Cancellation Function.....	22
2.1.13 Buffer Control.....	23
2.1.13.1 Buffer Control for CORBA Applications.....	23
2.1.13.2 Buffer Control for Transaction Applications.....	24
2.1.13.3 Buffer Control for IJServer EJB Applications.....	24
2.1.14 Degenerated Operation when the Automatic Restart of a WorkUnit Process Fails.....	24
2.1.14.1 Restoration of the WorkUnit in Degenerated Operation.....	25
2.1.15 Changing a WorkUnit's Number of Process Concurrency.....	25
2.2 Various Types of WorkUnit.....	25
2.2.1 CORBA WorkUnit.....	25
2.2.1.1 Using WorkUnits.....	26
2.2.1.2 CORBA WorkUnit Operation Functions.....	26
2.2.2 Processing with Transaction Applications.....	27
2.2.2.1 Application Runtime Environments Using WorkUnits.....	29
2.2.2.2 Creating Server Applications that Use APM.....	30
2.2.2.3 Server Object Registration.....	31
2.2.2.4 Manual Registration of Server Objects.....	31
2.2.2.5 Local Transaction Linkage.....	31
2.2.2.6 Global Transaction Linkage.....	32
2.2.2.7 Session Information Management.....	37
2.2.2.8 Server Application Process Modes (In the Case of Solaris and Linux).....	38
2.2.2.9 Specification of the Conditions for Restarting a Server Application Process after its Failure.....	39
2.2.3 Linking with Existing Systems.....	39
2.2.3.1 Wrapping Objects.....	39
2.2.3.2 Wrapper Definitions.....	39
2.2.4 Performing Processing Using General Applications.....	40
2.2.4.1 Using a WorkUnit.....	40

2.2.5 Timeout Monitoring.....	41
2.2.5.1 CORBA Application Timeouts.....	41
2.2.5.2 Monitoring Transaction Application Timeouts.....	45
2.2.5.3 Timeout Monitoring of Global Transaction Applications.....	46
2.2.5.4 Monitoring Session Information Management Function Timeouts.....	48
2.2.6 Design when Using the Operation Support Function of a WorkUnit.....	48
2.2.6.1 WorkUnit Exit Function.....	48
2.2.6.2 Process Salvage Exit Function.....	49
2.2.6.3 WorkUnit Process Information Notification Function.....	49
2.2.7 Various Exit Functions.....	50
2.2.8 Trouble Investigation Support Function.....	50
Chapter 3 Starting / Stopping / Surveillance of WorkUnits.....	53
3.1 Starting and Stopping WorkUnits.....	53
3.1.1 Starting and Stopping WorkUnits Using the Interstage Integration Command.....	53
3.1.2 Monitoring WorkUnits.....	53
3.1.2.1 Operating Status of WorkUnits.....	53
3.1.2.2 Operating Status of Objects of WorkUnits.....	54
3.1.2.3 Checking Application Process Information.....	56
3.1.3 Starting and Stopping WorkUnits Using the Interstage Management Console.....	56
3.1.4 Starting WorkUnits Automatically.....	56
3.2 Performance Monitoring Tool.....	57
3.2.1 Functions of Performance Monitoring Tool.....	58
3.2.1.1 Function of Outputting Log Information to the Performance Log File.....	58
3.2.1.2 Function of Monitoring the Real Time Performance Information by a Network Control Manager (Monitoring by MIB).....	59
3.2.2 Performance Monitoring Procedure.....	60
3.2.3 Registering to the SNMP Service.....	62
3.2.4 Creating a Performance Monitoring Environment.....	63
3.2.4.1 Starting Operation of Performance Monitoring Tool.....	63
3.2.5 Monitoring Operations.....	63
3.2.5.1 Starting Performance Monitoring.....	63
3.2.5.2 Starting a Business Application.....	64
3.2.5.3 Outputting the Performance Log File and Analyzing the Performance Information.....	65
3.2.5.4 Stopping the Application.....	66
3.2.5.5 Stopping the Performance Monitor.....	67
3.2.5.6 Deleting the Performance Monitoring Environment.....	67
3.2.5.7 Deletion from the SNMP Service.....	67
3.2.6 Analyzing the Performance Information and Taking Action.....	67
3.2.6.1 Function of Outputting Log Information to the Performance Log File.....	67
3.2.6.2 Performance Information Collected by the Network Control Manager with the Real Time Monitoring Function.....	70
3.2.6.3 Warnings Regarding the Evaluation of the Performance Information.....	71
3.2.7 Managing the Performance Log Files.....	71
3.2.8 Source Names of the Messages Displayed on the Event Viewer.....	72
3.3 Interstage Operation API.....	72
3.3.1 Function Overview.....	73
3.3.1.1 Interstage Operation API Environment Initialization and Collection.....	73
3.3.1.2 Interstage Operation Information Notification.....	73
3.3.1.3 Interstage Operation.....	75
3.3.1.4 Interstage System Information Notification.....	75
3.3.2 Compiling and Linking Applications.....	76
3.3.3 Examples of Use.....	76
3.3.4 Notes.....	80
3.3.4.1 Command Operations.....	80
3.3.4.2 Starting and Stopping WorkUnits.....	81
3.3.4.3 Operation in the Cluster System.....	81
3.3.4.4 Control Table Version-Level for Interstage Operation API.....	82
3.3.4.5 Parameter Information used by Interstage Operation API.....	82

3.4 Changing WorkUnits.....	83
3.4.1 Adding a WorkUnit (Transaction Applications).....	83
3.4.2 Adding a WorkUnit (EJB Applications).....	84
3.4.3 Deleting a WorkUnit.....	85
3.4.4 Changing a WorkUnit.....	86
3.5 Changing Server Applications.....	86
3.5.1 Adding a Server Application (Transaction Application).....	87
3.5.2 Adding a Server Application (EJB Application).....	88
3.5.3 Deleting Server Applications.....	89
3.5.4 Changing a Server Application (Transaction Application).....	90
3.5.5 Changing a Server Application (EJB Application).....	92
3.5.6 Active Changing of a Server Application (Transaction Applications Only).....	92
3.5.7 Dynamic Changing of the Number of Server Application Processes.....	94
Chapter 4 WorkUnit Operation of Each Application.....	96
4.1 Operating CORBA WorkUnits.....	96
4.1.1 Application Development.....	97
4.1.2 Compilation by IDL Compiler (IDLc Command).....	97
4.1.3 Creating CORBA Server Applications.....	97
4.1.4 Registration of Information on Server Application to Implementation Repository Definition.....	97
4.1.5 Generation of Object Reference.....	98
4.1.6 Specifying/Registering WorkUnit Definitions.....	98
4.1.7 Starting a WorkUnit.....	98
4.1.8 Stopping a WorkUnit.....	98
4.1.9 Operation Using the Interstage Management Console.....	99
4.1.9.1 Creating a CORBA WorkUnit.....	99
4.1.9.2 Deploying a CORBA Application.....	100
4.1.9.3 Closure/Closure Cancellation.....	103
4.1.9.4 Function for Maximum Number of Messages Retained in a Queue.....	103
4.1.9.5 Alarm Notification Function for Number of Messages in a Queue.....	103
4.1.10 Operation by Command Line Interface.....	103
4.1.10.1 CORBA WorkUnit Definitions.....	103
4.1.10.2 CORBA Application Queue Control.....	106
4.1.10.3 Global Transaction Linkage.....	107
4.2 Operating EJB WorkUnits.....	109
4.2.1 Specifying/Registering WorkUnit Definitions.....	109
4.2.1.1 Starting a WorkUnit.....	110
4.2.1.2 Stopping a WorkUnit.....	110
4.3 Operating Transaction Application WorkUnits.....	110
4.3.1 Operation Using the Object Priority Function.....	110
4.3.2 Operating Procedures for Object Priority Function.....	111
4.3.2.1 Create Application.....	111
4.3.2.2 Specify WorkUnit Definitions.....	111
4.3.2.3 Register WorkUnit Definitions.....	112
4.3.2.4 Start the WorkUnit.....	112
4.4 Operation in Utility WorkUnits.....	112
4.4.1 Operating Procedures.....	113
Chapter 5 Operating the Distributed Transaction Function	116
5.1 Procedure Required to Use Distributed Transaction Function	116
5.2 Setting Up the System Environment for the Distributed Transaction Function	117
5.2.1 Tuning the System	117
5.2.2 Determining If a Disk Partition Is Necessary	117
5.2.3 Setting the CORBA Service Operating Environment File.....	117
5.2.4 Setting Up the Database Linkage Service Environment Definition.....	118
5.2.5 Setting Up the Interstage Operating Environment Definitions.....	119
5.3 Creating the OTS System Environment	119
5.3.1 Using the Interstage Management Console.....	119

5.3.2 Using Commands.....	119
5.3.2.1 Creating an Interstage Operating Environment Definition.....	119
5.3.2.2 isinit Command and otsetup Command.....	120
5.3.2.3 Using a Local Naming Service (Recommended).....	120
5.3.2.4 Using Remote Naming Service.....	120
5.4 Creating a Resource Management Program.....	121
5.4.1 What is a Resource Management Program?.....	122
5.4.2 The XA Linkage Program.....	122
5.4.3 The Database Library.....	123
5.4.4 Creating a Resource Management Program.....	123
5.5 Creating the Environment for the Resource Management Program.....	123
5.5.1 Creating a Resource Definition File.....	124
5.5.1.1 Registering a Resource Definition.....	124
5.5.1.2 Environment Definition for Resource Management Program.....	125
5.6 Creating Definitions.....	125
5.6.1 Resource Manager Information.....	125
5.7 Starting the OTS System.....	125
5.8 Starting and Stopping a Resource Management Program.....	126
5.8.1 Environment Setting for Operation on a Host Other Than That of the OTS System.....	126
5.8.1.1 Sharing a Naming Service (Recommended).....	126
5.8.1.2 Not Sharing a Naming Service.....	127
5.9 Tracing Function.....	128
5.9.1 Dump File Collection Function.....	128
5.9.2 Trace Log Output Function.....	128
5.10 Notes.....	128
5.10.1 Migration from the Old Environment.....	128
Appendix A WorkUnit Definition.....	129
A.1 Syntax.....	129
A.2 Syntax of WorkUnit Definition File.....	129
A.3 Control Statement of WorkUnit Definition.....	132
A.3.1 WORK UNIT Section.....	132
A.3.1.1 Name.....	132
A.3.1.2 Kind.....	132
A.3.2 APM Section.....	133
A.3.2.1 Name.....	133
A.3.3 Control Option Section.....	133
A.3.3.1 Path.....	135
A.3.3.2 Current Directory.....	136
A.3.3.3 Remove Directory.....	136
A.3.3.4 Application Retry Count.....	137
A.3.3.5 Snapshot.....	137
A.3.3.6 Path for Snapshot.....	138
A.3.3.7 Path for Application.....	138
A.3.3.8 Library for Application.....	138
A.3.3.9 Environment Variable.....	139
A.3.3.10 Registration to Naming Service.....	139
A.3.3.11 Using Load Balance.....	140
A.3.3.12 Using Notification of User Information.....	140
A.3.3.13 Access Control.....	141
A.3.3.14 Access Control Base DN.....	141
A.3.3.15 Maximum Processing Time For Exit Program.....	141
A.3.3.16 WorkUnit Exit Program.....	142
A.3.3.17 Executable File of Exit Program for Salvage.....	143
A.3.3.18 WorkUnit Automatic Stop Mode.....	143
A.3.3.19 Request Assignment Mode.....	143
A.3.3.20 Traffic Director Monitor Mode.....	144

A.3.3.21 Output Of Stack Trace.....	144
A.3.3.22 Startup Time.....	145
A.3.3.23 Shutdown Time.....	145
A.3.3.24 Unconditional Reactivation of Process.....	146
A.3.3.25 Start Log.....	146
A.3.3.26 Process Degeneracy.....	147
A.3.3.27 Number of Revision Directories.....	147
A.3.4 Application Program Section.....	147
A.3.4.1 Destination.....	151
A.3.4.2 Destination Priority.....	152
A.3.4.3 PSYS.....	152
A.3.4.4 Executable File.....	152
A.3.4.5 Application Language.....	153
A.3.4.6 Concurrency.....	153
A.3.4.7 Maximum Processing Time.....	154
A.3.4.8 Terminate Process for Time out.....	154
A.3.4.9 Maximum Processing Time For Exit Program.....	154
A.3.4.10 Maximum Queuing Message.....	155
A.3.4.11 Queuing Message to Notify Alarm.....	155
A.3.4.12 Queuing Message to Notify Resumption.....	156
A.3.4.13 Environment Variable.....	156
A.3.4.14 Form.....	157
A.3.4.15 Pre Exit Program.....	157
A.3.4.16 Post Exit Program.....	158
A.3.4.17 Recovery Exit Program.....	158
A.3.4.18 Executable File for Exit Program.....	158
A.3.4.19 Access Control.....	159
A.3.4.20 Access Control BaseDN.....	159
A.3.4.21 Type of User Identification.....	159
A.3.4.22 User Name Param.....	160
A.3.4.23 User Base DN.....	160
A.3.4.24 User DN Param.....	161
A.3.4.25 Password Param.....	161
A.3.4.26 Bind Type.....	161
A.3.4.27 Using Wrapper Session Management.....	162
A.3.4.28 SessionID Param.....	162
A.3.4.29 Method Name to Begin Session.....	162
A.3.4.30 Maximum Session Active Time for Client.....	163
A.3.4.31 Maximum Processing Time for WRAPPER.....	163
A.3.4.32 Maximum Memory for EJB Application.....	164
A.3.4.33 CLASSPATH for Application.....	164
A.3.4.34 Java Command Option.....	164
A.3.4.35 Exit Program for Process Salvage.....	165
A.3.4.36 Executable File of Exit Program for Salvage.....	165
A.3.4.37 Exit Program for Terminating Process.....	165
A.3.4.38 Param for Executable File.....	166
A.3.4.39 Reset Time for Application Retry Count.....	166
A.3.4.40 Request Assignment Mode.....	166
A.3.4.41 Impl ID.....	167
A.3.4.42 Buffer Number.....	167
A.3.4.43 Buffer Size.....	167
A.3.4.44 Path.....	168
A.3.5 Nonresident Application Process Section.....	168
A.3.5.1 Concurrency.....	169
A.3.5.2 Pre Exit Program.....	169
A.3.5.3 Post Exit Program.....	169
A.3.5.4 Executable File for Exit Program.....	170

A.3.5.5 Maximum Processing Time for Exit Program.....	170
A.3.5.6 Dynamic Link Library.....	170
A.3.5.7 Exit Program for Process Salvage.....	171
A.3.5.8 Executable File of Exit Program for Salvage.....	171
A.3.6 Multiresident Application Process Section.....	171
A.3.6.1 Concurrency.....	172
A.3.6.2 Pre Exit Program.....	172
A.3.6.3 Post Exit Program.....	172
A.3.6.4 Executable File for Exit Program.....	173
A.3.6.5 Maximum Processing Time for Exit Program.....	173
A.3.6.6 Recovery Exit Program.....	173
A.3.6.7 Exit Program for Process Salvage.....	173
A.3.6.8 Executable File of Exit Program for Salvage.....	174
A.3.7 Resource Manager Section.....	174
A.3.7.1 Name.....	174
A.3.7.2 RM.....	175
Appendix B Interstage Operation API Sample Programs.....	176
B.1 File Configuration.....	176
B.1.1 WorkUnit Startup Program.....	177
B.1.2 WorkUnit Stop Program.....	178
B.1.3 WorkUnit/Object Information Acquisition Program.....	178
B.1.4 Object Close Program.....	178
B.1.5 Cancel Object Closure Program.....	179
B.1.6 Information Acquisition of the Object in the Implementation Repository ID Program.....	179
B.1.7 Acquiring/releasing of a system name list Program.....	180
B.2 Compiling and Linking.....	180
Appendix C Notes on OLTP Server Operations.....	182
C.1 Operation Using the Interface Information Check Functions.....	182
C.1.1 Procedure for Operating the Interface Information Check Function.....	184
Appendix D WorkUnit Automatic Start Setting File.....	187
D.1 Coding Format.....	187
D.1.1 Statement.....	187
D.1.2 Section.....	188
D.1.3 Comment Line.....	188
D.1.4 Space Line.....	189
D.2 WorkUnit Automatic Start Setting File Example.....	189
D.2.1 WorkUnit Name.....	189
D.2.2 User Name.....	189
Appendix E Procedure for CORBA WorkUnit Operation Using the Interstage Management Console.....	190
E.1 For Solaris.....	190
E.1.1 Procedure for Operation Using a Sample C Application.....	190
E.1.2 Procedure for Operation Using a Sample Java Application.....	192
E.1.3 Procedure for Operation Using a Sample C++ Application.....	194
E.1.4 Procedure for Operation Using a Sample COBOL Application.....	196
E.2 For Windows(R).....	198
E.2.1 Procedure for Operation Using a Sample C Application.....	199
E.2.2 Procedure for Operation Using a Sample Java Application.....	201
E.2.3 Procedure for Operation Using a Sample C++ Application.....	204
E.2.4 Procedure for Operation Using a Sample COBOL Application.....	206
E.3 For Linux.....	208
E.3.1 Procedure for Operation Using a Sample C Application.....	209
E.3.2 Procedure for Operation Using a Sample Java Application.....	211
E.3.3 Procedure for Operation Using a Sample C++ Application.....	213
E.3.4 Procedure for Operation Using a Sample COBOL Application.....	215

Appendix F CORBA WorkUnit Activation Change.....	218
F.1 Procedure for CORBA WorkUnit Activation Change.....	218
F.1.1 WorkUnit Configuration Change Phase.....	218
F.1.2 Preparation ("prepare") Phase.....	219
F.1.3 Switch to New Environment ("change") Phase.....	220
F.1.4 Old Environment Deletion or Restoration (Commit or Rollback) Phase.....	220
F.2 WorkUnit Configurations that can be Changed during Activation Change.....	220
F.3 Notes on Executing Activation Change.....	225
F.3.1 Relationship between Activation Change and Existing Functions.....	225
F.3.2 WorkUnit Environment after Restart.....	225
F.3.3 Stopping a WorkUnit while the Activation is being Changed.....	226
F.3.4 Current Directory.....	226
F.4 Activation Change Command References.....	227
F.4.1 ischangewudef.....	227
F.4.2 isinfchangewudef.....	228
F.4.3 ispreparewu.....	228
F.4.4 ischangewu.....	229
F.4.5 iscommitwu.....	230
F.4.6 isrollbackwu.....	231
F.4.7 ischeckwustat.....	231

Chapter 1 OLTP Server of Interstage Application Server

With its open technology, Interstage can be used to build mission-critical, highly reliable systems that build on the capabilities of legacy systems.

Interstage also provides communication methods required to develop various types of systems or to link with different systems.

This chapter explains what Interstage can do and how to use it.

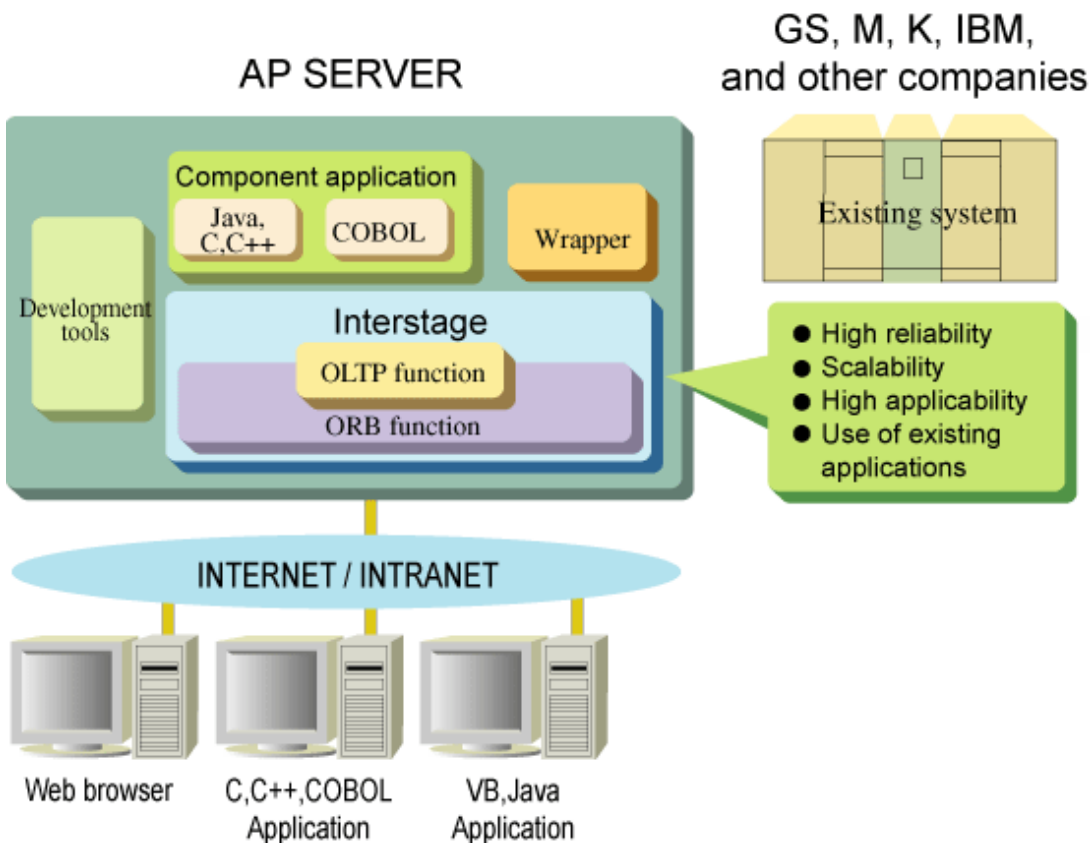
1.1 Application to Business Critical Systems

The essential requirements for development of business critical systems are high reliability, a high degree of scalability and high performance.

A product that satisfies these requirements is usually called an OLTP product. The OLTP services implemented by Interstage include resource sharing functions to conserve resources and queue control functions for response security. These functions, in combination with the core ORB functions and other application management functions of Interstage, achieve high performance levels.

Interstage maintains the stability, scalability, and high performance of a business- critical system and enables the construction of the latest type of Internet and extranet information systems (the following figure).

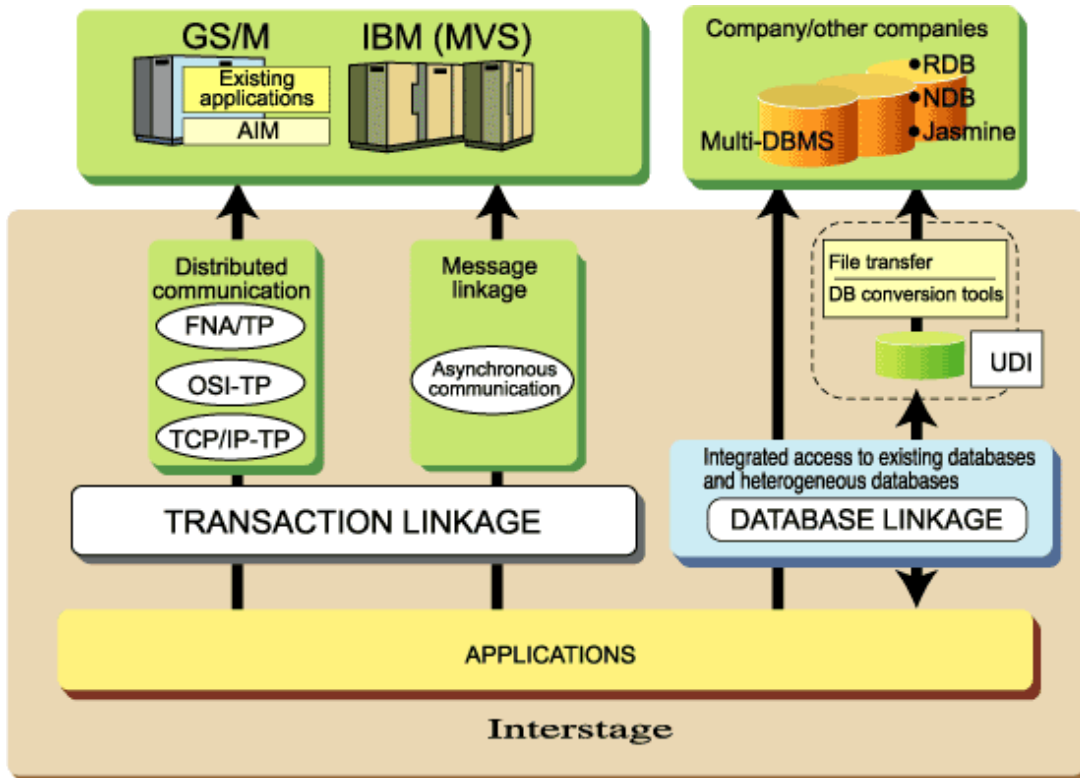
Figure 1.1 Interstage Application to Business Critical Systems



1.2 System Integration by Using Existing Applications

As well as providing such basic functions for new applications as an application server, Interstage also provides a wrapper function that encapsulates existing applications and databases. It is possible to develop new systems that fully utilize the Interstage basic functions in addition to services provided by existing applications (the following figure).

Figure 1.2 System Integration Using Existing Applications



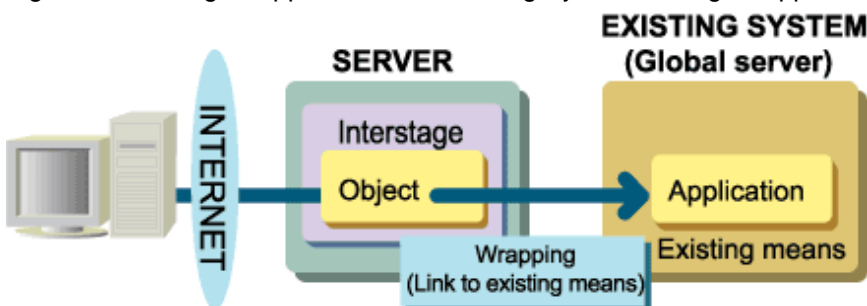
Note

FNA/TP is Fujitsu's network architecture that is supported on Solaris.

Using Application Wrappers

With Interstage, existing system applications can be used with either synchronous or asynchronous communication models, and are represented as distributed environment objects. Clients can link directly to applications on existing systems using wrappers (the following figure).

Figure 1.3 Linking to Applications on Existing Systems Using Wrappers



1.3 Client/Server Communication

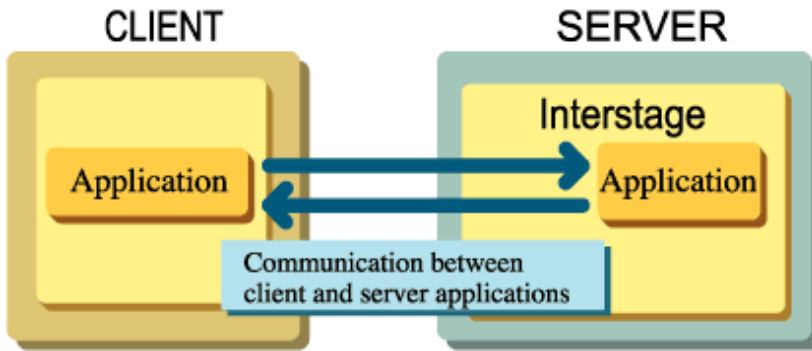
Interstage enables communication between client and server supported by CORBA. Similarly, applications can be linked between Windows® clients, UNIX servers (Solaris) and other CORBA based systems.

Communication between client and server is carried out through an ORB (Object Request Broker), which is present on both machines, and both client and server always exchange data via the ORB.

1.3.1 Client/Server Communication

Application communication between client and server is possible, as shown in the following figure.

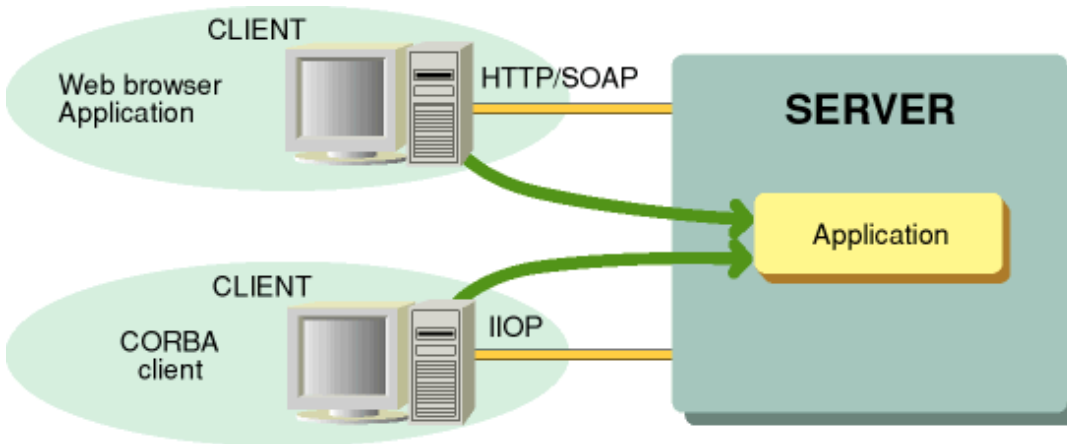
Figure 1.4 Client/Server Communication



As shown in the following figure, three types of client/server communication are used by Interstage:

- Client/Server Communication Using IIOP
- Client/Server Communication Using HTTP
- Client/Server Communication Using SOAP

Figure 1.5 Client/Server Communication using HTTP and IIOP



Client/Server Communication Using IIOP

IIOP (Internet Inter-ORB Protocol) is the communications protocol used to communication between a CORBA client and a CORBA server. A client can communicate only by preparing a Web browser.

The CORBA client can be developed in Java, C, C++ or Visual Basic.

Client/Server Communication Using HTTP

HTTP (Hypertext Transfer Protocol) is used as the communications protocol. Communication can be carried out between client and server via the Internet or an intranet using a Web browser or from a Java application.

In addition to the publication of HTML web pages, the Web server provides CGI applications, session control applications and server-resident applications, as well as services that can be easily created using COBOL (COBOL Web Subroutine).

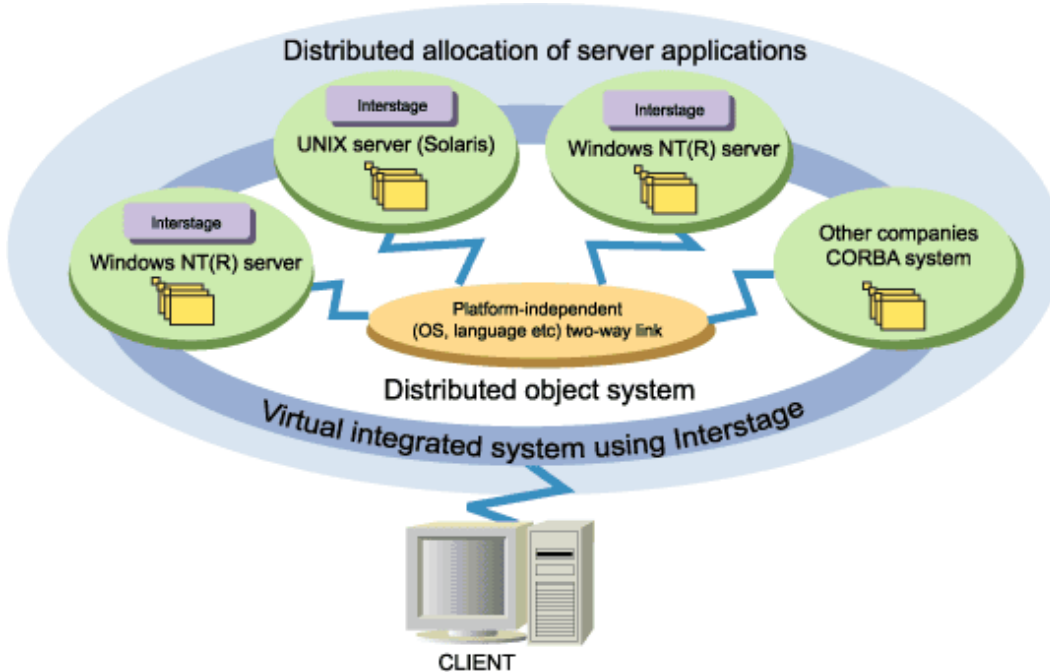
Client/Server Communication Using SOAP

The Simple Object Access Protocol (SOAP) is used as the communication protocol. This is used when the client/ server communication adopting SOAP is carried out. The World Wide Web Consortium (W3C) is promoting the standardization of SOAP which is a light protocol. SOAP uses internet standard HTTP and eXtensible Markup Language (XML) as the communication infrastructure.

1.4 Improved Distributed System Management

Using CORBA, Interstage has surmounted the problem of integrating different systems and development languages, allowing development of a single virtual integrated system (the following figure).

Figure 1.6 Development of a Single Virtual Integrated System



Information such as the object storage location is maintained in one place within the distributed object system. On the client side, by using this control information base, it is possible to develop applications without being aware of the final server configuration.

Specification of the linkage point can be omitted from programs making it easy to create new applications. System configuration can be changed centrally without needing to update client applications.

High Reliability by Using Transactions

Database operations can be secured using the commit and roll-back capability provided by the transaction function. Application systems that demand reliability can be developed easily using the transaction capability.

Windows32/64 Solaris32 Linux32/64

Using global transaction linkage data integrity is assured because the transaction can incorporate updates to multiple databases.

1.5 Distributed Transactions Windows32/64 Solaris32 Linux32/64

To maintain data integrity, database updating must be performed as an integrated, inseparable series of processing steps. This type of processing is referred to as transaction processing.

A transaction that occurs with a single object or resource is referred to as a local transaction.

A global transaction, on the other hand, is an operation that centrally manages local transactions with multiple objects or resources in such a way that they cooperate with each other as if they were a single transaction. In other words, a global transaction comprises multiple local transactions that can be handled as if they were a single transaction.

The Interstage Database Linkage Service can implement an X/Open distributed transaction processing (DTP) model and guarantees the ACID characteristics of distributed transactions ([Table 1.1 ACID Characteristics](#) and [Table 1.2 Components that Constitute a DTP Model](#)).

Table 1.1 ACID Characteristics

Characteristic	Description
Atomicity	Guarantees that every transaction is either completed or in its original state and that data that is subject to the operation (such as data in databases) is either completely updated or completely not updated.
Consistency	Guarantees that database data is free of inconsistencies regardless of whether or not transactions are completed.
Isolation	Guarantees that multiple transactions processed concurrently do not interfere with each other and do not affect data.
Durability	Guarantees that the results of transaction processing, once completed, are maintained even if errors occur.

Table 1.2 Components that Constitute a DTP Model

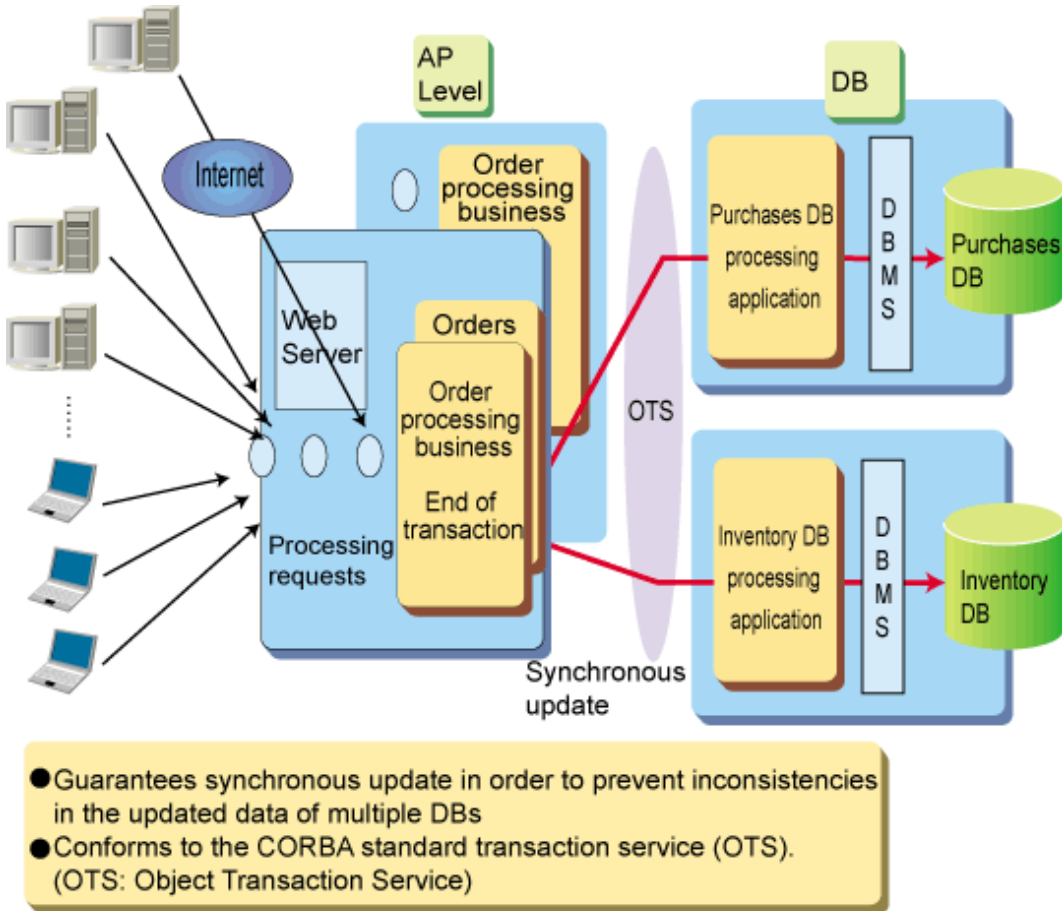
Component	Description
AP (Application Program)	An AP is a component installed by the application developer which controls transactions by using the service provided by TM or RM. A J2EE application such as EJB is an example of an AP.
TM (Transaction Manager)	A TM is a transaction service that manages transactions. Interstage provides the Database Linkage Service as a transaction service.
RM (Resource Manager)	An RM is a program that manages resource transactions. A database management system is an example of such a program.
cRM (Communication RM)	A cRM is a service that provides a communication base to enable transaction control. Interstage provides the CORBA Service as a cRM.

Consistent Processing of Distributed Databases

Interstage provides, as the database linkage service, the distributed transaction control function. This function conforms to the international standards (OMG/CORBA) of the distributed object environment and the Java distributed transaction control mechanism, which is a distributed database environment. This function enables consistent processing of the distributed databases (the following figure).

The databases supported are Symfoware, Oracle databases, and the SQL Server in a Windows® environment, thus extending interoperability.

Figure 1.7 Non-stop Processing of Distributed Databases



1.6 Functions for Reliable System Operation

Interstage provides an application management function that monitors operating status to ensure stable operation in systems built using Interstage.

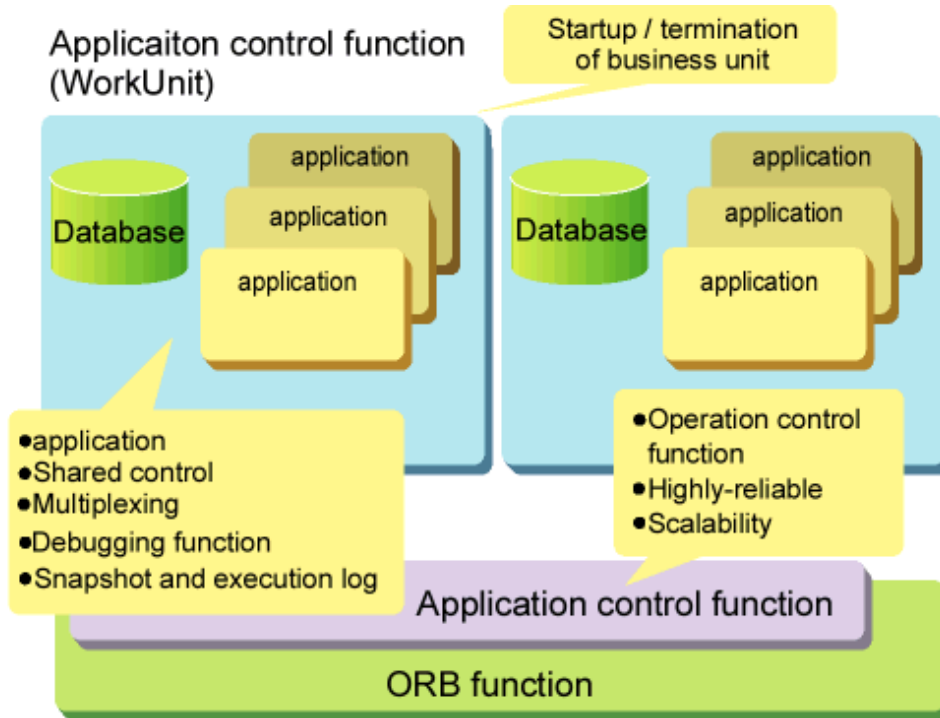
This section provides an overview of the Interstage operation functions.

1.6.1 Basic Operation Control

The application control function was created to improve business application operability (the following figure). It allows the construction of an environment for business units (WorkUnits) and provides the following functions:

- Business unit operation (startup/termination)
- Multi-client process sharing (conserving resources)
- Non-resident applications (conserving resources)
- Multi-level processes (improving scalability)
- Mid-execution snapshots/logs (improving maintainability)

Figure 1.8 Basic Operation Control



1.6.2 Performance Information Measurement

WorkUnit Types That Can Use the Performance Monitoring Tool

Interstage supports the following types of WorkUnits that can use the performance monitoring tool:

- Transaction application WorkUnits
- CORBA WorkUnits

The performance monitoring tools measure two types of performance information for jobs running as WorkUnits on Interstage.

The first type of performance information is measured in object units and gives a performance overview. This overview is used when a network control manager (*1), such as Systemwalker/CentricMGR, is used to give realtime displays of performance information and to monitor the performance of jobs running on Interstage. (Use of a network control manager to monitor and display performance information is known as realtime monitoring.)

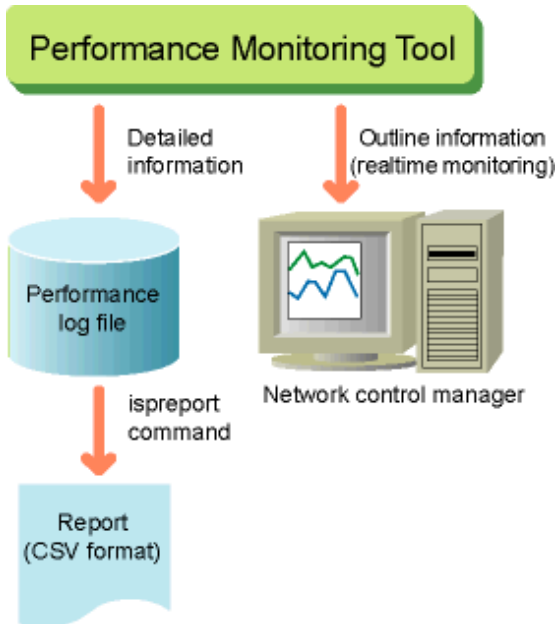
*1 A network control manager program monitors performance and displays the performance information on the monitoring server.

The second type of performance information is more detailed, and is measured in object, operation, and process units. This detailed information is saved in a performance log file, which is the information source for job performance analysis.

The files converted to CSV format can be used in spreadsheet programs such as Excel, allowing the presentation of information from the file in a graphic or other format that enables the user to analyze the performance information from various perspectives.

Types of performance information measurement are shown in the following figure.

Figure 1.9 Performance Information Measurement

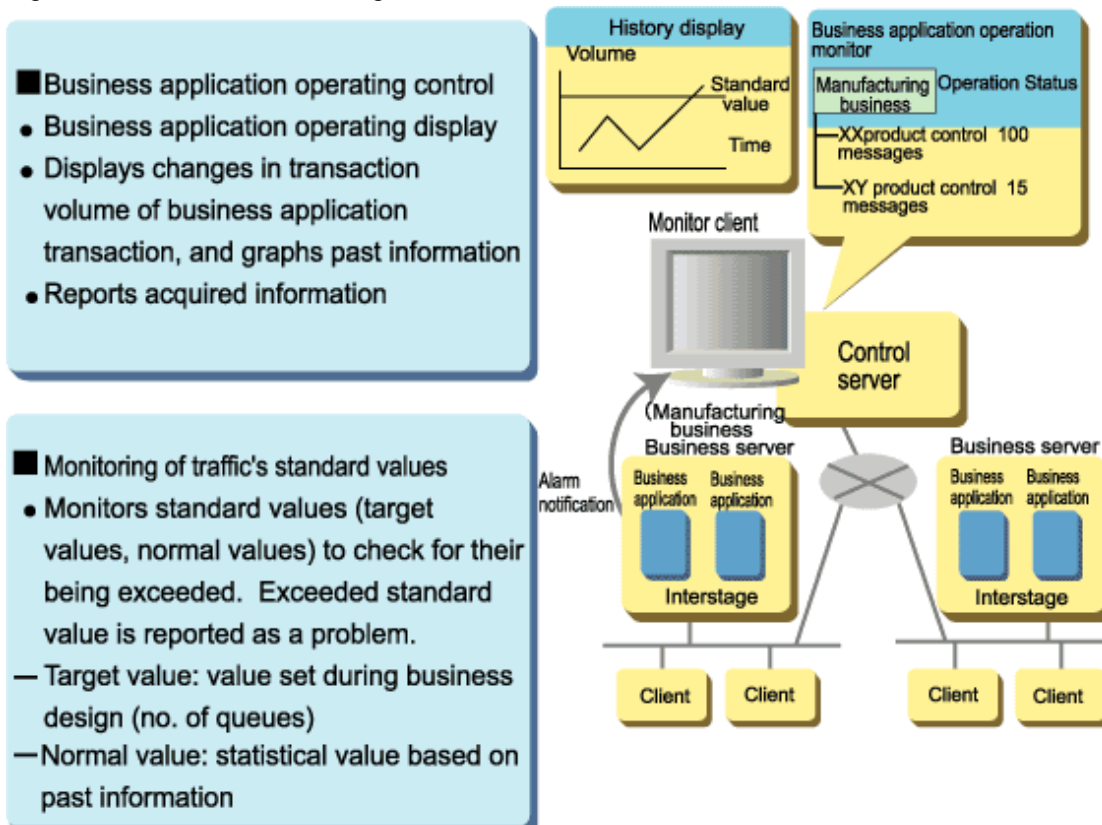


1.6.3 Realtime Monitoring

When a network control manager (monitoring client) such as Systemwalker/CentricMGR is used with Interstage, overview information can be displayed in real time. Thus, the operating status and performance of jobs in a distributed environment can be monitored in real time. Base values can be set at the network control manager. If the base values are exceeded during monitoring, an alarm is issued to facilitate detection of performance faults.

Select realtime display of overview information if realtime monitoring of job operating status and performance (and realtime detection of performance faults) is required (the following figure).

Figure 1.10 Realtime Monitoring



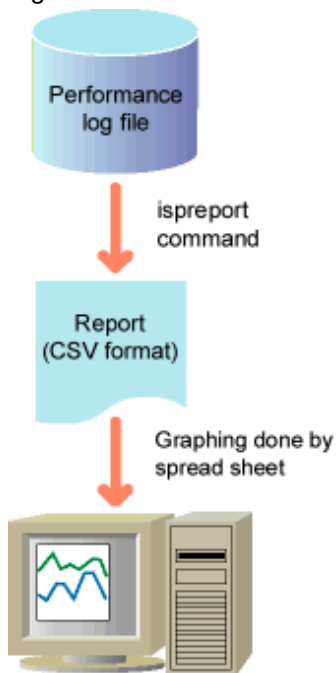
1.7 Performance Analysis

Performance monitoring tools measure overview information and detailed information simultaneously. If performance faults are detected by the realtime monitoring function, analyze the job performance by referring to the detailed information stored in the performance log files.

The performance log files are binary files. Use the command provided by the performance monitoring tools to convert these files to CSV format for analysis.

The files converted to CSV format can be used in spreadsheet programs such as Excel, allowing presentation of information from the file in a graphic or other format that enables the user to analyze the performance information from various perspectives (the following figure).

Figure 1.11 Performance Analysis



1.7.1 Automatic Centralized Monitoring

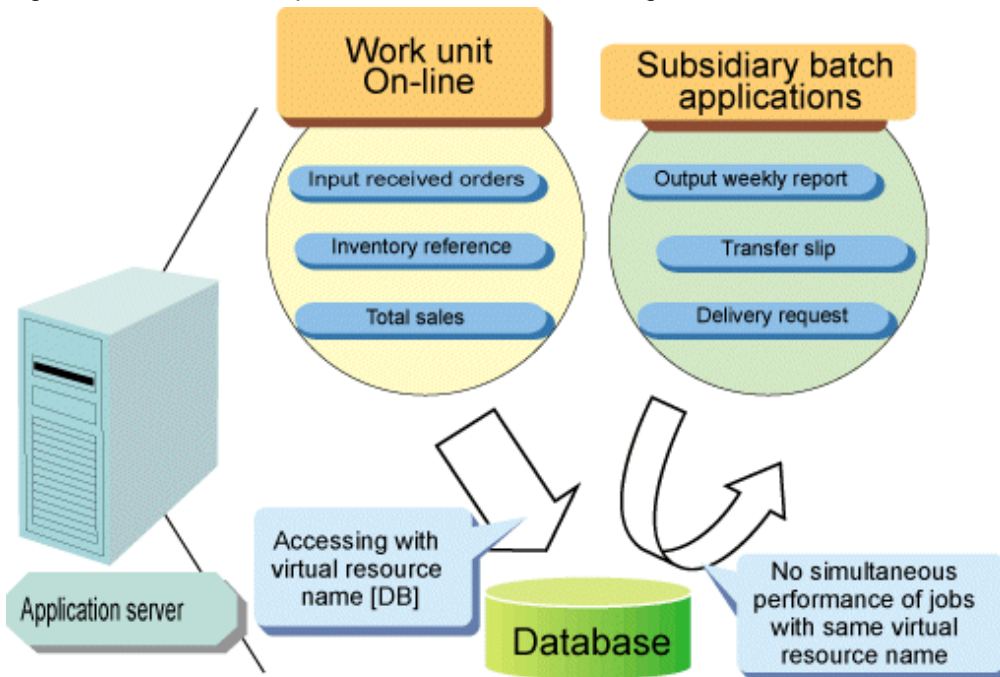
If you are running an Interstage business system on multiple servers, you can perform centralized monitoring of the system by linking it with Systemwalker/CentricMGR. This will simplify system operation. Systemwalker linkage offers the following functions:

- Monitoring function: You can monitor business system status from an operation control server in real time. You can grasp the entire system's status and find trouble spots more easily. WorkUnit and object operating status monitoring by Systemwalker CentricMGR can be used for CORBA WorkUnits and transaction application WorkUnits.
- Remote operation function: You can send Interstage commands to a remote business system from the operation control server. This enables remote system maintenance and administration.
- Application distribution and management function: You can administer Interstage software resources from the operation control server, distributing them in batch to servers and clients on-line. This makes it easier to control versions and distribute system software resources, and reduces any errors or problems that may occur.

1.7.2 Automatic Operation for Built-in Batch Programs

Core system design must include automatic operations that allow built-in batch applications. It must also include exclusive control for batch applications (the following figure).

Figure 1.12 Automatic Operation for Built-in Batch Programs



Systemwalker uses a jobnet to control batch jobs. Interstage can recognize its applications as on-line jobs. It is possible to apply exclusive control between on-line jobs and between batch applications.

When you link System Walker/OperationMGR you can automate the following system operations and controls.

- Automatic work unit execution

Work unit start and stop times registered in jobnet will start/stop according to schedule.

- Work unit and batch job exclusive access

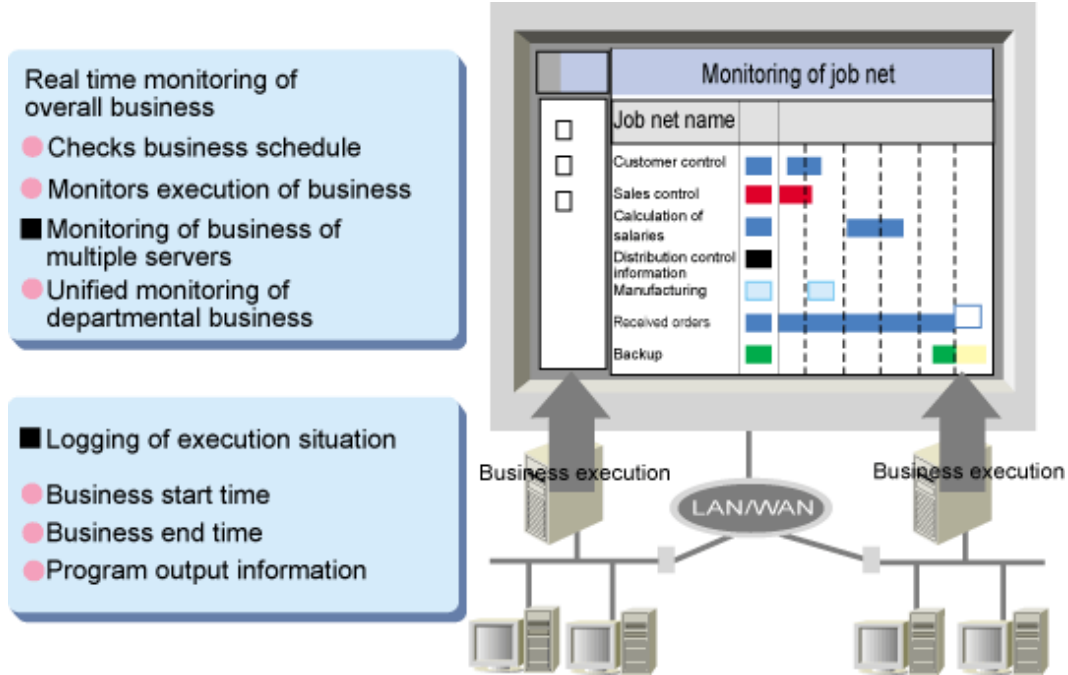
When you register jobs, specify the work unit's logical resource name (queue name, resource name). This will enable WorkUnits and batch jobs to have exclusive control of resources, and allow linked scheduling for WorkUnits and their subsidiary batch applications.

- Work unit monitoring/operation

It is easy to confirm work unit conditions as they are displayed, color-coded, in real time.

Systemwalker/OperationMGR's automatic operation can use a Gantt chart to monitor and automatically operate all jobs. When Interstage business applications are running, their work unit conditions are displayed in a Gantt chart. For easy reference, the Gantt chart color codes the start time, current time, and estimated completion time (the following figure).

Figure 1.13 Systemwalker/OperationMGR's Automatic Operation



Chapter 2 Designing the OLTP Server

This chapter provides information on designing the OLTP server.

2.1 Designing WorkUnits

2.1.1 Application Execution Environment Using WorkUnits

A WorkUnit is a unit of application operation. Both starting and stopping applications are defined in one WorkUnit.

WorkUnit is a distinct unit of operation which may contain multiple applications. However, if one application in the WorkUnit fails and causes an abnormal termination of the WorkUnit, all applications in the WorkUnit will be stopped. Minimize the risk of one application's failure affecting others by creating separate WorkUnits for important applications.

Note that the number of WorkUnits, objects, and processes that can run in the entire Interstage system is limited. The formula for calculating this limit is as follows:

$$[\text{Number of WorkUnits}] + [\text{total number of objects}] + [\text{total number of processes} \times 2] [\leq] 1,600$$

(The total number of objects is the total number of [Application Program] sections in the WU definition.)

To operate CORBA WorkUnits using the Interstage Management Console, refer to the Operation using Interstage Management Console information in each section.

The user name used to operate WorkUnits must satisfy the following two conditions:

- The user name must consist only of characters permitted by the OS.
- The user name must be no more than eight bytes.

2.1.2 Application Process Concurrency

When multiple clients simultaneously issue requests to a single application, the requests will be processed by creating multiple execution units (processes) that can be executed concurrently. The number of concurrent processes for processing multiple requests (process concurrency) can be defined in the WorkUnit definition.

If a WorkUnit control command is used to define a CORBA application, the process concurrency must be defined not only in the WorkUnit definition but also in the implementation repository using the *OD_impl_inst* command.

The process concurrency considerations include the processing time per process, the response time to the client, and the number of requests per time unit.

Information regarding the response time to the client can be collected using the performance monitoring function. For more information about the performance monitoring function, refer to the "Performance Monitoring" chapter in the Operator's Guide.

Defining an unnecessarily high process concurrency adversely affects the system resources, such as memory. Define a level of process concurrency appropriate for your system.

Setting Process Concurrency Using Interstage Management Console

Set "process concurrency" on the CORBA application deployment screen or environment setup screen. If the process concurrency is set using the Interstage Management Console, it will not need to be set again in the implementation repository using the *OD_impl_inst* command.

2.1.3 Automatic Application Restart

If an application terminates abnormally due to input of invalid data from the client, the application can be set to automatically restart. If automatic restart is defined, requests from clients after an abnormal termination can be processed automatically.

If an application terminates abnormally due to an error before processing can be completed, it may restart in an attempt to continue processing. You can configure the retry count to be the maximum number of times the application process terminates abnormally before the WorkUnit also terminates abnormally. If this happens, requests that are being processed in the WorkUnit are returned to the client as an error.

If [1] is specified for the retry count, the application restart is not performed. If [0] is specified for the retry count, the number of application restarts is unlimited and the WorkUnit application continues, even if the application process terminates abnormally.

If the process restart fails, the WorkUnit terminates abnormally and all applications running in the WorkUnit stop.

For a transaction application, abnormal termination of the WorkUnit occurs when the pre-exit program returns abnormally or terminates during process restarting. To prevent this problem, take measures so that a retry for recoverable errors is performed within the pre-exit program.

Degenerated Operation when the Restart of a Process Fails

When the restart of a process fails, the WorkUnit is terminated abnormally. However, for a WorkUnit in which two or more applications are operating concurrently, a function is provided to avoid the abnormal termination of the WorkUnit. This ensures the WorkUnit's continued operation, even if the restart of a process fails. This function continues the WorkUnit operation of the remaining processes after outputting the message to indicate the failed restart of the process. For details, refer to "[2.1.14 Degenerated Operation when the Automatic Restart of a WorkUnit Process Fails](#)".

Setting Retry Count using the Interstage Management Console

Set a numerical value (0-255) for the "retry count" on the WorkUnit definition screen or WorkUnit environment setup screen.

Determining when 'the number of times the application process terminates abnormally' is Reset in a WorkUnit

The WorkUnit is not an IJServer WorkUnit

If the application process runs normally (that is, the application returns) even once before the value set for 'number of times the application process terminates abnormally' is reached, the value is reset.

IJServer

The value is reset after the application first terminates abnormally once the time set for [Retry Count Set Time] in the Interstage Management Console WorkUnit:New or Interstage Management Console:Environment Settings window is exceeded.

2.1.4 Server Application Timer Function

The response time between the calling of the server application running under control of a WorkUnit and its return can be monitored by specifying the maximum processing time (time-out) of the application. Use this function to prevent a delay in response to the client (such a delay can occur because of application failure, or because of a processing loop). Specify the maximum processing time of the application in the WorkUnit definition.

The behavior of the application at time-out varies depending on the application type. The server application timer function is also valid for CORBA, J2EE, and transaction applications.

Refer to "[2.2.5 Timeout Monitoring](#)" for more details of the timer function.

Use the timer function as a corrective measure when an application return is delayed because of an error in the server application. For general operation, avoid using the timer function as it may cause frequent time-outs.

CORBA Application

The following modes can be selected for behavior at time-out:

- The system outputs a time-out message, and forcibly terminates the server application in which a time-out occurred. After forcibly terminating the process, the application returns with an error to the client. Note that in this mode the application of another thread running concurrently is also forcibly terminated even if it is processing. Use this mode carefully. This mode is effective for a server application running in process mode, or if each applicable process can be stopped when a time-out occurs.
- The system only outputs a time-out message, but does not terminate the server application process. For a server application running in multithread mode, an application running properly in one thread may be forcibly stopped due to a problem in an application running in another thread. This will avoid this situation. Note that in this mode the application may return normally after a message is output. When using this function, use it together with the client time-out function (period_receive_timeout) to determine whether any client time-out can be attributed to a delay in the server application.

J2EE Application

The following patterns may be selected as the behavior to be followed when a timeout occurs. In both cases, a thread dump is collected automatically when the timeout occurs.

- A message is output informing you that the timeout time was exceeded, and the server application process belonging to the timed-out processing is stopped by force. After this, an error/exception is returned to the client. Select this pattern if, after considering the fact that processing is executed in more than one thread on the corresponding process at the same time, you do not have a problem with the process being stopped by force.
- A message is output informing you that the timeout time was exceeded, but the server application process is not stopped. Care should be exercised if this pattern is used because the application is sometimes returned normally after this message is output. In addition, the output of the message from the same process does not occur until 10 minutes after output of the message following the first timeout.
- The thread dump is collected in the container information log (info.log). It is also output twice, once immediately after the timeout occurrence and then once again 10 seconds later. Because of this, a problem in an unchanged application that is run on a thread can be detected using 2 thread dumps.
- Configure the maximum processing time for the application in the Interstage Management Console WorkUnit settings.
- Processing for stopping the process by force occurs 10 seconds after the second thread dump is output. This prevents the process being stopped by force before the thread dump is output. How it works is that, 10 seconds after the message informing you that the timeout time was exceeded is output, the second thread dump is output. A further 10 seconds later, the process is stopped by force. For this reason, the process does not stop for at least 20 seconds.
- For this reason, even if the message is output the application might be returned normally until the process stops, and the process might be stopped by force after that.

Setting Maximum Processing Time of Application Using Interstage Management Console

Set "maximum processing time of application" on the WorkUnit definition screen or WorkUnit environment setup screen.

Transaction Application

- The application outputs a time-out message and forcibly terminates the server application process in which a time-out occurred. After forcibly terminating the process, the application returns with an error to the client.

Standard Setting of Server Application Timer

If the server application timer is to be used in the WorkUnit, the timer must be adjusted according to the client timer. The client timer function is as follows:

Client time-out

The client time-out is set in the operating environment file (config) in the CORBA service:

period_receive_timeout

This parameter specifies the time during which the client waits for a response after sending a request (issuance of server method). If no response is received from the server method within the specified time, a time-out is issued to the client application. Note that the value obtained by multiplying period_receive_timeout by 5 is the actual time in seconds.

The server application timer function and client timer function are used in separate roles as follows:

- The server application timer function mainly monitors delays in server application processing, such as server application loops.
- The client timer function is used to guarantee responses.

To prevent an invalid time-out in the client application, the wait time for response to a request (period_receive_timeout) must be set so that it satisfies the following expression:

$$\text{period_receive_timeout (T1)} > \text{queuing time} + \text{Maximum Processing Time (T2)}$$

For the maximum processing time of client WorkUnits when communication is performed between server applications, set the time obtained by adding 7 seconds or more to the maximum processing time of server WorkUnits.

When the maximum processing time is exceeded during application processing of server WorkUnits, it may take up to 7 seconds before Interstage actually detects the excess of the maximum processing time of the server and reports an error to the client.

Calculate the maximum processing time of client WorkUnits by considering the time during which a request stays in the server WorkUnit queue and the application processing time of client WorkUnits.

2.1.5 Current Directory

The work directory (current directory) in which a WorkUnit application runs can be specified.

Using the current directory enables the applications working under control of the WorkUnit to run in different work directories.

Current Directory Generation Management Function

Up to five generations of backup copies of the current directory of a WorkUnit can be maintained by setting up the "Number of Revision Directories" statement in the [Control Option] section of the WorkUnit definition.

Number of Revision Directories: 1

A backup copy is created when the WorkUnit is started, and it remains until it becomes older than the number of generations specified in the WorkUnit definition. The directory that was created when the WorkUnit was last started and has the same name as the WorkUnit is backed up as "workunit-name.old1." "workunit-name.old1" is backed up as "workunit-name.old2". Thus, this backup operation is repeated for each directory until the directory is renamed "workunit-name.oldn" where n is the specified number of generations. If "workunit-name.oldn" already exists, it is deleted and the directory one generation younger than "workunit-name.oldn" is renamed as "workunit-name.oldn" and preserved.

Even after the restart of the WorkUnit, files in the current directory, such as the standard output file (stdout), standard error output file (stderr), etc. (or core file for Solaris/Linux version) are saved. Therefore, in the past, after the occurrence of a problem, it was necessary to collect the files in the current directory as investigation data before restarting the WorkUnit. However, with the Current Directory Generation Management Function, the WorkUnit can be restarted so that priority is given to the recovery of regular operations, and, later, investigation data can be collected. Additionally, even if a long period elapses after the occurrence of a problem, the information for the investigation can easily be obtained.

Note

- The number of generations of the current directory to be backed up is 1 by default. Therefore, directories and files created the last time the WorkUnit was started necessarily remain in the directory specified by the current directory of the WorkUnit.

In consideration of the sizes of output files, secure sufficient free space in the disk for the directories specified by the current directory.

- The generation management function is valid for the following types of WorkUnits:

ORB, CORBA, UTY

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

- For the details of the current directory of the IJServer WorkUnit, refer to the "Environment where J2EE Applications are Operated (IJServer)" chapter in the J2EE User's Guide.
- According to the specified number of generations, estimate sufficient disk capacity for the current directory.
- For a CORBA WorkUnit, if "EXTP_CURRENTDIR_HISTORY=YES," (which specifies a compatible function of older versions), is included in the environment variables of the WorkUnit definition, five generations of backup copies of the current directory are maintained.
- A WorkUnit is backed up with the following current directory name in an extended system:

workunit-name.system-name.old*

Specification formats for CORBA applications and transaction applications

Table 2.1 Specification Formats for CORBA Applications and Transaction Applications

Platform	System	Specification format (*1)
Windows®	-	xxx\yyy\zzz
Solaris	Default system	xxx/yyy/zzz
Linux	Extended system (*2)	xxx/yyy.system-name/zzz

*1 xxx: Directory specified in the WorkUnit definition

yyy: Relevant WorkUnit name

zzz: Application execution process ID

*2 The extended system can be used only with Solaris.

The stdout file is allocated to standard output. It is used as the output destination of data placed on standard output by the server application.

The stderr file is allocated to standard error output. It is used as the output destination of data placed on standard error output by the server application.

According to the specifications of the OS standard I/O library, data placed on standard output and standard error output is put in the standard I/O library for buffering.

The standard output and standard error output must be flushed in order to securely output the buffered data to the stdout and stderr files.

When C language is used, for instance, issue `fflush(stdout)` and `fflush(stderr)`. Unless both standard outputs are flushed, the data buffered by the standard I/O library is not output to the stdout and stderr files.

In C++, use the standard C++ iostream library if the input/output stream (`cout`, `cerr`) is used to output data as standard output and standard error output. If an old iostream library is used, users must program standard output and standard error output to be mapped to the files. If a standard C++ iostream library is used, the `.h` file extension of `<iostream>` uses a header file that is not attached. Refer to information published by Microsoft® Limited, such as MSDN, for details of the differences between old iostream libraries and standard C++ iostream libraries.

When Java or COBOL is used, the stdout and stderr files are created but no data is output to the files.

When a COBOL application uses the DISPLAY statement, specify the compiler option SSOUT (environment variable information name) at compilation. Also, specify the file name of the data output destination in the environment variable information name of the Environment Variable: statement within the [Control Option] section or [Application Program] section in the WorkUnit definition. If the default is used as the output destination for the data, and the output of messages for which the `@WinCloseMSG` environment variable has been set to "OFF" is not stopped, the application may hang when the WorkUnit stops.

When Solaris/Linux is used, an application that terminates abnormally and outputs a core file will output it in the respective current directory.

Note

If Java is used to output data to files, the user must create a program so that standard output and standard error output are allocated to files.

If you don't want standard output and standard error output be allocated to the stdout and stderr files when a CORBA application is used, specify the following environment variable in the Environment Variable: statement of the [Control Option] section or [Application Program] section within the WorkUnit definition. No data will be output to the stdout and stderr files:

```
INTERSTAGE_WU_STDOUT_REDIRECTION = OFF
```

Setting Application Operation Current Directory in the Interstage Management Console

Set the "application operation current directory" on the WorkUnit definition screen or WorkUnit environment setup screen.

Notes on creating CORBA Applications and Transaction Applications using Microsoft® Visual C++.NET or Microsoft® Visual C++ 2005

For an application that is created using Microsoft® Visual C++.NET or Microsoft® Visual C++ 2005, data that is output to the standard output or standard error output is not output to the stdout or stderr file in the current directory.

To solve this problem and output data correctly, perform the following steps in the application:

Add the following code to the beginning of the program: (Note 1)

```
freopen("stdout", "w", stdout);  
freopen("stderr", "w", stderr);
```

Note 1: If Microsoft® Visual C++ 2005 is used for the build, the "warning C4996: 'freopen' was declared using an old format." warning may be output. This does not affect the program.

If a pre-exit program is to be used in a transaction application, add these statements to the beginning of the pre-exit program. If the code is added to the pre-exit program, the above processing need not be performed by the main routine or the post-exit program.

If a pre-exit program is not to be used, add the code to the beginning of the program and arrange the program so that the code is executed only once when the program is invoked for the first time.

2.1.6 Environment Variables

The environment variables used by the applications running under control of the WorkUnit can be defined explicitly. The environment variables that need to be reflected in application processes can be specified in the WorkUnit definition. Use environment variables if they are used by applications for database processing.

- Valid environment variables for application processes to be operated in the WorkUnit are described in both in the WorkUnit definition and in the Interstage runtime environment.

In addition, regarding environment variables specified in the Interstage runtime environment, the system environment variable is inherited in a Windows environment. In a Solaris or Linux environment, according to the start method of Interstage, different environment variables are inherited as follows:

When Interstage is started with the *isstart* command:

The environment variables specified in the *isstart* command runtime environment are inherited. In addition, if the operation is designed so that Interstage is automatically started using the system initialization script (*S99startis*) at the start of the server, the environment variables of the system initialization script runtime environment are inherited.

When Interstage is started from Interstage Management Console:

The environment variables specified in the Interstage JMX service runtime environment are inherited to Interstage. The Interstage JMX service is started with the *isjmxstart* command. Therefore, the environment variables specified in the *isjmxstart* command runtime environment are inherited. In addition, if the operation is designed so that Interstage is automatically started by the system initialization script (*S95isjmxstart*) at the start of the server, the environment variables of the system initialization script runtime environment are inherited.

- If the WorkUnit definition and Interstage runtime environment have the same environment variables, the settings of the former have priority.

The settings that are valid for individual environment variables are explained below:

- Environment variables other than PATH, LD_LIBRARY_PATH, and CLASSPATH

The environment variables described in the "Environment Variable" of the WorkUnit definition and those specified by the Interstage runtime environment are both valid.

- However, if the same environment variable names are specified, the one specified in the WorkUnit definition takes precedence.
- Also, if the same environment variable name is set in both the [Application Program] and [Control Option] sections, only the one specified in the [Application Program] section becomes valid.

LD_LIBRARY_PATH

The values of "Path" and "Library for Application" of the WorkUnit definition are specified before the value of LD_LIBRARY_PATH specified by the Interstage runtime environment.

Thus all environment variables are valid and given priority in the following order:

1. Value set for "Path" in the WorkUnit definition
2. Value set for "Library for Application" in the WorkUnit
3. Value set in the *isstart* command execution environment

Set the values for "Path" and "Library Application" in the WorkUnit definition and LD_LIBRARY_PATH in the *isstart* command execution environment so that the total path length does not exceed 16,384 bytes.

PATH

The value of "Path for Application" of the WorkUnit definition is specified before the value of PATH specified by the Interstage runtime environment.

Both values are valid but the value set in the WorkUnit definition is given priority.

However, configure the settings so that the total length of the paths set in "Path for Application" of the WorkUnit definition and set in PATH specified by the Interstage runtime environment will not exceed 4096 bytes.

CLASSPATH

CLASSPATH is valid in the CORBA WorkUnit. The variable is set in different ways for these two types of WorkUnits.

- CORBA WorkUnit

The value of "CLASSPATH for Application" of the WorkUnit definition is specified before the value of CLASSPATH specified by the Interstage runtime environment.

Note

The CLASSPATH set in the *isstart* command execution environment is not set as the java process argument (-classpath). The CLASSPATH set in the *isstart* command execution environment is ignored.

For this reason, set all classpaths, which are required for running EJB applications, for "CLASSPATH for Application" in the WorkUnit definition.

Setting CORBA WorkUnit Environment Variables Using the Interstage Management Console

Set CORBA WorkUnit environment variables using the Interstage Management Console as follows:

Environment variables other than PATH, LD_LIBRARY_PATH, and CLASSPATH

Environment variables can be set in the following two ways:

- Environment variable on the WorkUnit: New screen, or environment variable on the WorkUnit: Environment Setup screen
- Environment variable on the CORBA Application: Deployment screen, or environment variable on the CORBA Application: Environment Setup screen

If the same name is specified for the WorkUnit environment variable and CORBA application, the one specified in the CORBA application is valid.

The environment variable that was set during Interstage starting and the WorkUnit environment variable are both valid.

However, if the same environment variable name is specified, the value of the environment variable defined in the WorkUnit is valid.

LD_LIBRARY_PATH

Specify the library path in the detailed settings on the WorkUnit: New screen or WorkUnit: Environment Setup screen. The specified value is set before the value that has been set for LD_LIBRARY_PATH during Interstage startup.

A character string of up to 255 bytes, excluding null characters and n-size kana characters, can be specified per line. Up to 30 paths can be specified by using the linefeed character as delimiter.

PATH

Specify the path in the detailed settings on the WorkUnit: New screen or WorkUnit: Environment Setup screen. The specified value is set before the value that has been set for the environment variable PATH during Interstage startup.

A character string of up to 255 bytes, excluding null characters, can be specified per line. Up to 30 paths can be specified by using the linefeed character as delimiter.

CLASSPATH

Specify the classpath on the CORBA Application: Deployment screen or CORBA Application: Environment Setup screen. The specified value is set before the value that has been set for CLASSPATH during Interstage starting.

Specify one path with a character string of up to 255 bytes, excluding null. Up to 255 classpaths can be specified by delimiting them with a linefeed character.

Note

When the Interstage Management Console is used to start Interstage, Solaris/Linux takes over the environment variables that are used when the *ismngconsolestart* command or *isjmxstart* command is executed, and Windows takes over system environment variables.

For each of the environment variables PATH, and CLASSPATH that are set when the WorkUnit application starts, the total path length of the value set in the WorkUnit definition, and the value that has been set in the Interstage start environment, may exceed the following number of bytes:

PATH: 16,384 bytes

LD_LIBRARY_PATH: 16,384 bytes

CLASSPATH: 65,536 bytes

If a value exceeds the limit, the excess part is invalid.

2.1.7 Queue Control

Multiple clients may issue requests to the same application (object). Starting the application to concurrently process all requests places excessive load on the server.

To prevent this problem, use queue control to average the load.

Under queue control, the requests from the client are entered in the queue for the relevant application and processed sequentially by the application.

The queue unit varies depending on the application type.

CORBA Application

A queue is created for each implementation defined in the implementation repository.

A queue exists for each implementation repository and the requests for an interface are connected to the queue of the corresponding implementation repository. For this reason, if the application defined in the implementation repository includes multiple interfaces, the multiple interfaces share one queue.

This queue is also used when a CORBA application is operated without using a WorkUnit.

Note

Queue control using the *odsetque* and *odcntlque* commands is disabled for CORBA WorkUnits.

odsetque is not valid for Linux (64 bit).

Setting Classpath Using Interstage Management Console

Specify the classpath on the CORBA Application: Deployment screen or CORBA Application: Environment Setup screen.

Transaction Application

A queue is created for each object specified in the WorkUnit definition.

When the application consists of concurrent processes, the Least Recently Used (LRU) algorithm is used for request assignment to individual processes. This algorithm is used to minimize resource consumption by unnecessary processes. The round-robin algorithm can also be specified for CORBA applications and transaction applications. The assignment algorithms can be specified in the WorkUnit definition.

The round-robin algorithm enables using concurrent processes as evenly as possible. Note that requests are not completely equally assigned because they are assigned in a round-robin fashion to each process that can accept a request at the respective time.

- LRU algorithm

Of the server application processes waiting for requests, the process that last entered the request wait state is assigned a request from the client.

When a small number of requests are received from the client, request messages are assigned to only specific processes.

Specify LIFO for "Request Assignment Mode" in the WorkUnit definition.

- Round-robin algorithm

Of the server application processes waiting for requests, the process that first entered the request wait state is assigned a request message from the client.

When a small number of requests are received from the client, request messages are assigned evenly to individual processes.

Specify FIFO for "Request Assignment Mode" in the WorkUnit definition.

When using the process bind function for transaction applications, use the round-robin algorithm. The round-robin algorithm assigns a session start request evenly to each process and thus prevents performance deterioration by session concentration on one process.

When the process bind function is not used, use the LRU algorithm (default). However, use the round-robin algorithm to evenly assign request messages from the client to server application processes.

The LRU and round-robin algorithms differ as follows:

The LRU algorithm assigns a request message to the process that ran last, and the round-robin algorithm assigns a request to the process that ran first among other processes waiting for requests. Thus, it is highly probable that the process to which a request is assigned by the LRU algorithm still exists in memory, while a process to which a request is assigned by the round-robin algorithm may have been swapped out. The LRU algorithm may thus have better communication capability.

For more information about queue control, refer to "Queue Control" in "2.2 Various Types of WorkUnit".

2.1.8 Inhibiting and Permitting Queues

Each queue can be inhibited from, or permitted to, temporarily reject requests from the client, or restart accepting requests. The queue inhibit/permit function is used to:

- Restrict the use of a job according to the time zone.
- Temporary reject requests because the load is too high.
- Suppress requests before stopping a job so that the job stops after processing is completely finished.

If the client issues a request while the queue is inhibited, an error is reported to the client. The queue can be inhibited in the following units:

CORBA Application

- For each implementation
- For each interface installed by implementation

To inhibit the queue for a specific interface, define the interface to be inhibited in the implementation repository definition and then start the WorkUnit.

Setting Inhibit/Permit Queuing Using the Interstage Management Console

The Interstage Management Console enables closure/closure release using the CORBA Application: Status screen.

IJServer Application

- For each IJServer Work Unit

Setting Inhibit/Permit Queuing Using the Interstage Management Console

In the Interstage Management Console, it is possible to block/unblock using the IJServer WorkUnit : Operate window.

Transaction Application

- For each object specified in the WorkUnit definition

The *isinhbitque* command is used to inhibit the queue and the *ispermitque* command is used to permit it.

The inhibit/permit function is enabled only when the relevant application runs under control of a WorkUnit. When Solaris/Linux is used, only the user who starts the WorkUnit or the superuser can use the inhibit/permit function. Inhibiting or permitting queues can be performed using the *isinhbitque* or *ispermitque* command or Interstage operation API.

2.1.9 Maximum Queuing Function

The maximum number of requests that can be queued can be specified explicitly with this function. This can restrict requests from the client when the load placed on the server application exceeds a given level.

If the number of requests issued by the client exceeds the specified maximum queue number, an error is reported to the client.

The request being presently processed by the server application is not included in the number of queued requests. Specify the maximum queuing function in the WorkUnit definition.

The maximum queuing function can be used for CORBA applications, IJServer EJB applications, transaction applications, and wrapper WorkUnits (Windows® and Solaris version wrapper WorkUnits only).

For the errors reported to the client for CORBA, refer to the "Exception Information and Minor Codes Reported from the CORBA Service" chapter in the Messages manual.

Setting Maximum Queuing on the Interstage Management Console

The maximum number of messages that can be queued for CORBA applications can be set using "maximum number of queued messages" on the CORBA Application: Deployment screen or CORBA Application: Environment Setup screen of the Interstage Management Console.

The maximum queuing number for an IJServer EJB application can be set in "Maximum Number of Queuing Messages" of the Interstage Management Console WorkUnit : Create New window, or the IJServer WorkUnit : Environment Settings window.

2.1.10 Alarm Report Function for Stagnant Queues

An alarm is reported when the number of stagnant queues left unprocessed exceeds a specified number of queues to be monitored. Monitoring restarts when the number of stagnant queues is reduced to the specified number of queues.

This function thus enables real-time monitoring of the system load status.

The stagnant queue monitoring function can monitor the following three states:

- The number of stagnant queues exceeds the maximum queue number.
- The number of stagnant queues reaches a specified queue number (any).
- The number of stagnant queues reaches a specified monitor restart queue number (any).

Users can specify which state to monitor in the WorkUnit definition.

Working with Systemwalker CentricMGR enables centralized monitoring for a certain number of stagnant queues.

When the number of stagnant queues subjected to monitoring has reached the relevant state, state transition information is displayed on the screen of the Interstage CentricMGR operation control client.

This function thus enables real-time monitoring for state transition of the number of stagnant queues.

Refer to the "Performance Monitoring" chapter in the Operator's Guide for details of the linkage with Systemwalker CentricMGR.

The alarm report function for stagnant queues can be used for CORBA and IJServer EJB applications (running under control of the relevant WorkUnit), transaction applications, and wrapper WorkUnits (Windows® and Solaris version wrapper WorkUnits only).

Even when the application does not work with Systemwalker CentricMGR, a message is output when the respective event occurs. The number of stagnant queues can thus be checked by monitoring the respective message.

The following table lists the messages output for individual events. Refer to the Messages manual for more information.

Table 2.2 Messages Output for Individual Events

Type of application	Event	Output message
CORBA application	The number of stagnant queues exceeds the maximum queue number.	od11108
	The number of stagnant queues reaches the specified queue number (any).	od11107
	The number of stagnant queues reaches the monitor restart queue number (any).	od11109
IJServer EJB application	The number of stagnant queues exceeds the maximum queue number.	od11108
	The number of stagnant queues reaches the specified queue number (any).	od11107
	The number of stagnant queues reaches the monitor restart queue number (any).	od11109
Transaction application	The number of stagnant queues exceeds the maximum queue number.	td12034
	The number of stagnant queues reaches the specified queue number (any).	td12033
	The number of stagnant queues reaches the monitor restart queue number (any).	td12035
Wrapper WorkUnit	The number of stagnant queues exceeds the maximum queue number. (*1)	td12034
	The number of stagnant queues reaches the specified queue number (any).	td12033

Type of application	Event	Output message
	The number of stagnant queues reaches the monitor restart queue number (any).	td12035

*1 To use the maximum queue number for a wrapper WorkUnit, specify "Number of Maximum Session" in the component transaction service environment definition.

Setting Maximum Queued Messages Using the Interstage Management Console

The maximum number of queued messages that are subject to monitoring and the number of queued messages for restarting monitoring can be set on the CORBA Application: Deployment screen or CORBA Application: Environment Setup screen of the Interstage Management Console. Use "maximum number of queued messages under monitoring" and "number of queued messages for restarting monitoring" on the screen.

The monitor queuing number and monitor restart queuing number for IJServer EJB applications can be set in the Interstage Management Console WorkUnit : Create New window or the IJServer WorkUnit : Environment Settings window. Use "Number of Monitor Queuing Messages" or "Number of Monitor Restart Queuing Messages" as appropriate.

2.1.11 Priority Control

Priority can be assigned among the objects that constitute a job.

This function is useful when one job needs to be processed prior to others, for example when the CPU load is too high. Priority levels 1 to 255 can be assigned.

Priority control can be performed for CORBA applications and transaction applications.

For CORBA applications, priority can be assigned to the interfaces within the same implementation. Individual priority levels can be specified in the implementation repository definition.

Refer to "Priority Order Control" in "CORBA Application Queue Control" in the "WorkUnit Operation of Each Application" chapter for information on how to specify the priority in the implementation repository definition using the *OD_impl_inst* command.

For transaction applications, a priority can be assigned among applications within the same WorkUnit. Individual priority levels can be specified in the WorkUnit definition. Refer to "Operation Using the Object Priority Function" in "Operating Transaction Applications WorkUnits" in the "WorkUnit Operation of Each Application" chapter for information regarding priority control over transaction applications.

Setting Priority Control Using Interstage Management Console

Set "priority" on the CORBA Application: Deployment screen or CORBA Application: Environment Setup screen.

2.1.12 Queued Message Cancellation Function

Queued requests can be canceled and an exception posted to the client. This function is enabled in the following cases:

- The load is high, and it is estimated that that sending a reply for the queued request before the client times out will not be possible
- The load is high, and it can be decreased canceling a queued request for a low priority application in the WorkUnit

Queued requests can be canceled in the following units:

CORBA Applications

- For each implementation

Queued requests are canceled using the *iscancelque* command. They cannot be canceled using the Interstage Management Console.

The cancellation function is enabled only when the relevant application runs under control of a WorkUnit. When Solaris/Linux is used, only the user who starts the WorkUnit or the super user can use the cancellation function.

To cancel all queued requests that belong to a WorkUnit, stop the WorkUnit.

2.1.13 Buffer Control

Request data from the client is temporarily held in shared memory while it is queued.

The data area in the shared memory is referred to as a communication buffer, and the way of controlling the buffer area is referred to as buffer control. The communication buffer consists of several buffer areas, each of which holds request data. The size of the buffer is called the buffer size and the number of buffers is called the buffer number.

2.1.13.1 Buffer Control for CORBA Applications

A buffer of a given size (4096 bytes) is prepared for copying a piece of request data. Request data is copied to the buffer area and then queued. If the length of request data is equal to or less than the given length, the data can be accommodated in the buffer area. All for the request data is copied to the buffer area and is then entered into a queue (*1). The queue used is created for each implementation defined in the implementation repository. Refer to "CORBA Application" in "2.1.7 Queue Control". If buffer tuning is performed in the WorkUnit definition, a buffer area is created for each implementation. If buffer tuning is not performed, the default buffer is shared as a buffer area. Note that buffer tuning cannot be performed for CORBA applications that do not run in WorkUnits. The default buffer is shared as a buffer area for CORBA applications that do not run in WorkUnits. In addition, the buffer cannot be tuned for a CORBA application that does not perform the WorkUnit operation. A CORBA application that does not perform the WorkUnit operation shares the default buffer as a buffer area.

From a performance viewpoint, it is preferable that request data be fully accommodated in the buffer area and then queued. In the case of a job using many requests with data exceeding the given length (4096 bytes), the user can tune the buffer size and buffer number for the job. Do not to make the buffer size and buffer number too large, it will consume too much shared memory space and adversely affect the operation efficiency. A reasonable buffer size and buffer number must be defined for each job.

Buffer tuning is valid for CORBA applications running in WorkUnits. Buffer tuning can be performed for each CORBA application by specifying the "Buffer Number" and "Buffer size" statements in the WorkUnit definition. The tuned buffer uses a shared memory separate from the one used for the default buffer. For details of the definition, refer to "Buffer Number" and "Buffer Size" below.

If the buffer number is not tuned in the WorkUnit definition, the default buffer area is used. The default buffer area is shared for CORBA communication excluding the CORBA applications for which the buffer number is tuned in the WorkUnit definition. The default buffer normally does not require tuning. However, when message od11110 or od11111 is output, the number of default buffers may be too small. In this case, tune the buffer number by modifying the number_of_common_buffer parameter in the CORBA service operating environment file (config).

*1 If the size of request data exceeds the given length, the excess data is received when the server application executes processing. If a client in a multithread configuration issues multiple requests simultaneously, the next request is not processed until the data for the first request is read. Moreover, if a large amount of data is requested, the client application process for writing to a socket may time out and the error message od10965 may be output.

Buffer Size

Each request data item includes a data part that is actually set by the user and a control data part used by Interstage.

For CORBA applications, the validity of the buffer size can be checked by referring to the length of data collected for each operation by the performance monitoring tool.

Buffer Number

The buffer number needs to be determined from the number of concurrent requests from the client.

As many buffers are required as there are client requests concurrently enqueued. Determine the number of requests that are concurrently enqueued based on the application concurrency, application processing time, and the number of requests concurrently issued by the client.

The validity of the buffer number can be checked by referring to the number of accumulated queues obtained by the *isinfobj* command and the number of requests waiting for processing obtained by the performance monitoring tool. Refer to the "Performance Monitoring" chapter in the Operator's Guide for information on the performance monitoring tool.

Buffer Tuning Using the Interstage Management Console

Buffer tuning in units of CORBA applications can be performed by setting "number of communication buffers" and "communication buffer length" on the CORBA Application: Deployment screen or CORBA Application: Environment Setup screen.

2.1.13.2 Buffer Control for Transaction Applications

Buffer areas, each consisting of a given size (4096 bytes), are prepared for transaction applications. Request data that exceeds the given size is copied to two or more combined buffer areas and then enqueued. Transaction applications are used for general online processing. Therefore, do not use them for transmission of large amounts of data, such as in file transfer. The number of buffers can be tuned for transaction applications. It can be changed in the environment definition for the component transaction service.

Refer to the Tuning Guide for the environment definitions of the component transaction service.

2.1.13.3 Buffer Control for IJServer EJB Applications

The size of the buffer area for copying one item of request data is set at a length of [4096] bytes. This is used for copying and queuing request data. When the request data length does not exceed this limit, it can fit in the buffer area, and is copied and queued (*1).

If the buffer is tuned, a buffer area is created in the WorkUnit definition for each IJServer. If the buffer is not tuned, the default buffer is used.

The fact that the request data fits in the buffer area and can be queued enhances processing efficiency. If the number of requests means that the request data length exceeds [4096] bytes, the buffer length and number can be tuned by the user.

When the buffer length and number are such that they require the use of a large amount of common memory however, it is inefficient. For this reason, set an optimum buffer length and number for each business.

If the buffer number is not tuned in the WorkUnit definition, the default buffer area is used. If the buffer length and number are tuned, CORBA communication is shared except for IJServer EJB applications and CORBA applications.

Normally, there is no need to tune the default buffer. If od11110 or od11111 is output, however, it is possible that the default buffer size is insufficient. In this case, change the number_of_the_common_buffer parameter in the CORBA service operating environment file (config).

*1 If the size of request data exceeds the given length, the excess data is received when the server application executes processing. If a client in a multithread configuration issues multiple requests simultaneously, the next request is not processed until the data for the first request is read. Moreover, if a large amount of data is requested, the client application process for writing to a socket may time out and the error message od10965 may be output.

Buffer Size

One item of request data contains the control data part set by the actual user and the control data part used in Interstage.

The data length for each operation collected using the Performance Monitoring Tool can be referenced to check the validity of the buffer length.

Buffer Number

The buffer number depends on the number of simultaneous requests from the client.

Requests from the client must be queued simultaneously. The number depends on the number of simultaneous requests from the client as well as the application concurrency and processing time.

The validity of the buffer number can be checked by using the *isinfobj* command to refer to the number of messages in the queue and the number of processing wait requests collected using the Performance Monitoring Tool. Refer to the "Performance Monitoring Tool" chapter for details.

Buffer Tuning Using the Interstage Management Console

The buffer number and length for the IJServer EJB application can be set in 'Buffers' and 'Buffer length' in the Interstage Management Console WorkUnit: New window, or in the IJServer WorkUnit: Environment Settings window.

2.1.14 Degenerated Operation when the Automatic Restart of a WorkUnit Process Fails

When a process is forcibly stopped because an application has terminated abnormally or the maximum application processing time has elapsed, the process restarts because of the automatic process restart function. At this time, if the process restart fails due to an application preprocessing error or application preprocessing timeout, the WorkUnit terminates abnormally.

For a WorkUnit in which two or more applications are operating concurrently, a function to avoid this situation is supported. Even if a process fails to restart, this function continues the WorkUnit operation of the remaining processes, after outputting the message to indicate the failed restart of the process.

This function is valid for a CORBA WorkUnit and IJServer WorkUnit.

In this function, when the number of process concurrency in a WorkUnit is two or more, even when the automatic restart of the application process fails, the operation is continued in the state where the number of process concurrency is decreased by one. Also, the function to restore the degenerated application process is provided.

To use this function for a CORBA WorkUnit, set "YES" to the "Process degeneracy" statement in the "Control Option" section of the WorkUnit definition.

For an IJServer WorkUnit, in the [Environment Settings] window of Interstage Management Console, select [Continue WorkUnit Operation] from the [Control When the Automatic Restart of an Application Fails] items.

2.1.14.1 Restoration of the WorkUnit in Degenerated Operation

A function is provided to restore the WorkUnit whose operation is degenerated. This function performs the restart of processes and restores to the original process concurrency number the process concurrency that has been reduced due to the failed automatic restart of processes. If the number of process concurrency is dynamically changed, it will be restored to the changed number of process concurrency.

The generated WorkUnit is restored at the following opportunities:

Recovery by the operation of the command (*isrecoverwu*)

Recovery by the operation of Interstage Management Console

2.1.15 Changing a WorkUnit's Number of Process Concurrency

The number of process concurrency of server applications can be changed without stopping a WorkUnit in operation or changing the WorkUnit definition.

For details, refer to "Dynamic Changing of the Number of Server Application Processes" in the "Starting / Stopping / Surveillance of WorkUnits" chapter.

This function is valid only for a CORBA WorkUnit and the WorkUnit of a transaction application.

Note this function is supported only for the Enterprise Edition.

2.2 Various Types of WorkUnit

Interstage Application Server supports the following types of applications for operation under control of WorkUnits:

- CORBA application (CORBA WorkUnit)
- Transaction application
- J2EE application (IJServer WorkUnit)
- Wrapper object
- General application (application not under control of Interstage Application Server)

Of these WorkUnits, the IJServer WorkUnit and CORBA WorkUnit that use CORBA applications can be defined using the Interstage Management Console.

Using the Interstage Management Console to operate CORBA WorkUnits makes it easy to define WorkUnits and initialize application environments. It also enables centralized management of operation and status referencing.

2.2.1 CORBA WorkUnit

CORBA applications (server applications) are supported for WorkUnit operation. A WorkUnit running CORBA applications is referred to as a CORBA WorkUnit.

The Interstage Management Console, or WorkUnit management commands, can be used for CORBA WorkUnit definition.

Fujitsu recommends using the Interstage Management Console to operate CORBA WorkUnits.

2.2.1.1 Using WorkUnits

This section explains CORBA WorkUnits.

Application Execution Unit of CORBA WorkUnits

A CORBA WorkUnit uses the CORBA application registered in the implementation repository as its execution unit. Specify the implementation repository ID as the identifier of a CORBA application in the WorkUnit definition. For CORBA applications using the same operation unit (starting and stopping simultaneously), multiple implementation repository IDs can be specified for one WorkUnit.

Applicable CORBA Applications

Only a CORBA application with the server type "persistent" can run as a CORBA WorkUnit. (type=persistent of *OD_impl_inst*)

CORBA Application Operation Mode

"SYNC_END" can be used as an operation mode of the CORBA WorkUnit. In this mode, if the process returns from CORBA_BOA_impl_is_ready when the WorkUnit stops, the process must always be terminated.

If the operation mode is "SYNC_END" when the WorkUnit is stopped in normal stop or synchronous stop mode, the process returns from CORBA_BOA_impl_is_ready, but WorkUnit termination cannot be performed until the process is finished. For this reason, the process must always be terminated if it returns from CORBA_BOA_impl_is_ready.

2.2.1.2 CORBA WorkUnit Operation Functions

This section explains the major functions for CORBA WorkUnit operation.

Application Process Concurrency

If multiple clients are expected to issue requests simultaneously to one server application, the number of server application processes that can concurrently be processed (process concurrency) can be defined. Specify the process concurrency in the WorkUnit definition.

The amount of process concurrency should be determined according to the processing time per process, time required for response to clients, and the number of requests per time unit.

An appropriate process concurrency must be determined because setting it too high will adversely affect system resources, including memory.

Note

If WorkUnit management commands are used to define CORBA applications, the process concurrency must be defined not only in the WorkUnit definition but also in the implementation repository definition.

Note that the process concurrency in the WorkUnit definition must be lower than that in the implementation repository definition.

Setting Process Concurrency on the Interstage Management Console

When the Interstage Management Console is used to define a CORBA application, set 'process concurrency' on the application deployment screen.

If the process concurrency is set on the Interstage Management Console, it will not need to be set in the implementation repository definition.

Automatic Restart of Applications

If an application terminates abnormally due to input of invalid data from the client, the application can be automatically restarted. Thus, if automatic restart is defined, requests accepted from the client while the application was stopped can be processed automatically.

Automatic application restart can be specified in the WorkUnit definition. The value to be set is the number of consecutive abnormal terminations by which the WorkUnit terminates abnormally.

The number of consecutive abnormal terminations is the number of times the abnormal termination and restart sequence is repeated. The WorkUnit terminates abnormally when the specified number of consecutive abnormal ends is reached. At that point the WorkUnit

terminates abnormally, the application processes in the WorkUnit are all stopped and the request being processed in the WorkUnit returns with an error to the client.

Setting the Retry Count on the Interstage Management Console

When the Interstage Management Console is used to define a CORBA application, set the number of consecutive abnormal ends in the parameter "retry count" on the WorkUnit definition screen.

Timer Monitoring Function

During WorkUnit operation, the time between the calling of the CORBA application was and its return (its "response time") can be monitored. This monitoring function monitors application hangs and processing delays to prevent delays in responding to the client. Specify the monitoring time in the WorkUnit definition (Maximum Processing Time).

Whether to forcibly stop the application at occurrence of a time-out can also be specified in the WorkUnit definition (Terminate Process for Time out).

- A time-out message is output in this case and the application process is forcibly terminated. The following exception is returned to the client:

System exception	Minor code (in hexadecimal notation)
UNKNOWN	0x464a0072
	0x464a0872

- A time-out message is output but the application process is not terminated.

If the timer monitoring time is defined for the WorkUnit, notice the setting in the config file of the CORBA service. Refer to "[Timeout Monitoring during Operation of CORBA Applications](#)" for more information.

Queue Control

If requests from multiple clients are concentrated on a specific CORBA application and all requests are processed concurrently, the server may become overloaded, which can cause processing delays and frequent time-outs. The queue control function can average the load to prevent this problem. Refer to "CORBA Application Queue Control" in the "WorkUnit Operation of Each Application" chapter for more information.

2.2.2 Processing with Transaction Applications

This section explains the procedure to create transaction applications. Transaction linkage is broadly divided into two types:

Local Transaction Linkage

Each application links with only one database. A transaction is restricted to one server, and controlled on the server-side.

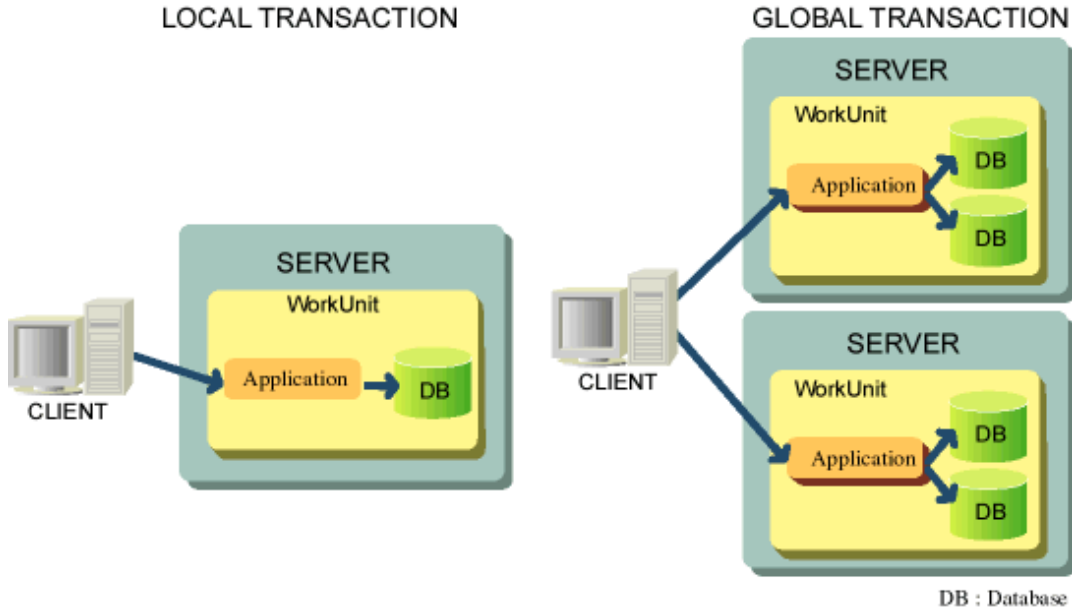
Global Transaction Linkage

An application links with multiple databases, and updates can be carried out simultaneously. The transaction can be controlled either on the client workstation or on the server.

Linkage is also possible with databases on separate server systems. Therefore, you can have a more flexible operational design because you can freely distribute resources such as transaction applications and databases. Global transactions are based on an XA interface specified by X/Open.

The following figure shows the operation of Local Transaction Linkage and Global Transaction Linkage.

Figure 2.1 Local and Global Transaction Linkage



Apart from operations involving CORBA applications, transaction-related operations are necessary in order to use an external transaction application.

[2.2.2.1 Application Runtime Environments Using WorkUnits](#) explains the tasks relating to the use of transaction applications. First, the tasks which you need to understand before performing Transaction Linkage are discussed. These include:

- Application execution environments that use WorkUnits
- Creating server applications that use APM
- Server object registration

Then, the tasks which are required depending on the transaction mode are discussed. These include:

- Local transaction linkage
- Global Transaction Linkage
- Session information management
- Linkage with multiple databases from one WorkUnit

The management of session information in transaction applications is also discussed.

Note

The number of objects that can be registered in a WorkUnit for each system scale must satisfy the following formula.

[Conditional expression]

Number of objects that can be registered = Internal limit for each system scale - number of transaction application WorkUnits - number of WorkUnits that use multiobject residence

[Internal limit]

small: 202

moderate: 402

large: 602

super: 1002

Example:

The maximum number of WorkUnits that can be started if there is one object for one WorkUnit is as follows:

small: 101

moderate: 201

large: 301

super: 501

2.2.2.1 Application Runtime Environments Using WorkUnits

The following operations are possible using a WorkUnit:

- You can separately control the environment necessary for executing the application. An application can be developed independently from the execution environment just by adding an object to the WorkUnit.
- Batch control of operations is possible by assembling into one WorkUnit objects that are to be combined by the operation. By starting or stopping a WorkUnit, you can group all objects defined in the WorkUnit together so that they can or cannot be accessed from a client.
- Applications are loaded into memory by starting the WorkUnit, and application pre-processing is executed. The main processing is carried out immediately in response to a client request during the operation. Pre-processing and post-processing can be carried out separately from the main processing. Post-processing is carried out when the WorkUnit terminates. Pre-processing (pre exit program) and post-processing (post exit program) are invoked for each process. Refer to "[2.2.2.2 Creating Server Applications that Use APM](#)" for details on the application processing mechanism.

The main issues to consider in defining and using the WorkUnit are:

Multi-control

If requests are simultaneously output from multiple clients to one application and multi-control is enclosed, then multiple execution units (process/es) are used to process the requests.

Queue Control

If a request is simultaneously output from multiple clients to one application, loading can be balanced by queue control.

Resident, Non-resident, and Multi-object Resident Operation of Applications

The client request can specify whether the application that is loading should be resident or non-resident. You can use system resources efficiently by using them differently according to the nature of the client's request.

You can select a resident operation in which one application is loaded in one process or a multi-object resident operation in which multiple applications are loaded in one process.

Application Timeout Monitor

You can specify the maximum processing time (timeout) of an application and monitor it.

Exit Program Timeout Monitor

You can specify the maximum processing time (timeout) of a pre exit and post exit programs to avoid loops and then monitor them.

Automatic Re-start at the Time of Application Error

You can re-start the application automatically if it terminates abnormally due to an input data error from the client. By setting automatic restart, client requests that have been received during the failure can be processed normally after restarting.

Getting Snapshots

The log of I/O information in response to requests from the client can be collected. From the WorkUnit file, you can obtain log information from start-up to termination of the WorkUnit for the purpose of debugging. Log information is output to the snapshot output path specified in the WorkUnit definition.

Output Folder

You can specify the current folder when an application, pre-processing exit program, or post-processing exit program is running. The standard output and standard error output are directed to this folder:

```
Output folder: xxx\yyy\zzz
xxx: Folder specified in the WorkUnit definition
yyy: Target WorkUnit name
zzz: Application execution process id
```

The stdout file is allocated to the standard output. This file is used as the output destination when data is output to the standard output by a server application.

The stderr file is allocated to the standard error output. This file is used as the output destination when data is output to the standard error output by a server application.

When the application terminates abnormally, core image files are output to this folder. When the snapshot output path is omitted, snapshots are also output to this folder.

If a WorkUnit exit or process salvage exit program is used, the exit program runs in the following folder:

```
Operation folder: xxx\yyy\zzz_exit
xxx: Folder specified in the WorkUnit definition
yyy: Target WorkUnit name
zzz: Execution process id of the exit program
```

The standard output file and standard error output file are output to the above folder.

The output folder is created when a WorkUnit is started, and is retained even after the WorkUnit is stopped. When the WorkUnit is restarted, the existing output folder will be deleted for each file under the folder and a new output folder will be re-created. However, if the output folder for the WorkUnit definitions is changed after the WorkUnit is stopped, the output folder created when the WorkUnit was started last time will not be deleted even when the WorkUnit is restarted. In this case, manually delete the unnecessary output.

Deleting Output Folders

You can specify that the output folder for a server application be automatically deleted when the application is stopped.

The output folder will be deleted when the server application stops because the WorkUnit is stopped or dynamically modified, or the number of concurrent processes is changed. This ensures that the output folders for the stopped server application do not remain on disk to take up excessive disk space.

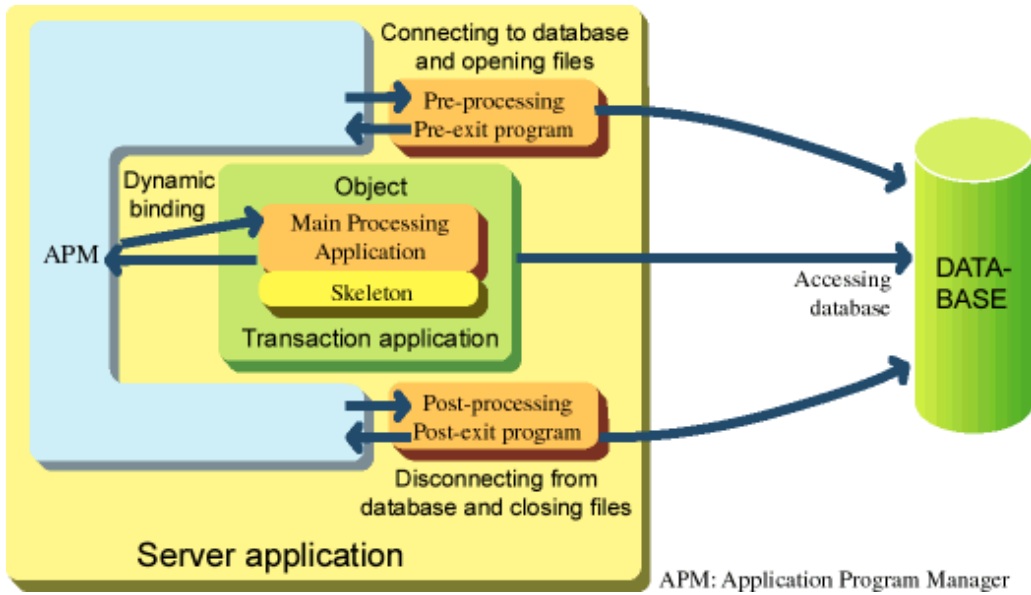
However, if files with a file size of 1 or more exist in the output folder, the output folder will not be deleted. Check whether these files are necessary, and manually delete any unnecessary files. In this case, the output folder will also be deleted and re-created when the application is restarted.

2.2.2.2 Creating Server Applications that Use APM

When carrying out transaction linkage, you can create a server application by using APM (Application Program Manager). A server application is executed by linking it dynamically with the APM.

The following figure shows the APM structure.

Figure 2.2 APM Structure



Ordinary applications are normally comprised of three processes:

Pre-processing

This carries out connection processing to the databases used and file opening processing.

This processing

This carries out database access processing.

Post-processing

This carries out disconnection from the databases used and file closing processing.

The pre-processing and post-processing required depended on the database processing system used. The pre-exit program and the post-exit program can be specified using APM. Therefore, server application developers only develop the main processes, and development efficiency is enhanced.

Create an APM for each database management system that you use, and specify it in the WorkUnit definition. However, you do not need to create an APM in the case of local transactions. Specify the APM name used for local transactions in the WorkUnit. Additionally, because only one APM can be specified for one WorkUnit, it is necessary to divide the WorkUnit into two or more WorkUnits if two or more database management systems are to be used.

2.2.2.3 Server Object Registration

The operations carried out automatically at the time of start-up if transaction linkage is carried out are:

- Server application start-up
- Object reference generation and registration to the Naming Service

However, registration to the server object Naming Service can also be carried out manually.

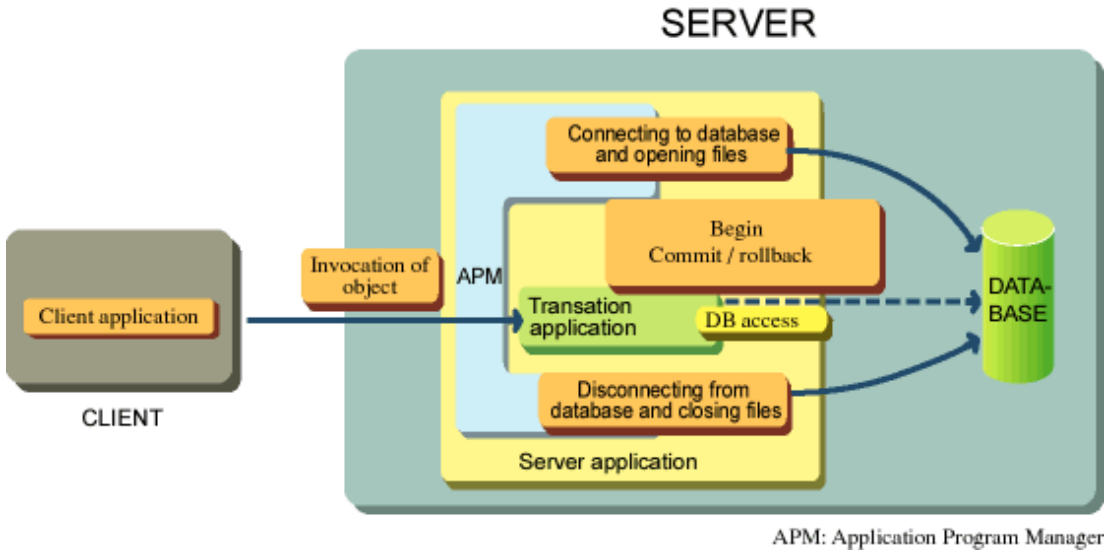
2.2.2.4 Manual Registration of Server Objects

1. Specify in each WorkUnit definition that you wish to register the object manually with the Naming Service.
2. Register the object to the Naming Service with the *OD_or_adm* command before starting the targeted WorkUnit. You need to have started Interstage at this time.

2.2.2.5 Local Transaction Linkage

Carry out local transaction linkage after specifying the transaction instruction in the server application. The following figure shows local transaction linkage:

Figure 2.3 Local Transaction Linkage



Creating a Client Application

Create a client application in the same way as creating a normal CORBA client application on the client-side, as the transaction instruction is not specified in the client application. No special steps are needed for local transaction linkage on the client-side.

Creating a Server Application

Create an application that includes a transaction instruction for the database management system being used, and the additional functions necessary for using the database management system (such as database connection and disconnection processes).

WorkUnit Definition

In addition to the normal WorkUnit definition, the following table shows the WorkUnit definition items that are necessary for local transaction linkage:

Table 2.3 WorkUnit Definition Items Necessary for Local Transaction Linkage

Definition item		Set content
APM	Name	TDNORM
Control option	Environment variable	Environment variables used when applications and exit programs operate
	Application library path	Folder and path where the application and exit program executable files are stored.

2.2.2.6 Global Transaction Linkage

Global Transaction Linkage enables simultaneous updates to linked multiple databases and maintains the integrity of each database. Global Transaction Linkage supports databases stored on other server systems.

CORBA compliant transaction management is carried out in the distributed object environment by the Database Linkage Service, which is comprised of two methods:

OTS System

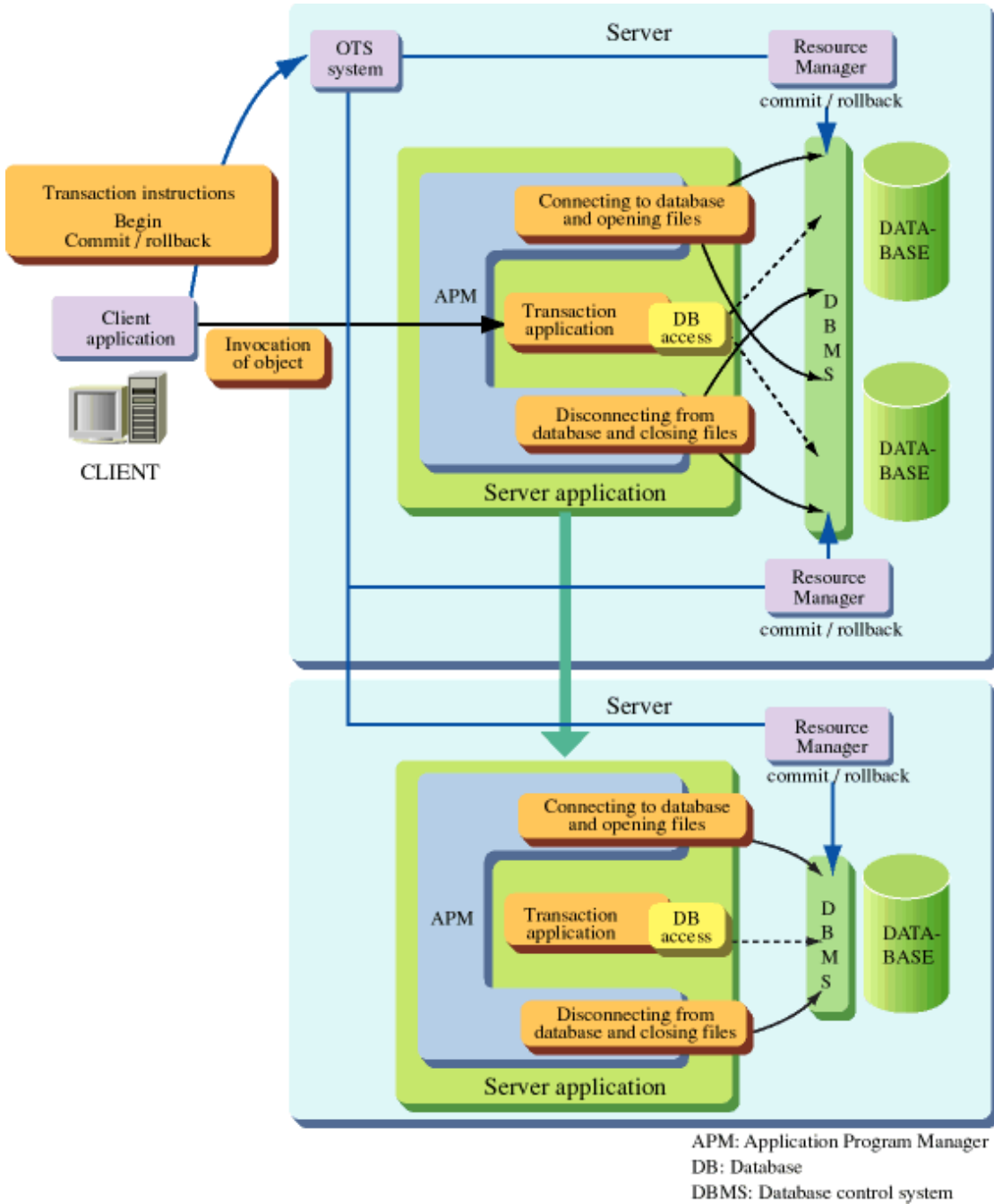
This carries out transaction management for the entire OTS system. The system manages transaction requests from the client and transaction recovery processing.

Resource Management Program

This is the program for executing transaction completion processing (commit/rollback). *commit* stores data that has been updated during the transaction. *rollback* restores to its pre-update status.

The following figure shows examples of Global Transaction Linkage in the case where transactions are controlled on the client-side. The database is maintained in multiple server machines.

Figure 2.4 Global Transaction Linkage Examples



Global transaction linkage procedures are:

- [Creating a Client Application](#)
- [Creating a Resource Management Program](#)
- [Creating a Server Application](#)
- [WorkUnit Definition](#)

You can control transactions on the client or server side when Global Transaction Linkage is used.

Creating a Client Application

This section explains the procedures necessary to use Global Transaction Linkage. The steps are:

1. Program specification

This specifies the process globally from the start to termination of the transaction, in order to request the transaction process in the database linkage service.

2. Development of load modules

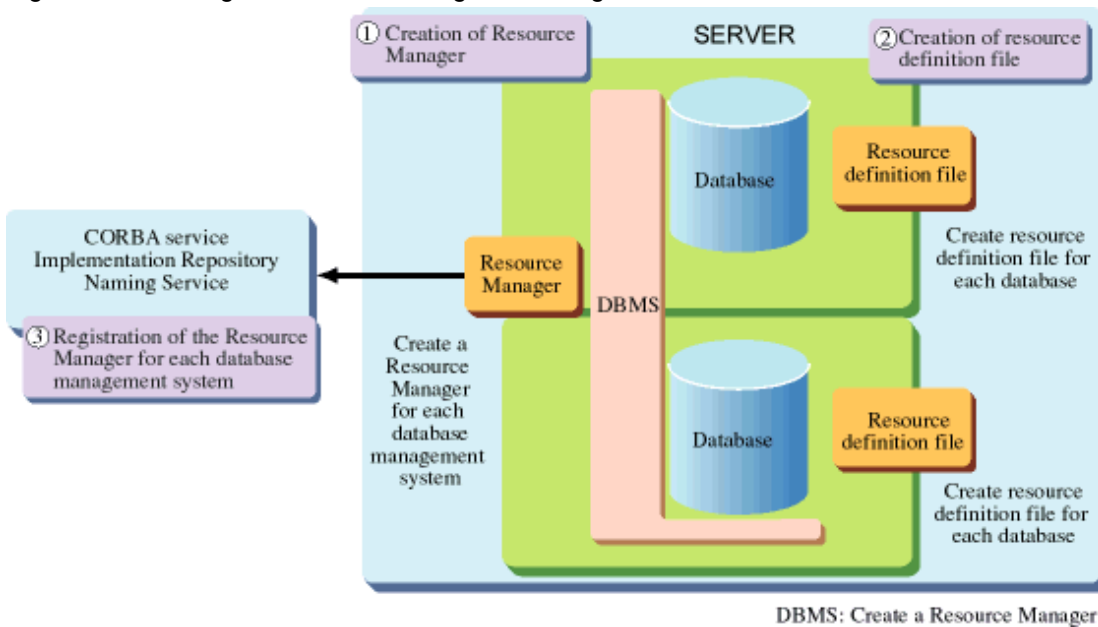
This links the client application to the library provided by the database linkage service for client applications.

Creating a Resource Management Program

Carry out transaction completion processing (commit/rollback) in the resource management program.

A resource management program is created for each database management system (Symfoware or Oracle or SQL Server). The following figure shows the procedure for creating a resource management program.

Figure 2.5 Creating a Resource Management Program



The steps to create a resource management program are:

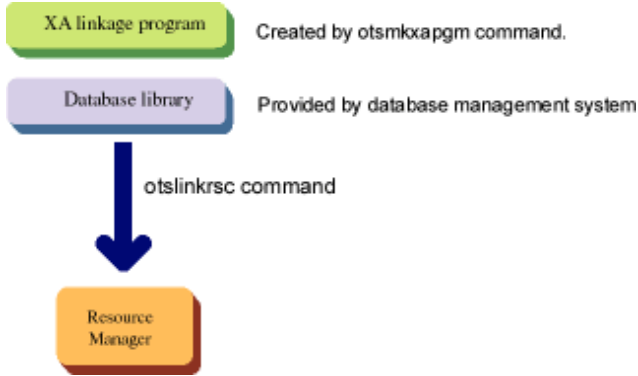
1. Creating a Resource Management Program Load Module

Use the *otslinkrsc* command in each database management system used (Symfoware, Oracle and SQL Server) to create a resource management program from the following libraries:

- Program for XA linkage (for linkage with XA interface)
- Database library provided by the database management system

The following figure shows the flow of creating a resource management program using the *otslinksrc* command.

Figure 2.6 Creating a Resource Management Program Using *otslinksrc* Command



Refer to the manual of the database you are using for details on the database library.

2. Creating a Resource Definition File

The resource definition file is the file referenced by the resource management program. It specifies the database type and open/close information. It is created for each database that is used (each instance in the case of Oracle).

Note

Resource definition files used in former versions cannot be used as is, but must be converted to the current version.

3. Registering the Resource Management Program to the CORBA Service

After creating the resource management program, register it in the Implementation Repository and Naming Service. Register the resource definition file by specifying its name at the *otssetrsc* command.

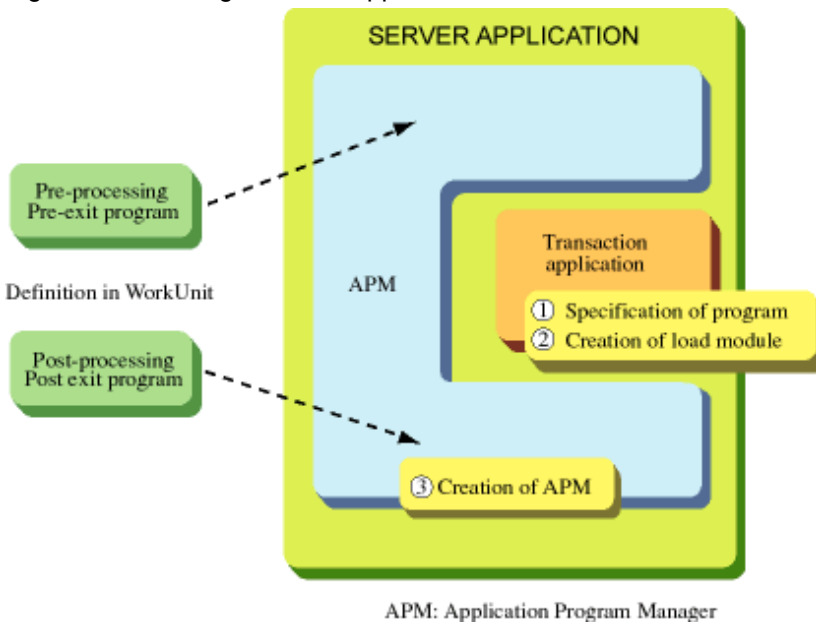
Note

Execute the *otssetrsc* command on the machine that invokes the resource management program.

Creating a Server Application

The following figure shows the procedures for creating a server application.

Figure 2.7 Creating a Server Application



The steps to create a server application are:

1. Specification of Program

Processing (user service implementation) for the database is described in the program.

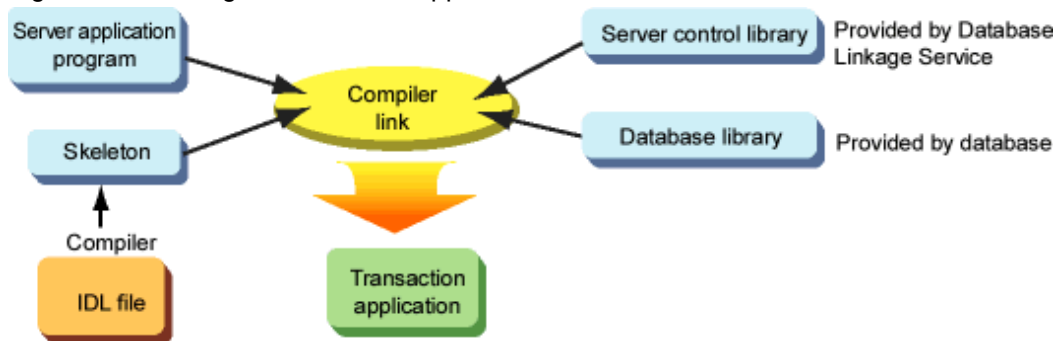
2. Creation of Load Modules

Create the following libraries and programs, and then create the transaction applications (load modules) by compiling and linking:

- Program for server applications (the program created in step 1)
- Skeleton
- Server management library (a library for transaction management)
- Database library

The following figure shows the flow of creating a transaction application.

Figure 2.8 Creating a Transaction Application

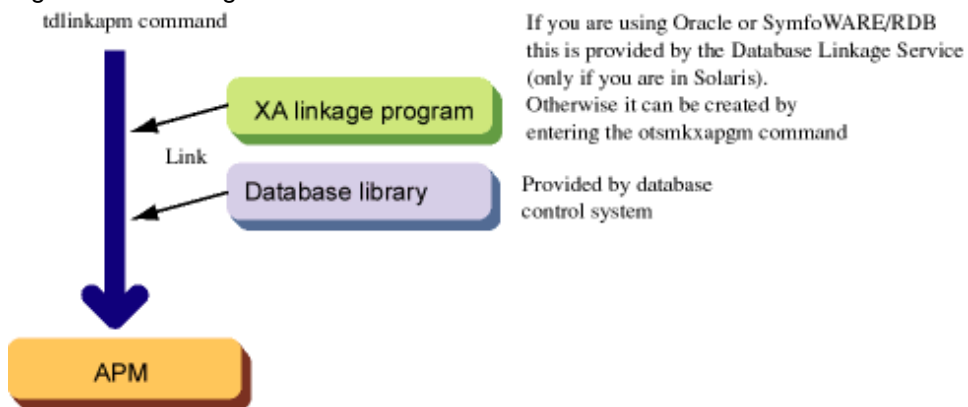


3. Creation of APM

The APM is created with the *tdlinkapm* command. When the *tdlinkapm* command is executed, create the APM by linking the XA linkage library, which is provided by the database management system with the XA linkage program prepared at the time of transaction application creation.

For information on libraries provided by the database control system, refer to the appropriate database manual. The following figure shows the flow of creating APM from the *tdlinkapm* command.

Figure 2.9 Creating APM



WorkUnit Definition

In addition to the normal WorkUnit definition, the WorkUnit definition items in the following table are necessary for Global Transaction Linkage:

Table 2.4 WorkUnit Definition Items Necessary for Global Transaction Linkage

Definition item		Set content
APM	Name	Specify the name of the created APM

Definition item		Set content
Control option	Environment Variable	Environment variable used when applications and exit programs operate.
	Application library path	Folder path where the application and exit program executable files are stored.
Resource manager information	Resource definition name	Specify the resource definition name used by the WorkUnit.
	Database system name	Specify the name of the database system used by the WorkUnit (up to a maximum of 32 units).

Note

When using the Database Linkage Service, WorkUnit definitions used in the former versions cannot be used as is, but need to be converted.

Linking to Multiple Databases from One WorkUnit

To link from a single WorkUnit to multiple databases, in the WorkUnit definitions, set the [Resource Manager] section for each database, and create the application.

Organizing applications that link to multiple databases into a single WorkUnit has these advantages over construction of systems in which WorkUnits are separated according to each database:

- Simplified development and operation
- Improved throughput
- Reduced maintenance

However, these points should be noted:

- When creating a server application that links to multiple databases, make sure that the actual number of databases is at least two.
- Within a single transaction, access all databases that the server application is able to link to.

For details, refer to the "Operating the Distributed Transaction Function" chapter.

2.2.2.7 Session Information Management

By using the session information management function in the transaction application, you can manage the session information when the application system is being constructed. Session information means any information that is held in each client on the server-side when there are multiple requests and responses between the client and server objects.

There is also a way to use the process binding function in order to manage session information.

Using the Session Information Management Function

Session information management is an application in which a single task concludes as a result of multiple requests and responses. It is used to temporarily store any data on the server side according to each processing request. Information needs to be recorded in databases for long-term storage.

Session Information Management Units

In the session information management function, session information is managed by the Session Information Management Object (SMO). The SMO can manage session information for each of the three identifiers that distinguish the client:

- Session ID
- Web server (InfoProvider Pro) client ID
- Client identifier

Session ID is an identifier used for identifying sessions by the Component Transaction Service. Session ID is a transaction application server object, and it is numbered by session ID numbering API supplied by the Component Transaction Service. The session ID is used to store session information, and it is used to transfer an operational parameter from the client to the server. Similarly, the server can uniquely identify the client of the process request source.

The session ID is also used in the process binding function and the AIM linkage session continuation.

When web linkage is performed using the HTML Page Editing Service (WebGateway), the client ID of the Web server (InfoProvider Pro) is the session identifier that specifies the Web browser of the process request source. This ID is used by the Web server session management facility. The client ID of the Web server is sent to the transaction application by the client ID notification function of the HTML Page Editing Service (WebGateway). If Web linkage is carried out, the client ID of the Web server can be used in exactly the same way as the session ID.

Refer to the HTML Page Editing Service User's Guide regarding the client ID notification function of the HTML Page Editing Service.

The Web server client ID is also used with the process binding function and with the session continuation of AIM linkage.

When calling a transaction application from the Web browser by using a servlet, use the session ID because the servlet is a CORBA client. The servlet records the session ID in the object that manages a session with the Web browser, enabling the session ID of the Web browser as the requester to correspond to the session ID of the Component Transaction Service.

The client identifier is the identifier for uniquely identifying the CORBA client process of the process request source when using a CORBA client. The client identifier is automatically assigned by the CORBA service. You can get the client identifier acquisition API in the transaction application.

The following table shows the assignment methods, advantages, and usage details for the Session Information Management Object.

Table 2.5 Session Information Management Object Assignment Methods

Identifier	Assignment method	Method of getting with server	Advantage	Usage function
Session ID	API with server object	Parameter passing from client	You can have sessions range with each application Client identification even with 1 thread unit is possible	Session information management Process binding Continuous session of AIM linkage
Web server client ID	Automatic assignment by Web server	Parameter passing from client	Client identification with Web client linkage is possible	Session information management Process binding Continuous session of AIM linkage
Client identifier	Automatic assignment by CORBA server	Issuing get API by server	You do not have to rotate the identifier by parameter. You do not have to rotate the identifier on the client-side	Session information management

Note

The session information area acquired by any object can be accessed even from an object that is not in the same server. Session information that exists in multiple servers cannot be managed.

2.2.2.8 Server Application Process Modes (In the Case of Solaris and Linux)

Server application processes have the two modes described below.

Thread Mode

This is the multi-thread mode of operation for server application processes. However, the server application only runs on the primary thread and does not operate with multiple threads. The thread mode starts multiple Interstage control threads.

The thread mode should be used in normal circumstances.

Process Mode

This is the single-thread mode of operation for server application processes. Use this mode only if the libraries of other products called from the server application cannot be called from processes running in multi-thread mode.

2.2.2.9 Specification of the Conditions for Restarting a Server Application Process after its Failure

When a server application process fails during application processing, the process is restarted by the automatic application restart function. However, for a transaction application, the restart of the process is not performed and the WorkUnit is terminated abnormally in the following cases: Because of the area destruction by the application, the failure occurs at a time except during the control of the application program (i.e., during the control of Interstage), or the server application is forcibly terminated externally (killed) while it is waiting for the receipt of a request.

The abnormal termination described above can be avoided, and the process can be restarted automatically even if the application process fails at a time except during the control of the application program.

To use this function, set "YES" to the "Unconditional Reactivation of Process" statement in the WorkUnit definition. Note this function is disabled by default.

2.2.3 Linking with Existing Systems

To link with applications on an existing system, use the wrapping object and wrapper definition.

2.2.3.1 Wrapping Objects

Wrapping objects are functions that display existing system applications as objects of the distributed object environment with Interstage.

Wrapping objects themselves can be related to existing system applications by the WorkUnit definition. A wrapping object appears from the client application as a communications partner, and is actually linked with the existing system application via the wrapping object on the server.

Because wrapping objects are provided as standard, there is no need to create a server application in order to implement them. By designing the interface section to allow linkage with Interstage, the resources (applications) of the existing system can be used effectively.

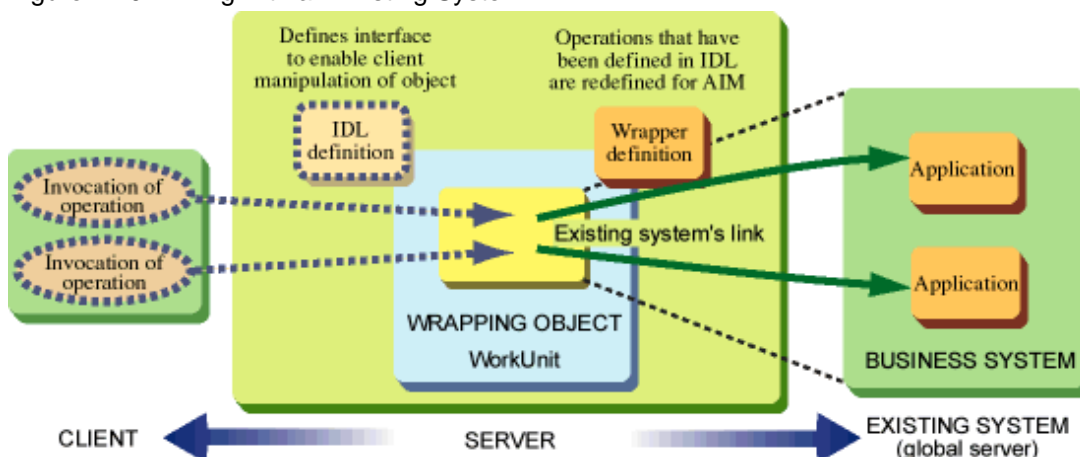
2.2.3.2 Wrapper Definitions

In order to relate an object (interface) that has been registered in the IDL file with an existing system application define this information:

- Relationship of operation name to application name on the existing system
- Relationship of the data format of each parameter and the variable format used by the application on the existing system (operation unit of the object defined by the IDL definition file)

The following figure shows the structure for linking with an existing system:

Figure 2.10 Linking with an Existing System



Only one wrapping object is defined in the WorkUnit. Similarly, you can unify the operating range of the WorkUnit with the operating range of the applications on the existing system.

For a detailed explanation of the wrapper definitions, refer to the NETSTAGE Director User's Guide. Refer also to the relevant AIM Manual regarding applications that are necessary in response to the existing connected system.

2.2.4 Performing Processing Using General Applications

General applications which are created by users and are not under Interstage can use WorkUnits just like transaction applications and EJB applications. This is called a utility WorkUnit.

This section explains the effects and notes when a WorkUnit is used with respect to general applications.

2.2.4.1 Using a WorkUnit

The following items are available when ordinary applications not under the control of Interstage were operated in the WorkUnit.

Batch Activation of General Applications

All the ordinary applications registered in the WorkUnit can be started in batch (as many as the process concurrency specifies) by starting the WorkUnit.

Batch Stop of General Applications

By stopping a WorkUnit, a batch stop of a group of general applications activated by the WorkUnit can be carried out. It is possible to set an exit program that specifies how to stop general applications. You need to create an exit program that specifies a stop appropriate to each application.

When an exit program is not set, ordinary application processes will be freed from Interstage monitoring, and the processes remain as they are. The user has to terminate the remaining processes separately.

Automatic Reactivation of General Applications

General applications are monitored and if one of them terminates abnormally it can be reactivated automatically. It is also possible to set the 'number of times the application process terminates abnormally' for a WorkUnit. On reaching the specified value, the WorkUnit terminates.

Resetting the 'number of times the application process terminates abnormally'

It is possible to reset the error termination count. One of the following two methods can be used to reset the count:

Reset by specifying the time

If a process does not terminate abnormally for a preset period of time, the error termination count that is already incremented can be reset. The time can be set in the WorkUnit definition.

Reset by the reset command

By executing the command, the error termination count that has been incremented can be reset. Execute the command or execute it from a general application using the System function or other method.

Process Multiplicity of General Applications

The process concurrency to be activated can be specified in the WorkUnit definition. Specify the concurrency if multiple processes need to be activated.

Specifying the Output Directory

It is possible to specify the output directory when general applications are running. The output directory has the following configuration. All general applications specified in the same WorkUnit are executed under the following directory. Thus, if you want to operate each application process in a different directory, you need to move the directory within the application.

```
Output directory: "X" / "Y" / "Z"
"X": Directory specified in the WorkUnit definition
"Y": Target WorkUnit name
"Z": Execution process id of application
```

Current Directory Generation Management Function

If the following environment variable is set in Environment Variable: Statement within the [Control Option] section of the WorkUnit definition, up to five generations of the current WorkUnit directory can be backed up:

```
EXTP_CURRENTDIR_HISTORY=YES
```

A backup is created when a WorkUnit is activated and up to five generations of the backup are managed. The directory with an existing WorkUnit name is backed up as WorkUnit-name.old1. WorkUnit-name.old1 is backed up as WorkUnit-name.old2 and thus sequentially backed up till WorkUnit-name.old5 is created. When the directory with WorkUnit-name.old5 already exists, the WorkUnit-name.old5 is deleted and WorkUnit-name.old4 is backed up as a new WorkUnit-name.old5.

Even after the WorkUnit is restarted, the standard output file (stdout) and standard error output file (stderr) in the current directory (or the core file in the Solaris or Linux version) are saved. Thus, investigation information can easily be obtained even after a lapse of days after a problem.

Process Activation Parameter Notification

It is possible to set the activation parameters for general applications activated together with a WorkUnit. Specify the activation parameters in the WorkUnit definition.

WorkUnit Auto-stop

When command or batch execution is used in general applications, some processes do not reside and instead stop after execution. Therefore, if each process does not restart after terminating and all the processes in the WorkUnit terminate, the WorkUnit automatically enters a state of termination. This is called the WorkUnit auto-stop. This function is useful for applications that stop after command or batch execution.

2.2.5 Timeout Monitoring

Five timeouts can be monitored in Interstage:

- CORBA application timeouts
- Transaction application timeouts
- Global transaction application timeouts
- Session information management function timeouts

2.2.5.1 CORBA Application Timeouts

This subsection explains the timeout monitoring function provided by the CORBA Service.

Timeout Monitoring during Operation of CORBA Applications

The CORBA Service provides a timeout monitoring function to monitor the application operation status so that hang-up of applications can be prevented. Timeout durations are specified for the operations of client/server applications and the applications are notified of such timeouts as they occur.

Types of Timeout Time Periods

Set the timeout time of the CORBA Service in the operation environment file (config) of the CORBA Service. For details including the initial value of each parameter, refer to "config" in the Tuning Guide.

`period_receive_timeout`

Wait time on the client between a request transmission and the response being received.

If this timeout period elapses without the receiving a response from the server method, the client application is notified of a timeout.

`period_server_timeout`

Monitoring time on the server (any type other than the Persistent type) between the server method activation and the CORBA_ORB_init method (ORB initialization method: method name of the C language interface) activation

If this time period elapses without the CORBA_ORB_init method being issued, the client is notified of a NO_IMPLEMENT exception (that is, the non-implementation of the server method).

period_idle_con_timeout

Monitoring time of non-communication (no request transmission from the client) on the server

If no request is sent from the client after this timeout period, connection to the client is disconnected to release memory resources used for request processing.

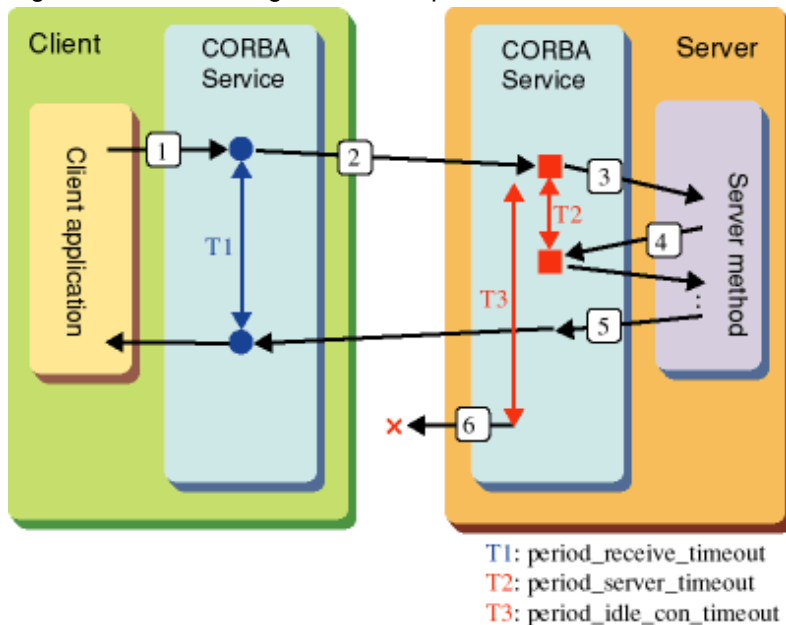
period_client_idle_con_timeout

Monitoring time (connection maintenance time after request return is complete) of non-communication (no request transmission to the server) in the client.

If the request is not sent to the server after this time is exceeded, the connection to the server is terminated. The connection is re-opened the next time a request is sent. In process mode, the connection to the server is not terminated when the timeout is exceeded, but the next time a request is sent the connection is terminated and then re-opened.

The following figure shows the flow of processing from a request transmission (server method execution request) to return and the related timeout time periods:

Figure 2.11 Processing a Client Request Transmission



1. Issue of requests from the client application.
2. Request transmission (connection established).
3. Server method activation (any type other than the Persistent type).
4. Issue of the CORBA_ORB_init method.
5. Server method return.
6. Connection terminated.

Guideline for the Timeout Time Setting

To establish CORBA application linkage, the timeout time must be set. When doing so, please remember these considerations.

period_receive_timeout (client application side)

T1 or a larger value must be set as the wait time between request transmission and response reception on the client application side. Consideration needs to be paid to the communication speed between server and client and the processing time of server methods.

Generally set a value equal to or larger than period_server_timeout on the server application side is suitable.

period_server_timeout (server application side)

T2 or a larger value must be set. Consideration should be given to the processing time between activation of the server method of the server application and the issue of the CORBA_ORB_init method.

period_idle_con_timeout (server application side)

The CORBA Service on the server side terminates the connection to the client if the period_idle_con_timeout (T3) period is exceeded after the connection is established on the first request transmission.

Since memory resources for request processing are reserved for each client in the CORBA Service, such resources are retained until the connection is terminated.

When setting period_idle_con_timeout, consider the memory capacity that can be used by the CORBA Service and the number of clients.

period_client_idle_con_timeout (client application side)

In thread mode applications, the CORBA Service on the client application side terminates the connection to the server if the value of period_client_idle_con_timeout is exceeded after all requests have been returned. The connection is re-opened the next time a request is sent. In process mode applications, the connection to the server is not terminated when the timeout is exceeded, but the next time a request is sent the connection is briefly terminated, re-opened, and the request sent.

This prevents a send error after a request is sent without the termination of the connection being detected on the client application side, if the connection to the server is terminated because of a timeout on the server side (the period_idle_con_timeout (T3) time is exceeded, for example).

Client and server only

Set a value less than or equal to the value set for period_idle_con_timeout (T3) on the server application side for period_client_idle_con_timeout (T4).

If there is a firewall between the client and server

period_client_idle_con_timeout and period_idle_con_timeout (T3) must be set after considering the firewall connection maintenance time.

Dynamic Change of the Timeout Time (period_receive_timeout)

You can dynamically change the period_receive_timeout parameter (specifying the response time from request issue to reply receipt) by invoking one of the following methods from the client application.

- CORBA_ORB_set_client_timer method (C interface method name)
Any time value change specified by this method is effective on the entire client application process.
- CORBA_ORB_set_client_request_timer method (C interface method name)
Any time value change specified by this method is effective only within the current thread of the client application.

Change the method of timeout for each development language as shown in the following table. For further details, refer to the Reference Manual (API Edition).

Table 2.6 Development Language Methods

Development language	Method name
C	CORBA_ORB_set_client_timer, CORBA_ORB_set_client_request_timer
C++	CORBA::ORB::set_client_timer, CORBA::ORB::set_client_request_timer
Java	com.fujitsu.ObjectDirector.CORBA.ORB.set_client_timer, com.fujitsu.ObjectDirector.CORBA.ORB.set_client_request_timer
COBOL	CORBA-ORB-SET-CLIENT-TIMER, CORBA-ORB-SET-CLIENT-REQUEST-TIMER
OOCOBOL	CORBA-ORB-SET_CLIENT_TIMER, CORBA-ORB-SET_CLIENT_REQUEST_TIMER

Timeout Monitoring of the Interface Repository

In the IDL compiler (*IDLc* command or *tdc* command) and interface information import (*odimportir* command), the processing time required for registration/deletion of interface information from the interface repository is monitored. This is explained in Timeout time of the interface repository.

For reference access to the interface repository, the timeout time (set by the config file) of the CORBA Service operation is valid.

Timeout Time of the Interface Repository

Set the timeout time of the interface repository to *ir_timeout* in the operation environment file (*irconfig*) of the interface repository. Changed parameter values are enabled after the interface repository is restarted.

ir_timeout

Specify the wait time for the response to a request made to the interface repository in the IDL compiler or *odimportir* command (the default value is 1,800 seconds. If 0 is specified, the monitoring of the wait time is disabled). A timeout is notified if the request does not return after this time elapses.

Action to be taken when a timeout occurs

If a longer time is needed for the registration/deletion of interface information in the interface repository (for example, when a large IDL file is compiled), the IDL compiler may terminate due to a timeout. In this case, the timeout may be avoided by increasing the value of *ir_timeout* and reactivating the interface repository.

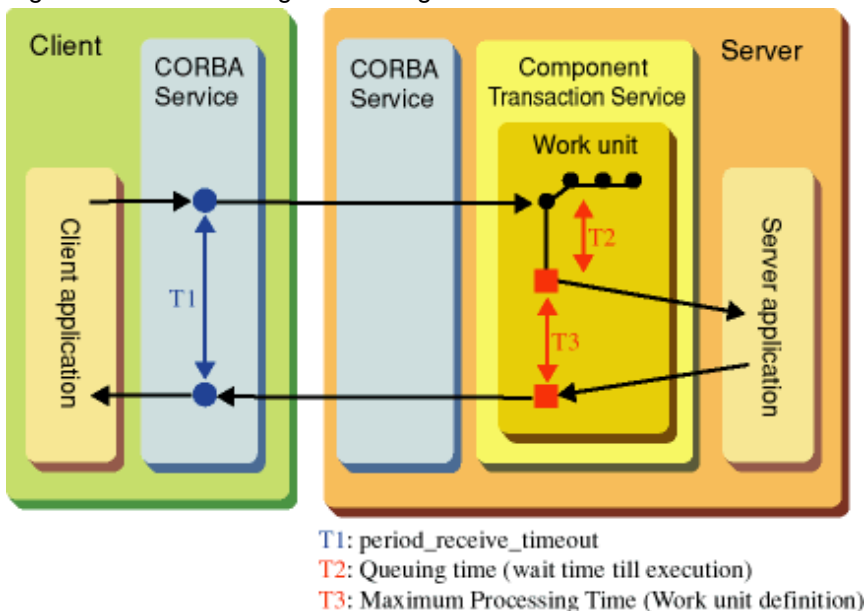
If a timeout occurs during registration/deletion of numerous interfaces (several hundred or more is not unrealistic in a large system), the value of the timeout period needs to be increased.

Timeout Time Setting in Linkage with Other Services/Applications

Maximum processing time of the server application in a WorkUnit

In a WorkUnit, the maximum processing time can be set and monitored for each application.

Figure 2.12 Determining Processing Time



When a CORBA client application and a WorkUnit are linked, *period_receive_timeout* (wait time till the request return) needs to be set so that the following requirements are satisfied:

***period_receive_timeout* (T1) > queuing time (T2) + maximum Processing Time (T3)**

Set the maximum processing time of client WorkUnits communicating between server applications, to be the maximum processing time of the server WorkUnits plus 7 seconds or more

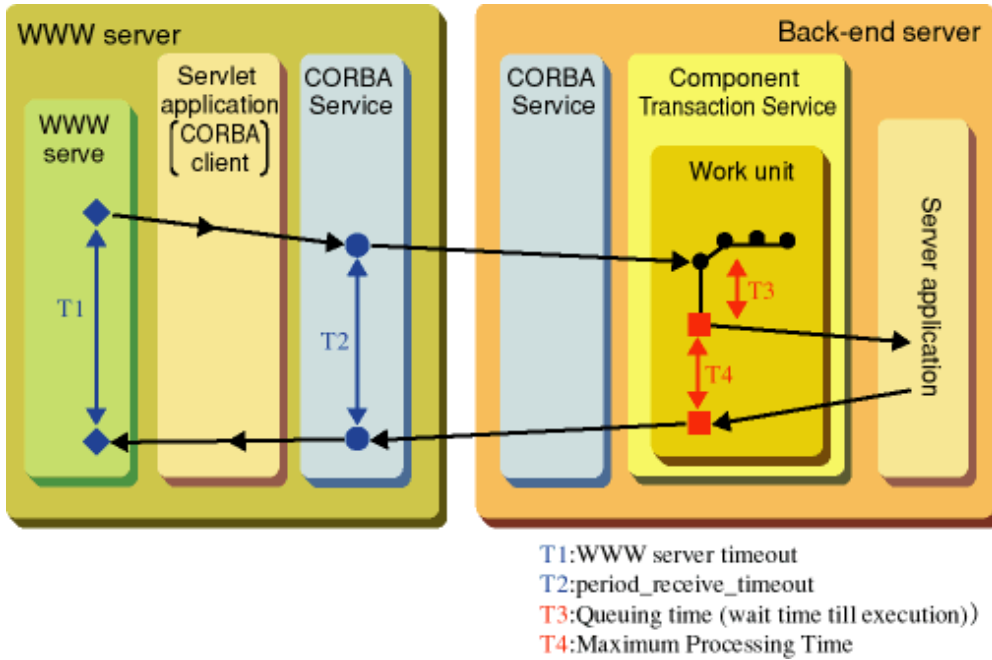
When the maximum processing time is exceeded during application processing of server WorkUnits, it may take up to 7 seconds before Interstage actually detects the maximum processing time contravention of the server and is able to report an error to the client.

Calculate the maximum processing time of client WorkUnits by considering the time during which a request stays in the server WorkUnit queue, and the application processing time of client WorkUnits.

Linkage with the Web Server

If a Servlet application (or a CGI application) activated by the Web server (InfoProvider Pro) is linked to the back-end job server/DB server as a CORBA client, the timeout time (timeout: application transmission/reception timeout time) of the Web server needs to be considered, in addition to the timeout time of the back-end job application.

Figure 2.13 Determining Processing Time for a Web-based Application



To avoid a timeout on the CORBA client (on the Web server), period_receive_timeout (wait time till the request return) needs to be set so that the following requirements are satisfied:

Web server timeout (T1) > period_receive_timeout (T2) > queuing time (T3) > transaction timeout (T4)

2.2.5.2 Monitoring Transaction Application Timeouts

Two timeouts can be monitored when using transaction applications:

- Server application timeouts
- Exit program timeouts

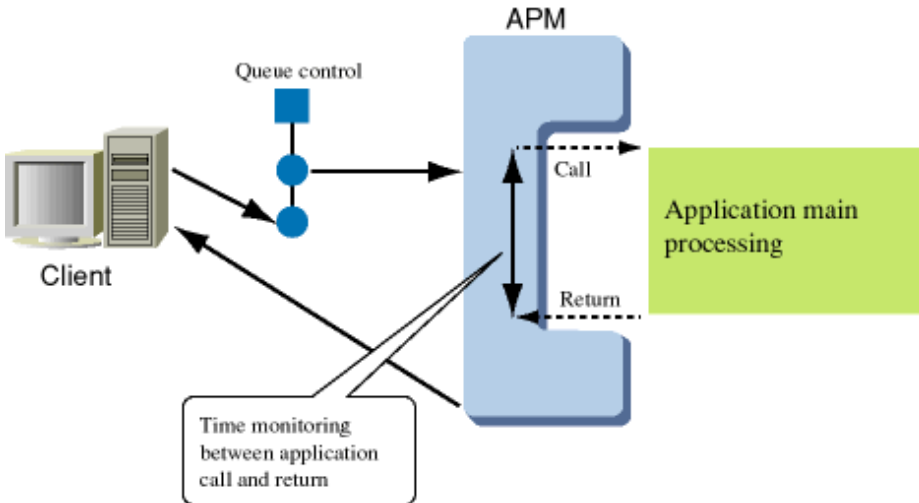
Timeout Monitoring of Server Applications

It is possible to set the maximum allowable processing time (timeout) of a server application. Timeout values are set for each application in the WorkUnit definitions.

If the maximum processing time is exceeded while a server application is processing, Interstage will forcibly terminate the application process, and server application error 10004 will be posted to the client application as the server application return value. Exception information is not posted.

Application processes forcibly terminated by Interstage are automatically restarted and are processed sequentially from the next request in the queue. However, if a successive abnormal termination count is set in the WorkUnit definitions and the number of timeouts reaches the successive abnormal termination count value, the WorkUnit is terminated abnormally.

Figure 2.14 Determining the Allowable Number of Timeouts



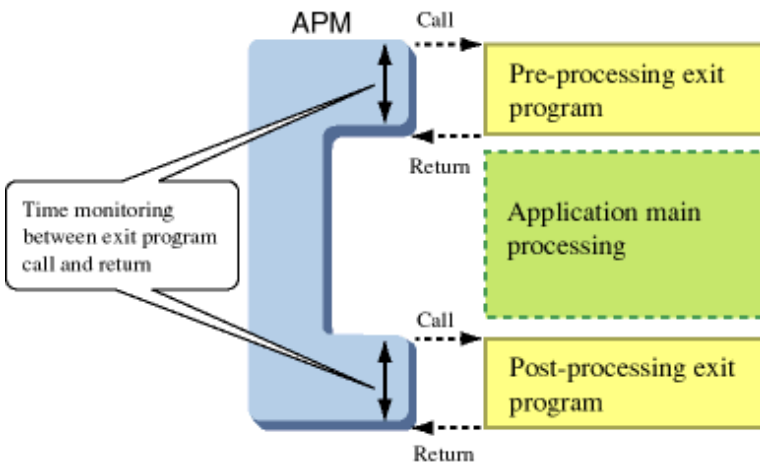
Regarding the relationship between the timeout monitoring of the transaction application and that of the CORBA application, refer to the CORBA application timeout monitoring.

Timeout Monitoring of Exit Programs

To avoid entering pre-exit and post-exit program loops, it is possible to set a maximum allowable processing time (timeout). The value of the timeout is defined in the WorkUnit definitions.

If the maximum processing time is exceeded while the pre-exit program is processing, the WorkUnit will fail to start. If the time is exceeded while the post-exit program is operating, the WorkUnit will terminate abnormally.

Figure 2.15 Determining Timeout Monitoring for Exit Programs



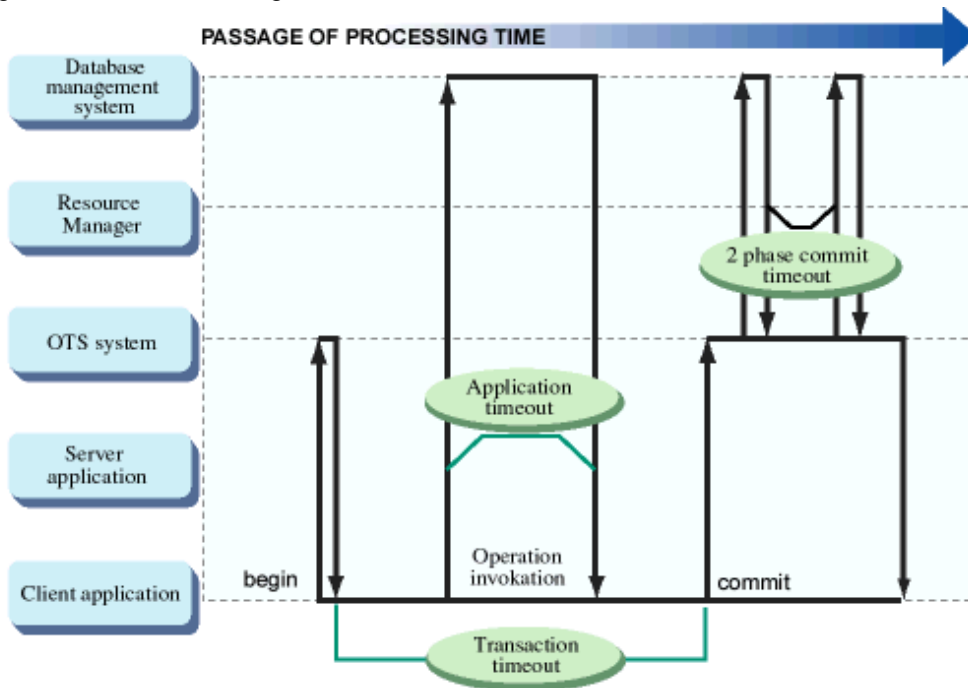
2.2.5.3 Timeout Monitoring of Global Transaction Applications

Three types of timeout monitoring are possible with transaction applications that perform global transaction linkage (global transaction applications):

- [Transaction Timeout](#)
- [Application Timeout](#)
- [Two-phase Commit Timeout](#)

The following figure shows the range of each timeout.

Figure 2.16 Timeout Ranges



Transaction Timeout

If a transaction does not terminate within a specified period, the transaction will be rolled back. The transaction timeout is set when the client application is created. Note that if the timeout is set to 0, timeout monitoring will be disabled.

A greater value than the value that includes the processing times of all applications running from begin to commit must be specified for the transaction timeout. For example, if an application is called twice after begin is called, the transaction timeout must be greater than the processing time during which the application is called twice.

For a transaction application, if the client does not call begin or commit, the transaction automatically starts when the transaction application is called. When the calling is completed, commit (or rollback) is called. In this case, the total value obtained by adding all processes (including other application calls) to be called from within one method of the transaction application must be specified as the transaction timeout time.

Specify the transaction timeout time using the following as a yardstick:

$$\text{CORBA Timeout} > \text{Tran Timeout} > \text{TD Timeout}$$

Note

- Tran Timeout: Transaction timeout time
- TD Timeout: Transaction application processing time
- CORBA Timeout: COBRA application processing time

Application Timeout

If an operation call does not complete within a specified period, control returns to the invoking side.

The application timeout is set in the operating environment file of the CORBA Service

Two-phase Commit Timeout

When a transaction does not conclude within the period specified in the two-phase commit timeout, the transaction is rolled back.

The two-phase commit timeout is set in the system environment definition file of the Database Linkage Service.

2.2.5.4 Monitoring Session Information Management Function Timeouts

In the session information management function, the session information area generated by issuing the create_info or create_info2 operation is automatically collected when no access occurs during the time specified at its generation.

If the session information area is automatically collected by this timeout monitoring function, it is posted to the status notification listener of the session information management function (when the status notification listener is already registered).

2.2.6 Design when Using the Operation Support Function of a WorkUnit

A WorkUnit can be used for transaction applications. This section explains the behavior of the operation support function, its relationship with the application environment and how it can be shared by applications. For information about how to use the WorkUnit with applications, refer to "[2.2.2 Processing with Transaction Applications](#)".

Functions that can be shared by applications are:

- [WorkUnit Exit Function](#)
- [Process Salvage Exit Function](#)
- [WorkUnit Process Information Notification Function](#)

2.2.6.1 WorkUnit Exit Function

An exit program can be set for each WorkUnit. The WorkUnit Exit program specified in the WorkUnit definition is called when the WorkUnit is activated, stopped or terminated abnormally.

This function is useful for freeing resources such as common memory where the resources are acquired when WorkUnits are activated and released when the WorkUnits are stopped or terminated abnormally.

A WorkUnit Exit program is called when a WorkUnit is activated and can return user information. This information can be referenced from the process information notification. For example, by retrieving the shared memory by the WorkUnit exit program at the time of the WorkUnit startup and returning the shared memory identifier as the user information of the WorkUnit exit program, you can pass the shared memory identifier retrieved at the time of the WorkUnit startup to the application process.

The WorkUnit exit function can monitor maximum processing time. When the maximum processing time elapses, the process to call the exit is forced to stop and a message is output. When an attempt is made to start the WorkUnit, the activation fails. If the function is called for other reasons, no operation error occurs.

If a WorkUnit exit program does not return due, for example, to a loop; the WorkUnit start/stop is not completed. For these reasons, you are advised to always monitor the maximum processing time.

WorkUnit Exit programs run in the following folder:

```
Operation folder:xxx\yyy\zzz_exit
xxx: Folder specified in the WorkUnit definition
yyy: Target WorkUnit name
zzz: Execution process id of the WorkUnit exit program
```

The standard output file and standard error output file are output to the above folder.

The WorkUnit exit function is available for the transaction application WorkUnit, and the general application WorkUnit (the utility WorkUnit).

The WorkUnit Exit function can be used for the WorkUnits of transaction applications.

Note

The WorkUnit Exit function is called once when the WorkUnit is activated by the exit program and one more time when the WorkUnit is stopped. This is a different function from the pre/post exit function called for each application process when starting/stopping the WorkUnit.

If the WorkUnit exit function and process collection exit function are used at the same time, the same load module needs to be used.

2.2.6.2 Process Salvage Exit Function

This function is useful when a process stops or resources or information related to a process set in the common memory need to be deleted. If an application stops (including an abnormal termination), a process salvage exit program set in the WorkUnit definition can be called.

Since the Process Salvage Exit program can be called from any process other than application processes, information that can only be referenced in application processes is unavailable. Keep this point in mind when creating a process salvage exit program.

This process salvage exit is called when a process stops due, for example, to a stop command or abnormal end.

The Process Salvage Exit function can monitor the maximum processing time. When the maximum processing time elapses, the process to call the exit is forced to stop and a message is output. No operation error occurs.

If a Process Salvage Exit program does not return due, for example, to a loop; the next WorkUnit operation remains in wait status. For these reasons, you are advised to always monitor the maximum processing time

Process Salvage Exit programs run in the following folder:

```
Operation folder: xxx\yyy\zzz_exit
xxx: Folder specified in the WorkUnit definition
yyy: Target WorkUnit name
zzz: Execution process id of the process salvage exit program
```

The standard output file and standard error output file are output to the above folder.

The Process Salvage Exit function can be used for the WorkUnits of transaction applications.

2.2.6.3 WorkUnit Process Information Notification Function

Applications can obtain information about local processes and the WorkUnit from environment variables. When activating each application from the WorkUnit, you should set information about processes and the WorkUnit to specific environment variables.

The following sections explain the information to be set for environment variables. This information is set to "transaction applications".

Process Serial Number

Set a unique serial number within the WorkUnit. When a process is restarted, the same number as that of the abnormally terminated process is set. This number is independent of the process ID.

Block allocation information for each process within the WorkUnit may be held in shared memory. If this is the case and you want to use the same shared memory block also in the restarted process after it terminated abnormally, this number can be used to determine the block to be used. However, since this number is only unique within the WorkUnit, only use the number for determining processes within the WorkUnit.

WorkUnit Name

Set the name of the WorkUnit that started the process.

The WorkUnit name is useful when applications need to be aware of WorkUnit names when performing processing.

Activation User Name

Set the name of the user who activated the WorkUnit.

The activation user name is useful when the name of the user activating the process needs to be recognized.

Process Activation Count

The count of process activation is set. The number is set as 1 the initial activation by the WorkUnit and is incremented by 1 each time a process is restarted.

The process activation count is useful when it is necessary to determine whether a process was activated for the first time or was reactivated.

WorkUnit Exit Return Information

Set the user return information as the return of the WorkUnit exit called when the WorkUnit is activated. If no WorkUnit exit is set in the WorkUnit definition, 0 is set.

Use this information if you need to obtain common, dynamic information within the WorkUnit.

For details on the WorkUnit exit, refer to "2.2.6.1 WorkUnit Exit Function".

2.2.7 Various Exit Functions

Interstage Application Server supports the following types of exit programs as WorkUnit operation support functions.

- Pre-exit program
- Post-exit program
- WorkUnit exit program
- Process recovery exit program
- Error exit program
- Process stop exit program

These types of exit programs have the following characteristics:

Table 2.7 Exit Program Characteristics

Exit program type	Call unit	Call timing	Example	WorkUnit type
Pre-exit program	Application process	Start of application process	Initial processing such as connection to database	ORB(*1)
Post-exit program	Application process	Normal termination of application process	Termination processing such as disconnection from database	ORB(*1)
WorkUnit exit program	WorkUnit	Start, normal stop, forced stop, or abnormal termination of WorkUnit	When the WorkUnit starts, the WorkUnit program allocates shared memory for use by the WorkUnit. When the WorkUnit stops, the program deallocates the shared memory. The identifier of the allocated shared memory can be passed to the application process when the program returns with the identifier as user information.	CORBA ORB(*1) UTY(*2)
Process recovery exit program	Application process	Normal stop, forced stop, abnormal termination, hot modification, or dynamic deletion of application process	When the process stops, the program deletes the resources and information related to the process from shared memory.	CORBA ORB(*1) UTY(*2)
Error exit program	Application process (session unit)	Detection of client think time-out while the process bind function is used	By issuing a session ID reference API, the program references the session ID (client identifier) and, based on the session ID, frees the area for the client object that caused a time-out in the transaction application.	ORB(*1)
Process stop exit program	Application process	Start of application process stop processing	The program stops the utility WorkUnit process.	UTY(*2)

*1 This is not valid for Linux (64 bit) or Windows (64 bit).

*2 This is not valid for Windows.

2.2.8 Trouble Investigation Support Function

When a CORBA WorkUnit process is started, a log file is created under the current directory of the WorkUnit, and a function is offered for the output of the process start parameter and the start environment variable.

If the start of a lone CORBA application is successful, but start of the CORBA WorkUnit fails, it is possible that the path of an environment variable required for WorkUnit configuration was not set. In this case, refer to this log file and check that the start parameter and environment variable are set correctly.

This function is enabled in CORBA WorkUnits.

To use this function, set "YES" in the "Start Log" statement of the "Control Option" in the WorkUnit configuration. Alternatively, in the Interstage Management Console [Environment Settings] window, select [Output] of the [CORBA WorkUnit Process Start Log] option.

[Log file output directory]

"[Current Directory]\[WorkUnit Name]\[Process ID]_info.log"

The log file is output in the following format.

Title	: Process start date
First line	: Start parameter
Second line and thereafter	: Environment variable

```
-----
                08/09/2004  08:39:44      Process start date
-----
"C:\Interstage\JDK6\bin\java.exe" "simple_s"  Start parameter
env[0] : ALLUSERSPROFILE=C:\Documents and Settings\All Users Environment variable
env[1] :
classpath=C:\Interstage\ODWIN\src\sample\complex\samplelist.Java\data/array;
.;C:\Interstage\lib\isadmin_scs.jar;C:\Interstage\jms\lib\fjmsprovider.jar;C
:\Interstage\J2EE\lib\isj2ee.jar;C:\Interstage\J2EE\lib\providerutil.jar;C:
\Interstage\J2EE\lib\fscontext.jar;C:\Interstage\Enabler\Runtime\Java\
Classes\jena.jar;C:\Interstage\Enabler\Runtime\Java\Classes\jcifs.jar;C:
\Interstage\Enabler\Runtime\Java\Classes\jenaj2ee.jar;C:\Interstage\lib;C:
\Interstage\lib\xmlpro.jar;C:\Interstage\lib\xmltrans.jar;C:\Interstage\lib
\xmltransx.jar;C:\Interstage\ODWIN\etc\class\ODjava4.jar;C:\Interstage\eswin
\lib\esnotifyjava4.jar
env[2] : CommonProgramFiles=C:\Program Files\Common Files
env[3] : COMPUTERNAME=Interstage
env[4] : ComSpec=C:\WINNT\system32\cmd.exe
env[5] : EXTPATH=C:\Interstage\bin
env[6] : FJIIOPATH=C:\Interstage\EJB\BIN
env[7] : INCLUDE=C:\Program Files\Microsoft Visual Studio .NET 2003\SDK\v1.1\include\
env[8] : IS_APL_INFO1=0
env[9] : IS_APL_INFO2=0
env[10] : IS_APL_SERIALNUM=1
env[11] : IS_APL_STARTNUM=1
env[12] : IS_APL_SYSNAME=default
env[13] : IS_APL_USRNAME=root
env[14] : IS_APL_WUNAME=ODSAMPLE
env[15] : IS_HOME=C:\Interstage
env[16] : IS_J2EEAPF=C:\Interstage\J2EE\var\deployment
env[17] : JAVA_HOME=C:\Interstage\JDK6
env[18] : LOGOFF_IGNORE=YES
env[19] : NUMBER_OF_PROCESSORS=1
env[20] : OD_HOME=C:\Interstage\ODWIN
env[21] : OD_IMPLID=ImpleArraytest
env[22] : OD_WORKUNIT=ON
env[23] : OD_WUCURRENTDIR=ON
env[24] : OD_WUIMPLID=ImpleArraytest
env[25] : OS=Windows_NT
env[26] : Os2LibPath=C:\WINNT\system32\os2\dll;
env[27] : Path=C:\Interstage\JDK6\bin;C:\WINNT\system32;C:\WINNT;C:\WINNT\System32\
Wbem;C:\INTERS-1\bin;C:\Interstage\bin;C:\Interstage\ODWIN\bin;C:\INTERS-1\
ID\Dir\bin;C:\INTERS-1\ID\Dir\sdk\c\bin;C:\INTERS-1\ID\Dir\sdk\c\lib\dynamic
env[28] : PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
env[29] : PROCESSOR_ARCHITECTURE=x86
env[30] : PROCESSOR_IDENTIFIER=x86 Family 15 Model 2 Stepping 9,
```

```
GenuineIntel
env[31] : PROCESSOR_LEVEL=15
env[32] : PROCESSOR_REVISION=0209
env[33] : ProgramFiles=C:\Program Files
env[34] : SystemDrive=C:
env[35] : SystemRoot=C:\WINNT
env[36] : TD_HOME=C:\INTERS~1\td
env[37] : TEMP=C:\WINNT\TEMP
env[38] : TMP=C:\WINNT\TEMP
env[39] : UJI_HOME=C:\Interstage\APC\
env[40] : USERPROFILE=C:\Documents and Settings\oms
env[41] : windir=C:\WINNT
```

Chapter 3 Starting / Stopping / Surveillance of WorkUnits

This chapter explains starting, stopping and surveillance of WorkUnits.

3.1 Starting and Stopping WorkUnits

WorkUnits can be started and stopped using the Interstage integration command, or Interstage management console. WorkUnits can also be started automatically. Note that the Interstage Management Console can only be used for CORBA WorkUnits and IJServer WorkUnits.

This section explains each of the above scenarios.

3.1.1 Starting and Stopping WorkUnits Using the Interstage Integration Command

This section explains how to start and stop WorkUnits using the Interstage integration command.

Starting WorkUnits

Start a WorkUnit by specifying the WorkUnit name in the *isstartwu* command.

```
isstartwu TDSAMPLE1
```

To start an EJB container, specify the EJB container name in the *isstartwu* command.

```
isstartwu ejb-container-name
```

Note

Transaction application WorkUnits and WRAPPER WorkUnits can also be started with the *tdstartwu* command.

Terminating WorkUnits

All WorkUnits are terminated by the *isstopwu* command.

```
isstopwu TDSAMPLE1
```

To stop an EJB container, specify the EJB container name in the *isstopwu* command.

```
isstopwu ejb-container-name
```

Note

Transaction application WorkUnits and WRAPPER WorkUnits can also be stopped with the *tdstopwu* command.

To stop an EJB container synchronously, specify "-s EJB container name" in the *isstopwu* command.

To stop an EJB container forcibly, specify "-c EJB container name" in the *isstopwu* command.

If a WorkUnit is stopped where the pop-up dialog indicating an application error is output, no-response may occur. In this case, a no-response is released by closing the pop-up dialog.

3.1.2 Monitoring WorkUnits

WorkUnits can be monitored using the Interstage integration command, or the Interstage Management Console.

This section explains monitoring WorkUnits using the Interstage integration command. Similar procedures can also be used with the other monitoring methods. Note that the Interstage management console can only be used for CORBA WorkUnits and IJServer WorkUnits.

3.1.2.1 Operating Status of WorkUnits

The operating status of the WorkUnits can be confirmed by the *islistwu* command.

```
islistwu
```

A WorkUnit may have one of the following statuses:

Information	Contents
Starting process	startproc
Executing (startup completed)	execute
Processing stopped	stopproc

Note

Transaction application WorkUnits and WRAPPER WorkUnits can also use the *tdlistwu* command.

3.1.2.2 Operating Status of Objects of WorkUnits

The *islistobj* or *isinfobj* command is used to verify the operating status of objects (Implementation Repository ID).

```
islistobj
```

The information in the following table can also be checked using the *islistobj* command.

Table 3.1 Object Operating Status Information - *islistobj* Command

Information	Contents
objectname/applicationname	The application name.
kind	The WorkUnit type.
status	The status of the object of the WorkUnit. Active: Operating Inhibit: Inaccessible

```
isinfobj objname
```

Where *objname* is the name of the object whose details are to be checked.

The following information can also be checked using the *isinfobj* command.

The object of a WorkUnit may have one of the following statuses:

Information	Contents
Object enabled (in use)	Active
Object disabled *1 (inhibited)	Inhibit

CORBA Applications

The following table shows the detailed information that is displayed for an object (Implementation Repository ID) of a running CORBA application.

Table 3.2 Object Operating Status Information - CORBA Applications

Information	Contents
impl ID	The Implementation Repository ID.
kind	"CORBA".
status	The status of the object. Active: Operating Inhibit: Inaccessible
object	The interface name registered in the Implementation Repository.

Information	Contents
procnum	Concurrent number of application processes for implementation repository ID
queue	The number of client requests remaining in the application.
accumulation	The accumulation count processed by the application.
thread	The number of application threads.
wuname	The name of the WorkUnit in which the application operates.

Transaction Applications or WRAPPER Applications

The following table shows the detailed information that is displayed in the case of a running transaction application or WRAPPER application.

Table 3.3 Object Operating Status Information - Transaction or WRAPPER Applications

Information	Contents
objectname	The name of the application object.
kind	The application type. TD : Object of transaction application WRAPPER : Object of wrapper application
status	The status of the object. Active : Operating Inhibit : Inaccessible
procnum	The concurrent number of application processes for the object.
queue	The number of client requests remaining in the application.
accumulation	The accumulation count processed by the application.
wuname	The name of the WorkUnit in which the application operates.

Note

Transaction application WorkUnits and WRAPPER WorkUnits can also be registered with the *tdinfobj* command.

IJServer Applications

The following table shows the detailed information that is displayed in the case of a running IJServer application.

Table 3.4 Object Operating Status Information - IJServer Applications

Information	Contents
IJServer name	The name of the IJServer WorkUnit.
kind	"IJServer".
IJServer type	The IJServer type. 1VM: The Web application and EJB application are operated on the same JavaVM. Web: The Web application is operated alone or is operated on one JavaVM with the EJB application on another JavaVM. EJB: The EJB application is operated alone or is operated on one JavaVM with the Web application on another JavaVM.
status	The status of the object. Active: Operating. Inhibit: Inaccessible (displayed only when the type is EJB).

Information	Contents
	Stop: Inactive (displayed when either of the two applications, Web and EJB applications to be operated on different JavaVMs, is not provided).
procnum	The concurrent number of application processes.
queue	The number of client requests remaining in the application (EJB applications only).
accumulation	The accumulation count processed by the application (EJB applications only).
instance	The number of application instances (EJB applications only).

3.1.2.3 Checking Application Process Information

Use the *islistapproc* command to display the process information for applications operated using the WorkUnit.

```
islistapproc [wuname]
```

Where wuname is the name of the WorkUnit for which application process information is to be checked.

WorkUnit application process information is shown in the following table.

Table 3.5 WorkUnit Application Process Information

Information	Contents
PID	The process ID.
wuname	The WorkUnit name.
kind	The WorkUnit type. CORBA: CORBA WorkUnit EJB: EJB WorkUnit TD: Transaction WorkUnit IJSERVER: IJSERVER WorkUnit
objectname/applicationname	The following information on each WorkUnit type is output. CORBA WorkUnit: Implementation Repository ID EJB WorkUnit: EJB application name Transaction WorkUnit: Object name IJSERVER WorkUnit: WorkUnit name Utility WorkUnit: Execution filename of the application

3.1.3 Starting and Stopping WorkUnits Using the Interstage Management Console

This section explains how to start and stop WorkUnits using the Interstage management console.

To start or stop a WorkUnit, press the Start or Stop button respectively in the WorkUnit: status window.

For further information on "Using the Interstage Management Console", refer to the Operator's Guide.

3.1.4 Starting WorkUnits Automatically

WorkUnits can be started automatically during Interstage startup by creating a WorkUnit Automatic Start setting file.

The procedure for automatically starting a WorkUnit is as follows.

- Shut down Interstage.
- Create a WorkUnit Automatic Start setting file and specify the WorkUnit that is to be started automatically. Refer to the "WorkUnit Automatic Start Setting File" appendix for further details.

- Start Interstage. The specified WorkUnit is then automatically started.

When the Interstage management console is used to operate the WorkUnit, the setting can be performed in "WorkUnit creation" or "WorkUnit environment settings".

Note

A WorkUnit that performs global transaction linkage cannot be started automatically.

Notes on Using a Pre-exit Program

To use a pre-exit program on a WorkUnit that is to be started automatically (to execute database connection processing), the database must be started before the WorkUnit is started; that is, before Interstage is started. Care must therefore be taken when starting up the machine.

When the system has been set up in such a way that Interstage starts automatically on machine startup, the order in which services are automatically started is unclear. The database system is therefore not guaranteed to start before Interstage. The database system may also be inactive when the pre-exit program of the WorkUnit to be started automatically is invoked so database processing in the pre-exit program may fail.

To address the above situation, take one of the following actions:

- If the database is inactive in pre-exit processing, do not assume an error but start database connection processing.

An example outline of processing is as follows:

(Preprocessing) Database connection processing. Returns normally even if the database system is inactive.

(This processing) Database operation. Starts with database connection processing if the database is not yet connected.

- Change the start-up type of the Interstage service (service name: INTERSTAGE) from Auto to Manual to prevent Interstage starting automatically when the server machine starts. Start Interstage after the server machine has started.

Note that although these notes use a database system as an example, similar care must be taken for other services used in the WorkUnit pre-processing.

Notes on Wrapper WorkUnits

To automatically start a wrapper WorkUnit, do not automatically start Interstage when the machine starts. Start Interstage after the NETSTAGE Director service has started.

3.2 Performance Monitoring Tool

The Performance Monitoring Tool is designed to collect the performance information of the objects of the transaction applications and the wrapper running on the application server.

The Performance Monitoring Tool supports the following two functions.

- Function of outputting log information to the performance log file

This function collects the performance information of specified objects in the performance log file. The accumulated performance information can be output in CSV format using the report output command.

- Function of monitoring the performance information in real time by a Network Control Manager (Monitoring by MIB)

By using the MIB (*2) monitoring function of a Network Control Manager such as Systemwalker/CentricMGR (*1), the performance information of the specified object can be displayed and monitored in real time.

Displaying and monitoring performance information using a Network Control Manager is called "real-time monitoring".

This section describes how to issue commands on the application server, and display performance information when using Systemwalker/CentricMGR as a Network Control Manager.

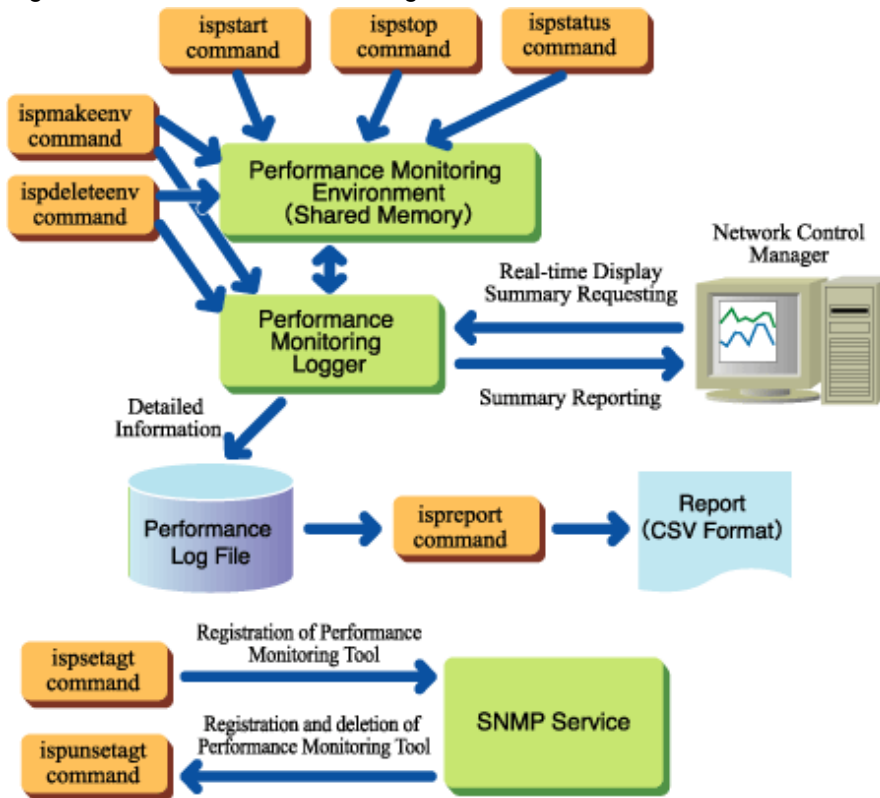
*1 Network Control Manager is a software program for displaying and monitoring performance information on the monitor server.

*2 MIB is the abbreviation of the Management Information Base. MIB is a management information area that has been defined for managing the system and TCP/IP information.

The Performance Monitoring Tool consists of the performance monitoring logger that collects performance information, and various commands.

The Performance Monitoring Tool provides the commands shown in the following figure.

Figure 3.1 Performance Monitoring Tool Commands



ispmakeenv command: Creates the performance monitoring environment and starts the performance monitoring logger

ispdeleteenv command: Deletes the performance monitoring environment and stops the performance monitoring logger

ispstart command: Starts monitoring performance

ispstop command: Stops monitoring performance

ispstatus command: Displays performance monitoring status information.

ispreport command: Outputs the performance log file report

ispsetagt command and *ispunsetagt* command: Registers and unregisters Performance Monitoring Tool in SNMP service

Performance monitoring logger: Collects performance monitoring information, reports performance information to the Network Control Manager, and creates performance log files.

3.2.1 Functions of Performance Monitoring Tool

This section describes the functions of the Performance Monitoring Tool

3.2.1.1 Function of Outputting Log Information to the Performance Log File

This function collects the performance information of specified objects in the transaction applications and the wrapper in the performance log file. The performance information is collected at the interval specified when creating a performance monitoring environment.

Since the accumulated performance information can be output in CSV format by executing the report output command, it is useful in the analysis of the performance information and the accumulation of statistical information.

The following information can be collected with this function.

Object Name

The name of the object from which performance information is collected

Operation Name

The name of the operation from which performance information is collected

Process ID

The process ID of the operation from which performance information is collected

Request Processing Time (maximum/average/minimum)

The processing time of the operation. The three types of information supplied: maximum, average, and minimum, are output at each interval to the performance log file.

Request Processing Wait Time (maximum/average/minimum)

The wait time from the acceptance of a request from a client application to the actual start of processing. The information is in object units. The three types of statistics supplied; maximum, average, and minimum, are output at each interval to the performance log file.

Number of Times the Operation has been Executed

The number of times the operation has been executed by the process within the interval time for the performance log file.

Number of Requests Received

The cumulative number of times the operation was executed from the start of performance monitoring.

Number of Requests Awaiting Processing

The maximum number of requests placed in the queue for the object within the interval time for the performance log file.

The above data items are output in either of the following units.

Information Output for each Operation

"Request processing time" and "Number of times the operation has been executed" are data items for the operation indicated in the "Operation name" and the process indicated in the "process ID". They are used to evaluate operations for each process.

Information Output for each Object

"Request processing wait time," "Number of requests received", and "Number of requests awaiting processing" are data items for the object indicated in the "Object name". They are used to evaluate operations for each process.

These data items enable detailed performance analysis.

3.2.1.2 Function of Monitoring the Real Time Performance Information by a Network Control Manager (Monitoring by MIB))

The real time monitoring function reports the performance information of the specified objects of the transaction applications and the wrapper as MIB information (*).

A Network Control Manager such as Systemwalker/CentricMGR supports the MIB monitoring function. By using the function of the Network Control Manager, the following operation can be performed.

Outputting the Report of the Statistical Information

The performance information can be displayed in graph or CSV format. It is useful in the collection of statistical information.

Monitoring the Performance Abnormality

By setting the threshold value for performance information such as "Number of requests awaiting processing" and monitoring it, any abnormality in performance information can be detected in advance. This function enables prompt response to abnormal events.

* Stands for Management Information Base. In order to manage system information and TCP/IP information, the MIB is the defined management information domain.

The following information can be collected with this function.

Object Name

The name of the object from which performance information is to be collected.

Request Processing Time (maximum/average/minimum)

The processing time for the object. The three types of information; maximum, average, and minimum, are output at each interval for real time monitoring.

Request Processing Wait Time (maximum/average/minimum)

The time between the acceptance of a request from a client application, and the actual start of processing. The three types of information; maximum, average, and minimum, are output at each interval for real time monitoring.

Number of Requests Received

The cumulative number of times the operation was executed from the start of performance monitoring.

Number of Requests Awaiting Processing

The maximum number of requests that were placed in the queue for the object, within the interval for real time monitoring, is output.

The above data items are collected for each object.

When this function is used, the information can also be output to the performance log file at the same time. For a more detailed performance analysis such as information in units of operations, analyze the information collected in the performance log file.

For the monitoring method using MIB information, refer to the manual of the Network Control Manager.

3.2.2 Performance Monitoring Procedure

To monitor and analyze the performance of Interstage job applications using the Performance Monitoring Tool, the five phases that must be performed are

1. Registering to the SNMP service
2. Starting the Performance Monitoring Tool
3. Monitoring Operation
4. Stopping the Performance Monitoring Tool
5. Deletion from the SNMP Service

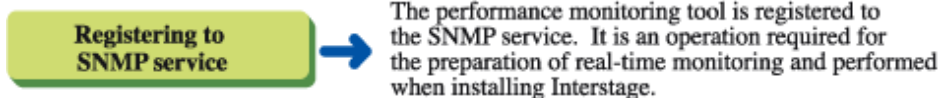
Each phase is explained in below and in [Figure 3.2 Installation Procedure](#) to [Figure 3.6 Deleting the Performance Monitoring Tool](#).

1) Registering to the SNMP Service

Register the Performance Monitoring Tool in the SNMP service.

Registers Interstage in the SNMP service.

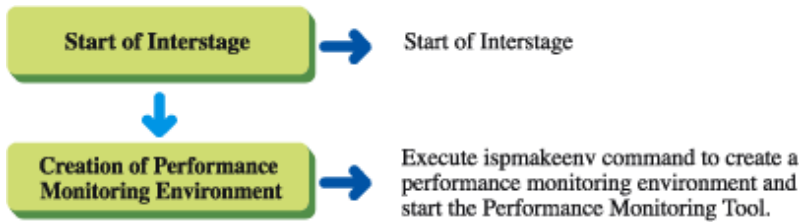
Figure 3.2 Installation Procedure



2) Starting the Performance Monitoring Tool

Perform this operation to start the Performance Monitoring Tool.

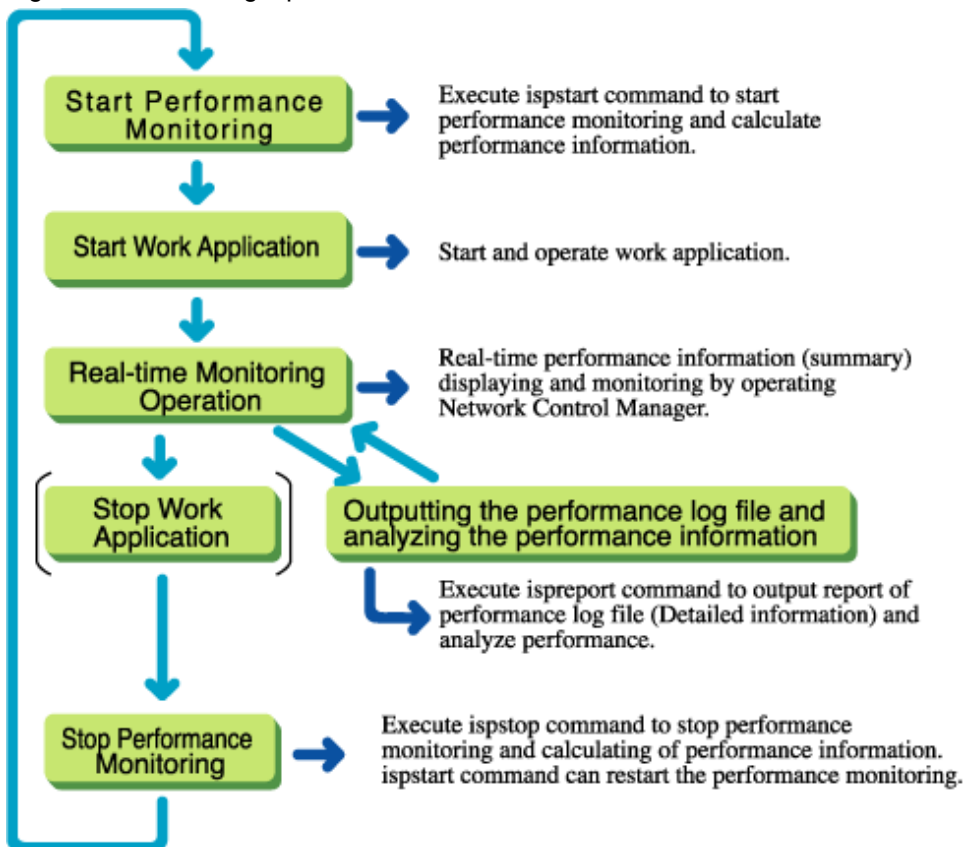
Figure 3.3 Starting the Performance Monitoring Tool



3) Monitoring Operation

Perform this operation to measure, monitor, and analyze performance information.

Figure 3.4 Monitoring Operation



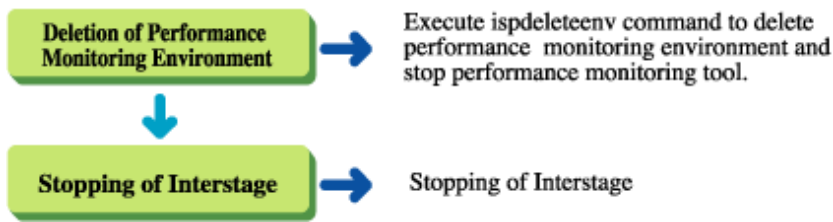
Notes

- Performance monitoring can be stopped by executing the *ispstop* command instead of stopping the application. However, performance information will not be measured after the *ispstop* command is executed. To restart measuring performance information, execute the *ispstart* command.
- After the *ispmakeenv* command is executed, start the business application (WorkUnit) that measures the performance. The performance of the business applications that are started before the execution of the *ispmakeenv* command are not measured.
- When a Network Control Manager such as Systemwalker/CentricMGR is used to display performance information in real time, do not start or stop performance monitoring while performance information is being displayed. Be sure to close the performance information display screen before starting performance monitoring, and perform real-time monitoring operation and display performance information after performance monitoring is started.

4) Stopping the Performance Monitoring Tool

Perform this operation to stop the Performance Monitoring Tool.

Figure 3.5 Stopping the Performance Monitoring Tool



Note

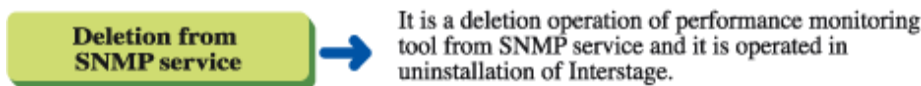
To create the performance monitoring environment again, first restart Interstage.

5) Deletion from the SNMP Service

Perform this operation when uninstalling Interstage.

Delete the Performance Monitoring Tool registered in the SNMP service.

Figure 3.6 Deleting the Performance Monitoring Tool



3.2.3 Registering to the SNMP Service

After Interstage is installed, the following operation is required to perform real-time monitoring with a Network Control Manager such as Systemwalker/CentricMGR. If real-time monitoring is not to be performed, the following operation need not be performed.

1) Registering to the SNMP Service

Execute the *ispsetagt* command to register the Performance Monitoring Tool in the SNMP service. After executing the *ispsetagt* command, re-start the SNMP service from the Service screen in Windows. The Performance Monitoring Tool can be registered in the SNMP service only when the SNMP service has been installed. Before executing the *ispsetagt* command, install the SNMP service.

The SNMP service is installed in Windows Server® 2003 and Windows Server® 2008 SNMP in the following way:

Windows Server® 2003

- Insert the CD-ROM of the OS into the CD drive.
- Select [Install Windows Option Component], then in [Windows Component Wizard]-[Admin and Monitor Tools], add [Simple Network Management Protocol (SNMP)].

Windows Server® 2008

- From [Server Manager]-[Feature], open [Add Feature], then select and install [SNMP Service].

2) Reading the MIB Definition File

To collect performance information from the Network Control Manager, the MIB definition file for performance information must be read using the Network Control Manager. Have the Network Control Manager read the MIB definition file from the machine in which Interstage has been installed, as follows:

- When the Network Control Manager operates under Windows®
Read TD_HOME\isp\mib\ispmibNT.my
- When the Network Control Manager operates under Solaris
Read TD_HOME\isp\mib\ispmibSol.my.

When using the Windows® version of Systemwalker/CentricMGR (operation management server), read the MIB definition file as follows:

- Use FTP or a similar application to copy the MIB definition file from the machine in which Interstage is installed to the machine in which Systemwalker/CentricMGR is installed.

- Activate Systemwalker/CentricMGR System Monitor.
- Choose Tools and then Extend MIB from the System Monitor screen to display the Extend MIB screen.
- Click the add button on the MIB Extension Operation screen to display the Select of Extend MIB File Selection screen.
- Select the MIB file from the Extension MIB File Selection screen and click the Open button. Then, click the Close button on the MIB Extension Operation screen.

3.2.4 Creating a Performance Monitoring Environment

3.2.4.1 Starting Operation of Performance Monitoring Tool

This section describes the construction of the environment for using the performance monitoring tool.

Starting Interstage

Execute the *isstart* command to start Interstage.

Creating the Performance Monitoring Environment

Use the *ispmakeenv* command to create a performance monitoring environment and activate the Performance Monitoring Tool.

In this case, the following two interval times are specified.

- Interval time for performance log file

The interval at which the performance information is output to the performance log file. The interval can be 5, 10, 20, or 30 minutes or 1, 2, 3, or 4 hours. The default is one hour.

- Interval time for real time monitoring

The interval for collecting the performance information to be notified to the Network Control Manager. Specify the interval time when performance monitoring is executed from the Network Control Manager. The interval can be specified between 1 and 60 minutes. The default is 5 minutes.

After the command *ispmakeenv* is executed, start the business application (WorkUnit) that measures the performance. Performance of the business applications that are started up before execution of the *ispmakeenv* command are not measured.

3.2.5 Monitoring Operations

This section describes the monitoring operation with the performance monitoring tool.

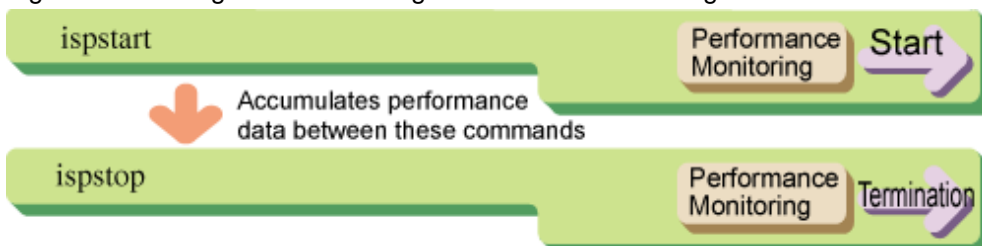
3.2.5.1 Starting Performance Monitoring

Start performance monitoring of the specific object with the *ispstart* command. The performance information is then output to the performance log file at the interval specified with the *ispmakeenv* command.

Performance information is collected until it is stopped by the *ispstop* command.

These commands are shown in the following figure.

Figure 3.7 Starting and Terminating Performance Monitoring



3.2.5.2 Starting a Business Application

Execute the *isstartwu* command to start the WorkUnit.

Real-time Monitoring

To display and monitor performance information using a Network Control Manager such as Systemwalker/CentricMGR, use the following procedure to display performance information on the Network Control Manager screen. This procedure is an overview of real-time monitoring operation when the Windows version of Systemwalker/CentricMGR operation management server is used. When using a Network Control Manager other than Systemwalker/CentricMGR, refer to the user's guide for the Network Control Manager to be used.

1. Activate Systemwalker/CentricMGR System Monitor.

2. Select the Application Server to be monitored

Click the application server name to be monitored on the System Monitor screen.

3. Display the Get MIB Data screen

Choose Tools | Specific System | Get/Set MIB Data | Get MIB Data on the System Monitor screen menu to display the Get MIB Data screen.

4. Display the List of Object Names and Check the Instance Number of the Object to be monitored

Make the following settings on the Get MIB Data screen and click the Retrieve button.

- Choose Not check Polling Interval.
- Specify DUMP as the Request Type.
- Specify "ispSumObjectName" in the MIB (performance information measurement item) to be displayed.

Specify the MIB according to the following procedure:

Click the Browse button to display the MIB Tree screen.

On the MIB Tree screen, expand the tree structure by double-clicking the options internet, private, enterprises, fujitsu, application, aplNetWork, aplNetFunction, aplInterstage, isPerformanceinf, ispSummaryTable, and ispSummaryTableEntry in that order. The performance information items that can be displayed in real-time monitoring will appear on the display. Click ispSumObjectName.

Click the Add button on the Get MIB Data screen.

5. Set the Reference Value

- On the System Monitoring screen, choose Policy | Setting Policy | Node | MIB State change to display the MIB State Change screen.
- Click the Add button on the MIB State Change screen to display the Threshold Details screen.
- Specify the MIB name (the display item name of the performance information to be monitored), instance number, and threshold value (reference value) on the Threshold Details screen. Refer to Step 4) for information about how to specify the MIB name.
- Choose On on the MIB Monitoring screen.

6. Display the Real-time Display Screen (Performance Information)

Make the following settings and click the Retrieve button.

- Set the polling time to 5 minutes or more.
- Specify GET as the Request Type.
- Specify the instance number of the object to be monitored.

When the list of object names is displayed, the following number appears. Specify this number as the instance number.

"ispSumObjectName : "ispSumObjectName. *number*. *object-name*"

- Specify the display item name of the performance information to be displayed in the MIB Name field. Refer to Step 4 for information about how to specify the MIB name.

The following table shows the performance information items that can be displayed in real-time monitoring. The display item name in this table means the item name of the performance information displayed by the Network Control Manager.

Table 3.6 Performance Information Items

Performance information item name	Unit	Display item name	Meaning
Object name	-	IspSumObjectName	Object name of the application for which performance information is measured
Maximum request processing time	ms	IspSumExecTimeMax	Maximum processing time (within a polling time) required for processing the object
Minimum request processing time	ms	IspSumExecTimeMin	Minimum processing time (within a polling time) required for processing the object
Averaged request processing time	ms	IspSumExecTimeAve	Averaged processing time (within a polling time) required for processing the object
Maximum request processing wait time	ms	IspSumWaitTimeMax	Maximum time (within a polling time) from the time of a client request to the initiation of a server application
Minimum request processing wait time	ms	IspSumWaitTimeMin	Minimum time (within a polling time) from the time of a client request to the initiation of a server application
Averaged request processing wait time	ms	IspSumWaitTimeAve	Averaged time (within a polling time) from the time of a client request to the initiation of a server application
Number of requests received	Number	IspSumRequestNum	Accumulated number of processes from the time of starting the performance monitoring up to the timing of this object.
Number of requests awaiting processing	Number	IspSumWaitReqNum	Maximum number of requests waiting for processing of this object, within the polling time.

Note

- Unless the list of object names is to be displayed, always specify 'GET' as the acquisition method when displaying performance information. If DUMP is specified for real-time display, an enormous amount of communications may be required between Systemwalker/CentricMGR and the Performance Monitoring Tool, resulting in a heavy load on the network as well as on the application server.
- When there is no object for which performance information can be displayed, 'NONE' will be displayed as the object name.

3.2.5.3 Outputting the Performance Log File and Analyzing the Performance Information

When the threshold value is exceeded during real-time monitoring and there is a possibility of performance abnormality analyze the detailed information saved as the performance log file.

Use the *ispreport* command to output a performance log file report. This command converts the performance information saved as the performance log file into CSV format and outputs it to the standard output. If the performance log file is to be converted into CSV format and output as a file, specify the output destination file name as below when executing the *ispreport* command.

'ispreport option > output-destination-file-name'

When the *ispreport* command is executed, performance information in the performance log file will be converted one record at a time and output to the standard output in the following format.

1. Line format

"D1,D2,D3, D4,D5,D6, D7,D8,D9, D10,D11,D12,D13, D14,D15,D16"

2. Output items in each line

The following table shows a list of output items in each line. Items D1 to D16 in the 'Item No.' column correspond to those shown in 1).

Table 3.7 Output Items

Item no.	Performance information item name	Unit	Meaning
D1	Data collection start date	-	Date the performance information measurement for the record was started
D2	Data collection start time	-	Time the performance information measurement for the record was started
D3	Data collection end date	-	Date the performance information measurement for the record was finished
D4	Data collection end time	-	Time the performance information measurement for the record was finished
D5	Object name	-	Object name of the application for which performance information is being measured
D6	Operation name	-	Operation name of the application for which performance information is being measured
D7	Process ID	-	Process ID of the application for which performance information is being measured
D8	Maximum request processing time	ms	The maximum processing time (within the interval time) of the operation processed by the concerned process
D9	Minimum request processing time	ms	The minimum processing time (within the interval time) of the operation processed by the concerned process
D10	Averaged request processing time	ms	The averaged processing time (within the interval time) of the operation processed by the concerned process
D11	Maximum request processing wait time	ms	Maximum wait time (within the interval time) from the time of a client request to the initiation of a server application
D12	Minimum request processing wait time	ms	Minimum wait time (within the interval time) from the time of a client request to the initiation of a server application
D13	Averaged request processing wait time	ms	Averaged wait time (within the interval time) from the time of a client request to the initiation of a server application
D14	Number of processes	Number	Number of times the operation has been executed by the process within the specified time interval
D15	Number of requests received	Number	Accumulated number of processes from the time of starting the performance monitoring up to the timing of this object.
D16	Number of requests awaiting processing	Number	The maximum number of requests waiting for processing of the concerned object in interval time

3.2.5.4 Stopping the Application

Execute the *isstopwu* command to start the WorkUnit.

3.2.5.5 Stopping the Performance Monitor

Performance Monitor is stopped with the *ispstop* command. When this command is executed, performance information is extracted and output on a performance log file stop.

3.2.5.6 Deleting the Performance Monitoring Environment

This section describes the deletion of the performance monitoring tool.

Deleting the Performance Monitoring Environment

Use the *ispdeleteenv* command to stop the Performance Monitoring Tool and delete the performance monitoring environment. Delete the performance monitoring environment after performance monitoring has stopped. If you do not do this, performance information will not be collected after the performance monitoring environment is deleted. To restart performance monitoring, create a new performance monitoring environment after restarting Interstage.

Stopping Interstage

Execute the *isstop* command to stop Interstage.

3.2.5.7 Deletion from the SNMP Service

Execute the *ispunsetagt* command to delete the Performance Monitoring Tool from the SNMP service. After executing the *ispunsetagt* command, use the Services dialog box to restart the SNMP service. This operation is required only when the Performance Monitoring Tool has been registered in the SNMP service by using the *ispsetagt* command.

- Windows®

Click [SNMP Service] for 'Administrative Tools' 'Services' and then execute [Stop] and [Start].

3.2.6 Analyzing the Performance Information and Taking Action

This section describes the method of analyzing the performance information collected in the performance log file and real-time monitoring, as well as the method for taking action.

3.2.6.1 Function of Outputting Log Information to the Performance Log File

This section describes the performance information that can be collected using the function of outputting log information to the performance log file, the evaluation method, and the method for taking action.

1. Collectable performance information

The function of outputting log information to the performance log file allows the accumulated information to be output in CSV format, by executing the *ispreport* command.

The performance information is described below.

Line format

"D1,D2,D3, D4,D5,D6, D7,D8,D9, D10,D11,D12,D13, D14,D15,D16"

Output items in each line

The following table shows a list of output items in each line. Items D1 to D16 in the 'Item No.' column correspond to those shown in 1).

Table 3.8 Output Items

Item no.	Performance information item name	Unit	Meaning
D1	Data collection start date	-	Date the performance information measurement for the record was started
D2	Data collection start time	-	Time the performance information measurement for the record was started

Item no.	Performance information item name	Unit	Meaning
D3	Data collection end date	-	Date the performance information measurement for the record was finished
D4	Data collection end time	-	Time the performance information measurement for the record was finished
D5	Object name	-	Object name of the application for which performance information is being measured
D6	Operation name	-	Operation name of the application for which performance information is being measured
D7	Process ID	-	Process ID of the application for which performance information is being measured
D8	Maximum request processing time	ms	The maximum processing time of the operation processed within interval time
D9	Minimum request processing time	ms	The minimum processing time of the operation processed within interval time
D10	Averaged request processing time	ms	The average processing time of the operation processed within interval time
D11	Maximum request processing wait time	ms	Maximum wait time from the time of request by the client to the initiation of the server application
D12	Minimum request processing wait time	ms	Minimum wait time from the time of request by the client to the initiation of the server application
D13	Average request processing wait time	ms	Average wait time from the time of request by the client to the initiation of the server application
D14	Number of processes	Number	Number of times the operation has been executed by the process within the specified time interval
D15	Number of requests received	Number	Accumulated number of processes, from the time of starting the performance monitoring to the timing of this object.
D16	Number of requests awaiting processing	Number	The maximum number of the requests waiting for processing to the concerned object in interval time

The *isreport* command outputs the interval time information in units of operations for each process. The output information consists of information for each process in units of operations and in units of objects.

The following is an example of output using the *isreport* command.

Example

The following is an example of the performance information output when monitoring the performance of objects OBJ001 and OBJ002. OBJ001 and OBJ002 have one operation. The process concurrency is 1.

Figure 3.8 Performance Information When Monitoring Objects

Interval time(start time,end time)	Object name	Operation name	Process ID	Request processing time	Request processing wait time	Number of processes received	Number of requests awaiting processing
2001-06-06, 22:36:27, 2001-06-06, 22:41:27,	OBJ001,	OPE011,	2692,	40, 0, 2,	250, 0, 5,	500,	500, 6
2001-06-06, 22:36:27, 2001-06-06, 22:41:27,	OBJ002,	OPE021,	2788,	30, 0, 2,	210, 0, 5,	100,	100, 5
2001-06-06, 22:41:27, 2001-06-06, 22:46:27,	OBJ001,	OPE011,	2692,	30, 0, 2,	140, 0, 6,	400,	900, 4
2001-06-06, 22:41:27, 2001-06-06, 22:46:27,	OBJ002,	OPE021,	2788,	30, 0, 2,	130, 0, 6,	200,	300, 3

} Data of interval time A
 } Data of interval time B

Information on an operation unit Information on an object unit

Example

The following is an example of the performance information output when monitoring the performance of an object OBJ001. OBJ001 has two operations. The process concurrency is 2.

Figure 3.9 Performance Information When Monitoring an Object

Interval time(start time,end time)	Object name	Operation name	Process ID	Request processing time	Request processing wait time	Number of processes received	Number of requests awaiting processing
2001-06-06, 22:36:27, 2001-06-06, 22:41:27,	OBJ001,	OPE001,	2692,	40, 0, 2,	250, 0, 5,	500,	1100,
2001-06-06, 22:36:27, 2001-06-06, 22:41:27,	OBJ001,	OPE002,	2692,	30, 0, 2,	130, 0, 6,	400,	1100,
2001-06-06, 22:36:27, 2001-06-06, 22:41:27,	OBJ001,	OPE001,	2788,	30, 0, 2,	210, 0, 5,	100,	1100,
2001-06-06, 22:36:27, 2001-06-06, 22:41:27,	OBJ001,	OPE002,	2788,	21, 0, 2,	230, 0, 6,	100,	1100,
2001-06-06, 22:41:27, 2001-06-06, 22:46:27,	OBJ001,	OPE001,	2692,	30, 0, 2,	140, 0, 6,	400,	2100,
2001-06-06, 22:41:27, 2001-06-06, 22:46:27,	OBJ001,	OPE002,	2692,	21, 0, 2,	130, 0, 6,	300,	2100,
2001-06-06, 22:41:27, 2001-06-06, 22:46:27,	OBJ001,	OPE001,	2788,	30, 0, 2,	130, 0, 6,	200,	2100,
2001-06-06, 22:41:27, 2001-06-06, 22:46:27,	OBJ001,	OPE002,	2788,	21, 0, 2,	140, 0, 6,	100,	2100,

} Data of interval time A
 } Data of interval time B

Information on an operation unit Information on an object unit

2. Evaluation and action

Reference each item in the following.

- Evaluation by operation (D8-D10, D14)

D8-D10 and D14 indicate the request processing time for the operation (indicated in D6 for the process indicated in D7) and the number of times the operation has been executed. With this information, a process can be evaluated for each operation.

- Evaluation by object (D11-D13, D15, D16)

D15 and D16 indicate the number of requests received, and the number of requests awaiting processing, for the object indicated in D5. With this information each object can be evaluated.

The methods of evaluating performance information and the actions to be taken are listed in the following table. If a performance abnormality was detected, take appropriate action using the table as a reference.

Table 3.9 Performance Items Details

Item no.	Performance information details	Action
1	In all the time slots when performance monitoring was performed, the maximum request processing time is long and the averaged request processing time is close to the maximum request processing time.	If the request processing time is longer than the target value, one or both of the following causes can be assumed: <ul style="list-style-type: none"> - The server application has a performance problem. - System workload is too high. Review the server application and system from the above viewpoints.
2	In a particular time slot, the maximum, minimum, and averaged request processing times are long. In a particular time slot, the maximum, minimum, and average request processing wait times are long	In a particular time slot, system workload is high. Check the workload status by measuring performance information of other server applications.
3	Although the maximum request processing time is long, the averaged request processing time is short and close to the minimum request processing time. Although the maximum request processing wait time is long, the averaged request processing wait time is short and close to the minimum request processing wait time.	One or both of the following causes can be assumed: <ul style="list-style-type: none"> - System work load have temporarily become high. - Under a particular condition, the server application has a performance problem. Review the server application and system from the above viewpoints.

Item no.	Performance information details	Action
4	In all the time slots when performance monitoring was performed, the maximum request processing wait time and average request processing wait time are long.	Processing capacity of the server application is insufficient for the number of requests from the client. Take action to increase the processing capacity, such as increasing the process multiplicity in the WorkUnit definition.
5	In a particular time slot, the number of processes and the number of requests waiting to be processed are high.	In a particular time slot, the number of requests to the server application has increased. If the processing capacity of the server application is insufficient for the number of requests from the client, take action to increase the processing capacity such as increasing the process multiplicity in the WorkUnit definition.

3.2.6.2 Performance Information Collected by the Network Control Manager with the Real Time Monitoring Function

This section describes the performance information that can be collected by the real time monitoring function, the method of evaluating it, and the various actions available.

1) Collectable performance information

The types of information that can be collected by using the real-time monitoring function are summarized below.

Table 3.10 Output Items

Performance information item name	Unit	Meaning
Object name	-	Object name of the application for which performance information is measured
Maximum request processing time	ms	Maximum processing time within the time required for processing the object
Minimum request processing time	ms	Minimum processing time within the time required for processing the object
Averaged request processing time	ms	Average processing time within the time required for processing the object
Maximum request processing wait time	ms	Maximum time within a polling time from the time of request by the client to the initiation of the server application
Minimum request processing wait time	ms	Minimum time from the time of request from by the client to the initiation of the server application
Averaged request processing wait time	ms	Average time from the time of request by the client to the initiation of the server application
Number of requests received	Number	Accumulated number of processes, from the time of starting the performance monitoring to the timing of this object.
Number of requests awaiting processing	Number	Maximum number of requests waiting for processing of this object, within the polling time.

2) Evaluation and action

The method of evaluating the performance information collected with the real time monitoring function and the method for taking action are listed in the table below.

If a performance abnormality was detected, take the action indicated in the table. Also, use the performance information output to the performance log file as a guide for evaluation.

Table 3.11 Performance Item Details

Item no.	Performance information details	Action
1	In each of the time slots when performance monitoring is performed, the maximum request processing time is long. The average request processing time is close to the maximum request processing time.	If the request processing time is longer than the target value, one or both of the following causes can be assumed: <ul style="list-style-type: none"> - There is a performance problem in the server application - System workload is too high Review the server application and system from the above viewpoints.
2	In a particular time slot, the maximum, minimum, and average request processing times are long.	In a particular time slot, the system workload is high. Check the workload status by measuring performance information of other server applications.
	In a particular time slot, the maximum, minimum, and average request processing wait times are long	
3	Although the maximum request processing time is long, the average request processing time is short and close to the minimum request processing time.	One or both of the following causes can be assumed: <ul style="list-style-type: none"> - System workload has temporarily become high - Under a particular condition, there is a performance problem in the server application Review the server application and system from the above viewpoints.
	Although the maximum request processing wait time is long, the averaged request processing wait time is short and close to the minimum request processing wait time.	
4	In all the time slots when performance monitoring was performed, the maximum request processing wait time and average request processing wait time are too long.	Processing capacity of the server application is insufficient for the number of requests from the client. Take action to increase the processing capacity, such as increasing the process multiplicity in the WorkUnit definition.
5	In a particular time slot, the number of processes and requests waiting to be processed are high.	In a particular time slot, the number of requests to the server application has increased. If the processing capacity of the server application is insufficient for the number of requests from the client, take action to increase the processing capacity such as increasing the process multiplicity in the WorkUnit definition.

3.2.6.3 Warnings Regarding the Evaluation of the Performance Information

There are a number of caveats about evaluating the performance information, which are shown below.

- If a value out of the range from 0 to 10,000 is specified as a reference value to the client application of the transaction application, its request is not reflected on the performance information.
- If the server application is terminated abnormally during processing, its request is not reflected in the performance information.

3.2.7 Managing the Performance Log Files

Make sure that there is enough disk space to create performance log files before starting the Performance Monitoring Tool. Use the following formula to estimate the required disk space.

```

Disk space required =
    shared-memory-size-specified-when-starting-performance-monitoring-tool
    x (time-from-when-performance-monitoring-tool-is-started-until-stopped
    / time-interval-specified-when-starting-performance-monitoring-tool)
```

Back up the performance log files and delete unnecessary files at regular intervals. Otherwise, the disk may be getting full. If the backed up and deleted files are to be output as a report, store these files in a folder and output a report with these files specified.

Performance log files will be created in the following folders.

- The folder specified in the *ispmakeenv* command parameter
- The folder specified in the ISP_LOG environment variable

Performance log files will be created in the folder specified in either the *ispmakeenv* command or the ISP_LOG environment variable according to the naming convention shown below. If both of them are specified, the folder name specified in the *ispmakeenv* command will take priority. If neither of them is specified, the folder name 'TD_HOME\isp\log' will be used as the default folder name. 'TD_HOME' is the interstage-installation-folder\td.

```
Performance log file name: ispYYYYMMDD.log
YYYYMMDD is a file creation date.
  YYYY: Year
  MM: Month (01 to 12)
  DD: Day (01 to 31)
```

The Performance Monitoring Tool creates a performance log file for the date when it is activated. If the Performance Monitoring Tool has been operated over several days, performance log files for these days will be created on the daily basis.

Note

After executing the *ispmakeenv* command, do not delete performance log files that are being created by the Performance Monitoring Tool. Otherwise, performance information may not be stored correctly. Use the *ispdeleteenv* command to delete performance log files.

3.2.8 Source Names of the Messages Displayed on the Event Viewer

For the source names of the messages output to the event viewer, either of the following can be selected.

F3FMis

'F3FMis' is shown as the source name displayed on the event viewer

ISPerf

'ISPerf' is shown as the source name displayed on the event viewer

In the initial setting, 'F3FMis' is output.

To output 'ISPerf', take the following steps.

Procedure

1. If the environment of the performance monitoring tool has already been created with the *ispmakeenv* command, delete the environment with the *ispdeleteenv* command.
2. Open the performance monitoring tool environment definition file (the following file) with the editor.

```
[Interstage installation folder]\td\etc\isp\ispconf.txt
```

3. Edit the line 'Msg Source=F3FMis' described in ispconf.txt as follows:

```
Msg Source=ISPerf
```

If a string other than 'ISPerf' is specified in 'Msg Source=', 'F3FMis' is displayed as the source name. To return the source name to 'F3FMis' after the source name is changed in the above procedure, specify 'Msg Source=F3FMis'.

3.3 Interstage Operation API

The Interstage operation API is used to operate WorkUnits and acquire information. This section covers the following points about the operation API group:

- [Function Overview](#)
- [Compiling and Linking Applications](#)
- [Examples of Use](#)

- [Notes](#)

The Interstage operation API provides an interface for the C language.

Note

Interstage operation APIs cannot be issued from multiple threads when they are used on a program operating in multi-thread.

The following functions of the Interstage operation API must not be issued from the pre-exit program and post-exit program of the WorkUnit, the WorkUnit exit program, or the process salvage exit program:

- Object information notification
- WorkUnit start
- WorkUnit stop
- Object close(*1)
- Cancel object close(*1)
- Information acquisition of the object in the Implementation Repository ID

*1 Object closure and object closure cancel are only supported in the Enterprise Edition.

3.3.1 Function Overview

The operation APIs provide the following functions:

- [Interstage Operation API Environment Initialization and Collection](#)

The initialization and collection function for using Interstage Operation APIs

- [Interstage Operation Information Notification](#)

The function used to acquire WorkUnit and object-related information

- [Interstage Operation](#)

The function used to start/stop WorkUnits and block/unblock objects

Solaris32

- [Interstage System Information Notification](#)

The function of acquiring the Interstage system list information

Note

Object closure, object closure cancel, and the reporting of Interstage system information are only supported in the Enterprise Edition.

3.3.1.1 Interstage Operation API Environment Initialization and Collection

The ISOPenv function initializes and collects the Interstage Operation API environment.

Interstage Operation API Environment Initialization

Initializes the Interstage Operation API environment. This API must be used to initialize the Interstage Operation API environment before using the Interstage Operation API. Issue this API in the application process only once at the beginning of the application.

Interstage Operation API Environment Collection

Collects the Interstage Operation API environment. Issue this API before terminating the application process.

3.3.1.2 Interstage Operation Information Notification

The ISOPnotify function reports Interstage operation information.

WorkUnit Name Notification

Reports information on all the WorkUnits defined when API is issued. Using this API repeatedly can retrieve a list of WorkUnit names.

This API provides the following information in addition to WorkUnit names.

- WorkUnit type

Object Name Notification

Reports information on the objects defined when API is issued. Using this API repeatedly can retrieve a list of object names registered in the specified WorkUnit.

This API provides the following information in addition to object names.

- Total number of objects registered in the WorkUnit
- Object type
- Number of concurrent processes when definitions registered
- DPCF communication path name Windows32 Solaris32
- Maximum number of queues
- Number of queues monitored
- Number of queues monitored and restarted
- EJB application mode
- Number of instances

The information posted depends on the type of WorkUnit.

Notes

- The meaning of 'number of concurrent processes' depends on whether the object is resident, or non-resident or multi-object non-resident.

For resident objects, this is the number of concurrent processes for the object.

For non-resident or multi-object non-resident objects, this refers to the maximum number of concurrent processes being run simultaneously by all non-resident and multi-object non-resident objects defined in the WorkUnit.

- Information on non-resident and multi-object non-resident object numbers of queues and numbers of requests collected by the information notification function relate to each object.

Windows32 Solaris32

- The maximum number of queues for WRAPPER WorkUnit type objects is adjusted to the system maximum number of queued sessions in the environment definitions for the Component Transaction Service. This value applies to the whole system, not each DPCF communication path.

If the DPCF communication path name in the WorkUnit definitions is shown in the Component Transaction Service environment definitions, the system maximum number of queued sessions is reported as the maximum number of queues.

If the DPCF communication path name in the WorkUnit definitions is not shown in the Component Transaction Service environment definitions, no load restrictions apply to the path. In this case, the system maximum number of queued sessions is ineffective, and the maximum number of queues will be reported as 0.

- Details on the number of queues monitored and the number of queues for which monitoring is restarted are only collected in the Enterprise Edition.

WorkUnit Information Notification

Reports WorkUnit information when API is issued. This API provides one of the following states: 'running', 'starting', 'stopping', and 'stopped' (stopped by API, stopped by command, or abnormal stop).

Object Information Notification

Reports object information when API is issued.

This API provides the following information in addition to object status:

- State of object
- DPCF communication path state Windows32 Solaris32
- Number of queues
- Number of cumulative processes
- Number of concurrent processes running
- EJB application mode
- Number of instances
- Queues monitoring state

The information reported depends on the type of WorkUnit in which the specified object is defined.

Note

Queue monitoring state details are only collected in the Enterprise Edition.

3.3.1.3 Interstage Operation

The ISOPoperate function reports Interstage operation.

WorkUnit Startup

Starts a WorkUnit. WorkUnit startup is synchronized. When a WorkUnit is started, the server application is also started.

WorkUnit Stop

Stops a WorkUnit. WorkUnit stop is synchronized. When a WorkUnit is stopped, the server application is also stopped. This API can also stop the WorkUnit started by the *tdstartwu* and *isstartwu* commands.

Object Close

Closes the specified object

Cancel Object Closure

Cancels the closure of the specified object

Note

The object closure and cancel object closure functions can be used if the Interstage Application Server Enterprise Edition is installed.

3.3.1.4 Interstage System Information Notification

Solaris32

The ISOPsystem function provides the Interstage system information notification function.

System Name List Notification

A list of systems generated when the API is issued can be extracted.

System Name List Release

Releases the area acquired when the system name list notification API was issued.

Note

The Interstage Operation API environment does not need to be initialized when this API is used. Issue this API with superuser authority. System name list reporting and system name list release can be used when Interstage Application Server Enterprise Edition is installed. This function cannot be used in Interstage Application Server Enterprise Edition.

3.3.2 Compiling and Linking Applications

Specify the following include file when compiling applications that use Interstage Operation API.

Include file name	Storage location
ISOP.h	Interstage installation folder\td\include

Specify the following library when linking applications that use Interstage Operation API.

Library name	Storage location	Use
F3FMistage.lib	Interstage installation folder\td\lib	Operation API run time (mandatory)

3.3.3 Examples of Use

The Interstage Operation API initializes and collects the environment. If the environment has failed to be initialized, collection processing will not be required.

Windows32

```
#include "ISOP.h"
~
ISOP_CTRL env;
/* initialize environment */
memset( (char *)&env, 0x0, sizeof( ISOP_CTRL ) );
env.request = ISOP_RINIENV;
env.pktvl = ISOP_006;
ISOPenv( (char *)&env );
if( env.result == -1 ){
    /* abnormal termination: Environment collection not required*/
    /* error processing */
}
~ /* execute Operation API*/
/* collect environment */
memset( (char *)&env, 0x0, sizeof( ISOP_CTRL ) );
env.request = ISOP_RTRMENV;
env.pktvl = ISOP_006;
ISOPenv( (char *)&env );
if( env.result == -1 ){
    /* abnormal termination */
    /* error processing */
}
}
```

When using the WorkUnit name or object name notification API to obtain multiple names, specify the same control table that was returned previously. Doing so will count the number of WorkUnit names or object names that have already been obtained. When all of the names are retrieved, "0" will be returned as the result.

```
~
ISOP_LSTWU lstwu;
~
memset( (char *)&lstwu, 0x0, sizeof( ISOP_LSTWU ) );
/* settings for obtaining the first name */
lstwu.opctrl.request = ISOP_RLSTWU;
lstwu.opctrl.pktvl = ISOP_006;
lstwu.checknum = 0;
do{ /*This loop retrieves all the names */
```

```

ISOPnotify( (char *)&lstwu );
    if( lstwu.opctrl.result == -1 ){
/* error processing */
/* Also processed here when no WorkUnit is found */
break;
    }
    ~ /* each processing */
}while( lstwu.opctrl. result == 1);

```

Solaris32

```

#include "ISOP.h"
#define DEF_ISOPENV    "ISOPenv"
~
ISOP_CTRL env;
void *handle;
void ( *ISOPENV )( char * );
int rtncode;
    /* open library */
handle = dlopen( "/opt/FSUNtd/lib/libistage.so", RTLD_LAZY );
if( handle == NULL ){
    /* abnormal termination */
    /* error processing */
}
/* obtain function address */
ISOPENV = (void (*)(char *))dlsym( handle, DEF_ISOPENV );
if( ISOPENV == NULL ){
    /* abnormal termination */
    /* error processing */
}
/* initialize environment */
memset( (char *)&env, 0x0, sizeof( ISOP_CTRL ) );
env.request = ISOP_RINIENV;
env.pktvl = ISOP_006;
(*ISOPENV)( (char *)&env );
if( env.result == -1 ){
    /* abnormal termination no need to collect environment */
    /* error processing */
}
~ /* execute Operation API */
    /* collect environment */
memset( (char *)&env, 0x0, sizeof( ISOP_CTRL ) );
env.request = ISOP_RTRMENV;
env.pktvl = ISOP_006;
(*ISOPENV)( (char *)&env );
if( env.result == -1 ){
    /* abnormal termination */
    /* error processing */
}
/* close library */
rtncode = dlclose( handle );
if( rtncode != 0 ){
    /* abnormal termination */
    /* error processing */
}

```

When using the WorkUnit name or object name notification API to obtain multiple names, specify the same control table that was returned previously. Doing so will count the number of WorkUnit names or object names that have already been obtained. When all of the names are retrieved, "0" will be returned as the result.

```

~
ISOP_LSTWU lstwu;
void ( *ISOPNOTIFY )( char * );

```

```

~
memset( (char *)&lstwu, 0x0, sizeof( ISOP_LSTWU ) );
/* settings for obtaining the first name */
lstwu.opctrl.request = ISOP_RLSTWU;
lstwu.pktv1 = ISOP_006;
lstwu.checknum = 0;
do{ /* This loop retrieves all the names */
    (*ISOPNOTIFY)( (char *)&lstwu );
    if( lstwu.result == -1 ){
        /* error processing */
        /* Also processed here when no WorkUnit is found */
        break;
    }
}
~ /* each process */
}while( lstwu.result == 1 );

```

When the Interstage system information notification API is used for system name list notification, a list of all system names can be obtained by one invocation.

To release the area of the system name list information retrieved, specify the invocation parameter structure used for system name list notification when the system name list release API is invoked.

```

~
ISOP_LSTSYS lstsys;
ISOP_LSTSYSDATA *data;
int i;
~

/* System name list acquisition */
memset(&lstsys, 0, sizeof(ISOP_LSTSYS));
lstsys.opctrl.request = ISOP_RLSTSYS;
lstsys.opctrl.pktversion = ISOP_006;

/* API invocation */
ISOPsystem((char*)&lstsys);
if( lstsys.opctrl.result != 0 ) {
    /* Abnormal end */
    /* Error processing */
}

/* Referencing to system name list information */
data = lstsys.bufaddr;
for(i=0; I < lstsys.num; i++, data++) {
    /* System name */
    printf("SYSTEM:%s\n", data->sysname);
    /* Profile */
    printf("PROFILE:%s\n", data->profile);
}

/* System name list release */
lstsys.opctrl.request = ISOP_RLSTFRESYS;
ISOPsystem((char *)&lstsys);
if( lstsys.opctrl.result != 0 ) {
    /* Abnormal end */
    /* Error processing */
}

```

Linux32

```

#include "ISOP.h"
#define DEF_ISOPENV    "ISOPenv"
~
ISOP_CTRL env;

```



```

void *handle;
void ( *ISOPENV )( char * );
int rtncode;
    /* open library */
handle = dlopen( "/opt/FJVSvd/lib/libistage.so", RTLD_LAZY );
if( handle == NULL ){
    /* abnormal termination */
    /* error processing */
}
/* obtain function address */
ISOPENV = (void (*)(char *))dlsym( handle, DEF_ISOPENV );
if( ISOPENV == NULL ){
    /* abnormal termination */
    /* error processing */
}
/* initialize environment */
memset( (char *)&env, 0x0, sizeof( ISOP_CTRL ) );
env.request = ISOP_RINIENV;
env.pktvl = ISOP_006;
(*ISOPENV)( (char *)&env );
if( env.result == -1 ){
    /* abnormal termination no need to collect environment */
    /* error processing */
}
~ /* execute Operation API */
    /* collect environment */
memset( (char *)&env, 0x0, sizeof( ISOP_CTRL ) );
env.request = ISOP_RTRMENV;
env.pktvl = ISOP_006;
(*ISOPENV)( (char *)&env );
if( env.result == -1 ){
    /* abnormal termination */
    /* error processing */
}
/* close library */
rtncode = dlclose( handle );
if( rtncode != 0 ){
    /* abnormal termination */
    /* error processing */
}
}

```

When using the WorkUnit name or object name notification API to obtain multiple names, specify the same control table that was returned previously. Doing so will count the number of WorkUnit names or object names that have already been obtained. When all of the names are retrieved, "0" will be returned as the result.

```

~
ISOP_LSTWU lstwu;
void ( *ISOPNOTIFY )( char * );
~
memset( (char *)&lstwu, 0x0, sizeof( ISOP_LSTWU ) );
/* settings for obtaining the first name */
lstwu.opctrl.request = ISOP_RLSTWU;
lstwu.pktvl = ISOP_006;
lstwu.checknum = 0;
do{ /* This loop retrieves all the names */
    (*ISOPNOTIFY)( (char *)&lstwu );
    if( lstwu.result == -1 ){
        /* error processing */
        /* Also processed here when no WorkUnit is found */
        break;
    }
}
~ /* each process */
}while( lstwu.result == 1 );

```

3.3.4 Notes

This section provides notes on using the Interstage Operation API. Do not use the *start* command provided by each service.

3.3.4.1 Command Operations

This section describes system behavior that occurs when this API conflicts with each command.

- Deleting WorkUnit definitions (*tddeldef/isdelwundef* commands)
- Stopping WorkUnits (*tdstopwu/isstopwu* commands)
- Changing WorkUnit definitions (*tdadddef/isaddwundef* commands)
- Changing WorkUnit definitions dynamically (*tdmodifywu* command)
- Changing the number of concurrent server applications (*tdmodifyprocnum* command)
- Adding WorkUnit definitions (*tdadddef isaddwundef* commands)
- Starting WorkUnits (*tdstartwu/isstartwu* commands)
- Stopping Interstage (*isstop* command)

Solaris32

- Adding the system to be operated (*iscreatesys* command)
- Deleting the system to be operated (*isdeletesys* command)

The Operation API's behavior in each of the above situations is described below. We recommend that the Operation APIs be used so that they do not conflict with these commands which change status.

Note

The functions for adding and removing systems to be operated are only supported in the Enterprise Edition.

Deleting WorkUnit Definitions

When WorkUnit definitions are deleted, the function for specifying WorkUnits will set `ISOP_ENOWU` in the detailed error information. To obtain the latest status, issue the WorkUnit name notification API and the object name notification API.

Stopping WorkUnits

Use the WorkUnit information notification API to check whether the WorkUnit has stopped. This API can also obtain the status of the WorkUnit stopped by the *tdstopwu* or *isstopwu* commands or abnormal stop status.

Changing WorkUnit Definitions and Changing WorkUnits Dynamically

The WorkUnit name notification API and the object name notification API always obtain the latest definitions.

Changing the Number of Concurrent Server Applications

When the number of concurrent server applications is dynamically changed, the number of concurrent server applications that is reported by the object name notification API is the number before the dynamic change. However, the object information notification API will return the latest dynamically changed number. If the number of concurrent server applications has been dynamically changed, use the object information notification API to return the number.

Adding WorkUnit Definitions

After adding WorkUnit definitions, use the WorkUnit name notification API to retrieve all the defined WorkUnit names.

Starting WorkUnits

When a WorkUnit has been started by the *tdstartwu* or *isstartwu* command, the WorkUnit information notification API will not report the reason for WorkUnit stop.

Adding the System to be Operated Solaris32

After adding a system to be operated, use the system name list notification API function to retrieve all of the generated system names.

Note

The addition of systems to be operated is a function only supported in the Enterprise Edition.

Deleting the System to be Operated Solaris32

After the system to be operated is deleted, ISOP_ENOSYSTEM is posted to the detailed error information if an API is issued with the deleted system name specified. To obtain the latest status, issue the system name list notification API.

Note

The addition of systems to be operated is a function only supported in the Enterprise Edition.

Stopping Interstage

When the system to be operated is stopped, ISOP_ESTPIS is posted to the detailed error information for all functions excluding system information notification.

3.3.4.2 Starting and Stopping WorkUnits

This API can stop a WorkUnit in normal stop or forced stop mode.

If the WorkUnit is stopped in normal stop mode when it is communicating with a client (linked to a client), an error (ISOP_EREQREJECT) will be returned. When a WorkUnit is stopped in synchronous stop mode, the request being processed is executed, then the WorkUnit is stopped.

The notes below explain the following points about stopping WorkUnits:

- [Forced WorkUnit Stop](#)
- [Collecting WorkUnit Information](#)

Forced WorkUnit Stop

When a WorkUnit is forcibly stopped, APIs may terminate abnormally but the WorkUnit may fail to stop. If the WorkUnit definitely must be stopped, use the WorkUnit information notification function to check, and take the appropriate action.

Collecting WorkUnit Information

The WorkUnit information notification function returns the following statuses as the stop modes in the return information **stat**. Take the action appropriate to each case.

- ISOP_DWUSTOP

The Interstage Operation API has stopped the WorkUnit.

- ISOP_DWUSTOP_COM

The WorkUnit started by the Interstage Operation API has been stopped by the *tdstopwu* or *isstopwu* command.

- ISOP_DWUSTOP_ABEND

The WorkUnit started by the Interstage Operation API has terminated abnormally.

Solaris32 Linux32

- ISOP_DWUSTOP_AUTO

The utility WorkUnit started by the Interstage Operation API has been stopped automatically.

3.3.4.3 Operation in the Cluster System

Windows32

When using Interstage Operation APIs in an environment in which Interstage operates in a cluster system, use the following service.

- Interstage API service

This service is registered when installing Interstage.

For a concrete operation method, refer to High Availability System Guide.

Use this service only when operating Interstage in the cluster configuration. If Interstage is operated in a configuration other than the cluster configuration, do not start this service.

3.3.4.4 Control Table Version-Level for Interstage Operation API

This section describes the following notes applicable when the version of the parameter structure control table for the Interstage Operation API is upgraded or when the old version of the control table is used.

The Interstage Operation API compares the value for the version of the control table set as a parameter when the API is called, and determines the range of functions used by the API.

If the value for the version of the control table specified as a parameter when the API is called is older than, or the same as, the current version, the range of functions supported up to the version specified by the API is used. In this case, the API simply returns the value for the version of the control table set as the parameter.

If the value for the version of the control table specified as a parameter when the API is called is newer than the current version, the API uses the range of functions supported up to the current version, but does not use the range of functions supported by the newer version. In this case, the API returns the value for the current version in the area for the version of the control table set as the parameter.

The following table lists the values for the versions of control tables set when an API is called, and the values for the version of the control table used and returned by the API as the current version.


Table 3.12 Control Table Values

Value of version of control table set when API called	API uses	Value of version of control table set when API returned
Windows32 Solaris32 ISOP_001	Functions supported by ISOP_001	ISOP_001
Windows32 Solaris32 ISOP_002	Functions supported by ISOP_002	ISOP_002
Windows32 Solaris32 ISOP_003	Functions supported by ISOP_003	ISOP_003
ISOP_005	Functions supported by ISOP_005	ISOP_005
ISOP_006	Functions supported by ISOP_006	ISOP_006
Other value	Functions supported by the latest control table version of each API	Windows32 Linux32 ISOP_006 Solaris32 System name list notification: ISOP_001 System name list release: ISOP_001 Other value: ISOP_006

3.3.4.5 Parameter Information used by Interstage Operation API

The Interstage Operation API uses structure members, some of which are defined in Interstage. These definitions are shown in the following table. Always append \0 to the end of each character string.

Table 3.13 Structure Member Definitions

Member name included in structure	Size	Settable value
WorkUnit name	36 bytes	The WorkUnit name can contain up to 36 bytes of alphabetic letters, numbers, hyphens and underscores, and must begin and end with an alphabetic letter.
Object name	255 bytes	<p>When type of the WorkUnit is ORB and WRAPPER, an object name is a character string that consists of alphanumeric characters, underscore and slash of 255 bytes or less including one or more slash and starts with an alphanumeric character. Slash cannot be used for the top and the last letters and two or more consecutive slashes cannot be used.</p> <p>When the WorkUnit type is EJB, an object name is a character string (of 255 bytes or less) that consists of alphanumeric characters, hyphens, slashes, colons, periods or underscores.</p> <p>A slash cannot be used for the first or last character and two or more consecutive slashes cannot be used.</p>
 System name	8 bytes	Alphanumeric characters in up to 8 bytes

3.4 Changing WorkUnits

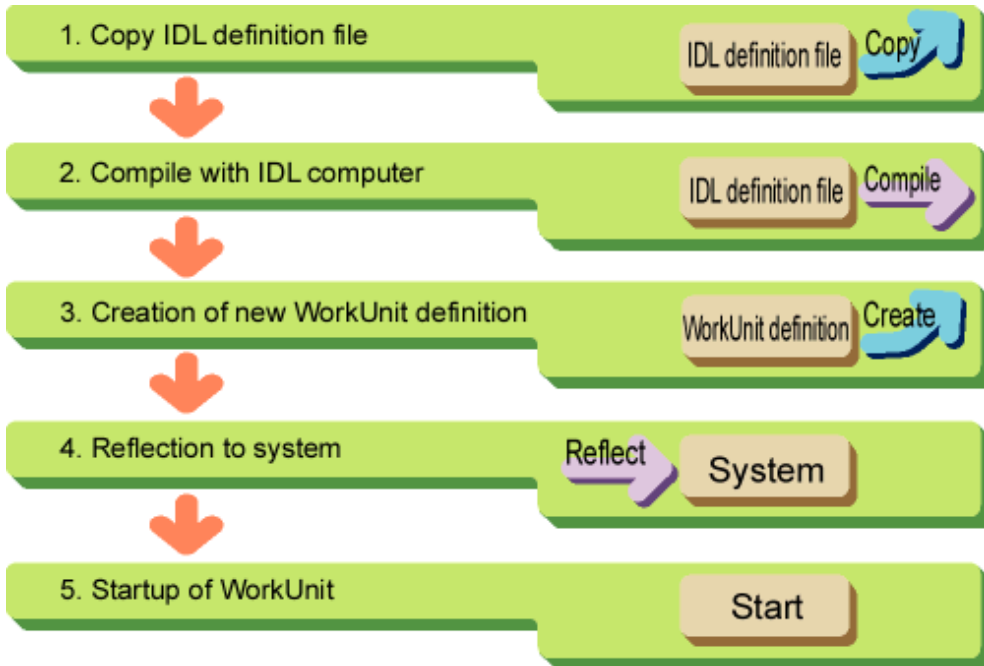
Transaction and EJB applications can be run as WorkUnits. This section explains how to add, change and delete the WorkUnits of transaction and EJB applications.

- [Adding a WorkUnit \(Transaction Applications\)](#)
- [Adding a WorkUnit \(EJB Applications\)](#)
- [Deleting a WorkUnit](#)
- [Changing a WorkUnit](#)

3.4.1 Adding a WorkUnit (Transaction Applications)

The following figure shows the procedure for adding a new transaction application WorkUnit to the system during operation.

Figure 3.10 Adding a New Transaction Application WorkUnit



1) Copy IDL Definition File

Copy the application program and the IDL definition file created in the start-up environment to the application environment.

2) Compile with IDL Compiler

Use the IDL compiler to compile the IDL file in the application environment. An execution module for the added application is created on the basis of the generated skeleton code. It is recommended that the interface information check function be used when compiling the IDL definition file. For details on the interface information check function, refer to "Operation Using the Interface Information Check Function" in the "Notes on OLTP Server Operations" appendix.

3) Create New WorkUnit Definitions

Create a new definition for the added WorkUnit

4) Update System

Use the *isaddwundef* command to copy the WorkUnit definition to the system.

```
isaddwundef create-def.wu
```

If you specify the *-o* option, then any existing definition with the same name will be overwritten. If no definition of that name exists, then a new definition is registered. If you omit the *-o* option, the file will only be registered if there is no existing definition of the same name.

Note

Transaction application WorkUnits can also be registered with the *tdadddef* command.

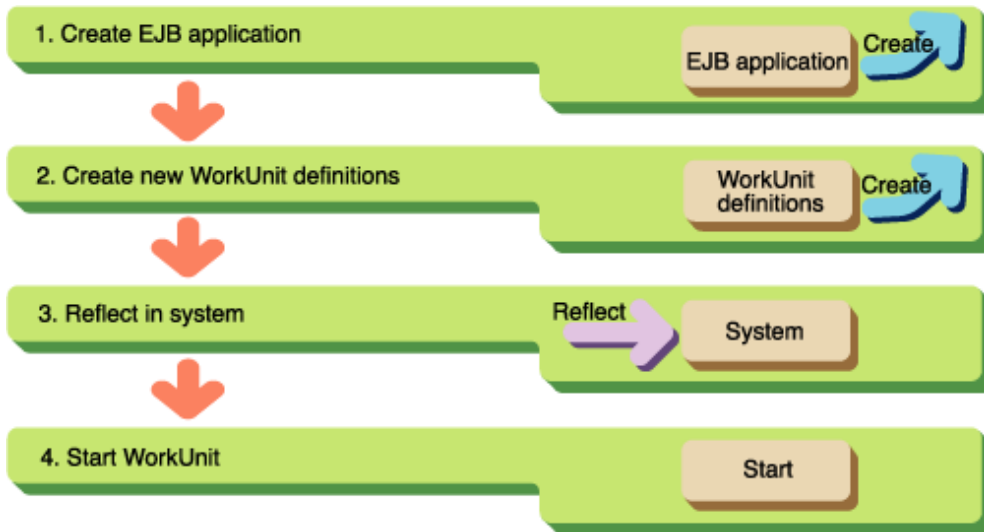
5) Start WorkUnit

Start up the added WorkUnit.

3.4.2 Adding a WorkUnit (EJB Applications)

The following figure shows the procedure for adding a new EJB application WorkUnit to the system during operation.

Figure 3.11 Adding a New EJB Application WorkUnit



1) Create New EJB Application

Create the new EJB application.

2) Create New WorkUnit Definitions

Create a new definition for the added WorkUnit

3) Update System

Use the *isaddwundef* command to copy the WorkUnit definition to the system.

```
isaddwundef create-def.wu
```

If you specify the *-o* option, then any existing definition with the same name will be overwritten. If no definition of that name exists, then a new definition is registered. If you omit the *-o* option, the file will only be registered if there is no existing definition of the same name.

4) Start WorkUnit

Start up the added WorkUnit.

3.4.3 Deleting a WorkUnit

The procedure for deleting WorkUnits from the system during operation is illustrated in the following figure.

When a CORBA WorkUnit is used, this procedure can be executed using the Interstage management console.

Figure 3.12 Deleting a WorkUnit



1) Terminate WorkUnit

If the WorkUnit to be deleted is active, you must close it down.

```
isstopwu TDSAMPLE1
```

2) Delete WorkUnit

Delete the corresponding WorkUnit definition.

```
isdelwundef TDSAMPLE1
```

Note

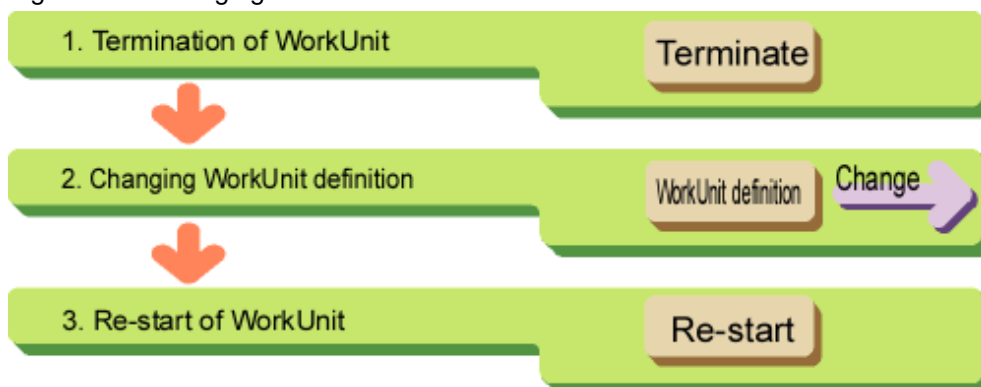
Transaction application WorkUnits can also be stopped with the *tdstopwu* command. WorkUnit definitions can also be deleted with the *tddeldef* command.

3.4.4 Changing a WorkUnit

This section describes the procedure for changing a WorkUnit, when the resource (database name or file name) used by an application has been altered. The following figure outlines this procedure. These changes can be made during operation. You should also follow this sequence of operations after making any changes to application processing, which become necessary due to alterations in resource allocation.

When a CORBA WorkUnit is used, this procedure can be executed using the Interstage management console.

Figure 3.13 Changing a WorkUnit



1) Terminate WorkUnit

Terminate the WorkUnit to be changed.

```
isstopwu TDSAMPLE1
```

2) Change WorkUnit Definition

Change the WorkUnit definition. Copy the new definition to the system with the *isaddwundef* command.

```
isaddwundef -o create-def.wu
```

3) Re-start WorkUnit

Re-start the WorkUnit.

```
isstartwu TDSAMPLE1
```

Note

Transaction application WorkUnits can also be started and stopped with the *tdstartwu* and *tdstopwu* commands respectively. WorkUnit definitions can also be registered with the *tdadddef* command.

3.5 Changing Server Applications

Server applications can be changed in the following ways. The procedures for adding and modifying transaction and EJB applications are different.

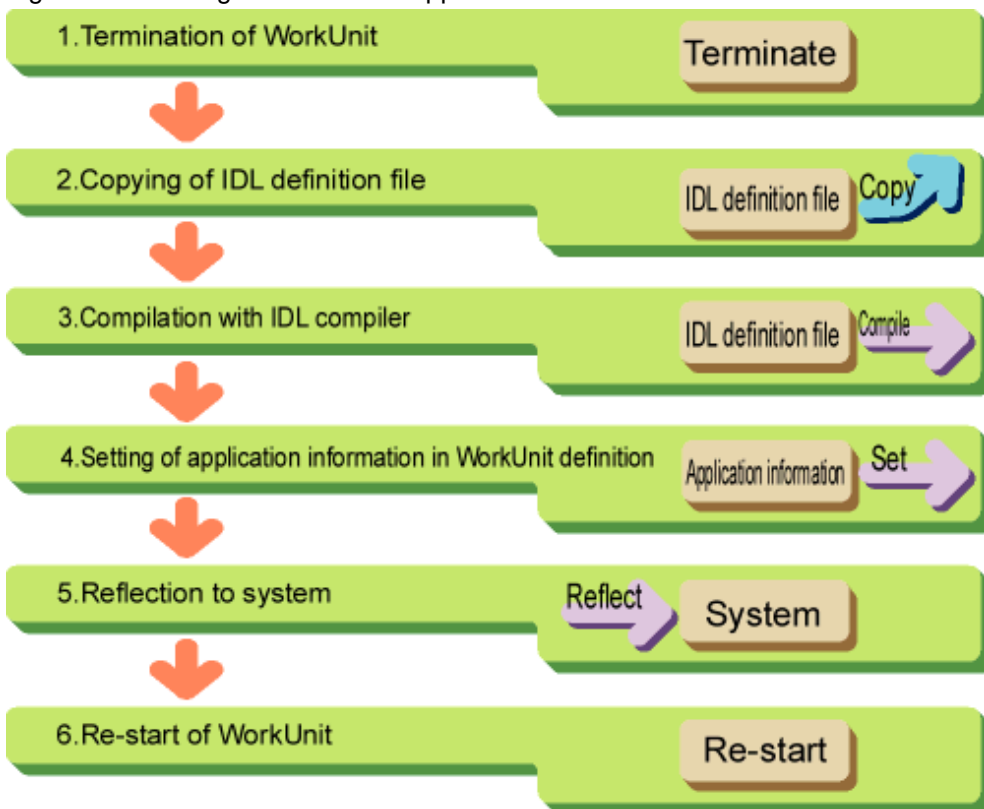
Active change is available for transaction applications only.

- Adding a Server Application (Transaction Application)
- Adding a Server Application (EJB Application)
- Deleting Server Applications
- Changing a Server Application (Transaction Application)
- Changing a Server Application (EJB Application)
- Active Changing of a Server Application (Transaction Applications Only)
- Dynamic Changing of the Number of Server Application Processes

3.5.1 Adding a Server Application (Transaction Application)

The procedure for adding a transaction application to an active WorkUnit is illustrated in the following figure.

Figure 3.14 Adding a Transaction Application



1) Terminate WorkUnit

Terminate the WorkUnit to which you are going to add the new application.

```
isstopwu TDSAMPLE1
```

2) Copy IDL Definition File

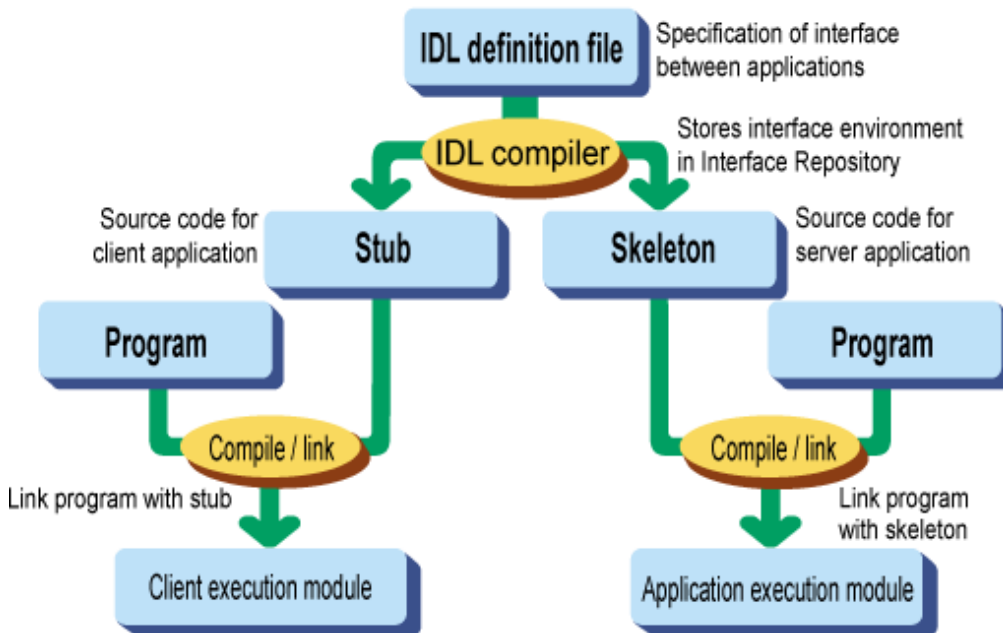
Copy the application program and the IDL definition file, created in the startup environment, to the application environment.

3) Compile with IDL Compiler

Use the IDL compiler to compile the IDL definitions in the application environment as shown in the following Figure. Then create an execution module for the added application on the basis of the generated skeleton code. Next, regenerate the application on the client-side execution module, using the stub created at the same time as the compile operation.

It is recommended that the interface information check function be used when compiling the IDL definition file. For details on the interface information check function, refer to "Operation Using the Interface Information Check Function" in the "Notes on OLTP Server Operations" appendix.

Figure 3.15 Compiling the IDL Definitions



4) Set the Application Information in the WorkUnit Definition File

Set information relating to the new application in the WorkUnit definition file.

5) Update System

Use the *isaddwundef* command to copy the altered WorkUnit definition file to the system.

```
isaddwundef create-def.wu
```

If you specify the *-o* option, then any existing definition with the same name will be overwritten. If no definition of that name exists, then a new definition is registered. If you omit the *-o* option, the file will only be registered if there is no existing definition of the same name.

6) Re-start WorkUnit

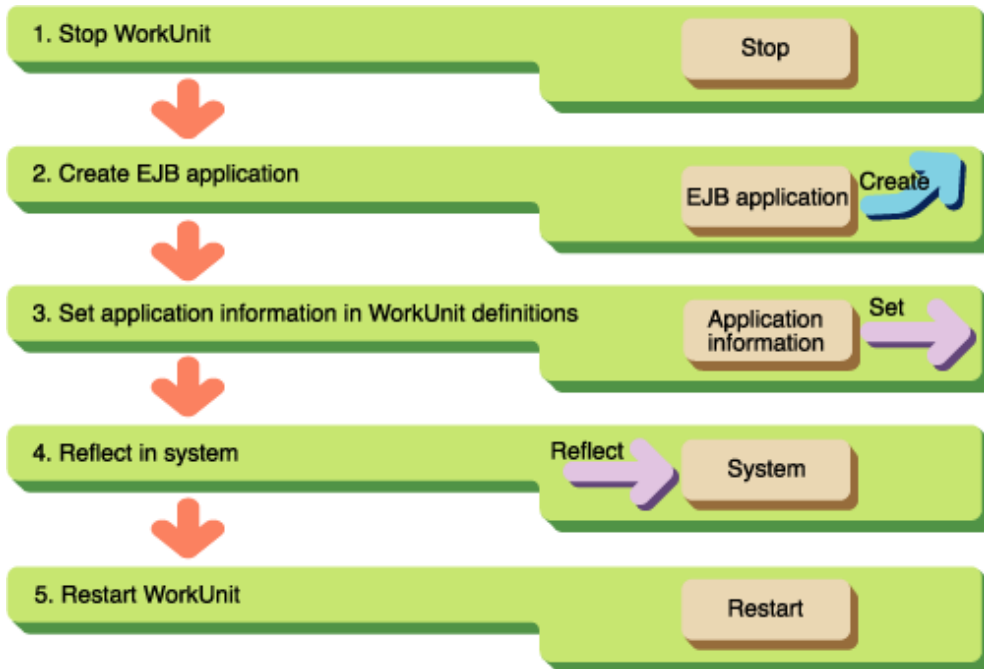
Re-start the WorkUnit that has been changed.

```
isstartwu TDSAMPLE1
```

3.5.2 Adding a Server Application (EJB Application)

The procedure for adding an EJB application to an active WorkUnit is illustrated in the following figure.

Figure 3.16 Adding an EJB Application



1) Terminate WorkUnit

Terminate the WorkUnit to which you are going to add the new application.

```
isstopwu EJBSAMPLE1
```

2) Create EJB Application

Create the EJB application.

3) Set the Application Information in the WorkUnit Definition File

Set information relating to the new application in the WorkUnit definition file.

4) Update System

Use the *isaddwundef* command to copy the altered WorkUnit definition file to the system.

```
isaddwundef EJBWU.wu
```

If you specify the *-o* option, then any existing definition with the same name will be overwritten. If no definition of that name exists, then a new definition is registered. If you omit the *-o* option, the file will only be registered if there is no existing definition of the same name.

5) Re-start WorkUnit

Re-start the WorkUnit that has been changed.

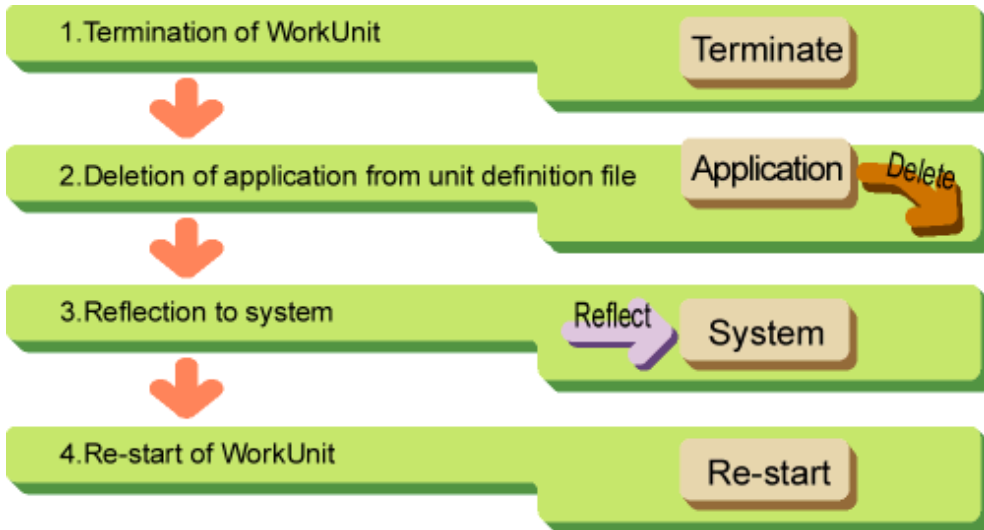
```
isstartwu EJBWU
```

3.5.3 Deleting Server Applications

An application is deleted from an active WorkUnit by the procedure illustrated in the following figure.

This procedure can be executed using the Interstage management console.

Figure 3.17 Deleting an Application from an Active WorkUnit



1) Terminate the WorkUnit

Terminate the WorkUnit containing the application that is to be deleted.

```
isstopwu TDSAMPLE1
```

2) Delete the Application from the WorkUnit Definition

Delete the relevant application from the WorkUnit definition file.

3) Update the System

Use the *isaddwundef* command to copy the altered WorkUnit definition file to the system.

```
isaddwundef -o create-def.wu
```

4) Re-start WorkUnit

Re-start the WorkUnit that has been changed.

```
isstartwu TDSAMPLE1
```

Note

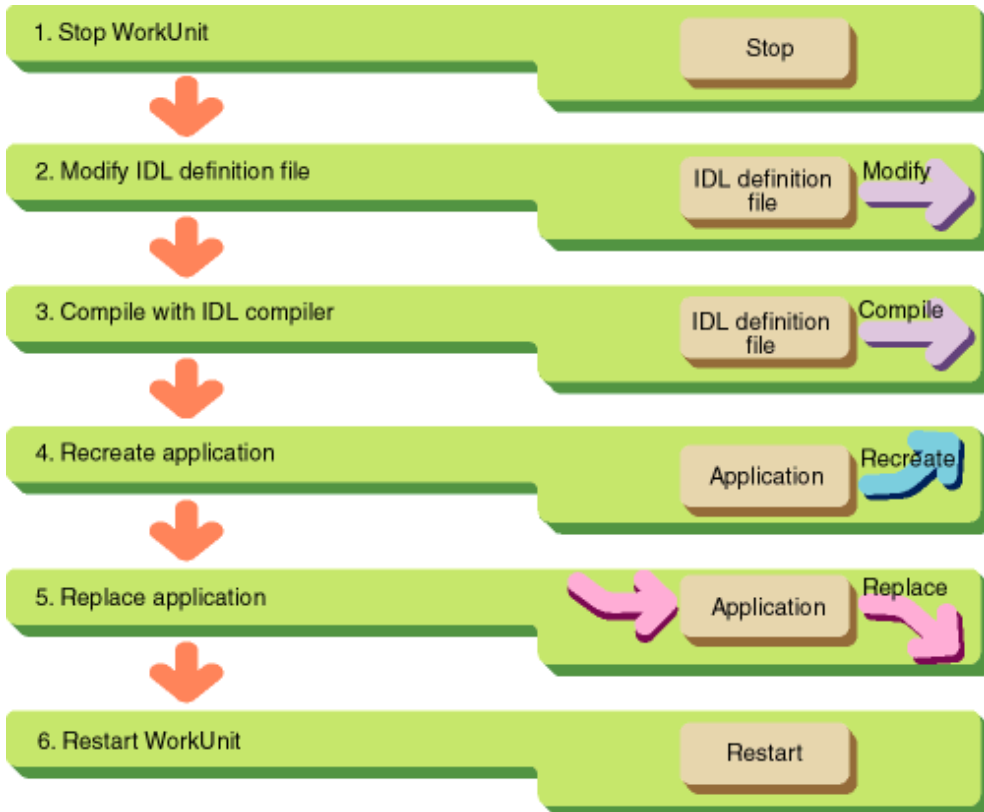
Transaction application WorkUnits can also be started and stopped with the *tdstartwu* and *tdstopwu* commands respectively. WorkUnit definitions can also be registered with the *tdadddef* command.

3.5.4 Changing a Server Application (Transaction Application)

The procedure for changing a transaction application in an active WorkUnit is illustrated in the following figure.

If the import/output elements in the client have altered, due to changes in business processing, or if the interface presented to the application has changed, the IDL definition information must also be modified. The method for changing an application, and modifying the IDL definition file, is described in this section.

Figure 3.18 Changing a Transaction Application in an Active WorkUnit



1) Terminate the WorkUnit

Terminate the WorkUnit in which the application is going to be changed.

```
isstopwu TDSAMPLE1
```

2) Modify the IDL Definition File

Modify the IDL definition file.

3) Compile with IDL Compiler

Use the IDL compiler to generate a skeleton and stub from the IDL definition file.

It is recommended that the interface information check function be used when compiling the IDL definition file.

4) Recreate Application

Modify the application and create an execution module using the generated skeleton code. Next, regenerate the client-side execution module, using the generated stub code.

5) Replace Application

To replace the existing application with a new one, you should change the contents of the execution module, rather than altering the path name (application library path) of the execution module.

6) Re-start WorkUnit

Re-start the WorkUnit.

```
isstartwu TDSAMPLE1
```

Note

Transaction application WorkUnit can also be started and stopped with the *tdstartwu* and *tdstopwu* commands respectively.

3.5.5 Changing a Server Application (EJB Application)

The procedure for changing an EJB application in an active WorkUnit is illustrated in the following figure.

Figure 3.19 Changing an EJB Application in an Active WorkUnit



1) Terminate the WorkUnit

Terminate the WorkUnit in which the application is going to be changed.

```
isstopwu EJBWU
```

2) Recreate the Application

Recreate the EJB application.

3) Restart WorkUnit

Restart the WorkUnit.

```
isstartwu EJBWU
```

3.5.6 Active Changing of a Server Application (Transaction Applications Only)

You can change an application in an active WorkUnit and update the system accordingly, without having to terminate the WorkUnit (this is called active changing).

The active changing of a server application is supported only in the Enterprise Edition.

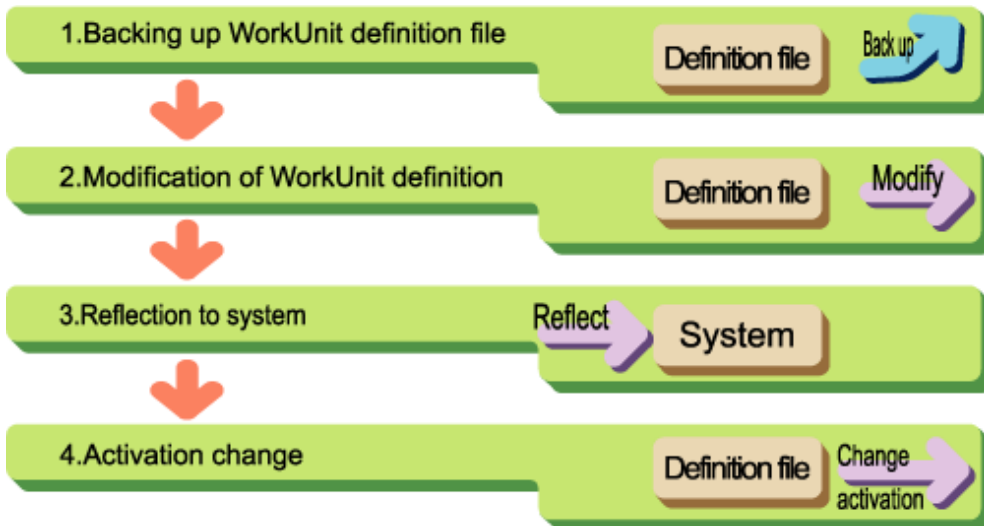
Note

The active changing procedure can be used for switching to a modified application, without altering the interface. It is not valid if the interface has been changed. Refer to "3.5.4 Changing a Server Application (Transaction Application)" for information on changing the interface.

Active changing can be applied to transaction applications only.

The procedure for changing an active application is illustrated in the following figure.

Figure 3.20 Active Changing of a Server Application



Note

Pay attention to the following points during this process:

- The active change procedure cannot be used in the following cases:
 - If you are changing an operation parameter
 - If the WorkUnit type is set to a type other than ORB
- If the new execution file is to have the same file name as the original, assign version information, or something similar, to prevent the file from being overwritten when the WorkUnit is in use.
- If the active change process fails, any alterations made to the WorkUnit definition file cannot be undone.
- No other processing can be carried out while an active application change is being performed. In some instances, there may be a temporary fall in system performance.
- In a system using the High Availability (HA) function, active changes made to the current node will not be reflected in the standby node. Also, the active change process cannot be performed for the standby node.
- The following statements in the WorkUnit definitions cannot be dynamically changed.
 - WorkUnit name
 - WorkUnit type
 - APM name
 - DPCF communication path name
 - Resident, non-resident, or multi-object-resident mode
 - Bind format
 - Session ID notification parameter
 - Session start method
 - Maximum client decision-making time
 - Error exit program name
 - AIM application monitoring time
 - Object priority
 - Maximum number of queues
 - Number of queues monitored

- Number of queues for monitoring restart

1) Back up WorkUnit Definition Files

Make back-up copies of the WorkUnit definition files currently in use, in case errors occur during the active change process.

2) Modify WorkUnit Definition File

Modify the WorkUnit definition file currently in use.

3) Update System

Use the *isaddwundef* command to copy the modified WorkUnit definition file to the system.

```
isaddwundef -o create-def.wu
```

4) Make Active Change

Use the *tdmodifywu* command to change the modified WorkUnit definition file.

The active change process can be used for:

- Switching to a modified server application
- Changing the environment variables used by server application

When replacing an application of the same name, supply version number information to the execution file name to avoid overwriting on the execution file.

```
tdmodifywu TDSAMPLE1
```

Notes

1. This command can only be executed by the user who has activated the WorkUnit.
2. Active changing of WorkUnits is an operation function that modifies WorkUnit definitions without stopping applications and starts the WorkUnit according to the modified definitions.
3. WorkUnit definition files can also be reflected in the system with the *tdadddef* command.

3.5.7 Dynamic Changing of the Number of Server Application Processes

The process concurrency of server applications for a WorkUnit can be changed without stopping the WorkUnit or changing the WorkUnit definition. This is referred to as dynamically changing the number of processes.

Dynamic changing of the number of processes is only supported in the Enterprise Edition, and can only be performed for CORBA WorkUnits and WorkUnits of transaction applications.

This section explains the procedure for dynamically changing the number of processes.

1) Dynamically Changing the Number of Processes

Change the application process concurrency using the *ismodifyprocnum* command or the Interstage Management Console.

Notes

- A registered WorkUnit definition cannot be modified using this function.
- For a CORBA WorkUnit, the number of processes for an application maintaining instance data for each client application (*iswitch = ON*) can be increased, but not decreased. (The number of applications cannot be restored to the defined value after the number of processes is increased).
- For WorkUnits of transaction applications, this command cannot be used for an object that uses the process binding function.
- For WorkUnits of transaction applications, this command cannot be used during execution of the activation change command (*tdmodifywu*).
- If the HA function has been used, modifications made using this command will not be reflected in the standby system.

- If the process concurrency is repeatedly increased and decreased using this command, a directory with the same name as the process ID name remains. This directory is created in a directory with the WorkUnit name in the current directory defined in the WorkUnit definition, and may use hard disk resources unnecessarily.

When changing the process concurrency frequently, check the process IDs of the applications currently running, and delete directories whose names indicate process IDs that are not currently running.

- For WorkUnits of transaction applications, if this command is used for a non-resident or multi-object resident type object, the process concurrency of objects operating as non-resident types or multi-object resident types will be changed.

 Solaris Linux32/64

- This command can only be executed by the user who started the WorkUnit or by the root user.

Chapter 4 WorkUnit Operation of Each Application

This chapter explains WorkUnit operation for each application.

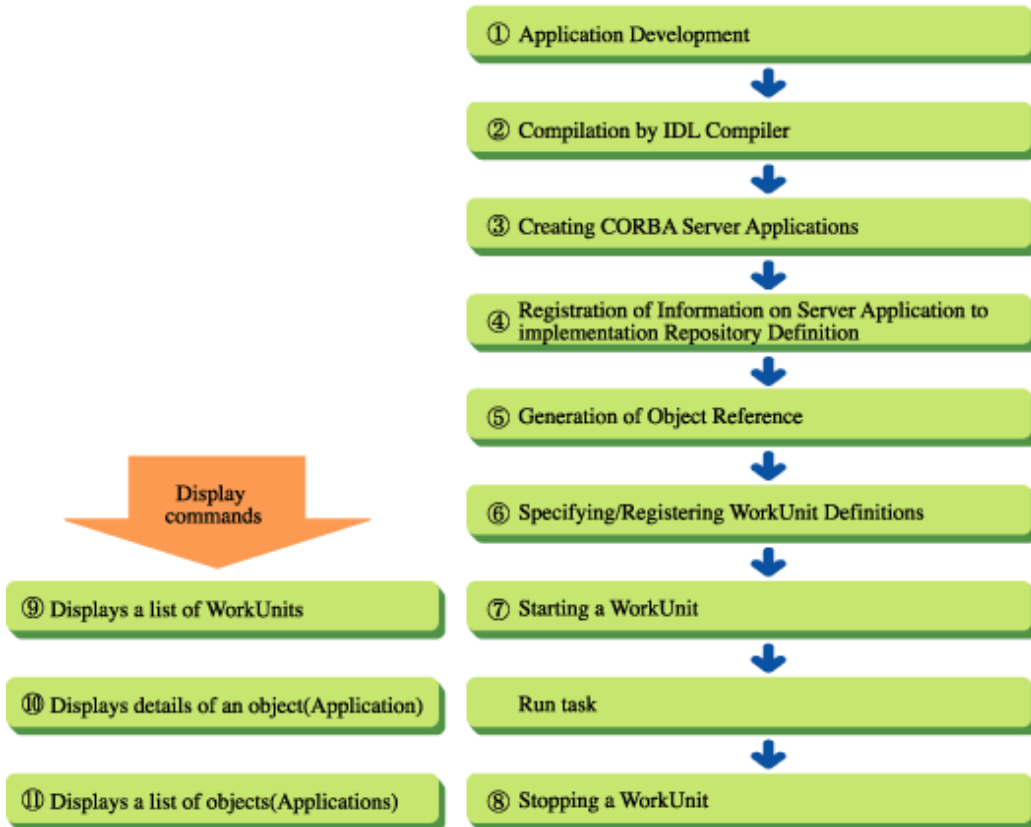
4.1 Operating CORBA WorkUnits

A CORBA WorkUnit (a WorkUnit of a CORBA application) can be operated using one of the following two methods:

- [Operation Using the Interstage Management Console](#)
- [Operation by Command Line Interface](#)

This section explains the environment setup and the operating procedure required for a CORBA WorkUnit to function.

Figure 4.1 Operating CORBA WorkUnits



The following five separate steps are required when a CORBA WorkUnit is used.

- a. CORBA application making phase (1 - 3).
- b. CORBA application environment and setting phases (4 - 5).
- c. CORBA WorkUnit environment and setting phases (6).
- d. CORBA WorkUnit operation phase (7 - 8).
- e. CORBA WorkUnit reference phase (9 - 11).

With a conventional CORBA WorkUnit, WorkUnit definitions and CORBA application definitions were implemented using the WorkUnit operation command and CORBA Service operation command. The operations from b to e above can now be managed collectively using the Interstage management console.

Refer to the Distributed Application Development Guide (CORBA Service Edition) and Reference Manual (Command Edition) for CORBA application development details (programming, IDL compiler, and so on).

4.1.1 Application Development

The CORBA application program operated under the control of the WorkUnit is made. As for the method of the programming, it is similar to the case to operate outside under the control of the WorkUnit.

4.1.2 Compilation by IDL Compiler (IDLc Command)

The IDL definition is made, the Interface Repository is registered by the IDLc command, and the skeleton is output. As for the IDL definition, it is similar to the case to operate outside under the control of the WorkUnit.

4.1.3 Creating CORBA Server Applications

The CORBA server application (execution module) is made from the skeleton made by made CORBA application program and IDLc command. As for the method of making the CORBA server application, it is similar to the case to operate outside under the control of the WorkUnit.

Note

It is necessary to end the process when "SYNC_END" is used with the CORBA WorkUnit as an operation mode or it returns from CORBA_BOA_impl_is_ready. The stop of the WorkUnit becomes a waiting state though it returns from CORBA_BOA_impl_is_ready in case of "SYNC_END" when the WorkUnit is usually stopped and stopped synchronously until the process is ended. Therefore, it is necessary to end the process when returning from CORBA_BOA_impl_is_ready without fail.

When a CORBA application is operated on the WorkUnit, the standard output and standard error output from that CORBA application are redirected to the following files.

- Standard output
"Current-directory of the WorkUnit definition"\WorkUnit-name"\process-id"\stdout.
- Standard error output
"Current-directory of the WorkUnit definition"\WorkUnit-name"\process-id"\stderr.

Data output to the standard output and standard error output is temporarily stored in the standard input-output library according to the OS standard input-output library specifications.

Flushing must be performed for the standard output and standard error output to ensure that the stored data is reliably output to the stdout and stderr files.

For example, issue fflush(stdout) and fflush(stderr) in C language. The data stored in the standard input-output library is not output to the stdout and stderr files until flushing is performed.

Refer to the Reference Manual (Command Edition) for details on storing server application information in the Implementation Repository definition using the command line interface.

4.1.4 Registration of Information on Server Application to Implementation Repository Definition

Information on the CORBA application operated with the WorkUnit is set. It is basically similar to the case to operate outside under the control of the WorkUnit. Notes of definition information are shown below when operating on the WorkUnit.

Start Type of Server Application (type)

The start type of the server application should set "Persistent server" type when operating under the control of the WorkUnit.

The Maximum In Process Multiplicity (proc_conc_max)

It is necessary to set larger value than the process multiplicity set by the WorkUnit definition. Please note the failure in the start of the WorkUnit when smaller value than the process multiplicity set by the WorkUnit definition is set.

Refer to the Reference Manual (Command Edition) for details on storing the server application information in the Implementation Repository definition by using the command line interface.

4.1.5 Generation of Object Reference

To access the made server application from other applications as a target, the object reference to identify the object is made. It registers in service of the naming of the object reference made at the same time. It is basically similar to the case to operate outside under the control of the WorkUnit.

4.1.6 Specifying/Registering WorkUnit Definitions

Specify and register in the WorkUnit definitions the CORBA application information required for operation in the WorkUnit.

- Specifying WorkUnit definitions

Please refer to [4.1.10.1 CORBA WorkUnit Definitions](#).

- Registering WorkUnit definitions

Register in Interstage the WorkUnit definitions is specified. Use the *isaddwundef* command.

4.1.7 Starting a WorkUnit

Start the CORBA application specified in the WorkUnit definitions. Specify the WorkUnit name in the *isstartwu* command. In addition to starting a WorkUnit via a command, a WorkUnit can be started from the Interstage operation API, Interstage management console, or Systemwalker OperationMGR.

Note

Please do not delete the execution module of the CORBA application set to the WorkUnit definition, and do not overwrite while operating it with the WorkUnit. The CORBA application processing and the operation of the pertinent WorkUnit might hang up when going.

If a WorkUnit fails to start, the start parameters and environment variables of the process can be output to the log file, and checked using the process start log collection function.

Specify "YES" in the "Start Log" statement of the WorkUnit definition to enable the process start log collection function, and check that the log file is output to the current directory.

4.1.8 Stopping a WorkUnit

Stop the WorkUnit that has been started.

A Normal stop and a synchronously stop are used when stopping usually. When it wants to stop in the emergency, the forced stop is used.

- Normal stop

Stops the WorkUnit provided that all applications defined in the WorkUnit have stopped processing. The WorkUnit cannot be stopped if it is still processing a task. In such cases, wait until processing is completed, and run the command again.

Note

CORBA_BOA_impl_is_ready returns as operation on the application when the operation mode of the CORBA application is "SYNC_END" or the WorkUnit Normal stops. In this case, it is necessary to end the process as processing of the application. Please note that the stop usually enters the state of hanging up about the WorkUnit when making it to the waiting state without ending the process. Please stop the WorkUnit compulsorily in that case by the compulsion stop command. For instance, there is a possibility of hanging up when the thread is generated with the processing of the application for the Java application, and the generated thread has not been stopped normally. Please collect the thread generated with the application by the application.

- Synchronous stop

After the demand processing it when the command is executed is executed, the WorkUnit is stopped. The demand which is the waiting state is annulled (The error is notified to the client).

Note

After the demand under processing is completed, it stops at a synchronous stop. Therefore, the WorkUnit stop will hang up when the application under processing is not completed. In that case, please stop the WorkUnit compulsorily by the compulsion stop command.

CORBA_BOA_impl_is_ready returns as operation on the application when the operation mode of the CORBA application is "SYNC_END" or the WorkUnit usually stops. In this case, it is necessary to end the process as processing of the application. Please

note that the stop usually enters the state of hanging up about the WorkUnit when making it to the waiting state without ending the process. Please stop the WorkUnit compulsorily in that case by the compulsion stop command. For instance, there is a possibility of hanging up when the thread is generated with the processing of the application for the Java application, and the generated thread has not been stopped normally. Please collect the thread generated with the application by the application.

- Forced stop

The forced outage stops compulsorily when the application is processing it. The application under processing is stopped compulsorily while processing it, and annulled for the demand which is the waiting state (The error is notified to the client). The forced outage can be compulsorily stopped by executing the compulsion stop command at the state of the WorkUnit operation of the following other than.

- The WorkUnit start is being processed.
- The WorkUnit is being normal terminated.
- The WorkUnit is being synchronously terminated.

A WorkUnit can be stopped using a command (isstopwu) or by using the Interstage operation API, Interstage management console, or Systemwalker OperationMGR.

4.1.9 Operation Using the Interstage Management Console

This section explains the environment creation and operating procedure required for using a CORBA application via the Interstage management console.

Note

A CORBA application linking to a global transaction cannot be operated via the Interstage management console.

4.1.9.1 Creating a CORBA WorkUnit

WorkUnit: A WorkUnit can be created using the window for new creation. Specify the following information and click the New button to create the WorkUnit.

- WorkUnit name
- WorkUnit type

Detailed settings can be made as required. The items and their contents are explained in the following table.

Table 4.1 CORBA WorkUnit Detailed Settings

Interstage management console item name (WorkUnit definition item name)	Required item?	Specified contents
WorkUnit name (Name)		Specify a WorkUnit (CORBA) identification name.
WorkUnit type (Kind)		Select CORBA.
Application folder (Path)		Specify an application subdirectory.
Current directory for application operation (Current Directory)	Required	Specify the work directory in which the application operates.
retry count (Application Retry Count)	Omissible	Specify an ABEND count for the application process at which the WorkUnit stops abnormally.
Path (Path for Application)	Omissible	Specify the path used by an application during operation.

Interstage management console item name (WorkUnit definition item name)	Required item?	Specified contents
Library path (Library for Application)	Omissible	Set the path used by an application during operation (environment variable LD_LIBRARY_PATH). This item is valid for systems other than Windows.
Environment variable (Environment Variable)	Omissible	Set the environment variable used by an application during operation.
Exit program name for WorkUnit (WorkUnit Exit Program)	Omissible	Specify the name of the exit program to be issued once during WorkUnit starting and stopping.
Maximum processing time for exit program (Maximum Processing Time for Exit Program)	Omissible	Specify the maximum processing time for exit program monitoring.
The executable file name of the process collection exit program (Executable File of Exit Program for Salvage)	Omissible	Set the execution filename of the WorkUnit exit program and process release exit program.
Request dispatch mode (Request Assignment Mode)	Omissible	Specify the mode in which a request message from a client is distributed to the server application process waiting for the request.
Automatic start of WorkUnit	Omissible	Specify whether or not the WorkUnit is to be started during Interstage startup.
Stack trace collection (Output of Stack Trace)	Omissible	Specify whether or not stack traces are to be collected.
Time waiting for WorkUnit to start up (Startup Time)	Omissible	Specify the monitoring time until the WorkUnit has started.
Forcible shutdown time for a process (Shutdown Time)	Omissible	Specify the monitoring time until the WorkUnit is stopped completely.
Collection of the process start log (Start Log)	Omissible	The log of the CORBA WorkUnit at the beginning of the process is collected.
Degenerated process operation (Process Degeneration)	Omissible	If automatic restarting of a WorkUnit process fails, operation continues in a state where one process is removed (degenerated operation).
Number of backup generations for the current directory (Number of Revision Directories)	Omissible	The number of backup generations for the current WorkUnit directory can be specified.

The created WorkUnit can be changed using the environment setting in the WorkUnit Name operation window.

Note

If the definition of a WorkUnit being started is deleted using the isdelwudef command, a CORBA WorkUnit of the same name cannot be created from the Interstage management console unless that WorkUnit is stopped. In this situation, stop the WorkUnit being started and retry the operation.

4.1.9.2 Deploying a CORBA Application

CORBA WorkUnit: An application can be deployed from the Arrange window.

Set the following information and click the Deploy button. This will deploy the application so that it can be operated on the WorkUnit.

- Implementation Repository ID
- Execution program file
- Start specification

Note

Files such as libraries to be referenced dynamically from the execution program file of the application to be deployed must be stored on the server where that CORBA WorkUnit operates. During the deployment operation, file transfer is not performed. Store the execution program file in the directory specified in the application folder for the WorkUnit.

Detailed settings can also be made as required. Setting items include the settings for each CORBA application and the interface definition settings in each CORBA application. Details of setting items and their contents are explained in the following table.

Table 4.2 Setting Items for each CORBA Application

Interstage management console definition item name (WorkUnit definition item name or definition item name stored in implementation repository)	Required item?	Specified contents
Implementation Repository ID (Impl ID/rep_id(*1))	Required	Specify the Implementation Repository ID to identify a CORBA application. The same Implementation Repository ID cannot be specified more than once for the same WorkUnit.
Execution program file (Executable File)	Required	Set the application and exit program module names.
Start specification	Required	Checked off or selected to start a WorkUnit after application deployment.
Locale (locale(*1))	Omissible	Specify the code system for the object reference for which the code system for the application is specified.
Process concurrency (Concurrency/proc_conc_max(*1))	Omissible	Specify the application process concurrency.
Number of processing threads (thr_conc_init(*1)/thr_conc_maximum(*1))	Omissible	Specify the number of application processing threads for each process. This number increases or decreases automatically with loads, so specify the initial and maximum values.
Maximum processing time of application (Maximum Processing Time)	Omissible	Set the time (in seconds) to be used as the maximum processing time for an application during monitoring.
Control when the maximum processing time of an application is exceeded. (Terminate Process for Time out)	Omissible	Specify the processing to be applied to the process corresponding to the application when the maximum application processing time has been exceeded.
Maximum number of messages in queue (Maximum Queuing Message)	Omissible	Set the maximum number of messages to be retained in a queue. An alarm can then be issued as soon as the number of messages in the queue exceeds this value.
Number of messages in a queue to trigger alarm (Queuing Message To Notify Alarm)	Omissible	Set the number of messages in a queue on which an alarm is to be issued. If the number of messages in the queue reaches this number, an alarm is issued.

Interstage management console definition item name (WorkUnit definition item name or definition item name stored in implementation repository)	Required item?	Specified contents
Number of messages in a queue at which monitoring is to be restarted (Queuing Message To Notify Resumption)	Omissible	Set the number of messages in a queue on which monitoring for alarm issuing is to be restarted. When this number is reached (after the number of messages in the queue has exceeded the number at which an alarm is triggered), monitoring of the number of messages in the queue is restarted.
Class path (CLASSPATH for Application)	Omissible	Set the class path to be used when a CORBA application (Java) operates.
Environment variable (Environment Variable)	Omissible	Set the environment variables used by an application and exit program during operation. Use the following specification format: "Environment variable = value". The PATH and LD_LIBRARY_PATH parameters cannot be specified.
Name of process collection exit program (Exit Program for Process Salvage)	Omissible	Set the name of the process release exit program. Note that only the name of a C program can be specified.
The executable file name of the process collection exit program (Executable File of Exit Program for Salvage)	Omissible	Set the filename of the execution file for the process release exit program
Start parameter (Param for Executable File)	Omissible	Specify the start parameter for the application. For a Java application, an application class name to be specified in a Java command must be set.
Request dispatch mode (Request Assignment Mode)	Omissible	Specify the mode in which a request message from a client is distributed to a server application process waiting for that request.
Number of communication buffers (Buffer Number)	Omissible	Set the number of queue buffers.
Communication buffer size (Buffer Size)	Omissible	Set the data item size for which a single queue operation is to be performed in accordance with a request.
Instance retaining function (iswitch(*1))	Omissible	Specify whether or not an application is to retain instance data for each client application. This item is valid for applications in the C++, Java, and OOCOBOL programming languages.
SSL information addition to object reference (ssl(*1))	Omissible	Specify the SSL information addition rules that apply when an object reference is created for a server application.
Operation mode (mode(*1))	Omissible	Specify the operation mode for a CORBA application.

*1 Implementation Repository definition item

Table 4.3 Setting Items for each Interface

Interstage management console definition item name	Required item?	Specified contents
Interface Repository ID	Omissible	Set the Interface Repository ID of an object.
Addition to Naming Service	Omissible	Specify whether or not the object is added to a Naming Service.
Naming service registration name	Omissible	If an object is added to a Naming Service, specify its name.
Priority order	Omissible	Specify the priority order of objects.
Library path name	Omissible	Specify the path name to a library.
COBOL dynamic skeleton interface	Omissible	Set the Interface Repository ID of an object.

4.1.9.3 Closure/Closure Cancellation

To close or cancel closure for a queue or interface for CORBA applications, click the Close or Disclose button in the Status.

4.1.9.4 Function for Maximum Number of Messages Retained in a Queue

To use the function to determine the maximum number of messages left in a queue, set the maximum value in the WorkUnit definition.

If there is a request to retain messages in the queue which exceeds the maximum number of messages in the queue, the following exception is returned to the client.

System exception	Minor code (hexadecimal notation)
NO_RESOURCE	0x464a0094
	0x464a0894

4.1.9.5 Alarm Notification Function for Number of Messages in a Queue

To use the alarm notification function when a queue contains a certain number of messages, set the following details in the WorkUnit definition: The number of messages in a queue at which an alarm is to be issued, and the number before monitoring for alarms will be restarted.

- Maximum number of messages in a queue

The maximum number of messages retained in a queue. When this value is exceeded, a message is output. A later message is also output notifying the user that a new alarm will not be issued for this event until the number of messages reaches that at which monitoring will restart.

- Number of messages in a queue before an alarm is issued

Set the number of messages in a queue at which an alarm is issued (a message is output). If the number of messages retained in the queue reaches this value, a message is output. A later message is also output notifying the user that a new alarm will not be issued for this event until the number of messages reaches that at which monitoring will restart.

- Number of messages in a queue before monitoring is restarted

Set the number of messages in a queue at which monitoring for alarms is to be restarted. When this value is reached after an alarm for an excessive number of queue elements was issued, monitoring of the number of messages in the queue is restarted and a message is output to that effect.

4.1.10 Operation by Command Line Interface

This section explains the environment creation and operating procedure for using a CORBA WorkUnit via the command line interface.

4.1.10.1 CORBA WorkUnit Definitions

The CORBA WorkUnit definition items and their contents are shown in the following table.

Table 4.4 CORBA WorkUnit Definition Items

Section name	Definition name	Required item?	Specification details
WORK UNIT	Name (WorkUnit name)	Required	Specify the WorkUnit name
	Kind (WorkUnit type)	Required	Set as "CORBA".
Control Option	Path (application library path)	Required	Set the path in which general applications exist.
	Current Directory (current directory)	Required	Specify the directory in which applications run. The application runs under the directory/WorkUnit name/process ID directory specified for this item.
	Application Retry Count (continuous abnormality termination count)	Omissible	Set the number of abnormal terminations after which automatic retry disabling occurs for the application. This item is ignored when the "WorkUnit Automatic Stop Mode:" statement is set to "YES".
	Path for Application (path used by the application)	Omissible	Specify the path (environment variable PATH) used when the application operates.
	Library for Application (Library path used by application)	Omissible	Specify the library path (environment variable LD_LIBRARY_PATH) used when the application operates.
	Environment Variable (environment variable)	Omissible	Specify the environment variable used when the application operates.
	Maximum Processing Time For Exit Program (maximum processing time for exit program)	Omissible	Specify the monitoring value for the maximum exit program processing time (process stop exit). When it is set in the [Control Option] section, this value becomes valid for the application under the WorkUnit. If this value is also set in the [Application Program] section, the value set in the [Application Program] section becomes valid for the application.
	Executable File of Exit Program for Salvage (the executable file name of the process collection exit program)	Omissible	When using the process stop exit, set the executable file in which the process stop exit program is set. This specification can be set in the [Control Option] section or the [Application Program] section. If an executable file has not been set for the [Application Program] section, the executable file set for the [Control Option] section is enabled.

Section name	Definition name	Required item?	Specification details
	Request Assignment Mode (Request dispatch mode)	Omissible	Specify the mode in which a client request message is distributed to the server application process waiting for the request.
Application Program	Impl ID (Implementation Repository ID)	Required	Specify the Implementation Repository ID to identify a CORBA application. The same Implementation Repository ID cannot be specified more than once for the same WorkUnit.
	Executable File (Execution program file)	Required	Set the application and exit program module names.
	Concurrency (process concurrency)	Omissible	Specify the concurrency of the applications to be started.
	Maximum Processing Time (Maximum processing time of application)	Omissible	Set the time (in seconds) that is to be used as the maximum processing time for an application during monitoring.
	Terminate Process for Time out (Control when the maximum processing time of an application is exceeded)	Omissible	Specify the processing that is to be applied to the process corresponding to the application when the maximum processing time has been exceeded for that application.
	Maximum Processing Time for Exit Program (Maximum processing time for exit program)	Omissible	Specify the maximum processing time for which the exit program is monitored.
	Maximum Queuing Message (Maximum number of messages in queue)	Omissible	Set the maximum number of messages to be retained in a queue. An alarm can then be issued as soon as the number of messages in the queue exceeds this value.
	Queuing Messages To Notify Alarm (Number of messages in a queue to trigger alarm)	Omissible	Set the number of messages in a queue at which an alarm is to be issued. An alarm is issued if the number of messages in the queue reaches this number.
	Queuing Messages To Notify Resumption (Number of messages in a queue at which monitoring is to be restarted)	Omissible	Set the number of messages in a queue at which monitoring for alarm issuing is to be restarted. When this number is reached after the number of messages in the queue has exceeded the number at which an alarm is triggered, monitoring of the number of messages in the queue is restarted.
CLASSPATH for Application (Class path)	Omissible	Set the class path to be used when a CORBA application (Java) operates.	

Section name	Definition name	Required item?	Specification details
	Exit Program for Process Salvage (Name of process collection exit program)	Omissible	Set the name of the process release exit program. Note that only the name of a C program can be specified.
	Destination Priority (object priority)	Omissible	Specify the priority for the object. A priority from 1 to 255 can be specified, with higher numbers indicating higher priority.
	Request Assignment Mode (Request dispatch mode)	Omissible	Specify the mode in which a client request message is distributed to a server application process waiting for the request.
	Buffer Number (Number of communication buffers)	Omissible	Set the number of queue buffers.
	Buffer Size (Communication buffer size)	Omissible	Set the data item size for which a single queue operation is to be performed in accordance with a request.

Refer to WorkUnit Definition in Appendix A for the WorkUnit definition syntax rules.

4.1.10.2 CORBA Application Queue Control

Priority Order Control

If priority control is to be used by a CORBA application, it is necessary to register the priority in the Implementation Repository.

Priority is specified for the Interface Repository ID with the detailed information file specified by the -ax option of the OD_impl_inst command. The specification method is as follows

```
intfID = priority;solib[, [prefix][, inherit[, ...]]]
```

- intfID

The Interface Repository ID of the object is specified.

- priority

The priority (0-255) of the object is specified. When priority control is not used, 0 is specified.

- solib

The path name in the library is specified.

- prefix

When the first character string prefix of the function name is specified with the -S option of the IDLc command, when specifying it (C language only).

- inherit

The Interface Repository ID inherited to is specified.

Closure or Closure Cancellation

This section explains how to close or cancel closure for a queue or interface.

Closure is performed using the isinhibitque command.

```
isinhibitque WU1 IDL:test1:1.0 IDL:test1/intf1:1.0
```

Closure cancellation is performed using the ispermitque command.

```
ispermitque WU1 IDL:test1:1.0 IDL:test1/intf1:1.0
```

Note

It is necessary to register the interface name which performs the close processing in the Implementation Repository beforehand when the interface is closed.

If a request is sent from a client while the queue is closed, the following exception is returned to the client.

System exception	Minor code (hexadecimal notation)
NO_RESOURCE	0x464a0094
	0x464a0894

Function for Maximum Number of Messages Retained in a Queue

To use the function for the maximum number of messages retained in a queue, set the maximum value in the WorkUnit definition.

If there is a request to retain messages in the queue for which the maximum number of messages in the queue would be exceeded, the following exception is returned to the client.

System exception	Minor code (hexadecimal notation)
NO_RESOURCE	0x464a0094
	0x464a0894

Alarm Notification Function for the Number of Messages in a Queue

To use the alarm notification function for a certain number of messages in a queue, set the following details in the WorkUnit definition: The number of messages in a queue at which an alarm is issued and that before monitoring for alarms will be restarted.

- Maximum number of messages in a queue

The maximum number of messages retained in the queue. When this value is exceeded, a message is output. A later message is also output notifying the user that a new alarm for this event will not be issued until the number of messages reaches the number at which monitoring will restart.

- Number of messages in a queue before an alarm is issued

Set the number of messages in the queue at which an alarm is issued (a message is output). If the number of messages retained in the queue reaches this value, a message is output. A later message is also output notifying the user that a new alarm for this event will not be issued until the number of messages reaches that at which monitoring will restart.

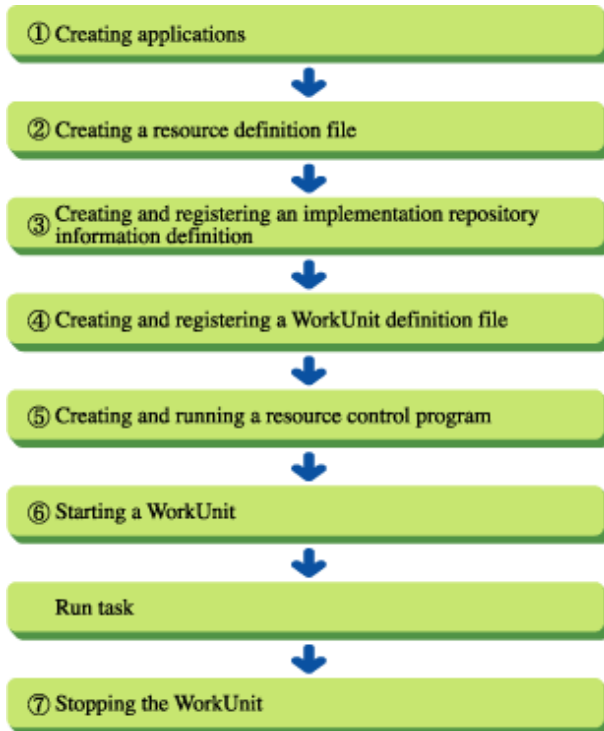
- Number of messages in a queue before monitoring is restarted

Set the number of messages in a queue at which alarm monitoring is to be restarted. When this value is reached after an alarm for an excessive number of queue elements was issued, monitoring of the number of messages in the queue is restarted and a message to that effect is output.

4.1.10.3 Global Transaction Linkage

This section explains how to operate applications in the CORBA WorkUnit to implement global transaction linkage.

Figure 4.2 Implementing Global Transaction Linkage



To implement global transaction linkage using the CORBA WorkUnit, the operations in (1), (2), (3), and (5) are required in addition to those required using CORBA applications for local transactions.

The Interstage management console can be used for (4) and (6).

(1) Creating applications

This section explains how to operate applications in the CORBA WorkUnit to implement global transaction linkage.

Create CORBA application programs used for global transaction linkage. Application programs used for global transaction linkage can be created in the same way as for those running not under control of WorkUnits.

(2) Creating a resource definition file

Create a resource definition file. A resource definition file can be created in the same way as one used not under control of a WorkUnit.

(3) Creating and registering an implementation repository information definition

Create and register an implementation repository information definition.

(4) Creating and registering a WorkUnit definition file

Create and register a WorkUnit definition file for the CORBA application used to implement global transaction linkage.

Refer to [4.1.6 Specifying/Registering WorkUnit Definitions](#) for information on how to create and register a WorkUnit definition file.

(5) Creating and running a resource control program

Create and run a resource control program. A resource control program can be created and run in the same way as one used not under control of a WorkUnit.

(6) Starting a WorkUnit

Start the WorkUnit that defines the CORBA application used to implement global transaction linkage.

Refer to [4.1.7 Starting a WorkUnit](#) for information on how to start the WorkUnit.

(7) Stopping the WorkUnit

After operation is finished, stop the WorkUnit.

Refer to [4.1.8 Stopping a WorkUnit](#) for information on how to stop the WorkUnit.

4.2 Operating EJB WorkUnits

This section explains how to create the environment and the procedures for operating EJB applications in WorkUnits. The procedure is outlined in the following figure.

Figure 4.3 Creating the Environment and Operating Procedures for EJB WorkUnits



Operations 1 to 8 are required in order to operate an EJB application in a WorkUnit.

4.2.1 Specifying/Registering WorkUnit Definitions

Specify and register in the WorkUnit definitions the EJB application information required for operation in the WorkUnit.

- Specifying WorkUnit definitions

- Registering WorkUnit definitions

Register in Interstage the WorkUnit definitions in which the EJB application information is specified. Use the *isaddwundef* command.

4.2.1.1 Starting a WorkUnit

Start the EJB application specified in the WorkUnit definitions. Specify the WorkUnit name in the *isstartwu* command.

4.2.1.2 Stopping a WorkUnit

Stop the WorkUnit that has been started.

WorkUnits can be stopped in two ways:

- Normal stop

Stops the WorkUnit provided that all applications defined in the WorkUnit have stopped processing. The WorkUnit cannot be stopped if it is still processing a task. In such cases, wait until processing is completed, and run the command again.

- Forced stop

Stops the WorkUnit even if an application is still performing processing. Specify the WorkUnit name in the *isstopwu* command.

4.3 Operating Transaction Application WorkUnits

This section explains the Object Priority Function:

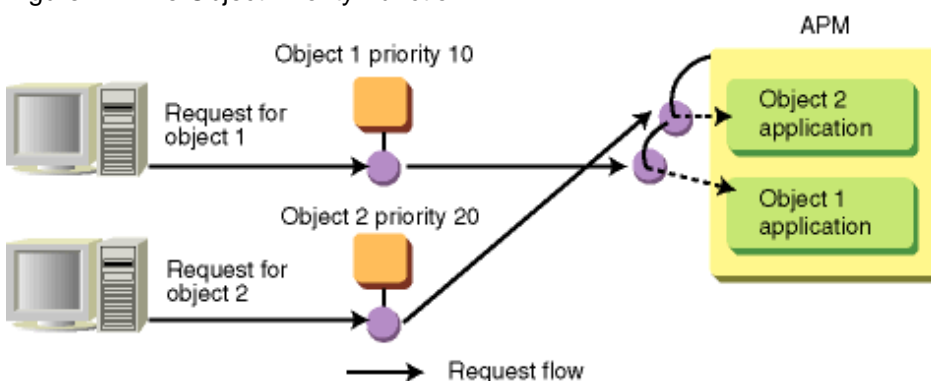
- [Operation Using the Object Priority Function](#)
- [Operating Procedures for Object Priority Function](#)
 - [Create Application](#)
 - [Specify WorkUnit Definitions](#)
 - [Register WorkUnit Definitions](#)
 - [Start the WorkUnit](#)

4.3.1 Operation Using the Object Priority Function

The object priority function allows the specification of priorities for transaction application objects.

This function makes it possible to assign priorities to processing as shown in the following figure.

Figure 4.4 The Object Priority Function



Multiple objects are set in a single APM process. The APM receives data in order from the highest priority objects down. Therefore, if data for an object with a high priority resides in the queue, data for an object with a low priority is not processed.

Windows32/64 Solaris

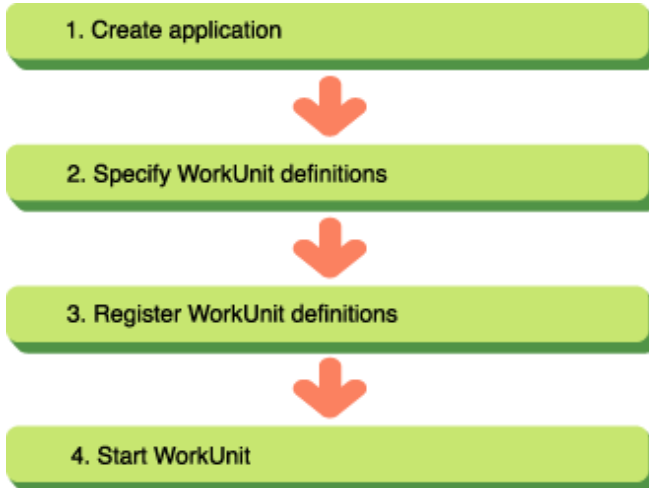
This function can be used when the application language is C or COBOL.

This function can be used when the application language is C.

4.3.2 Operating Procedures for Object Priority Function

This section describes the procedures for using the object priority function. The procedure is shown in the following figure and described in the following text.

Figure 4.5 Operating Procedures for the Object Priority Function



4.3.2.1 Create Application

Create the application

Note

C++ applications cannot use this function.

4.3.2.2 Specify WorkUnit Definitions

Specify the priority for the object in the WorkUnit definitions.

The following table explains the WorkUnit definitions required when using the object priority function.

Note

The object priority function can only be used for objects specified in the WorkUnit definitions.

Table 4.5 WorkUnit Definition Specifications

Section name	Definition name	Details of specifications
Application Program	Destination (object name)	Specify the name of the object using priority
	Destination Priority (object priority)	Specify the priority for the object. Priority from 1 to 255 can be specified, with higher numbers indicating higher priority
	Form (resident or non-resident mode)	Specify NONRESIDENT or MULTIRESIDENT
Nonresident Application Process *1	Concurrency (process concurrency)	Specify the application concurrency
	Pre Exit Program (pre-exit program)	Specify the exit program name if an exit program, started when the WorkUnit is started, is used.

Section name	Definition name	Details of specifications
	Post Exit Program (post-exit program)	Specify the exit program name if an exit program, started when the WorkUnit is stopped, is used.
	Executable File for Exit Program (name of executable file for exit program)	Specify the executable file name for the exit program if the exit program is created as an executable file separate from the application.
	Maximum Processing Time for Exit Program (maximum processing time for exit program)	Specify the monitoring time (seconds) for the maximum processing time for the exit program.
Multiresident Application Process *2	Concurrency (process concurrency)	Specify the application concurrency
	Pre Exit Program (pre-exit program)	Specify the exit program name if an exit program, started when the WorkUnit is started, is used.
	Post Exit Program (post-exit program)	Specify the exit program name if an exit program, started when the WorkUnit is stopped, is used.
	Executable File for Exit Program (name of executable file for exit program)	Specify the executable file name for the exit program if the exit program is created as an executable file separate from the application.
	Maximum Processing Time for Exit Program (maximum processing time for exit program)	Specify the monitoring time (seconds) for the maximum processing time for the exit program.

*1 Required if NONRESIDENT is specified in the Application Program Form.

*2 Required if MULTIRESIDENT is specified in the Application Program Form.

Note

Apart from these definitions, the definitions are otherwise as normal.

4.3.2.3 Register WorkUnit Definitions

Register the WorkUnit definitions in the system using the isaddwufdef command.

Note

WorkUnit definitions can also be added with the tdaddddef command.

4.3.2.4 Start the WorkUnit

Start the WorkUnit specified in the WorkUnit definitions using the isstartwu command.

Note

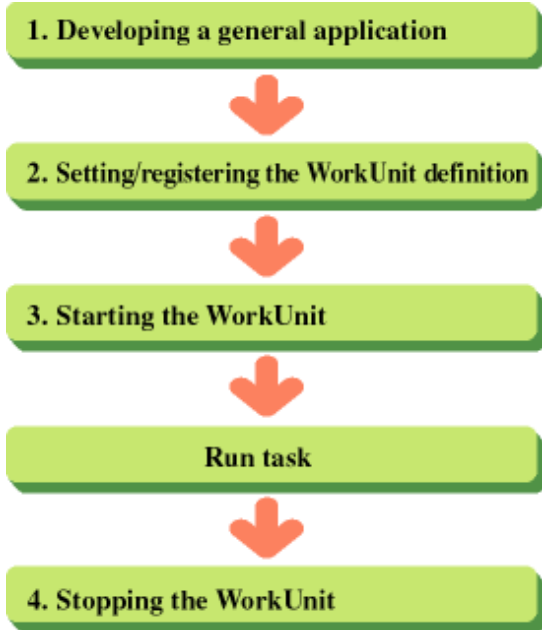
WorkUnits can also be started with the tdstartwu command.

4.4 Operation in Utility WorkUnits

When operating a general application out of the control of Interstage on a WorkUnit, create the environment and perform operation as shown below.

Figure 4.6 Operation in Utility WorkUnits



4.4.1 Operating Procedures

When operating a general application on a WorkUnit, the operations in Steps 1 to 4 are required.

1) Developing a General Application

Create a general application that is to be operated in a WorkUnit. Before using it in a WorkUnit, test the general application by starting it independently.

2) Setting/Registering the WorkUnit Definition

Set and register the information of the general application required for the operation of the WorkUnit.

- Setting the WorkUnit definition

In the WorkUnit definition, set the information for operating a general application on a WorkUnit.

Table 4.6 Setting the WorkUnit Definition

Section name	Definition name	Details of specifications
WORK UNIT	Name (WorkUnit name)	Specify the WorkUnit name
	Kind (WorkUnit type)	Specify UTY to operate general applications that are not under Interstage on the WorkUnit.
Control Option	Path (application library path)	Set the path in which a general application exists.
	Current Directory (current directory)	Specify the directory in which applications run. The application operates under the directory/WorkUnit name/process ID directory specified for this item.
	Application Retry Count (continuous abnormality termination count)	Set the number of times the application was terminated abnormally until automatic retry disabling occurs. When "WorkUnit Automatic Stop Mode:" statement is set to "YES," this item is ignored.
	Path for Application (path used by the application)	Specify the path (environment variable PATH) used when the application operates.

Section name	Definition name	Details of specifications
	Library for Application (Library path used by application)	Specify the library path (environment variable LD_LIBRARY_PATH) used when the application operates.
	Environment Variable (environment variable)	Specify the environment variable used when the application operates.
	Maximum Processing Time For Exit Program (maximum processing time for exit program)	Specify the monitoring value for the maximum processing time of the exit program (process stop exit). When it is set in the [Control Option] section, this value becomes valid for the application under the WorkUnit. If this value is also set in the [Application Program] section, the value set in the [Application Program] section becomes valid for the application.
	Executable File of Exit Program for Salvage (the executable file name of the process collection exit program)	When using the process stop exit, set the execution file in which the process stop exit program is set. Set it in the [Control Option] section or the [Application Program] section. If an executable file has not been set for the [Application Program] section, the executable file set for the [Control Option] section is enabled.
	WorkUnit Automatic Stop Mode (Automatic stop mode)	Set to stop the WorkUnit automatically.
Application Program	Executable File (execution file name)	Specify the general application that runs in the WorkUnit.
	Concurrency (process concurrency)	Specify the concurrency of the applications to be started.
	Environment Variable (environmental variable)	Specify the environment variable used when the application operates. When the environment variable of the same name is specified in "Environment Variable" in [Control Option], this item becomes valid for the application set in [Application Program].
	Param for Executable File (starting parameter)	Set the parameter to be passed when starting applications. The parameters are set in order starting from the parameter defined first.
	Reset Time for Application Retry Count (Reset Time for Application Retry Count (retry count reset time))	Set the continuous non-stop time until the present abnormal termination count is reset.
	Exit Program for Terminating Process (process stop exit program name)	Set the name of the program that stops the started application.
	Executable File of Exit Program for Salvage (the executable file name of the process collection exit program)	When using the process stop exit, set the execution file in which the process stop exit program is set. Set in the [Control Option] section or the [Application Program] section. When it is also set in the [Control Option] section, the execution file set in this item section becomes valid for the application set in [Application Program].
	Maximum Processing Time For Exit Program (output program maximum processing time)	Specify the monitoring value for the maximum processing time of the exit program (process stop exit). If the monitoring time is also registered in the [Control Option] section, the value here becomes valid for the relevant exit program.

- Registering the WorkUnit definition

Register the WorkUnit definition in which the information on general applications is set in Interstage. For registration, use the *isaddwundef* command.

3) Starting a WorkUnit

Start the general applications set in the WorkUnit definition. To start the applications, use the *isstartwu* command with the WorkUnit name specified.

4) Stopping the WorkUnit

Stop the WorkUnit during starting processing.

There are two types of stopping methods: one is to stop with a command and the other is to wait for all the applications to stop and stop the WorkUnit.

For stopping applications with a command, the following two methods can be used. For normal stop, specify the WorkUnit name in the *isstpwu* command. For forced stop, also specify *-c*.

Normal Stop

Release the application processes started on the WorkUnit from the monitoring target. That is, monitoring of general applications by the WorkUnit is canceled and the command returns control. Since the processes of the general applications are left after the WorkUnit stops and the command returns, you need to stop the processes.

It is also possible for the user to create a process stop exit program that stops application processes and set it in the WorkUnit definition so that it can be called when the WorkUnit is stopped. Since normal termination is notified to the exit program, this method is effective to terminate the WorkUnit by discriminating from the forced stop.

Forced Stop

Release the application processes started on the WorkUnit from the monitoring target. That is, monitoring of general applications by the WorkUnit is canceled and the command returns control. Since the processes of the general applications are left after the WorkUnit stop command is returned, you need to stop the processes.

It is also possible for the user to create a process stop exit program that stops application processes and set it in the WorkUnit definition so that it can be called when the WorkUnit is stopped. Since forced termination is notified to the exit program, this method is effective to terminate the WorkUnit by discriminating from normal stop.

The WorkUnit is stopped when all applications have been stopped, as follows:

WorkUnit Automatic Stop

Like commands and batch, there are applications that do not reside as processes and stop after execution. For such applications, do not restart a process even if it stops and stop the WorkUnit after all the processes in the WorkUnit stop. The WorkUnit automatic stop is an effective function for commands and batch. The WorkUnit automatic stop is set in the WorkUnit definition.

Chapter 5 Operating the Distributed Transaction Function

Windows32/64 Solaris32 Linux32/64

This chapter describes the procedure and environment setup required to use the distributed transaction function.

It describes how to create the environment in order to use the *isinit* and *isstart* commands, and how to use the *otssetup* command to set up the environment.

Note

Windows32 Solaris32

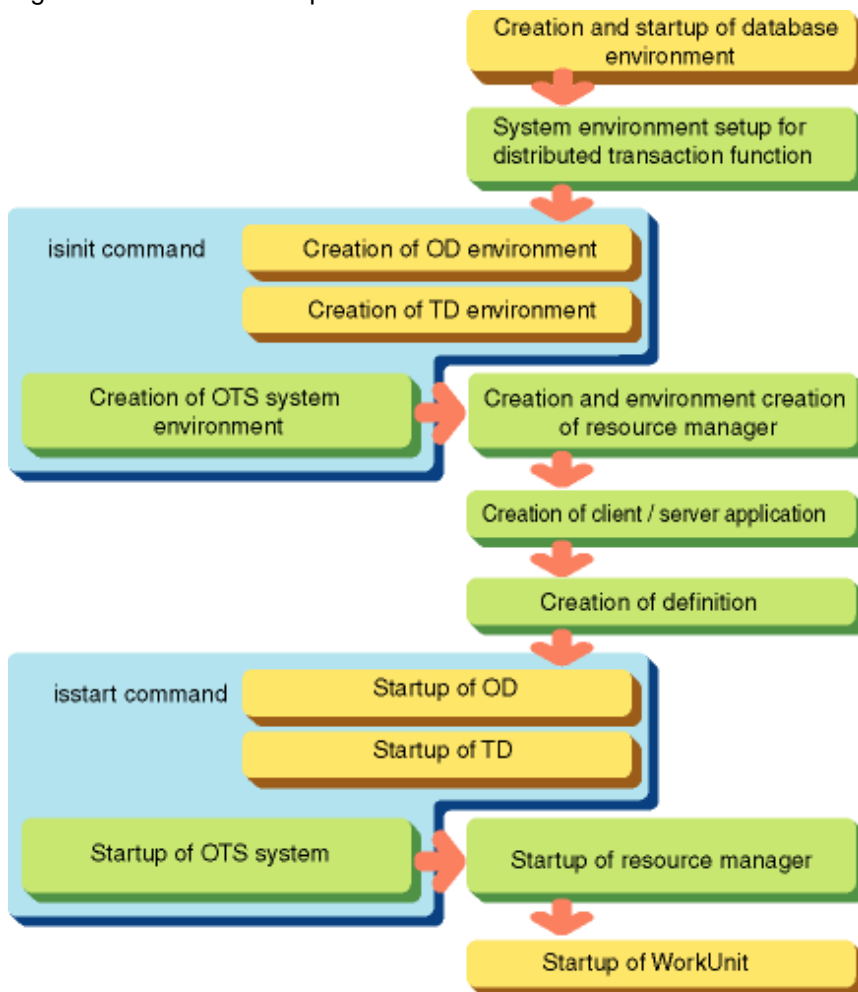
When using the Database Linkage Service, the resources used in the old environment (such as resource definition files and WorkUnit definitions) need to be migrated.

5.1 Procedure Required to Use Distributed Transaction Function

Windows32/64 Solaris32 Linux32/64

Set up the environment and follow the sequence shown in the following figure in order to use the Distributed Transaction Function in Interstage. The Distributed Transaction Function provides a Database Linkage Function. This is described with the appropriate procedures for setting up the environment

Figure 5.1 Procedure Required to Use Distributed Transaction Function

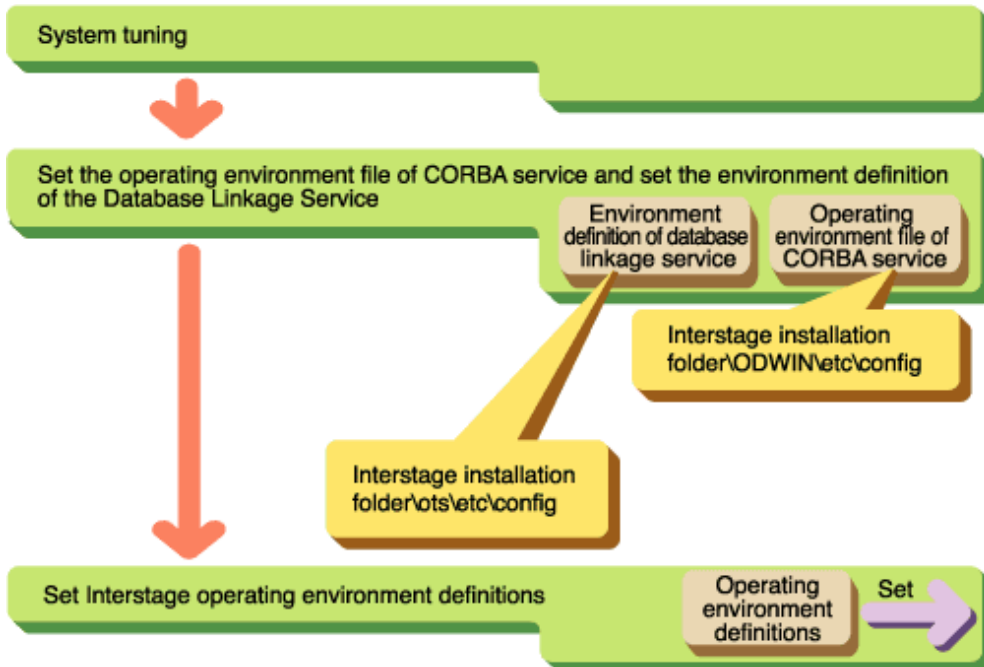


OD: CORBA Service TD: Component Transaction Service
OTS: Database Linkage Service

5.2 Setting Up the System Environment for the Distributed Transaction Function Windows32/64 Solaris32 Linux32/64

The system environment should be set in accordance with the following procedure. Note that the environment definitions vary according to the environment type (CORBA applications and transaction applications). The following figure illustrates the procedure.

Figure 5.2 Setting Up the System Environment for the Distributed Transaction Function



5.2.1 Tuning the System Solaris32 Linux32/64

To use the Distributed Transaction Function, the following resources will require tuning. Tuning of resources to use by the Database Linkage Service is done by the ini file (Interstage installation folder\ots\etc\ots.ini)

- Shared memory
- Semaphore resources
- Message queue

5.2.2 Determining If a Disk Partition Is Necessary Solaris32 Linux32/64

To create system log files for use of the distributed transactions feature using a raw device, a disk partition must be created.

Calculate the required area size using the formula below:

```
maximum number of transactions * X + 1 ( Kbyte)
```

- If up to 4 resources can join one transaction: X = 4
- If more than 4 resources can join one transaction: X = number of resources that can join the transaction

5.2.3 Setting the CORBA Service Operating Environment File

This is only required if a CORBA application uses the distributed application function.

When the Distributed Transaction Function is used in a CORBA application, modify the initial value of the operating environment file of the CORBA Service (Interstage installation folder\ODWIN\etc\config). The setting value is calculated in the formula shown in the following table.

Refer to [5.2.5 Setting Up the Interstage Operating Environment Definitions](#) for information on how transaction applications use the Distributed Application Function.

For information on the meaning of the parameters to be changed, and instructions on setting them, refer to the Tuning Guide.

If the Naming Service runs on a separate machine from the OTS system because of the system configuration, change the operating environment file of the CORBA Service on the machine where the Naming Service is running

Table 5.1 CORBA Service Operating Environment File Settings

Parameter	Formula for Estimating the Quantity to be Used
max_IIOp_resp_con (See Note 1)	If the Database Linkage Service is to be used, add the following values to max_IIOp_resp_con. Value to be added = $3 + x + y + z$ x: Level of process concurrency when Resource Manager is started (default = 6) (See Note 2) y: Number of client applications to be connected z: Number of server application processes to be started
max_IIOp_init_con (See Note 1)	If the Database Linkage Service is to be used, add the following value to max_IIOp_init_con. Value to be added = $2 + x + y$ x: Level of process concurrency when the Resource Manager is started (default = 6) (See Note 2) y: Number of server application processes to be started
max_IIOp_req_per_con	If the Database Linkage Service is to be used, specify a value greater than the concurrency level of the OTS system. (See Note 3)
max_process	If the Database Linkage Service is to be used, add the following value to max_process. Value to be added = $1 + x + y + z$ x: Level of process multilevel concurrency when Resource Manager is started (default = 6) (See Note 2) y: Number of client applications to be started z: Number of server application processes to be started
period_receive_timeout	Please note the following points. Specify a value greater than the timeout monitor period of the transaction. Specify a value at least double the period_idle_con_timeout value.

Notes

1. If this value has been increased from its initial value, the system resources (shared memory et cetera) will need to be tuned.
2. The concurrency level of the Resource Manager is the value specified by the -m option when the *otssetrsc* command is executed + 1
3. The concurrency level of the OTS system is the value specified by the -m option when the *otssetup* command is executed.

5.2.4 Setting Up the Database Linkage Service Environment Definition

The environment definition of the Database Linkage Service (Interstage-installation-folder\ots\etc\config) also needs to be changed in line with operational requirements.. This must be done for both CORBA Service and transaction applications, with the initial values modified according to the operation.

5.2.5 Setting Up the Interstage Operating Environment Definitions

Do this if a transaction application uses the distributed transaction function. It is not required for CORBA applications.

If Interstage is to be operated using a transaction application, use the *isinit* and *isstart* commands. Do not directly use commands such as *otssetup*, *otsstart*, and *otsstop* which are provided by the Database Linkage Service. If using a transaction application, set the Interstage operating environment definitions (Interstage installation folder\td\etc\isreg\isinitdef.txt) before executing the *isinit* and *isstart* commands.

The following definition values required for the Database Linkage Service are automatically set by Interstage according to the system scale. Therefore, change the initial values when needed. Be sure to set the setup type and system log file name.

```
# Set OTS setup type
OTS Setup mode = SYS

# Multilevel concurrency of Database Link Service
OTS Recovery=2

# Be sure to set a system log file name
# to be used by the Database Link Service
OTS path for system log=

# Maximum number of Database Link Service transactions
# If the system is small,
# the maximum number of clients that can connect should be set in
accordance with the size of the system.
OTS maximum Transaction=50
```

5.3 Creating the OTS System Environment

This operation is required only when a CORBA application is used.

It is not required when a transaction application is used.

The Distributed Transaction Function is provided by the OTS System, user applications and resource management programs. The OTS System manages the transaction information.

OTS System

The OTS System consists of multiple objects. It manages transaction information, controls recovery and so on.

5.3.1 Using the Interstage Management Console

The Interstage management console is used to perform environment settings for the transaction service (OTS).

Note

If detailed setup settings are changed, the transaction service environment is rebuilt. This means that all of the previously stored JTS resource definition information is deleted. To reuse these resources in global transactions, select and apply "Use Global Transaction" in the resource environment setting.

5.3.2 Using Commands

An OTS system is usually created as an extension of the environment setup for Interstage as a whole, using the *isinit* command. However, the *otssetup* command is used when the Naming Service is running by a different node from the OTS System.

5.3.2.1 Creating an Interstage Operating Environment Definition

An Interstage operating environment definition is automatically created when an Interstage system definition file is registered.

For details on Interstage operating environment definition, refer to the Interstage Application Server Operator's Guide.

5.3.2.2 isinit Command and otssetup Command

Use the *isinit* command to initialize Interstage.

To initialize Interstage by using the *isinit* command, an Interstage operation environment definition must be created to set required items.

Refer to the Interstage Application Server Operator's Guide for details on the Interstage operation environment definition.

To use the Database Linkage Service, Interstage must be initialized by type2. To use an EJB application, specify *ejb*.

```
isinit type2 ejb
```

To use the remote Naming Service, *isinit* must be executed by type3. In type3, an environment setting for the OTS system is not made.

Therefore, use the *otssetup* command to make environment settings for the OTS system and resource management program.

The following explains how to set the operation environment by the *otssetup* command and delete the operation environment.

```
otssetup -f setup information file
```

To delete the OTS system, specify the following and execute the command.

```
otssetup -d
```

5.3.2.3 Using a Local Naming Service (Recommended)

This section explains the setting method for using the Naming Service of the same host as the one where the Database Linkage Service (OTS system) operates.

For the following definition items required for the Database Linkage Service, Interstage sets values according to the system scale. To change the values appropriately for the operation environment, change the values of the following items.

However, be sure to set a setup type and system log file name.

Setting of Interstage Operation Environment Definition

```
# Set OTS setup type
OTS Setup mode = sys

# Thread concurrency of OTS system
OTS Multiple degree = 5

# Recovery process concurrency of database linkage service
OTS Recovery = 2

# Be sure to set a system log file name
# to be used by the Database Link Service
OTS path for system log =

# Maximum number of Database Link Service transactions
# If the system is small,
# the maximum number of clients that can connect should be set in
accordance with the size of the system.
OTS maximum Transaction = 50
```

Interstage Initialization

Use the *isinit* command to initialize Interstage.

```
isinit type2 ejb
```

5.3.2.4 Using Remote Naming Service

This section explains the setting method for using the Naming Service of a host other than the one where the Database Linkage Service (OTS system) operates.

When the remote Naming Service is used, the *isinit* command cannot be used to initialize the OTS system.

After an environment setting for the Naming Service is made, the *otssetup* command must be used to initialize the OTS system.

Setting of Interstage Operation Environment Definition

```
NS USE = remote
NS Host Name = host name where Naming Service to be used operates
NS Port Number = port number of Naming Service to be used
```

Interstage Initialization

Use the *isinit* command to initialize Interstage.

```
isinit type3
```

Creating Setup Information File

```
# Setting of OTS setup type
MODE = SYS

# Thread concurrency of OTS system
OTS_FACT_THR_CONC = 5

# Recovery process concurrency of database linkage service
OTS_RECV_THR_CONC = 2

# Be sure to set system log file name used in database linkage service.
LOGFILE =

# Maximum number of transactions of database linkage service
# When system scale is small
# Specify the maximum number of clients connected according to system scale.
TRANMAX = 100
```

Environment Setting for Database Linkage Service

Use the *otssetup* command to initialize OTS.

```
otssetup -f setup-information-file
```

5.4 Creating a Resource Management Program Windows32/64 Solaris32

Linux32/64

This section describes how to create a Resource Management program. An overview of the process is shown in the following figure. It is not necessary to create a Resource Management Program for JTS.

Figure 5.3 Creating a Resource Management Program



Note

There are no rules for the sequence in which resource management and user application programs are created. Either can be created first.

5.4.1 What is a Resource Management Program?

A Resource Management Program is a system program running in a processing space independent from the OTS system, and operates as an object on the machine where the database is running. The Resource Management Program runs in multiprocessing mode (6 concurrent processes by default) and cannot run in multithread mode.

There are two types of Resource Management Programs; the resource management programs for OTS and those for JTS. The Resource Management Program for OTS is required when a C, C++, or COBOL application performs global transaction linkage. The Resource Management Program for JTS is required when an EJB application performs global transaction linkage.

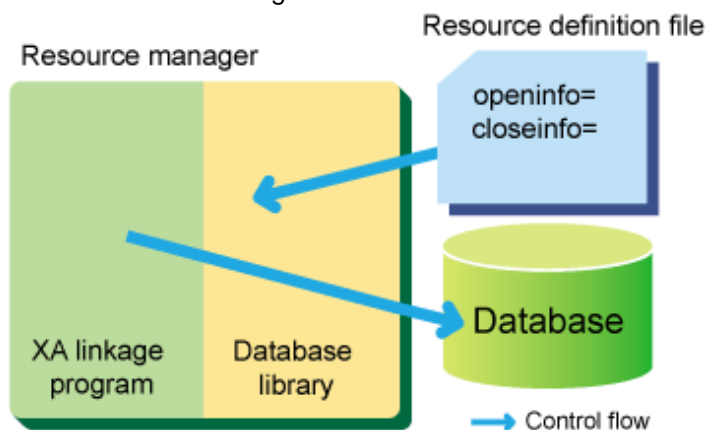
When a transaction completion request (commit/rollback) is issued from the client application, the Resource Management Program receives it via the OTS system and reports transaction completion to the database. If the Resource Management Program is started while an unsolved transaction remains, the unsolved transaction is also recovered.

Depending on the operating environment, in some cases the Resource Management Program can also be started on a separate machine from the one where the OTS system is running.

One is made in case of Oracle on every instance, in case of Symfoware/RDB on every the database, in case of SQLServer to SQLServer, and you must start a Resource Management Program for OTS.

Resource Management Program for JTS does not need to be created.

Figure 5.4 Resource Manager



As shown in the Figure 5-4, the Resource Management Program for OTS consists of an XA linkage program and a database library.

For the Resource Management Program for JTS, no XA linkage program exists.

When the Resource Manager for OTS is started, it connects to the database based on the information in the resource definition file. Therefore, when the Resource Management Program is started, the database described in the resource definition file needs to have been started.

When the Resource Manager for JTS is started, it reads all the resource definition files with RSCTYPE=JTS specified in the resource definition file registered with the *otssetrsc* command.

Database Library

The database library is the library provided by the database.

It is used when creating Resource Management Program and user application programs. Refer to the specific database manual for details of libraries.

JDBC Class Library

For the Resource Manager for JTS, the JDBC library for JDBC2.0OP (Optional Package) provided by the vendor is required.

5.4.2 The XA Linkage Program

An XA linkage program is a program that links to the database using the XA interface.

If an XA linkage program is to be used, you will have to create it using the *otsmkxapgm* command.

Note

In the Resource Manager for JTS, a XA linkage program does not need to be created.

The preparation of the program for the XA linkage

```
otsmkxapgm -s xaosw -r "/libpath:'C:\ORACLE\RDBMS\XA' oraxa10.lib" -o  
c:\temp\ots\oraxa.dll
```

5.4.3 The Database Library

The database library shows the library provided by the database.

It is used when creating Resource Managers and server applications. Further details on the library will be found in the manual specific to each database.

5.4.4 Creating a Resource Management Program

A Resource Manager can be created by linking together the XA linkage program and the database library published by the database vendor, using the *otslinkrsc* command.

Note

In the Resource Manager for JTS, a XA linkage program does not need to be created.

Example of the Resource Management Program [Windows32/64](#)

```
otslinkrsc -l xa_linkpgm -r "library" -o name
```

- l xa_linkpgm: name of XA linkage program
- r "library": library published by database vendor
- o name: name of Resource Manager to be created

Example of the Resource Management Program [Solaris32](#) [Linux32/64](#)

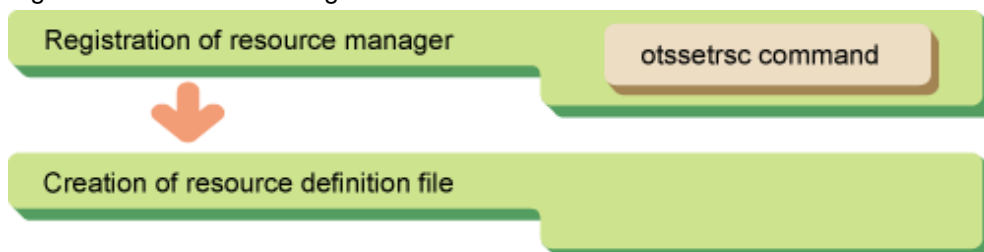
```
otslinkrsc [-t {thread | process} ] -l xa_linkpgm -r "library" -o name
```

- t {thread | process} : mode of process or thread
- l xa_linkpgm: name of XA linkage program
- r "library": library published by database vendor
- o name: name of Resource Manager to be created

5.5 Creating the Environment for the Resource Management Program [Windows32/64](#) [Solaris32](#) [Linux32/64](#)

Create the environment using the procedure shown in the following figure before starting the new Resource Manager program.

Figure 5.5 Resource Manager Environment



Next, you will need to create a resource definition file to store the information comprising the OPENINFO and CLOSEINFO character strings for the XA interface required by each database vendor.

5.5.1 Creating a Resource Definition File

A resource definition file contains the resource definition required for each resource (such as a database). A resource definition file must be created for each database, and must describe, in text format, items such as the `OPENINFO` and `CLOSEINFO` character strings for each database type name. In the Resource Manager for JTS, the information for acquiring the mounting class of `javax.transaction.xa.XADataSource` (`initialContextFactory`, `providerURL`, etc.) provided by each database vendor from JNDI is described in text format. The name of this resource definition file is specified as a parameter when registering or starting the Resource Manager. When starting the Resource Manager for JTS, it does not need to be set.

Typical settings for the resource definition file are shown in the following subsections.

An Example of the Resource Definition for OTS of Oracle

```
# resource definition name
NAME=ORACLE_DEF
# environment variables
ENVIRON ORACLE_SID=orac
# name of RM to be used, and OPENINFO and CLOSEINFO character strings
RMNAME=Oracle_XA
OPENINFO=Oracle_XA+Acc=P/system/manager+SesTm=0
CLOSEINFO=
```

An Example of the Resource Definition for JTS of Oracle

```
# resource definition name
name=ORACLE_JTS_DEF
# initialContextFactory, providerURL used
type=DBMS
lookUpName=jdbc/xa/OracleXADataSource
initialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory
providerURL=file:/JNDI/xa
rscType=JTS
```

An Example of Resource Definition for OTS of Symfoware/RDB

```
# resource definition name
NAME=ORACLE_DEF
# name of RM to be used, and OPENINFO and CLOSEINFO character strings
RMNAME=RDBII
OPENINFO=TO 'DB1' USER 'system/manager'
CLOSEINFO=
```

SQL Server Resource Definition File

Windows32

```
# name of RM to be used, and OPENINFO and CLOSEINFO character strings
RMNAME=MS_SQL_Server
OPENINFO=Tm=OTS,RmRecoveryGuide=197BAA60-8011-11d2-B342-0000E20F0756
CLOSEINFO=Tm=OTS,RmRecoveryGuide=197BAA60-9011-11d2-B342-0000E20F0756
```

The resource definition file can be created using a text editor. Refer to the Tuning Guide for details of the format of the resource definition file.

5.5.1.1 Registering a Resource Definition

To validate the resource definition created for each resource, register the resource definition using the `otssetrsc` command. A JTS resource definition can be registered via the Interstage management console.

The database service may be uninstalled from and reinstalled in the same environment. If the same resource definition is to be used subsequently, the registered information need not be deleted by the `otssetrsc` command prior to installation.

Notes

- Up to 32 resource definitions can be registered.
- If the CORBA Service is reinstalled or initialized, the resource definition must be re-registered.

5.5.1.2 Environment Definition for Resource Management Program

To implement an environment definition for a JTS resource management program that is to be used by the distributed transaction function of a J2EE application, perform either of the following:

- Update config file
- Update RMP property

Updating a config File

On the PATH= line defined in the config file, specify in full path the path to the java command to be used by the resource management program for JTS. Refer to the Tuning Guide for details.

Updating the RMP Property

For "RecoveryTarget=" defined in the RMP property file, specify the names of the resources to be recovered from a failure, delimited by a blank. If the recovery objects are not set, the failure recovery is not made during start of the resource management program for JTS. Normally specify this option.

Refer to the Tuning Guide for details.

5.6 Creating Definitions Windows32 Solaris32 Linux32

This involves creating the WorkUnit definitions. This is only necessary for transaction applications using the Distributed Transaction Function.

This section explains the definition information for the use of the Distributed Transaction Function.

5.6.1 Resource Manager Information

Specify the [Resource Manager] database information when using global transactions (*1).

Example of definition of Resource Manager information

```
#Resource Manager information
[Resource Manager]
  Name:RDBII_DEF
  RM:RDBII

[Resource Manager]
  Name:ORACLE_DEF
  RM:Oracle_XA
```

*1 This is only specified if the WorkUnit type is ORB.

5.7 Starting the OTS System Windows32/64 Solaris32 Linux32/64

To use the Interstage management console to start up the OTS system, start Interstage to synchronously start up the OTS system. To start up only the OTS system, use the *otsstart* command. To start the OTS system, the *otsstart* command is executed. Execute this command when a CORBA application uses the Distributed Transaction Function. When using transaction applications, the *isstart* command starts the OTS system together with all the services.

For that reason, execute the following command to start the CORBA Service for CORBA applications:

```
otsstart
```

To stop the OTS system, the *otsstop* command is executed. However, if a transaction application is to be run, the *isstop* command will be used, so this operation is unnecessary.

```
otsstop
```

5.8 Starting and Stopping a Resource Management Program

Windows32/64 Solaris32 Linux32/64

To start the Resource Management Program, the *otsstartsrc* command is executed. This is carried out on the machine on which the Resource Management Program is started. Do this separately for CORBA and transaction applications. The CORBA Service must be started before the following command is executed:

Resource Management Program for OTS

```
otsstartsrc -pg C:\temp\ots\resource1.exe -n resourcedef1
```

C:\temp\ots\resource1.exe: name of Resource Manager

resourcedef1: The resource definition name when registered with *otssetsrc*

To stop the Resource Manager, the *otsstopsrc* command is executed.

```
otsstopsrc -n resourcedef1
```

Resource Management Program for JTS

```
otsstartsrc -j
```

```
otsstopsrc -j
```

When the Interstage management console is used to set up a transaction service, start Interstage to start the resource management program for JTS as the transaction service (JTSRMP).

5.8.1 Environment Setting for Operation on a Host Other Than That of the OTS System

This section explains the environment setting required for operating the resource management program on a host other than the one on which the OTS system operates.

5.8.1.1 Sharing a Naming Service (Recommended)

A Naming Service is shared between the host on which the OTS system operates and the one on which the resource management program operates.

1. Specify the following in the Interstage operating environment definition for the host on which the resource management program operates.

```
NS USE = remote
NS Host Name = name of host where Naming Service to be used operates
NS Port Number = port number of Naming Service to be used
```

2. At the host on which the resource management program operates, select *type3* in the *isinit* command to initialize Interstage.
3. At the host on which the resource management program operates, execute the *isstart* command to start Interstage. (The OTS system is not started up.)
4. At the host on which the resource management program operates (Windows only), use the *net* command to start the *ObjectTransactionService*.
5. At the host on which the OTS system operates, use the *isstart* command to start up that OTS system. (*1)

6. At the host on which the resource management program operates, specify the following in the setup information file. (*2)

```
MODE = RMP
```

7. At the host on which the resource management program operates, use the otssetup command to initialize Interstage.
8. At the host on which the resource management program operates, use the otsstartsrc command to start that resource management program.

*1 If the OTS system uses a non-local Naming Service, set the Naming Service to be used in the Interstage operating environment definition in the same way as the setup for a resource management program to initialize Interstage. Refer to [5.3.2.4 Using Remote Naming Service](#) for details.

*2 Do not specify HOST and PORT.

5.8.1.2 Not Sharing a Naming Service

Different Naming Services are used at the host on which the OTS system operates and the one on which the resource management program operates. (The resource management program uses the local Naming Service.)

1. At the host where the resource management program operates, specify the following in the Interstage operation environment definition file.

```
OTS Setup mode = rmp
OTS Host = name of host where OTS system operates
OTS Port= port number of CORBA service at node where OTS system operates
OTS Locale= locale of host where OTS system operates
```

2. At the host where the resource management program operates, select type2 in the isinit command to initialize Interstage.
3. At the host where the resource management program operates, use the isstart command to start Interstage. (The OTS system is not started up.)
4. At the host where the OTS system operates, use the isstart command to start up that OTS system.
5. At the host where the resource management program operates, use the otsstartsrc command to start that resource management program.

Different Naming Services are used at the host on which the OTS system operates and the one on which the resource management program operates. (The resource management program uses a remote Naming Service.)

1. At the host where the resource management program operates, specify the following in the Interstage operation environment definition file.

```
NS USE = remote
NS Host Name = name of host where Naming Service to be used operates
NS Port Number = port number of Naming Service to be used
```

2. At the host where the resource management program operates, select type3 in the isinit command to initialize Interstage.
3. At the host where the resource management program operates, use the isstart command to start Interstage. (The OTS system is not started up.)
4. At the host where the resource management program operates, create a setup information file to set the following:

```
OTS Setup mode = rmp
OTS Host = name of host where OTS system operates
OTS Port = port number of CORBA service of host where OTS system
operates
OTS Locale = locale of host where OTS system operates
```

5. At the host where the resource management program operates (Windows only), use the net command to start the ObjectTransactionService service.
6. At the host where the resource management program operates, use the ostsetup command to specify and set up the setup information file.

7. At the host where the resource management program operates, use the `isstart` command to start Interstage. (The OTS system is not started up.)
8. At the host where the OTS system operates, use the `isstart` command to start up that OTS system.
9. At the host where the resource management program operates, use the `otsstartsrc` command to start that resource management program.

Notes

- This section contains only the information required to operate the resource management program on another host. Refer to the Tuning Guide for details on the Interstage operation environment definition file and setup information file.
- Refer to [5.3.2.4 Using Remote Naming Service](#) for details on the settings when the OTS system uses the Remote Naming service.

5.9 Tracing Function Windows32/64 Solaris32 Linux32/64

The following two tracing functions can be used in the Database Linkage Service.

- Dump file collection function
- Trace log output function

5.9.1 Dump File Collection Function

By using the `otsgetdump` command, the dump file collection function can collect the dump files of the OTS system and resource management program. This facilitates recovery from a failure, if any on the system, by referencing the output information and investigating the transaction information. Refer to the Reference Manual (Command Edition) for details on using the `otsgetdump` command.

5.9.2 Trace Log Output Function

By using the trace log output function, detailed information on failures in the communication layer and those in a resource (such as a database) linked with the Database Linkage Service can be collected. The operation of the data linkage service can also be checked.

The trace log output function is used for the following three processes:

- Command
- Application
- Resource Management Program

Command Trace Log

This function automatically outputs a trace log when the processing of the Database Linkage Service is executed, by using a command provided by the Database Linkage Service and Interstage integration command. The contents of the file are separately output for each command regardless of whether that command is normal or abnormal.

5.10 Notes Windows32/64 Solaris32 Linux32/64

This section provides notes on linking to Interstage under Interstage V3.1 and higher.

5.10.1 Migration from the Old Environment

When converting from an V3.1 environment or earlier, the environment and definitions must be converted.

Appendix A WorkUnit Definition

WorkUnit definition is described in the following format.

Note

Upon completion of structuring the application environment, creating a backup copy of the resources is recommended for the possible crash of resources.

For the procedure of creating a backup copy, see Maintenance (Backup of resources) in the Interstage Application Server Operator's Guide.

A.1 Syntax

The syntax of a WorkUnit definition is the same as that for the Component Transaction Service environment definition. Refer to the Component Transaction Service environment definition section for details.

A.2 Syntax of WorkUnit Definition File

A WorkUnit definition is specified in the following format:

[WORK UNIT] section

WORK UNIT information

[APM] section

APM name

[Control Option] section

Control option

[Resource Manager] section

Resource Manager information

[Application Program] section

Application information

[Nonresident Application Process] section

Nonresident application information

[Multiresident Application Process] section

Multi-object resident application information

The following is an example of a WorkUnit definition.

WORK UNIT

Definition name and type

Name: WorkUnit name

Kind: WorkUnit type

APM

APM name

Name: APM name

Control Option

Control option

Path: application-library-path

Path: application-library-path

To define multiple application library paths, write multiple "Path:" statements.

- Current Directory: current-folder
- Remove Directory: Whether or not the APM (server application) current folder is to be deleted
- Application Retry Count: abend-limit
- Snapshot: snapshot-acquisition-specification
- Path for Snapshot: snapshot-output-path
- Path for Application: application-use-path
- Path for Application: application-use-path

To define multiple application paths write multiple "Path for Application:" statements.

- Environment Variable: Environment variable
- Environment Variable: Environment variable

To define multiple environment variables, write multiple "Environment Variable:" statements.

- Registration to Naming Service: Registration status of Naming Service
- Using Load Balance: Use load balance function Y/N (*1)
- Using Notification of User Information: Notify user identification information Y/N
- Access Control: Access control used Y/N
- Access Control Base DN: Base ID for entry subject to access control

*1 This is not valid for Linux (64 bit).

Resource Manager

Resource manager information

File: Resource definition file

RM: RM name

To define multiple resource manager information, write multiple [Resource Manager] sections.

Application Program

Application information

Destination: Destination name (object name)

Destination Priority: Object priority

PSYS: DPCF communication path

Executable File: Execution file name

Application Language: Application language

Code Conversion For String: Code conversion information (string)

Concurrency: Process multi level

Maximum Processing Time: Application maximum processing time

Terminate Process for Time out: Forced termination when maximum processing time has elapsed Y/N

Maximum Processing Time for Exit Program: Maximum processing time for exit program

Maximum Queuing Message: Maximum number of messages in queues

Queuing Message to Notify Alarm: Number of queues monitored

Queuing Message to Notify Resumption: Number of queues to resume monitoring

Environment Variable: Environment variable

Environment Variable: Environment variable

To define multiple environment variables, write multiple "Environment Variable:" statements.

From: FormResident or nonresident type

Pre Exit Program: Pre exit program

Post Exit Program: Post exit program

Recovery Exit Program: Name of recovery exit program

Executable File for Exit Program: Executable file for exit program

Access Control: Access control is used Y/N

Access Control Base DN: Base ID for entry subject to access control

Type of User Identification: Type of user identification information

User Name Param: User name notification parameter

User Base DN: User name detection starting point DN

User DN Param: User ID notification parameter

Password Param: Password notification parameter

Bind type: Bind type

Using Wrapper Session Management: AIM linkage session inheritance function used Y/N

SessionID Param: SessionID notification parameter

Method Name to Begin Session: Method (operation) to begin session

Maximum Session Active Time for Client: Maximum thinking time for client

Maximum Processing Time for WRAPPER: AIM application monitoring time

Maximum Memory for EJB Application: Maximum memory for EJB applications

CLASSPATH for Application: Class path for applications

CLASSPATH for Application: Class path for applications

...

To define multiple class paths for applications, write multiple "CLASSPATH for Application:" statements. To define multiple application information, write multiple [Application Program] sections.

Nonresident Application Process

NONRESIDENT APPLICATION INFORMATION

Concurrency: Process multi level

Pre Exit Program: Pre exit program name

Post Exit Program: Post exit program name

Executable File for Exit Program: Executable file name for exit program

Dynamic Link Library: Dynamic link library name

Dynamic Link Library: Dynamic link library name

...

To define multiple dynamic link libraries, write multiple "Dynamic Link Library:" statements.

Maximum Processing Time for Exit Program:

Multiresident Application Process

MULTIRESIDENT APPLICATION INFORMATION

Concurrency: Process multi level

Pre Exit Program: Pre exit program name

Post Exit Program: Post exit program name

Executable File for Exit Program: Executable file name for exit program

Maximum Processing Time for Exit Program: Maximum processing time for exit program

Recovery Exit Program: Recovery exit program name

A.3 Control Statement of WorkUnit Definition

After the beginning of the WorkUnit definition is declared in the [WORK UNIT] section, WorkUnit information is described as follows:

- [APM] section
- [Control Option] section
- [Resource Manager] section
- [Application Program] section
- [Nonresident Application Process] section
- [Multiresident Application Process] section

If the WorkUnit type is WRAPPER or EJB, sections other than [Control Option] and [Application Program] are ignored.

Some sections can be omitted, depending on the type of WorkUnit and the functions used by them. Where they can be omitted, omit the section name and the statement.

A.3.1 WORK UNIT Section

Declare the start of the WorkUnit definition.

Synopsis

[WORK UNIT]

Name:

WorkUnit name

Kind:

WorkUnit type

A.3.1.1 Name

WorkUnit name

Explanation

Set the WorkUnit name.

The WorkUnit name can contain up to 36 bytes of alphanumeric characters, hyphens, and underscores, and must begin and end with an alphanumeric character.

A.3.1.2 Kind

WorkUnit type

Explanation

Set the WorkUnit type.

- CORBA: Specified when CORBA applications operate in a WorkUnit.
- ORB: Specified when transaction applications operate in a WorkUnit.
- EJB: Specified when EJB applications operate in a WorkUnit.
- WRAPPER: Specified for AIM linkage. No application can operate in the applicable WorkUnit.
- UTY: Specified when a general application outside the control of Interstage is operated in a WorkUnit.

Note

"UTY" is supported for Solaris and Linux, but not for Windows.

"WRAPPER" is supported for Windows and Solaris, but not for Linux.

A.3.2 APM Section

Specify APM name.

Synopsis

APM

Name:

APM name

A.3.2.1 Name

APM name

Explanation

Set APM name.

Specify the APM name in alphanumeric characters consisting of 23 or less bytes.

Set "TDNORM" to the NAME of the [APM] section only if the application language in the WorkUnit definition is C++. Set "TDNORM" to the APM name when not using the global transaction linkage function. When the global transaction linkage function is used, specify the APM name created by the APM creation command (tdlinkapm).

Be sure to specify this statement when the WorkUnit type is ORB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.3 Control Option Section

Specify control option of WorkUnit.

Ensure that this section is only set up at one location in the WorkUnit definition.

Synopsis

Control Option

Path:

application library path

Current Directory:

current-folder

Remove Directory:

Whether or not APM (server application) current folder is to be deleted

Application Retry Count:

abend-limit

Snapshot:

snapshot-acquisition-specification

Path for Snapshot:

snapshot output path

Path for Application:

application-use-path

Environment Variable:

environment variable

Registration to Naming Service:

registration type of Naming Service

Using Load Balance:

This is not valid for Linux (64 bit).

use load balance function Y/N

Using Notification of User Information:

notify user identification information Y/N

Access Control:

existence of the access control execution

Access Control Base DN:

cardinal point distinct name of access control applicable entry

Maximum Processing Time for Exit Program:

maximum processing time for exit program

WorkUnit Exit Program:

WorkUnit exit program name

Executable File of Exit Program for Salvage:

executable file of exit program for salvage

WorkUnit Automatic Stop Mode

WorkUnit automatic stop mode

Request Assignment Mode

Request assignment mode

Traffic Director Monitor Mode

Existence of IPCOM monitoring

Output of Stack Trace

Existence of stack trace acquisition at the time of timeout detection

Startup Time

WorkUnit startup waiting time

Shutdown Time

WorkUnit forced stop waiting time

Unconditional Reactivation of Process

Unconditional restart of the process

Start Log

Collection of the process start log

Process Degeneracy

Degenerated process operation

Number of Revision Directories

Number of backups of the current directory

Syntax

- The following statements are mandatory if the WorkUnit type is "ORB":
 - "Path:" statement
 - "Current Directory:" statement
- The following statement is mandatory if the WorkUnit type is "EJB":
 - "Current Directory:" statement
- Up to 10 of the following statements can be specified. To create multiple specifications, write multiple statements:
 - "Path:" statement
- Up to 30 of the following statements can be specified. To create multiple specifications, write multiple statements:
 - "Path for Application:" statement
- Input a newline /n at the end of the "Environment Variable:" statement line (a 2 byte code is automatically added to the string length).
- Multiple "Environment Variable:" statements can be specified, however, the length of the strings specified in the "Environment Variable:" statements is restricted to 4096 bytes.
- If the "Snapshot:" statement is set to DISABLE, the "Path for Snapshot:" statement is ignored.
- If the "Snapshot:" statement is set to ENABLE and the "Path for Snapshot:" statement is omitted, snapshot information is output to the current application runtime directory.

A.3.3.1 Path

Application library path

Explanation

Set the path to the directory where the application and exit program executables are stored.

When the application language type is C++, set the path to the directory where the application and exit program executables are stored.

When the WorkUnit type is CORBA and the application language is Java, specify the path to the Java execution file directory.

This statement can be specified in up to 10 statements. To specify more than one statement, the specification must be repeated. Note that the same path cannot be duplicated when specifying the statements.

A string of up to 255 bytes without spaces, beginning with "/".

Be sure to specify the absolute path in this statement. The relative path and current directory "." cannot be specified.

Be sure to specify this statement when the WorkUnit type is CORBA, ORB, EJB, or UTY.

Range of Support

OS	Windows, Solaris, Linux
----	-------------------------

Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, ORB, EJB, UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.3.2 Current Directory

Explanation

Set the path to create the current folder for the application and exit programs.

A string of up to 255 bytes without spaces, beginning with "/".

Always specify an existing path in this statement. If the path cannot be located, the WorkUnit fails to start.

Be sure to specify the absolute path in this statement. The relative path and current directory "." cannot be specified.

A directory with the WorkUnit name is created in the directory specified by this statement. In this created directory, a directory with the process ID is created. This process ID directory is actually the current directory.

If the directory specified in this statement has insufficient disk space, the WorkUnit may fail to start. In addition, in a Solaris or Linux system, when an application process is terminated abnormally, a core file is output. Therefore, secure enough disk space.

Be sure to specify this statement when the WorkUnit type is CORBA, ORB, EJB, or UTY.

In the case of the Java language, the stdout and stderr files are created but data is not output to them.

When the Java language is used to output data to the files, the program must be prepared in such a way that the user allocates the standard output and standard error output to those files.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, ORB, EJB, UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.3.3 Remove Directory

Specify whether the current directories of the application and exit program (process ID directories) are to be deleted when APM is stopped.

Explanation

Specify whether or not the APM (server application) current folder is to be deleted when APM stops.

- YES: Delete the APM (server application) current folder.
- NO: Do not delete the APM (server application) current folder.

This statement can be omitted. If so, the default is "NO".

Note that, if files that have a file size of 1 or more are stored in the current folder, the current folder is not deleted even if "YES" is set for this statement.

This statement is valid when the WorkUnit type is ORB.

If activation change or process concurrency has repeatedly been increased and decreased for the WorkUnit of a transaction application, many directories with process IDs will remain. This statement provides a function to prevent the accumulation of directories of finished processes.

Even if "NO" is specified in this statement, the directories and files in the current directory are deleted when the WorkUnit is restarted.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	CORBA

A.3.3.4 Application Retry Count

abend-limit

Explanation

An integer value from 0 to 127 specifying the consecutive abended application count until automatic restart is disabled.

This statement is optional. If it is specified as zero, or not specified, re-starting is performed unconditionally.

This statement is valid when the WorkUnit type is CORBA, ORB, EJB, or UTY.

Note

- When 1 is specified in this statement, the application process is not restarted and the WorkUnit stops abnormally.
- When 2 or more is specified in this statement, the WorkUnit stops abnormally if execution of a server application ends abnormally and restarts consecutively until the specified number of retries is reached. When the server application recovers normally, the counter is reset.
- In CORBA WorkUnits in which logic is used to generate the object reference in the first method as with "Factory" or object unit process bind, the counter is reset even if this method is operating normally. Even if an ABEND occurs during business method invocation, a value of 2 or more may therefore be invalid.
- For an EJB WorkUnit, the counter is reset when the "create" method operates normally. Even if an ABEND occurs during business method invocation, a value of 2 or more may therefore not be valid.
- For a utility WorkUnit, specify a retry count reset time (in the Reset Time for Application Retry Count: statement in the Application Program section) so that the counter is reset when the specified time has elapsed.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, ORB, EJB, UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.3.5 Snapshot

snapshot-acquisition-specification

Explanation

Set whether a file output snapshot is obtained.

- ENABLE: Obtain a snapshot
- DISABLE: Do not obtain a snapshot

The default is "DISABLE".

If "ENABLE" is specified in this statement, the snapshot information file is output in the current directory. To change the output directory of the snapshot information file, specify the "Path For Snapshot" statement.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.3.6 Path for Snapshot

Snap shot output path

Explanation

This statement can be omitted. If so, the current folder is set as the default value.

A string of up to 255 bytes without spaces, starting with "/".

Be sure to specify the absolute path in this statement. The relative path and current directory "." cannot be specified.

This statement is only valid when the WorkUnit type is ORB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.3.7 Path for Application

Explanation

Set the path to be used by the application and exit programs at run time.

This statement can be specified in up to 30 statements. To specify more than one statement, the specification must be repeated. Note that the same path cannot be duplicated when specifying the statements.

If two or more paths are specified, they are set in the "PATH" environment variable in the order of specification. Note that the value in the "PATH" environment variable specified in the Interstage startup environment is added after the value set by this statement.

A string of up to 255 bytes without spaces.

This statement is optional. The default value is the path used by the application.

If this statement is omitted, it is assumed that there is no path used by the application.

This statement is valid when the WorkUnit type is CORBA, ORB, EJB, or UTY.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, ORB, EJB, UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.3.8 Library for Application

Explanation

Set the library path to be used by the application and exit programs at run time.

This statement can be specified in up to 30 statements. To specify more than one statement, the specification must be repeated. Note that the same path cannot be duplicated when specifying the statements.

A string of up to 255 bytes without spaces.

A value that can be set in the LD_LIBRARY_PATH environment variable can be specified in this statement.

This statement is optional. The default value is the Library path used by the application.

If this statement is omitted, it is assumed that there is no Library path used by the application.

This statement is valid when the WorkUnit type is CORBA, ORB, EJB, or UTY.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, ORB, EJB, UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.3.9 Environment Variable

Explanation

Set the environment variable to be used when the application and exit programs are run.

This statement can be specified more than once. In this case, the character strings specified in all of the Environment Variable statements must total 4,096 or fewer bytes.

Specify the setting in the following format:

"environment variable= value"

Checks other than character type and length are not performed.

This statement is optional. If it is not specified, no default library is assumed.

The following word cannot be used because it is the reserved word of this statement.

- PATH
- LD_LIBRARY_PATH

If addition is required, specify the addition using the "Path for Application" or "Library for Application" statement.

This COBOL runtime check can be avoided by setting the following environment variable in this statement.

- CBR_CODE_CHECK=no

This statement is valid when the WorkUnit type is CORBA, ORB, EJB, or UTY.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, ORB, EJB, UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.3.10 Registration to Naming Service

Explanation

Specify the Naming Service registration type.

- MANUAL: manual registration. Specify the objects in advance.
- AUTO: auto registration default.

Interstage performs the following operations automatically:

- Registers the objects in the Naming Service before the WorkUnit starts.
- Deletes the objects from the Naming Service when the WorkUnit stops.

Note: When manually registering the object in the Naming Service or load balance function, specify the following Implementation Repository ID per operation type in the -a option of the OD_or_adm or *odadministerlb* commands. Also, start Interstage first, and then register the objects.

If the WorkUnit type is ORB FUJITSU-Interstage-TDLC

If the WorkUnit type is WRAPPER FUJITSU-Interstage-TDRC

Always use manual registration if a proxy link is used. Use the OD_or_adm command with the -x option to specify the Inbound Proxy information. If SSL is used for communication, ensure that the objects are registered manually. In such cases, specify the -s option of the OD_or_adm command.

This statement is only valid when the WorkUnit type is ORB.

Load balance function is not valid for Linux (64 bit).

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB, WRAPPER

Note

"WRAPPER" is supported on Windows and Solaris, but unsupported on Linux.

A.3.3.11 Using Load Balance

This is not valid for Linux (64 bit).

Explanation

Specify whether to use load balance Y/N.

- YES: Use load balance
- NO: Do not use load balance

This statement is optional. The default value is NO.

If the Naming Service has already been set, this setting becomes effective.

This statement is only valid when the WorkUnit type is ORB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB, WRAPPER

Note

"WRAPPER" is supported on Windows and Solaris, but unsupported on Linux.

A.3.3.12 Using Notification of User Information

Explanation

Specify whether to report user identification information.

- YES: Report user identification information
- NO: Do not report user identification information

This statement is optional. The default value is NO.

This statement is only valid when the WorkUnit type is ORB.

Range of Support

OS	Windows, Solaris
Edition	Enterprise Edition

WorkUnit type	ORB
---------------	-----

A.3.3.13 Access Control

Explanation

Specify the existence of the execution of the access control for the WorkUnit.

- YES: A WorkUnit is made the target of the access control.
- NO: A WorkUnit isn't made the target of the access control.

This statement can be omitted. When this statement is omitted, NO is set up as an optional value.

And, regardless of the designation of the "Access Control" statement of the [Application Program] section, it is effective. When both designations are given, access control for the WorkUnit is carried out first, and next access control for the object is carried out.

The EJB WorkUnit is not included in the security functions of the Component Transaction Service.

This statement is only valid when the WorkUnit type is ORB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB, WRAPPER

Note

"WRAPPER" is supported on Windows and Solaris, but unsupported on Linux.

A.3.3.14 Access Control Base DN

Explanation

Specify the distinct name (DN) of entry to find the WorkUnit registered in InfoDirectory, when the WorkUnit needs to be accessed.

The above statement is mandatory when the "Access Control" statement of the [Control Option] section is set to YES.

The length of the DN should not exceed 1023 bytes.

Setup example:

- Access Control Base DN: "ou=ISResouce, o=Fujitsu, c=jp"

Note: Specify the DN of the top most directory in the folder class so that the access control can look up the WorkUnit in the InfoDirectory. If the WorkUnit fails to start, due to it not being registered, specify the next highest directory in the hierarchy.

This statement is only valid when the WorkUnit type is ORB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB, WRAPPER

Note

"WRAPPER" is supported on Windows and Solaris, but unsupported on Linux.

A.3.3.15 Maximum Processing Time For Exit Program

Explanation

Specifies the monitoring time (in seconds) of the maximum processing time of the exit program.

An integer value of 1 to 1800.

This statement can be omitted. If this statement is omitted, 300 is set as a default value.

This statement is valid when the WorkUnit type is CORBA, ORB, EJB, or UTY.

Note

A value in this statement is valid for each of the following exit programs. If the processing time of each exit program exceeds the specified time, the process where that exit program operates is forcibly aborted.

- Pre exit program

If the time is exceeded, the program causes WorkUnit startup to fail.

- Post exit program

WorkUnit stop processing continues even if the time is exceeded.

- Error exit program

If the time is exceeded, the program stops the application process. If automatic restart is set, the program restarts the application process. If automatic restart is not set, the WorkUnit stops abnormally.

- WorkUnit exit program

If the time is exceeded when the program is executed at WorkUnit startup, the program causes WorkUnit startup to fail. If the time is exceeded when the program is executed at the time of WorkUnit stop processing, the WorkUnit stop processing continues.

- Process release exit program

WorkUnit stop processing continues even if the time is exceeded.

- Process termination exit program

WorkUnit stop processing continues even if the time is exceeded.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, ORB, EJB, UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.3.16 WorkUnit Exit Program

Explanation

Set the function name of the exit program to be executed before the process starts at start of the WorkUnit and after the process is released at stop of the WorkUnit.

A character string of up to 31 bytes.

This statement can be omitted.

This statement is valid when the WorkUnit type is CORBA, ORB, EJB, or UTY.

Note

- The WorkUnit exit program is executed even after control fails to start the process and then fails to start the WorkUnit, or after the WorkUnit stops abnormally and then its process is released.
- When this statement is set, be sure to specify the exit program execution file name in the Executable File of Exit Program for Salvage statement.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition

WorkUnit type	CORBA, ORB, EJB, UTY
---------------	----------------------

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.3.17 Executable File of Exit Program for Salvage

Explanation

Specifies an executable file name of the exit program for salvage.

A character string of up to 31 bytes. This statement does not distinguish between upper case and lower case.

This statement can be omitted.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, ORB, EJB, UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.3.18 WorkUnit Automatic Stop Mode

Explanation

To implement a batch program on the utility WorkUnit, specify YES. If YES is specified, the current process is not restarted even if it terminates, but the WorkUnit is automatically stopped when all of the processes under that WorkUnit terminate.

- YES: Automatically stops WorkUnit.
- NO: Does not automatically stop WorkUnit.

This statement is only valid when the WorkUnit type is UTY.

If "YES" is specified in this statement, an application is not restarted even if it terminates abnormally, and the WorkUnit completes normally when all of the processes are completed. Therefore, specify this statement when a batch program is to be started in the WorkUnit.

Range of Support

OS	Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.3.19 Request Assignment Mode

Explanation

Specify mode in which a request message from client is assigned to server application process waiting for requests.

- LIFO: A request message from the client is assigned to the process that last entered the wait state among all server application processes waiting for requests.
- FIFO: A request message from the client is assigned to the process that first entered the wait state among all server application processes waiting for requests.

When this statement is specified, it applies to all of the applications in a WorkUnit.

This statement can be omitted. If it is omitted, "LIFO" is set as the default value.

This statement is valid when the WorkUnit type is CORBA, ORB, or EJB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, ORB, EJB

A.3.3.20 Traffic Director Monitor Mode

Explanation

Specifies whether or not the WorkUnit is to be the subject of IPCOM monitoring.

- YES: Is to be monitored by IPCOM.
- NO: Is not to be monitored by IPCOM.

This statement is valid when the WorkUnit type is CORBA, ORB, EJB, or UTY.

When using the multi-system function, this statement is only effective in the default system. It is not effective if specified by the extended system.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, ORB, EJB, UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.3.21 Output Of Stack Trace

Explanation

- YES: Outputs stack trace.
- NO: Does not output stack trace.

By specifying "YES" in this statement, you can locate the hang-up position in the stack trace file when an application hangs up and a timeout occurs. Note that the stack trace file is output in the current directory.

This statement can be omitted. If this statement is omitted, "YES" is set as the default value.

This statement is valid when the WorkUnit type is CORBA or ORB.

Note

When YES is specified in this statement, be sure to apply the following OS patches:

If the OS patch is not applied, the stack trace output processing may hang up due to an operating system problem and it may not be possible for WorkUnit collection processing to continue from that point on.

- Solaris7: 106541-22

Range of Support

OS	Solaris
Edition	Enterprise Edition
WorkUnit type	CORBA, ORB

A.3.3.22 Startup Time

Explanation

Specify a monitoring time until the WorkUnit startup is completed. If the application process start processing exceeds the specified time, the process is shutdown to stop the restart processing. In this situation, the WorkUnit start processing terminates abnormally.

A value from 0 to 65,535 can be specified in this statement.

This statement can be omitted. If this statement is omitted, 300 is set as the default value.

Note that when 0 is specified, start time monitoring is not performed. The WorkUnit start processing may not be canceled.

This statement is only valid when the WorkUnit type is ORB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	CORBA

A.3.3.23 Shutdown Time

Explanation

If the stopping of the WorkUnit was executed, specify the maximum wait time (in seconds) for the process to completely stop. If the application process stop processing exceeds the specified time (there was a hangup), the process is shutdown so that the stop processing completes normally.

A value from 0 to 65,536 can be specified in this statement.

This statement can be omitted. If this statement is omitted, 180 is set as the default value.

Note that when 0 is specified, shutdown time monitoring is not performed. The WorkUnit stop processing may not be canceled.

This statement is only valid when the WorkUnit type is CORBA.

[Estimate Method]

Shutdown Time is the baseline value (time) used for determining that Application Process Stop Processing Time in the stopping of the WorkUnit is abnormal (there was a hangup). When tuning Shutdown Time, try to ensure that Application Process Stop Processing Time follows the formula below:

$$\text{Shutdown Time (secs)} > \text{Application Process Stop Processing Time (secs)}$$

- Application Process Stop Processing Time

In most cases, the default value can be used. The exception is when mode was set to SYNC_END (the activation method does not return even when the server application is active) after server application activation, and post-processing has been described after the activation method.

For details on the server application movement modes after activation, refer to the Reference Manual (Command Edition), section "OLTP System Operation Edition", "CORBA Service Operation Commands", "OD_impl_inst" .

Customer transactions are expected to be complete when the WorkUnit is stopped. For this reason, the default value is set to 180 seconds, to allow confirmation of hangups.)

If WorkUnit is stopped during a transaction, then try to ensure that Shutdown Time exceeds the combined total for the Application Processing Time and Application Process Stop Processing Time:

$$\text{Shutdown Time (secs)} > \text{Application Processing Time (secs)} + \text{Application Process Stop Processing Time (secs)}$$

- Application Processing Time

If the WorkUnit is stopped while the application is being processed, then set the longest time required for the application to complete normally.

If synchronous stop of the WorkUnit is executed while the application is being processed, then it waits for completion of requests being processed before stopping the processes. If this element is not taken into account, then the shutdown will occur while requests are still being processed in the synchronous stopping of the WorkUnit.

- Application Process Stop Processing Time

Refer to the formula above.

Example:

If Application Processing Time is 120 and Application Process Stop Processing Time is 5, set a minimum of 126 for Shutdown Time.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	CORBA

A.3.3.24 Unconditional Reactivation of Process

Unconditional restart of the process

Explanation

If a WorkUnit process (with the WorkUnit type "ORB") of the Component Transaction Service is terminated in a state other than that in which an application program is being controlled, this statement is specified if the process needs to be restarted.

- YES: The process is restarted.
- NO: The process is restarted only when an application program is being executed.

Note that even when "YES" is specified, if the number of application terminations has reached the "Application Retry Count" (the number of contiguous abnormal terminations), the restart is canceled and the WorkUnit is stopped.

This statement can be omitted. If it is omitted, "NO" is set as the default value.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.3.25 Start Log

Collection of the process start log

Explanation

The log of the CORBA WorkUnit at the beginning of the process is collected.

- YES: The start log is collected.
- NO: The start log is not collected.

If "YES" is specified, the log will be created in the current directory of the WorkUnit, and the process start parameters and environment variables will be output.

This statement can be omitted. If it is omitted, "NO" is set as the default value.

Specifying this statement enables the collection of diagnostic information which can be used to determine whether there is an error in the process start parameters and environment variables when a CORBA WorkUnit fails to start during the development phase.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	CORBA

A.3.3.26 Process Degeneracy

Degenerated process operation

Explanation

During WorkUnit operation, a process is restarted with the automatic restart function if abnormal termination of a process occurs. If the process restart fails, the operation can be continued in a state where one process is removed (degenerated operation).

- YES: If the restart fails, the number of processes is degenerated and the operation continues.
- NO: If the restart fails, the WorkUnit is stopped.

This function is valid when the process concurrency is two or more. If processes fail to restart repeatedly and the number of processes becomes 0, the WorkUnit is stopped.

This statement can be omitted. If it is omitted, "NO" is set as the default value.

It is recommended that "YES" is specified in this statement so that WorkUnit operation can continue for any remaining processes even if the restart fails after abnormal process termination.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	CORBA

A.3.3.27 Number of Revision Directories

Number of backups of the current directory

Explanation

The number of backups of the current directory of the WorkUnit can be specified. By specifying this statement, the current directories of the WorkUnit for past operations can be backed up for as many generations as specified.

A value from 0 to 5 can be specified for this statement.

If 0 is specified, the current directories of the WorkUnit for past operations are not backed up. The current directory at the previous start is deleted at the start of the WorkUnit.

This statement can be omitted. If it is omitted, 0 is set as the default value.

It is recommended that "1" is specified in this statement so that data output in the current directory remains, even if the WorkUnit is terminated abnormally. It is then possible to later collect the data in the current directory after starting the WorkUnit to recover the operation.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, ORB, EJB, UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.4 Application Program Section

Specify application information.

Synopsis

Application Program

Destination:

Destination name (object name)

Destination Priority:

Object priority

PSYS:

DPCF communication path name

Executable File:

Execution file name

Application Language:

Application language

Concurrency:

Process multi level

Maximum Processing Time:

Application maximum processing time

Terminate Process for Time out:

Forced termination when maximum processing time has elapsed Y/N

Maximum Processing Time for Exit Program:

Maximum processing time for exit program

Maximum Queuing Message:

Maximum number of messages in queues

Queuing Message to Notify Alarm:

Number of queues monitored

Queuing Message to Notify Resumption:

Number of queues to resume monitoring

Environment Variable:

Environment variable

Form:

Multi-object resident, resident or nonresident type

Pre Exit Program:

Pre exit program

Post Exit Program:

Post exit program

Recovery Exit Program:

Abnormal exit program

Executable File for Exit Program:

Exit program execution file

Access Control:

The existence of the access control execution

Access Control Base DN:

The cardinal point distinct name of access control applicable entry

Type of User Identification:

User identification information type

User Name Param:

User name notification parameter

User Base DN

User name retrieval base ID

User DN Param:

User distinct name notification parameter

Password Param:

Password notice parameter

Bind Type:

Bind type

Using Wrapper Session Management:

AIM linkage session inheritance function used Y/N

SessionID Param:

Session ID notice parameter

Method Name to Begin Session:

The method (the operation) that starts the session

Maximum Session Active Time for Client:

The maximum time of the client thinking time

Maximum Processing Time for WRAPPER:

AIM application watch time

Maximum Memory for EJB Application:

Maximum amount of memory for EJB applications

CLASSPATH for Application:

Class path used by application

Java Command Option:

Java command specifying option

Exit Program for Process Salvage:

Exit program name for process salvage

Executable File of Exit Program for Salvage:

Executable file of exit program for salvage

Exit Program for Terminating Process

Exit program for terminating process

Param for Executable File

Param for executable file

Reset Time for Application Retry Count

Reset time for application retry count

Request Assignment Mode

Request assignment mode

Impl ID

Implementation Repository ID

Buffer Number

The number of communication buffers

Buffer Size

The size of the communication buffers

Path

application path

Note

If the process binding function is used in a WorkUnit of a transaction application (a WorkUnit with the type "ORB"), note the following:

- If an application development language is C or COBOL, "INSTANCE" cannot be set in the Bind Type statement. An attempt to register a WorkUnit definition with such a setting ends in failure.
- If an application development language is C++, "PROCESS" cannot be set in the Bind Type statement. An attempt to register a WorkUnit definition with such a setting ends in failure.
- An attempt to register a WorkUnit definition ends in failure when "PROCESS" or "INSTANCE" is set in the Bind Type statement and a Method Name to Begin Session statement is not set.
- An attempt to register a WorkUnit definition ends in failure when "PROCESS" or "INSTANCE" is set in the Bind Type statement and a Session ID Param is not set.

Syntax Rules

- A maximum of 256 [Application Program] sections can be specified.
- If NONRESIDENT or MULTIRESIDENT is specified at the "Form:" statement of the [Application Program] section, the following statements are ignored:
 - "Environment Variable:" statement
 - "Concurrency:" statement
 - "Pre Exit Program: " statement
 - "Post Exit Program: " statement
 - "Executable File for Exit Program: " statement
 - "Maximum Processing Time For Exit Program : " statement
- If the WorkUnit type is ORB, the following statements are mandatory:
 - "Destination:" statement
 - "Executable File:" statement
- If the WorkUnit type is WRAPPER, the following statements are mandatory:
 - "Destination:" statement
 - "PSYS:" statement
- If the WorkUnit type is EJB, the "Destination:" statement is mandatory:
- The "Code Convert For String:" statement can be omitted. Other statements are ignored.
- A newline code 2 bytes is automatically added to the string length in "Environment Variable:" statement settings.
- Multiple "Environment Variable:" statements can be specified.

When specifying multiple "Environment Variable:" statements, the total number of strings specified at the "Environment Variable:" statement must be less than or equal to 4096.

- If "CPP" is specified at the "Application Language:" statement of the [Application Program] Section, NONRESIDENT or MULTIRESIDENT cannot be specified at the "Form:" statement.
- If NONRESIDENT or MULTIRESIDENT is specified at the "Form:" statement of the [Application Program] section, the same language must be set at all "Application Language" statements.
- If NONRESIDENT is specified in the "Form:" statement of the [Application Program] section, PROCESS cannot be set in the "Bind Type:" statement.
- If the [Application Program] section is coded more than once in one WorkUnit definition, NONRESIDENT and MULTIRESIDENT cannot both be specified in the "Form:" statement.

Note

- The destination name specified at other [Application Program] sections in the same WorkUnit definition cannot be specified at the "Destination:" statement.
- Other executable file names specified in the [Application Program] section of the same WorkUnit definition, and the exit program executable file name specified in the [Nonresident Application Process] section and the "Executable File for Exit Program" statement in the [Multiresident Application Process] cannot be specified in the "Executable File" statement.
- The "Code Conversion For String:" statement can be omitted. If it is omitted, the following is assumed, depending on the WorkUnit type:
 - If the WorkUnit type is ORB, DISABLE is assumed. The code conversion type between type applications is required at connection.
 - If the environment variable specified at the "Environment Variable:" statement of the [Application Program] section is the same as that specified at the "Environment Variable:" statement of the [Control Option] section, the environment variable specified at the "Environment Variable:" statement of the [Application Program] section is used.

A.3.4.1 Destination

Destination name (object name)

Explanation

For EJB WorkUnits, specify STATEFUL or STATELESS Session Bean (high speed calling Bean)

If the WorkUnit type is not EJB, specify a character string of up to 255 bytes, including one or more slashes, consisting of alphanumerics, underscores, and slashes, and starting with an alphabetic character. However, two or more consecutive slash characters cannot be used in place of the single slash at the beginning and end. In the object IDL definition, specify the module name and interface name, separated by a slash.

EJB WorkUnit destination names must be character strings of up to 255 bytes, consisting of alphanumeric characters, hyphens, slashes, colons, periods or underscores. However, two or more consecutive slash characters cannot be used in place of the single slash at the beginning and end.

This statement is valid when the WorkUnit type is CORBA, ORB, or EJB.

Notes

- When manually registering an ORB WorkUnit object, set the destination name specified at destination, to the -n option of the OD_or_adm or odadministerlb commands when using the load balance function by replacing the slash / with two colons "::". Refer to the example below.
- When manually registering an object, register it after starting Interstage.

Example

WorkUnit Definition :

```
Destination: MOD1/INTF1
```

OD_or_adm command:

```
OD_or_adm -c IDL:MOD1/INTF1:1.0 -a FUJITSU-Interstage-TDLC -n MOD1::INTF1
```

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	ORB, EJB, UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.4.2 Destination Priority

Object priority

Explanation

Specifies the priority for an object.

Specify an integer between 1 and 255. Higher values indicate higher priorities.

The default value is 10.

This statement is only valid if NONRESIDENT or MULTIRESIDENT is specified in the "Form:" statement.

Note:

To use priority control for CORBA applications, it is necessary to register priority in the Implementation Repository. For details, refer to "Priority Control" In "Designing the OLTP Server".

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.4.3 PSYS

DPCF communication path

Explanation

If the WorkUnit type is WRAPPER, set the information to indicate the position of the AIM application of the global server to be linked.

This is an alphanumeric string of up to 8 bytes.

This statement is not necessary if the WorkUnit type is not "WRAPPER."

Range of Support

OS	Windows, Solaris
Edition	Enterprise Edition
WorkUnit type	WRAPPER

Note

"WRAPPER" is supported on Windows and Solaris, but unsupported on Linux.

A.3.4.4 Executable File

Executable file name

Explanation

When the WorkUnit type is CORBA and the application is in Java language, specify a Java execution file name. If the application is coded in Java, the executable file specifies the Java executable name ("java.exe" for Windows and "java" for Solaris/Linux).

If the WorkUnit type is "ORB", specify the module name for the application and exit programs.

To create an exit program in a different executable file to the application, specify the filename for the exit program of this section.

If the application language is C++, specify the application and program executable names.

The following three file names cannot be specified in this statement:

- The execution file name specified in another Application Program section in the same WorkUnit definition
- The exit program executable file name specified in the Executable File for Exit Program statement in the Nonresident Application Process and Multiresident Application Process sections
- The collection exit program executable file name specified in the Executable File of Exit Program for Salvage statement in the Nonresident Application Process and Multiresident Application Process sections.

The exit program executive file name of our section is set up when application is made with the C language, COBOL and an exit program is made as another executive file with the application.

A string of up to 31 bytes.

This name cannot be omitted.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	CORBA, ORB, UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.4.5 Application Language

Explanation

Specify the application language.

- C: C
- CPP: C++
- COBOL: COBOL

If CPP is specified in this statement, NONRESIDENT and MULTIRESIDENT cannot be specified in the "Form:" statement.

This statement is optional. The default value is "C".

Be sure to specify this statement when the WorkUnit type is ORB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.4.6 Concurrency

Process multiple level

Explanation

Set process multiple level of application. Integer value 1 to 255.

This statement is optional. The default value is 1.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition

WorkUnit type	CORBA, ORB, EJB, UTY
---------------	----------------------

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.4.7 Maximum Processing Time

Explanation

An integer value from 0 to 86400 specifying the maximum processing time monitoring value seconds for the application.

This statement is optional. A time watch isn't done when this statement is optional or when 0 is specified.

This statement is valid when the WorkUnit type is CORBA, ORB, or EJB.

Note

If the application processing time exceeds the time specified in this statement, the process where the exit program is operating is forcibly stopped. The process is not forcibly stopped, however, if the WorkUnit type is CORBA or EJB, and NO is specified in the Terminate Process for Time out statement.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, ORB, EJB

A.3.4.8 Terminate Process for Time out

Explanation

For EJB WorkUnits, specifies whether the process containing the application is forcibly terminated when the maximum application processing time has passed.

- YES: Process is forcibly terminated
- NO: Process is not terminated

The default value is YES.

This statement is only valid if a value other than 0 was specified in the Maximum Processing Time statement.

This statement is valid when the WorkUnit type is CORBA or EJB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, EJB

A.3.4.9 Maximum Processing Time For Exit Program

Explanation

An integer value from 1 to 1800 specifying the maximum processing time monitoring value seconds for the exit program.

This statement is optional. If this statement is omitted, the value specified in the same statement of the [Control Option] section is used.

If this statement is omitted and the same statement of the [Control Option] section is omitted, 300 is set as a default value.

This statement is valid when the WorkUnit type is CORBA, ORB, EJB or UTY.

Note

The values in this statement are valid for each of the following exit programs. If the processing time of each exit program exceeds the specified time, the process where the exit program is operating is forcibly stopped.

- Pre exit program

If the time is exceeded, the program causes WorkUnit startup to fail.

- Post exit program

WorkUnit stop processing continues even if the time is exceeded.

- Error exit program

If the time is exceeded, the program stops the application process. If automatic restart is set, the program restarts the application process. If automatic restart is not set, the WorkUnit stops abnormally.

- Process release exit program

WorkUnit stop processing continues even if the time is exceeded.

- Process termination exit program

WorkUnit stop processing continues even if the time is exceeded.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, ORB, EJB, UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.4.10 Maximum Queuing Message

Explanation

Specifies the maximum number of messages to be queued at the message destination. When this value is exceeded, an alarm is posted.

Specify an integer from 0 to 2147483647.

The set value must also be smaller than BufferNumber.

This statement can be omitted. If omitted, or if 0 is specified, the number of messages is unlimited. In such a case, the alarm notification when the number has exceeded the maximum queuing number, is not performed. Also, even when this statement is specified, the alarm notification when the number has exceeded the maximum queuing number is not performed if the monitoring queuing number is omitted.

This statement is only valid for ORB type WorkUnits.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	CORBA, ORB, EJB

A.3.4.11 Queuing Message to Notify Alarm

Explanation

Specifies the number of queued messages at which an alarm is posted. An alarm is posted when the number of queued messages reaches this value.

Specify an integer from 1 to 2147483647.

For ORB type WorkUnits, if a value other than 0 is specified in Maximum Queuing Message, a higher value cannot be specified in this statement.

This statement can be omitted. If it is omitted, no alarm is posted.

Note

If this statement is specified in Standard-J Edition and the WorkUnit type is ORB or WRAPPER, an error occurs when the definition is registered, while the statement is ignored if the WorkUnit type is EJB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	CORBA, ORB, EJB, WRAPPER

Note

"WRAPPER" is supported on Windows and Solaris, but unsupported on Linux.

A.3.4.12 Queuing Message to Notify Resumption

Explanation

Specifies the number of queued messages at which monitoring to post an alarm is resumed. Once the number of messages has passed the level in Queuing Message to Notify Alarm, and then reaches the value specified in this statement, monitoring of the number of messages queued is resumed.

Specify an integer from 0 to 2147483647. Note that the value may not be higher than that in Queuing Message to Notify Alarm.

This statement can be omitted. The default is 70% (ignoring decimals) of the value specified in Queuing Message to Notify Alarm.

Note

If this statement is specified in Standard-J Edition and the WorkUnit type is ORB or WRAPPER, an error occurs when the definition is registered, while the statement is ignored if the WorkUnit type is EJB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	CORBA, ORB, EJB, WRAPPER

Note

"WRAPPER" is supported on Windows and Solaris, but unsupported on Linux.

A.3.4.13 Environment Variable

Explanation

Specifies the environment variable to be used when the application and exit programs run. Specify a character string of up to 4096 bytes.

Specify the value in the following format:

"environment variable = value"

Character type and length is not checked.

This statement can be specified more than once. In this case, the character strings specified in the Environment Variable statement must total 4,096 or fewer bytes.

This statement is optional. When this statement is omitted, the environment variable specified by the "Environment Variable :" statement of the [Control Option] section at the time of the practice is used.

If the environment variable specified in the Environment Variable statement in the Application Program section and that specified in the Environment Variable statement in the Control Option section are the same, the environment variable specified in the Application Program section becomes valid.

Note that the following word cannot be used because it is the reserved word of this statement.

- PATH

- LD_LIBRARY_PATH

This statement is valid when the WorkUnit type is CORBA, ORB, EJB, or UTY.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, ORB, EJB, UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.4.14 Form

Explanation

Specify resident, non-resident, or multiresident type.

- resident: RESIDENT
- non-resident: NONRESIDENT
- multiresident: MULTIRESIDENT

The default value is RESIDENT.

This statement is only valid when the WorkUnit type is ORB.

When "NONRESIDENT" is specified, be sure to set up the [Nonresident Application Process] section.

When "MULTIRESIDENT" is specified, be sure to set up the [Multiresident Application Process] section.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.4.15 Pre Exit Program

Explanation

Set the name of the exit program to be started when starting the WorkUnit. This cannot be set if the application language is C++.

The exit program name referred to as "function name" in C, and "ProgramID" in COBOL can consist of the following:

- Function name C: Alphanumeric and underscore characters up to a maximum 31 bytes.
- ProgramID COBOL: Alphanumeric and hyphen characters up to a maximum 30 characters, and must include one or more alpha characters. The first and last characters must not be a hyphen -.

This statement is optional.

This statement is only valid when the WorkUnit type is ORB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.4.16 Post Exit Program

Explanation

Set the name of the post exit program to be started when terminating the WorkUnit. This cannot be set if the application language is C++.

The post exit program name referred to as "function name" in C, and "ProgramID" in COBOL can consist of the following:

- Function name C: Alphanumeric and underscore characters up to a maximum 31 bytes.
- ProgramID COBOL: Alphanumeric and hyphen characters up to a maximum 30 characters, and must include one or more alpha characters.

The first and last characters must not be a hyphen -.

This statement is optional.

This statement is only valid when the WorkUnit type is ORB.

In addition, hyphens and underscores cannot be used at the same time.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.4.17 Recovery Exit Program

Explanation

Specify the name of the recovery exit program started when client thinking time has passed, when using the Process Binding Function. This statement cannot be specified when C++ is used.

The name of the abnormal exit program is set up.

The recovery exit program name referred to as "function name" in C, and "ProgramID" in COBOL can consist of the following:

- Function name C: Alphanumeric and underscore characters up to a maximum 31 bytes.
- ProgramID COBOL: Alphanumeric and hyphen characters up to a maximum 30 characters, and must include one or more alpha characters.

The first and last characters must not be a hyphen -.

This statement is optional.

This statement is only valid when the WorkUnit type is ORB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.4.18 Executable File for Exit Program

Explanation

When the pre exit, post exit, and error exit programs are stored in the execution file of a non-application program, specify the execution file name of an exit program.

Exit program execution file

When creating an exit program as an executable file, independent of an application, set the filename of the exit program.

This statement is only valid when the WorkUnit type is ORB.

A string of up to 31 bytes.

This statement does not distinguish between upper case and lower case.

If this statement is omitted, the value specified in the "Executable File:" statement is used.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.4.19 Access Control

Explanation

Specify the existence of the execution of the access control for the object.

- YES: An object is made the target of the access control.
- NO: An object isn't made the target of the access control.

This statement is optional. The default value is "NO".

And, regardless of the designation of the Access Control statement of the [Control Option] section, it is effective. When both designations are given, access control for the WorkUnit is carried out first, and next access control for the object is carried out.

This statement is only valid when the WorkUnit type is ORB.

Note

The EJB WorkUnit is not included in the security functions of the Component Transaction Service.

Range of Support

OS	Windows, Solaris
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.4.20 Access Control BaseDN

Explanation

Specify the distinct name (DN) of entry to find the object registered in InfoDirectory when the object needs to be accessed.

This statement is mandatory when the "Access Control" statement of the [Application Program] section is set to YES.

The length of the DN should not exceed 1023 bytes.

This statement is only valid when the WorkUnit type is ORB or WRAPPER.

Note: Specify DN of the top most directory in the folder class so that the access control can look up the object with InfoDirectory. If the WorkUnit fails to start, due to the object not being registered, specify the next highest directory in the hierarchy.

Range of Support

OS	Windows, Solaris
Edition	Enterprise Edition
WorkUnit type	ORB, WAPPER

A.3.4.21 Type of User Identification

Type of user identification

Explanation

Specify the type of user identification.

- DN: Identification name (DN)

- NAME: User name

The default value is "DN".

This statement is only valid when the WorkUnit type is ORB or WRAPPER.

Range of Support

OS	Windows, Solaris
Edition	Enterprise Edition
WorkUnit type	ORB, WAPPER

A.3.4.22 User Name Param

User name communication parameter

Explanation

Specify the parameter that sets the user name in calling operations when access control is applied to WorkUnits or objects.

This statement is mandatory when access control is used, and when NAME is specified in the Type of User Identification statement.

WorkUnits can only be started if string parameters are specified in this statement. Character strings must not exceed 128 bytes and must follow IDL identifier syntax.

This statement is only valid when the WorkUnit type is ORB or WRAPPER.

Note

For operations with objects subject to access control, access control is only applied to operations with in or inout parameters specified in this statement.

Range of Support

OS	Windows, Solaris
Edition	Enterprise Edition
WorkUnit type	ORB, WAPPER

A.3.4.23 User Base DN

User retrieval base point ID

Explanation

States the DN of the tree in which user information is stored on the directory server.

If user information is specified in multiple entries, specify the DN of the highest level of the common tree for each of the entries. The user base DN specified in this statement is treated as the top level for retrieving user information.

As the character string specified in User Base DN is passed directly to the directory server, specify the character string that will be used by the directory server.

This statement is mandatory if access control is used, and if NAME is specified in the Type of User Identification statement.

Specify the DN in up to 1023 bytes.

This statement is only valid when the WorkUnit type is ORB or WRAPPER.

Range of Support

OS	Windows, Solaris
Edition	Enterprise Edition
WorkUnit type	ORB, WAPPER

A.3.4.24 User DN Param

Explanation

Specifies the parameter that defines a user's unique name when you carry out access control for the WorkUnit or the object. This statement is mandatory when access control is carried out and when DN is specified in the Type of User Identification statement.

The start of the WorkUnit fails when the pattern of the parameter specified by this statement is except for string. The string that can be specified follows the regulation of the identifier of the IDL grammar within 128 bytes.

Note: The call of the operation which doesn't have the parameter of the type in specified by this statement by the operation of the object that access control is carried out, or the type inout becomes the outside of the object of the access control.

This statement is only valid when the WorkUnit type is ORB or WRAPPER.

Range of Support

OS	Windows, Solaris
Edition	Enterprise Edition
WorkUnit type	ORB, WAPPER

A.3.4.25 Password Param

Explanation

Specifies the parameter that defines a user's password when you carry out access control for the WorkUnit or the object.

This statement is indispensable when access control is carried out.

Note: The call of the operation which doesn't have the parameter of the type in specified by this statement by the operation of the object that access control is carried out, or the type inout becomes the outside of the object of the access control.

A character except for the ASCII code can't be used for the password by the access control.

This statement is only valid when the WorkUnit type is ORB.

Range of Support

OS	Windows, Solaris
Edition	Enterprise Edition
WorkUnit type	ORB, WAPPER

A.3.4.26 Bind Type

Explanation

Process Binding Function bind type is specified.

- DISABLE: Process Binding Function is not used.
- PROCESS: Process
- INSTANCE: Instance

This statement is optional.

The default value is "DISABLE".

When the application development language is C or COBOL, INSTANCE cannot be set. When the application development language is C++, PROCESS cannot be set.

When PROCESS or INSTANCE is set in this statement, be sure to set the Method Name to Begin Session statement and SessionID Param statement.

When the application development language is C or COBOL, INSTANCE cannot be set in the Bind Type statement. Doing so causes WorkUnit definition registration to fail.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.4.27 Using Wrapper Session Management

Explanation

AIM linkage session inheritance function used Y/N

Specifies whether the AIM linkage session inheritance function is used.

- YES: Used
- NO: Not used

This statement is only valid for WRAPPER type WorkUnits.

This statement is optional.

The default value is "NO".

This statement is only valid when the WorkUnit type is ORB.

Range of Support

OS	Windows, Solaris
Edition	Enterprise Edition
WorkUnit type	WRAPPER

A.3.4.28 SessionID Param

Explanation

The parameter name to set up Session ID defined with IDL definition is set up.

This statement is optional.

Be sure to specify this statement when you set up "PROCESS" or "INSTANCE" in the Bind Type statement.

When AIM linkage is used for session inheritance and the session ID is used in the session information management object, this statement is mandatory.

The string that can be specified follows the regulation of the identifier of the IDL grammar within 128 bytes.

This statement is only valid when the WorkUnit type is ORB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB, WRAPPER

Note

"WRAPPER" is supported on Windows and Solaris, but unsupported on Linux.

A.3.4.29 Method Name to Begin Session

Explanation

The method (the operation) that starts the session is set up.

Be sure to specify this statement when you set up "PROCESS" or "INSTANCE" in the Bind Type statement.

For WRAPPER type WorkUnits, specify the operation name that starts sessions specified in the wrapper definition. This statement is mandatory if AIM linkage is used for session inheritance.

The string that can be specified follows the regulation of the identifier of the IDL grammar within 128 bytes.

This statement is only valid when the WorkUnit type is ORB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB, WRAPPER

Note

"WRAPPER" is supported on Windows and Solaris, but unsupported on Linux.

A.3.4.30 Maximum Session Active Time for Client

Explanation

The maximum time of the client thinking time is set up in second unit.

This statement is valid only when PROCESS or INSTANCE is set in the Bind Type statement.

An integer value from 0 to 86400.

This statement is optional. The default value is 300.

When 0 is specified, a time watch isn't done.

This statement is valid only when the WorkUnit type is ORB.

Note

- A session is interrupted when a request from the client application during the session exceeds the specified time but does not arrive at the server application. If an error exit program has been registered, the program is executed.
- In the error exit program, collect the information of the session being continued.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB, WRAPPER

Note

"WRAPPER" is supported on Windows and Solaris, but unsupported on Linux.

A.3.4.31 Maximum Processing Time for WRAPPER

Explanation

Specify the monitoring value (seconds) for response time from the AIM application.

An integer value from 0 to 1800. When 0 is specified, a time watch isn't done.

This statement is optional.

The default value is 30.

This statement is only valid when the WorkUnit type is WRAPPER.

Range of Support

OS	Windows, Solaris
Edition	Enterprise Edition
WorkUnit type	WRAPPER

A.3.4.32 Maximum Memory for EJB Application

Maximum memory size for EJB applications

Explanation

Specify the maximum memory for the Java VM used by EJB applications.

Specify an integer between 16 and 2047 (megabytes).

This statement is optional.

The default value is the JDK/JRE default value.

This statement is only valid when the WorkUnit type is EJB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	EJB

A.3.4.33 CLASSPATH for Application

Class path for application

Explanation

Specify the class path used when EJB applications are started.

Specify a character string of up to 255 bytes.

This statement is optional.

The default is no class path.

This statement can be specified up to 30 times. To specify multiple statements, repeat the statement. Note that the same path cannot be specified more than once.

A value that can be set in the CLASSPATH environment variable can be specified in this statement.

This statement is valid when the WorkUnit type is CORBA or EJB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, EJB

A.3.4.34 Java Command Option

Java-command specifying option

Explanation

Specifies the option to be set to the Java command used for starting an EJB application.

A character string of up to 4096 bytes.

This statement can be omitted. If omitted, it is assumed that there are no options

This statement is valid when the WorkUnit type is EJB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	EJB

A.3.4.35 Exit Program for Process Salvage

Exit program for process salvage

Explanation

Specifies the name of the Exit program for process salvage. Only a program name of C language can be specified.

Alphanumeric characters and underscores of up to 31 bytes.

This statement can be omitted.

This statement is valid when the WorkUnit type is CORBA, ORB, EJB, or UTY.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, ORB, EJB, UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.4.36 Executable File of Exit Program for Salvage

Executable file of exit program for salvage

Explanation

Specifies the executable file name of the exit program for process salvage.

A character string of up to 31 bytes. This statement does not distinguish between upper case and lower case.

This statement can be omitted.

This statement is valid when the WorkUnit type is CORBA, ORB, EJB, or UTY.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, ORB, EJB, UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.4.37 Exit Program for Terminating Process

Explanation

Specifies the process exit program name for the application process which operates as a WorkUnit.

Set a character string of 31 or fewer bytes consisting of alphanumeric characters and underscores in this statement.

This statement is valid only for the "UTY" WorkUnit type. For the "UTY" type, also be sure to specify a program whose process is to be stopped with this statement. If this specification is omitted, processes of a utility WorkUnit cannot be stopped.

This statement can be omitted.

This statement is valid when the WorkUnit type is UTY.

Range of Support

OS	Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.4.38 Param for Executable File

Explanation

Specify the parameter to be passed when a utility WorkUnit application is started, or the start parameter to be set for the CORBA application. For an application in Java language, specify the class name for the Java application.

This statement can be specified in up to 255 statements and is set as a parameter in the specified order.

A character string of 65,025 or fewer bytes can be specified.

If this statement is specified more than once, the total of the following two items must be 65,025 or fewer bytes:

- The total number of bytes in the character string of each start parameter specified in each statement
- The total number of bytes (the number of parameters x 1 byte)

This statement can be omitted.

This statement is valid when the WorkUnit type is CORBA or UTY.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	CORBA, UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.4.39 Reset Time for Application Retry Count

Explanation

The present retry count (number of continuation attempts after abnormal termination). The continuation non-stopped time until the specification is reset in the Application Retry Count statement of the [Control Option] section is set up per second.

An integer value from 0 to 86400 can be set.

This statement can be omitted.

If this statement is omitted, a retry count is not reset. A retry count is also not reset if 0 is specified.

This statement is valid when the WorkUnit type is UTY.

Range of Support

OS	Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	UTY

Note

"UTY" is supported on Solaris and Linux, but unsupported on Windows.

A.3.4.40 Request Assignment Mode

Explanation

Specify mode in which a request message from client is assigned to server application process waiting for requests.

- LIFO: A request message from the client is assigned to the process that last entered the wait state among all server application processes waiting for requests.

- FIFO: A request message from the client is assigned to the process that first entered the wait state among all server application processes waiting for requests.

If this statement is specified, It is valid for the application of this section.

This statement can be omitted. If it is omitted, LIFO is used by default..

This statement is valid when the WorkUnit type is CORBA, ORB, or EJB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, ORB, EJB

A.3.4.41 Impl ID

Explanation

Specifies the Implementation Repository ID to be started.

A string of up to 255 bytes can be specified in this statement.

An Implementation Repository ID specified in other [Application Program] sections of the same WorkUnit definition cannot be specified in this statement.

This statement should be specified when the WorkUnit type is CORBA.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	CORBA

A.3.4.42 Buffer Number

Explanation

An integer value from 1 to 1048576 can be set.

When this statement is specified, the "Buffer Size:" statement is mandatory.

This statement can be omitted. If this statement is omitted, the default buffer is used.

Refer to "Buffer Control" for information about the communication buffer.

This statement is valid when the WorkUnit type is CORBA or EJB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, EJB

A.3.4.43 Buffer Size

Explanation

An integer value from 4096 to 2147483647 can be set.

When this statement is specified, the "Buffer Number:" statement is mandatory.

This statement can be omitted. If this statement is omitted, the default buffer is used.

This statement is valid when the WorkUnit type is CORBA or EJB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, EJB

A.3.4.44 Path

Explanation

Specify the path to the directory containing the executable files of the application programs.

If the WorkUnit type is "CORBA" and the application language is Java, specify the path to the storage directory of Java executables.

A string of up to 255 bytes without spaces, beginning with "/".

Be sure to specify the absolute path in this statement. The relative path and current directory "." cannot be specified.

Be sure to specify this statement when the WorkUnit type is CORBA, ORB, EJB, or UTY.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition, Standard-J Edition
WorkUnit type	CORBA, EJB

A.3.5 Nonresident Application Process Section

Specify the definition corresponding to non-resident type.

This section is only valid when the WorkUnit type is ORB.

This section is mandatory when "NONRESIDENT" is specified in the "From" statement.

Synopsis

Nonresident Application Process

Concurrency

Process multi level

Pre Exit Program

Pre exit program name

Post Exit Program

Post exit program name

Executable File for Exit Program

Executable file name for exit program

Maximum Processing Time for Exit Program:

Maximum processing time for Exit Program

Dynamic Link Library

Dynamic link library name

Exit Program for Process Salvage

Exit program name for process salvage

Executable File of Exit Program for Salvage

Executable file of exit program for salvage

Syntax Rules

- Only one [Nonresident Application Process] section can be specified at the WorkUnit.
- If NONRESIDENT is specified in the "Form:" statement of the [Application Program] section, the following statement must be specified in the [Nonresident Application Process] section:
 - "Concurrency:" statement
- If NONRESIDENT is not specified in the "Form:" statement of any [Application Program] section, the [Nonresident Application Process] section is ignored.
- The filename specified in the "Executable File:" statement of the [Application Program] section cannot be specified in the "Executable File for Exit Program:" statement.

A.3.5.1 Concurrency

Explanation

Set the process concurrency of the application.

Status: Process multi level

An integer value 1 to 255 can be set for the process concurrency of the application.

This statement is mandatory.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.5.2 Pre Exit Program

Explanation

Set the name of the exit program to be started when starting the WorkUnit.

The pre exit program name referred to as "function name" in C, and "ProgramID" in COBOL can consist of the following:

- Function name C: Alphanumeric and underscore characters up to a maximum 31 bytes.
- ProgramID COBOL: Alphanumeric and hyphen characters up to a maximum 30 characters, and must include one or more alpha characters. The first and last characters must not be a hyphen -.

This statement is optional.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.5.3 Post Exit Program

Explanation

Set the name of the exit program to be started when terminating the WorkUnit.

The post exit program name referred to as "function name" in C, and "ProgramID" in COBOL can consist of the following:

- Function name C: Alphanumeric and underscore characters up to a maximum 31 bytes.
- ProgramID COBOL: Alphanumeric and hyphen characters up to a maximum 30 characters, and must include one or more alpha characters. The first and last character must not be a hyphen -.

This statement is optional.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.5.4 Executable File for Exit Program

Explanation

When creating an exit program as an executable file independent of an application, set the filename of the exit program.

If the Pre Exit Program or Post Exit Program statements are specified, this statement is mandatory.

A string of up to 31 bytes without spaces.

This statement does not distinguish between upper case and lower case en- and em- sized characters.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.5.5 Maximum Processing Time for Exit Program

Explanation

Set an integer value 1 to 1800 as the maximum processing time for an exit program, in seconds. This statement can be omitted. If omitted, the value specified in the same statement of the [Control Option] section is used.

If this statement was omitted and the same statement of the [Control Option] section was also omitted, 300 is set as a default value.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.5.6 Dynamic Link Library

Explanation

When using a database from a non-resident application, specify the dynamic link library name

If the application is executed in non-resident mode, it may malfunction because the database library connected to the application is released from the process when the application execution ends. If the library is to reside in the process, specify this statement. The specified library is not released from the process even when the application ends.

Up to 10 statements of this type can be specified.

A string of up to 31 bytes without spaces.

This statement does not distinguish between upper case and lower case en- and em- sized characters.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.5.7 Exit Program for Process Salvage

Explanation

Specifies the exit program name for process salvage. Only a program name of C language can be specified.

The name must comprise alphanumeric characters and underscores of 31 or fewer bytes.

This statement can be omitted.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.5.8 Executable File of Exit Program for Salvage

Explanation

Specifies the executable file name of the exit program for salvage.

A character string of up to 31 bytes. This statement does not distinguish between upper case and lower case.

This statement can be omitted.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.6 Multiresident Application Process Section

Specify the definitions for multi-object resident mode.

This section is only valid when the WorkUnit type is ORB.

This section is mandatory when "MULTIRESIDENT" is specified in the "From" statement.

Synopsis

Multiresident Application Process

Concurrency

Process concurrency

Pre Exit Program

Pre-exit program name

Post Exit Program

Post-exit program name

Executable File for Exit Program

Executable file name for exit program

Maximum Processing Time for Exit Program

Maximum processing time for exit program

Recovery Exit Program

Recovery exit program name

Exit Program for Process Salvage

Exit program name for process salvage

Executable File of Exit Program for Salvage

Executable file name of exit program for salvage

Syntax Rules

- Specify only one [Multiresident Application Process] section per WorkUnit.
- If MULTIRESIDENT is specified in the "Form:" statement of the [Application Program] section, the following statement must be specified in the [Multiresident Application Process] section:
 - "Concurrency:" statement
- If MULTIRESIDENT is not specified in a "Form:" statement in an [Application Program] section, the [Multiresident Application Process] section is ignored.
- The file name specified in the "Executable File:" statement in the [Application Program] section cannot be specified in the "Executable File for Exit Program:" statement.

A.3.6.1 Concurrency

Explanation

Specify the application concurrency.

Specify an integer between 1 and 255. This statement is mandatory.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.6.2 Pre Exit Program

Explanation

Specify the name of the exit program started when the WorkUnit is started.

Specify up to 31 bytes of alphanumeric and underscores (C function names), or up to 30 alphanumeric characters and hyphens (COBOL program Ids) including at least one alphabetic character. Do not use hyphens at the beginning or end.

This statement is optional.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.6.3 Post Exit Program

Explanation

Specify the name of the exit program started when the WorkUnit is stopped.

Specify up to 31 bytes of alphanumeric and underscores (C function names), or up to 30 alphanumeric characters and hyphens (COBOL program Ids) including at least one alphabetic character. Do not use hyphens at the beginning or end.

This statement is optional.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.6.4 Executable File for Exit Program

Explanation

Specify the executable file name for the exit program if it was created as an executable file separate from the application.

This statement is mandatory if the Pre Exit Program, Post Exit Program or Recovery Exit Program is specified.

Specify a character string of up to 31 bytes. This statement does not distinguish between upper and lower case en- and em-size characters.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.6.5 Maximum Processing Time for Exit Program

Explanation

Specify the monitoring value (seconds) for the maximum processing time for the exit program.

Specify an integer between 1 and 1800.

This statement can be omitted. When omitted, the value specified in the same name statement of the [Control Option] section is used.

If this statement was omitted and the same statement of the [Control Option] section was also omitted, 300 is set as a default value.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.6.6 Recovery Exit Program

Explanation

Specify the name of the recovery exit program that is started when client thinking time is exceeded, when the Process Binding Function is used. Do not, however, set the name when the application language is C++.

Specify up to 31 bytes of alphanumeric characters and underscores (C function names), or up to 30 alphanumeric characters and hyphens (COBOL program Ids) including at least one alphabetic character. Do not use hyphens at the beginning or end.

This statement is optional.

This statement is only valid when the WorkUnit type is ORB.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.6.7 Exit Program for Process Salvage

Explanation

Specifies the exit program name for process salvage. Only a program name of C language can be specified.

Alphanumeric characters and underscores of up to 31 bytes.

This statement can be omitted.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.6.8 Executable File of Exit Program for Salvage

Explanation

Specifies the executable file name of the exit program for salvage.

A character string of up to 31 bytes. This statement does not distinguish between upper case and lower case.

This statement can be omitted

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.7 Resource Manager Section

Specify resource manager information.

Synopsis

Resource Manager

Name:

Resource definition name

RM:

Database system name

Syntax Rules

- 32 [Resource Manager] sections can be specified.
- When specifying one [Resource Manager] section, the RM statement can be omitted.
- The resource definition files of different [Resource Manager] sections in the same WorkUnit definition cannot be specified in the "File" statement.

Note

- When not using global transactions, this section can be omitted. If it is not specified, omit the section name also.
- If the Database Linkage Service is used, the WorkUnit definition used in an earlier version cannot be used as it is.

A.3.7.1 Name

Explanation

Resource definition name

Specifies the name of the resource definition that defines the resource used by this WorkUnit.

Up to 36 bytes of alphanumeric characters, hyphens, and underscores. This statement does not distinguish between upper case and lower case.

Hyphens and underscores cannot be used at the beginning or end of the Resource definition name.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

A.3.7.2 RM

Explanation

Database system name

Specify the system name of the database to be used by the WorkUnit in a string of up to 31 bytes without spaces.

Range of Support

OS	Windows, Solaris, Linux
Edition	Enterprise Edition
WorkUnit type	ORB

Appendix B Interstage Operation API Sample Programs

This appendix describes how to create Interstage Operation API applications.

The different types of Interstage Operation API sample programs are as follows:

- [B.1.1 WorkUnit Startup Program](#)

The WorkUnit startup program starts all the WorkUnits defined in the `tdaddddef` and `isaddwudef` commands.

- [B.1.2 WorkUnit Stop Program](#)

The WorkUnit stop program stops all the WorkUnits that have been started.

- [B.1.3 WorkUnit/Object Information Acquisition Program](#)

The WorkUnit/object information acquisition program obtains information about all the WorkUnits and objects defined in the `tdaddddef` and `isaddwudef` commands.

- [B.1.4 Object Close Program](#)

The Object Close Program shows how to close a specified object.

- [B.1.5 Cancel Object Closure Program](#)

The Cancel Object Closure Program shows how to cancel the closure of a specified object.

- Information acquisition of the object in the Implementation Repository ID

In information acquisition of the object in the Implementation Repository ID, information on the defined object in all Implementation Repository IDs is acquired.

Solaris

- Acquiring/releasing of a system name list

Acquiring a system name list acquires information about generated systems.

Releasing a system name list releases information about acquired systems.

Note

Interstage Operation API can be used when one of the following features has been custom-installed:

- Multilanguage Service
- J2EE Compatible

B.1 File Configuration

The file configuration of each application is shown in the following subsections. Item numbers listed in the following sections are as follows:

(1) WorkUnit Startup Program

This is an application for starting all the defined WorkUnits.

(2) Makefile for WorkUnit Startup Program

This is a Makefile for making the binary file of the WorkUnit startup program. Correct it according to the environment.

(3) WorkUnit Stop Program

This is an application for stopping all the WorkUnits that have been started.

(4) Makefile for WorkUnit Stop Program

This is a Makefile for making the binary file of the WorkUnit stop program. Correct it according to the environment.

(5) WorkUnit/Object Information Acquisition Program

This is an application for obtaining information about all the defined WorkUnits and objects.

(6) Makefile for WorkUnit/Object Information Acquisition Program

This is a Makefile for making the binary file of the WorkUnit/object information acquisition program. Correct it according to the environment.

(7) Object Close Program

This is an application for closing the specified object.

(8) Makefile for Object Close Program

This is a Makefile for making the binary file of the object close program. Correct it according to the environment.

(9) Cancel Object Closure Program

This is an application for canceling the closure of a specified object.

(10) Makefile for Cancel Object Closure Program

This is a Makefile for making the binary file of the cancel object closure program. Correct it according to the environment.

(11) Information Acquisition of the Object in the Implementation Repository ID Program

This is an application for acquiring the information of the object in the Implementation Repository ID.

(12) Makefile for Information Acquisition of the Object in the Implementation Repository ID Program

This is a Makefile for making the binary file of the information acquisition of the object in the Implementation Repository ID Program. Correct it according to the environment.

(13) Acquiring/releasing of a system name list Program

This is an application for acquiring/releasing of a system name list.

(14) Makefile for Acquiring/releasing of a system name list Program

This is a Makefile for making the binary file of the acquiring/releasing of a system name list program. Correct it according to the environment.

(15) Project File

This is a project file for compiling server applications under Microsoft® Visual C++®.

(16) Solution File

This is a file for managing project configuration under Microsoft® Visual C++®.

B.1.1 WorkUnit Startup Program

The following table provides details of the files associated with this program.

WorkUnit Startup Program

Windows32/64

This program is located in Interstage installation folder\td\sample\ISOP_API\STRWU.

Item no.	File contents	File name
(1)	WorkUnit startup program	strwu.c
(15)	Project file	strwu.vcproj
(16)	Solution file	strwu.sln

Solaris Linux32/64

This program is located in \$TD_HOME/sample/ISOP_API/STRWU.

Item no.	File contents	File name
(1)	WorkUnit startup program	strwu.c

Item no.	File contents	File name
(2)	Makefile for WorkUnit Startup Program	Makefile

B.1.2 WorkUnit Stop Program

The following table provides details of the files associated with this program.

WorkUnit Stop Program

 Windows32/64

This program is located in Interstage installation folder\td\sample\ISOP_API\STPWU.

Item no.	File contents	File name
(3)	WorkUnit stop program	stpwu.c
(15)	Project file	stpwu.vcproj
(16)	Solution file	stpwu.sln

 Solaris  Linux32/64

This program is located in \$TD_HOME/sample/ISOP_API/STPWU.

Item no.	File contents	File name
(3)	WorkUnit stop program	stpwu.c
(4)	Makefile for WorkUnit Stop Program	Makefile

B.1.3 WorkUnit/Object Information Acquisition Program

The following table provides details of the files associated with this program.

WorkUnit/Object Information Acquisition Program

 Windows32/64

This program is located in Interstage installation folder\td\sample\ISOP_API\NOTIFYWU.

Item no.	File contents	File name
(5)	WorkUnit/object information acquisition program	notifywu.c
(15)	Project file	notifywu.vcproj
(16)	Solution file	notifywu.sln

 Solaris  Linux32/64

This program is located in \$TD_HOME/sample/ISOP_API/NOTIFYWU.

Item no.	File contents	File name
(5)	WorkUnit/object information acquisition program	notifywu.c
(6)	Makefile for WorkUnit/Object Information Acquisition Program	Makefile

B.1.4 Object Close Program

The following table provides details of the files associated with this program.

Object Close Program

 Windows32/64

This program is located in Interstage installation folder\td\sample\ISOP_API\INHOBJ.

Item no.	File contents	File name
(7)	Object close program	inhobj.c
(15)	Project file	inhobj.vcproj
(16)	Solution file	inhobj.sln

This program is located in \$TD_HOME/sample/ISOP_API/INHOBJ.

Item no.	File contents	File name
(7)	Object close program	inhobj.c
(8)	Makefile for Object Close Program	Makefile

B.1.5 Cancel Object Closure Program

The following table provides details of the files associated with this program.

Cancel Object Closure Program



This program is located in Interstage installation folder\td\sample\ISOP_API\PMTOBJ.

Item no.	File contents	File name
(9)	Cancel object closure program	pmtobj.c
(15)	Project file	pmtobj.vcproj
(16)	Solution file	pmtobj.sln

This program is located in \$TD_HOME/sample/ISOP_API/PMTOBJ.

Item no.	File contents	File name
(9)	Cancel object closure program	pmtobj.c
(10)	Makefile for Cancel Object Closure Program	Makefile

B.1.6 Information Acquisition of the Object in the Implementation Repository ID Program

The following table provides details of the files associated with this program.

Information Acquisition of the Object in the Implementation Repository ID Program



This program is located in Interstage installation folder\td\sample\ISOP_API\NOTIFYIMPL.

Item no.	File contents	File name
(11)	Information Acquisition of the Object in the Implementation Repository ID Program	notifyimpl.c
(15)	Project file	notifyimpl.vcproj
(16)	Solution file	notifyimpl.sln

This program is located in \$TD_HOME/sample/ISOP_API/NOTIFYIMPL.

Item no.	File contents	File name
(11)	Information Acquisition of the Object in the Implementation Repository ID Program	notifyimpl.c
(12)	Makefile for Information Acquisition of the Object in the Implementation Repository ID Program	Makefile

B.1.7 Acquiring/releasing of a system name list Program

The following table provides details of the files associated with this program.

Acquiring/releasing of a system name list Program



This program is located in \$TD_HOME/sample/ISOP_API/LSTSYS.

Item no.	File contents	File name
(13)	Acquiring/releasing of a system name list Program	lstsys.c
(14)	Makefile for Acquiring/releasing of a system name list Program	Makefile

B.2 Compiling and Linking

This section describes the procedure for compiling and linking applications.



We assume that these applications are compiled using Microsoft® Visual C++® projects.

We recommend that the files provided be copied to a folder and customized according to the environment of the copy destination before using the sample programs.

Compiling and Linking Applications

Use Microsoft® Visual C++® projects to compile the applications. Executing the project build compiles and links the files required for creating server applications.

When the build terminates normally, strwu.exe, stpwu.exe, notifywu.exe, inhobj.exe or pmtobj.exe will be created in the same folder as the project file.

This section describes the procedure for compiling and linking applications.

We recommend that the files provided be copied to a folder and customized according to the environment of the copy destination before using the sample programs.

The following explanation uses the following coding:

\$CURRENT: Indicates the directory containing the files used by each application

```
>cd $CURRENT
```

```
>make clean
```

- a. Execute the *make* command for the WorkUnit startup program.

Execution of the *make* command compiles and links the files required for writing the application. If the *make* command terminates normally, *strwu* is created in the current directory where the *make* command was executed. If the *make* command fails, execute the following command:

>*make* clean

- b. Execute the *make* command for the WorkUnit stop program.

Execution of the *make* command compiles and links the files required for writing the application. If the *make* command terminates normally, *stpwu* is created in the current directory where the *make* command was executed. If the *make* command fails, execute the following command:

>*make* clean

- c. Execute the *make* command for the WorkUnit/object information acquisition program.

Execution of the *make* command compiles and links the files required for writing the application. If the *make* command terminates normally, *notifywu* is created in the current directory where the *make* command was executed. If the *make* command fails, execute the following command:

>*make* clean

- d. Execute the *make* command for the object close program.

Execution of the *make* command compiles and links the files required for writing the application. If the *make* command terminates normally, *inhobj* is created in the current directory where the *make* command was executed. If the *make* command fails, execute the following command:

>*make* clean

- e. Execute the *make* command for the cancel object closure program.

Execution of the *make* command compiles and links the files required for writing the application. If the *make* command terminates normally, *pmtobj* is created in the current directory where the *make* command was executed. If the *make* command fails, execute the following command:

>*make* clean

- f. Execute the *make* command for the WorkUnit list acquisition/release program.

By executing the *make* command, the files required for creating an application are compiled and linked. If the *make* command terminates normally, *lstsys* is created in the current directory in which the *make* command was executed.

If the *make* command fails, execute the following command:

>*make* clean

Appendix C Notes on OLTP Server Operations

This appendix provides notes on OLTP server operation.

C.1 Operation Using the Interface Information Check Functions

When requesting processing from a client application to a server application, this function checks that there is no difference in the interface information between the two.

These functions help to avoid the following occurrence if any inconsistency in the interface information occurs between the client and the server:

- Request data from the client application that was received by the server application shows an illegal value.
- Memory shortage error.
- No response to processing request
- Abnormal termination of Interstage

The interface information check function can be used if a client application is created in the following range and linked to the CORBA Service or Component Transaction Service.

- One of the following is used in the client applications
 - Static start interface of the CORBA client created in the following languages
 - C language
 - C++ language
 - Java
 - COBOL
 - Portable-ORB
- Either of the following services is used (see the following figures).
 - CORBA Service or

- Component Transaction Service

Figure C.1 CORBA Service

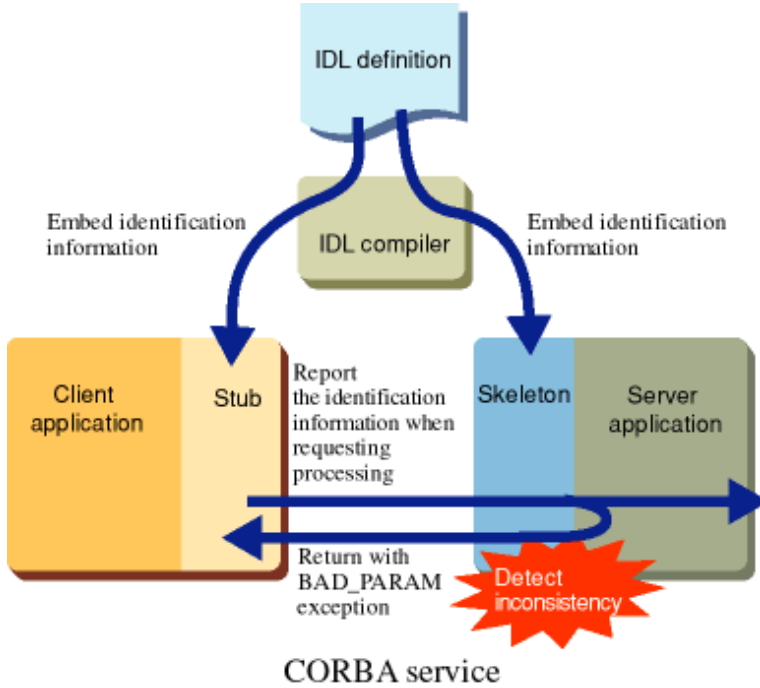
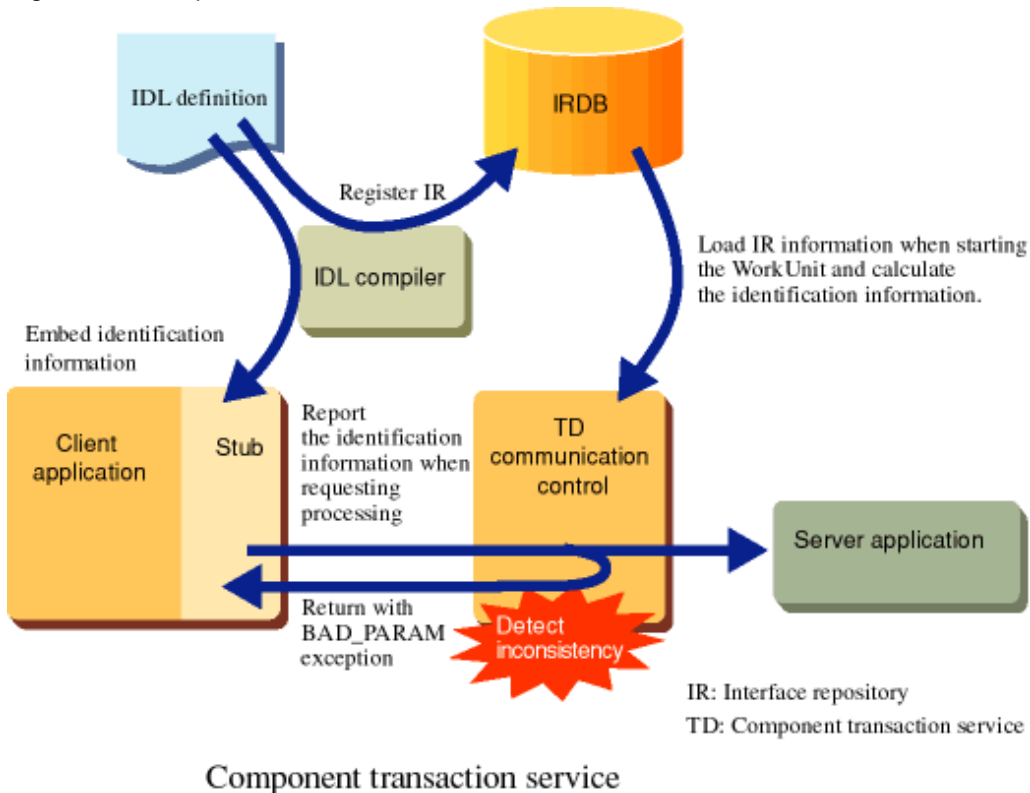


Figure C.2 Component Transaction Service



Note

To use this function, Interstage Application Server V4.0 or later must be installed in both the server and client environments. If the client environment uses one of the following, an execution error occurs in the client application.

- The run time of Interstage Application Server is V3.1 or earlier is installed and
- By using the stub file corresponding to this function, execute the created client application

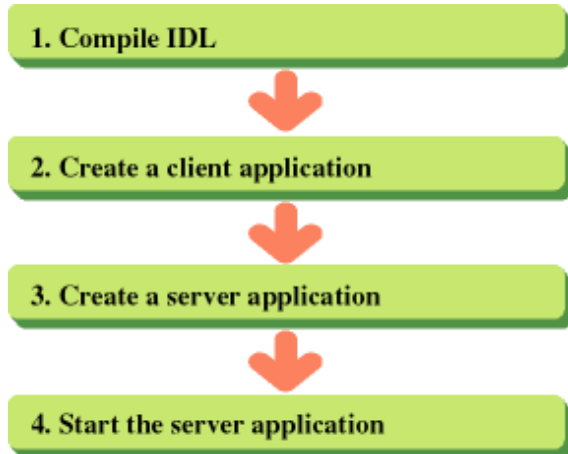
C.1.1 Procedure for Operating the Interface Information Check Function

This section describes the procedure for operating the interface information check function.

CORBA Service

The following figure shows the Interface Information Check Function with a CORBA Service.

Figure C.3 Interface Information Check Function with a CORBA Service



1) Compiling IDL

Execute IDL compilation with the *IDLc* command and create a stub file and a skeleton file. Specify the *-f* option for the *IDLc* command.

For the *IDLc* command, refer to the Reference Manual (Command Edition).

2) Creating a Client Application

Create a client application. Be sure to use the stub file created in Step 1) above. For details on how to create a client application, refer to the Distributed Application Development Guide (CORBA Service Edition).

3) Creating a Server Application

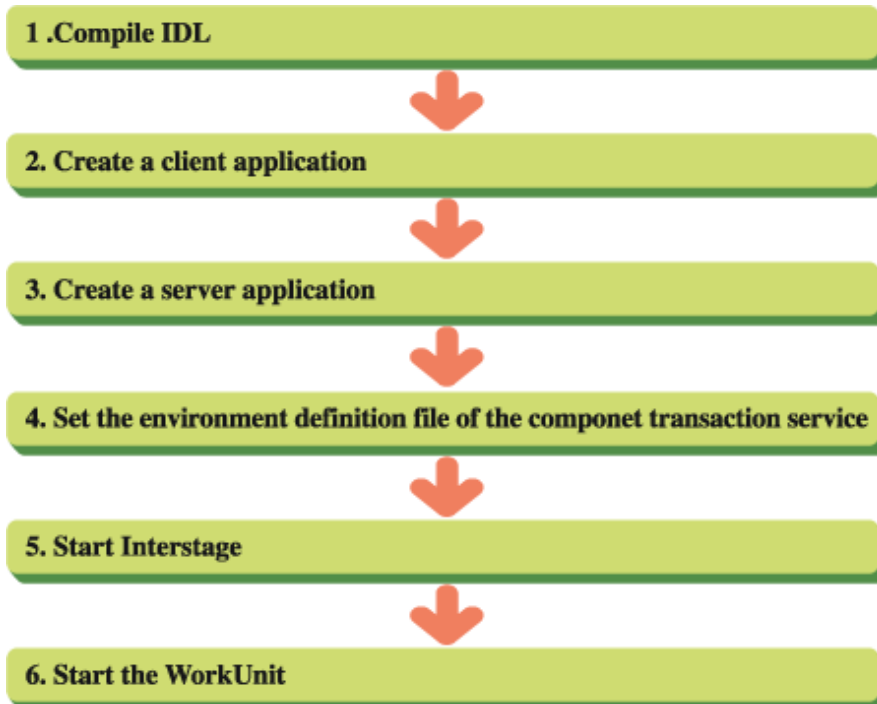
Create a server application. Be sure to use the skeleton file created in Step 1) above. For details on how to create a server application, refer to the Distributed Application Development Guide (CORBA Service Edition).

4) Starting the Server Application

Start the server application.

Component Transaction Service

Figure C.4 Interface Information Check Function with a Component Transaction Service



1) Compiling IDL

Execute IDL compilation with the *tdc* command to create a stub file and a skeleton file. Specify the *-f* option for the *tdc* command.

For information about the *tdc* command, refer to the Reference Manual (Command Edition).

2) Creating a Client Application

Create a client application. Be sure to use the stub file created in Step 1) above. For details on how to create a client application, refer to the Distributed Application Development Guide (CORBA Service Edition).

3) Creating a Server Application

Create a server application. Though there is no particular risk in the creation of a server application, be sure to use the skeleton file created concurrently with the stub file in Step 1) above.

4) Setting the Environment Definition File of the Component Transaction Service

Specify whether to use the interface information check function in the environment definition file.

Table C.1 Environment Definition Information

Section name	Definition name	Details of specifications
SYSTEM ENVIRONMENT	Using Interface Check	Specify whether to use the interface information check function. "YES": The interface information check function is used. "NO": The interface information check function is not used. The default is "NO."

5) Starting Interstage

Start Interstage with the *isstart* command.

For information about the *isstart* command, refer to the Reference Manual (Command Edition).

6) Starting the WorkUnit

Start the WorkUnit with the *isstartwu* command.

For the *isstartwu* command, refer to the Reference Manual (Command Edition).

Appendix D WorkUnit Automatic Start Setting File

This appendix explains the WorkUnit automatic start setting file that is created to automatically start a WorkUnit when Interstage starts.

D.1 Coding Format

The file coding format normally consists of the following elements. If the file coding format contains an error, a syntax error occurs and all of the coded information becomes invalid.

- Statement
- Section
- Comment line
- Space line

D.1.1 Statement

A statement is a line for setting information. The following format is used for a statement.

```
workunit-name1@user-name1[:workunit-name2@user-name1](\n)
```

Each statement consists of a WorkUnit name, an @ (at sign), a user name, and a ":" (colon). The statement coding rules are as follows:

- To omit a statement, delete the relevant statement or omit only parameter values.
- A comment cannot be inserted into the statement line.

The individual elements of a statement are explained below.

WorkUnit Name

Specify the name of the target WorkUnit according to the following rules:

- Specify the name using up to 36 bytes of alphanumeric characters, hyphens, underscores.
- Note that en-size uppercase and lowercase letters are not distinguished and em-size uppercase and lowercase letters are not distinguished. No name can begin or end with a hyphen, or an underscore. If a tab or space is specified at the beginning of the target line, it is ignored.

User Name

Specify a user name according to the following rules:

- Specify the user name using up to 20 bytes of alphanumeric characters.
- If a tab or space is specified at the beginning of the target line, it is ignored.

: (colon)

Use a colon as a delimiter between sets of WorkUnit names and user names according to the following rules:

- If a space or tab is inserted before or after the colon it is ignored.
- Indicate the end of the setting with a space, tab, linefeed ('\n'), or EOF.
- Upper case and lowercase letters are distinguished.
- Only one character string can be specified.
- If a space or tab is to be included, it must be enclosed within double quotes.
- If two or more settings are to be specified, provide as many statements as there are settings.

@ (at sign)

Use this sign as a delimiter between a WorkUnit name and user name.

Note

A space or tab cannot be inserted before or after a colon.

D.1.2 Section

A section is a group of statements. It can be coded in the following format:

```
[section-name] (\n)
  statement (\n)
  statement (\n)
```

Each section consists of "[section-name]" and two or more statements. The following rules apply:

- Only "Work Unit" can be specified for the section name.
- A section begins with "[section-name]" and continues until the next section or EOF appears.
- To omit a section, delete the entire section or change it to a comment.
- A section that consists of "[section-name]" alone without statements cannot be specified.
- No statements other than "[section-name]" can be included on the same line as "[section-name]".
- Each section name must always be enclosed in square brackets "[]".
- A character string consisting of alphanumeric characters and/or spaces can be specified for a section name. The character string must begin with an alphanumeric character.
- Uppercase and lowercase letters used to specify a section name are not distinguished.
- No command can be inserted on the same line as "[section-name]".

Examples of normal section specifications are provided below.

Example 1: A section contains one statement.

```
[Section] (\n)
  Statement
```

Example 2: A section contains three statements.

```
[Section] (\n)
  Statement
  Statement
  Statement
```

Examples of statements that cause a syntax error are shown below.

Example 3: A comment is inserted onto the line where [section name] is provided.

```
[Section]      # This is a Section (\n)
  Statement
```

Example 4: The square brackets of [section name] are not correctly paired.

```
[Section (\n)
  Statement
```

D.1.3 Comment Line

A comment line is used to insert a comment into a file. Write a comment as follows:

```
# comment (\n)
```

The following rules apply to the comment line:

- Specify a number sign "#" at the beginning of the comment line.

D.1.4 Space Line

A space line can be inserted. It is ignored during analysis.

D.2 WorkUnit Automatic Start Setting File Example

An example of a WorkUnit automatic start setting file is shown below.

```
[Work Unit]
ODWU@user1:EJBWU@user2
TDWU1@user3
TDWU2@user4
```

D.2.1 WorkUnit Name

Specify the names of the WorkUnits to be started automatically when Interstage starts. The WorkUnits are started in the specified order.

For CORBA WorkUnits and IJServers, multiple WorkUnits can be specified by delimiting them with a colon (:) so that they can be started in parallel.

Refer to Appendix A "WorkUnit Definition" for the WorkUnit name specification procedure.

D.2.2 User Name

The user name specified immediately after an @ (at sign) does not affect the starting of a WorkUnit, and cannot perform display and specification via the Interstage management console.

The character sequence is only required for syntax reasons, so specify a dummy user name as an alphanumeric character sequence of less than eight characters.

Appendix E Procedure for CORBA WorkUnit Operation Using the Interstage Management Console

This appendix describes the procedure for CORBA WorkUnit operation using the Interstage management console.

E.1 For Solaris

- [Procedure for Operation Using a Sample C Application](#)
- [Procedure for Operation Using a Sample Java Application](#)
- [Procedure for Operation Using a Sample C++ Application](#)
- [Procedure for Operation Using a Sample COBOL Application](#)

E.1.1 Procedure for Operation Using a Sample C Application

This section explains the procedure for creating an environment in which a CORBA WorkUnit (CORBA application WorkUnit) can be operated using the Interstage Management Console.

If a CORBA application is to be run in a WorkUnit, an application execution environment must be created according to the following procedure. Using a sample C application, this section explains how to create a WorkUnit, operate the application, and then delete the WorkUnit.

- Creating a CORBA application
- Starting the Interstage Management Console
- Creating a CORBA WorkUnit
- Deploying the CORBA application
- Starting the CORBA WorkUnit
- Running the Client Application
- Stopping the CORBA WorkUnit
- Undeploying the CORBA application
- Deleting the CORBA WorkUnit

This section explains the procedure for starting the sample application located at /opt/FSUNod/src/samples/complex/samplelist.C/data/any.

1) Creating a CORBA Application

Create a server application according to the execution procedure provided in the Distributed Application Development Guide (CORBA Service Edition). Here, the sample application provided under /opt/FSUNod/src/samples/complex/samplelist.C/data/any is used.

```
# OD_HOME=/opt/FSUNod
# export OD_HOME (Set OD_HOME in the environment variable.)
# LD_LIBRARY_PATH=$OD_HOME/lib:$LD_LIBRARY_PATH
# export LD_LIBRARY_PATH (Set the library path in the environment variable.)
# cd /opt/FSUNod/src/samples/complex/samplelist.C/data/any
# make (Create an application.)
```

2) Starting the Interstage Management Console

Start the Interstage Management Console by specifying "https://(host-name):12000/IsAdmin" from the Web browser.

3) Creating a CORBA WorkUnit

Create a WorkUnit definition to enable the CORBA application to run in a WorkUnit.

1. Select [System], [WorkUnits], and then the [Create a new WorkUnit] tab.
2. Select "CORBA" as the WorkUnit type.
The default "WU001" is entered for the WorkUnit name.
3. Click the [WorkUnit Settings [Show]] link to display detailed settings and enter /opt/FSUNod/src/samples/complex/samplelist.C/data/any for the Application Folder and enter /opt/FSUNod/src/samples/complex/samplelist.C/data/any for the Application Working Directory. Click the Create button.

4) Deploying the CORBA Application

Deploy the CORBA application so that it can run in the WorkUnit.

1. Select the [View WorkUnit Status] tab and click on the newly created WorkUnit (WU001) in the WorkUnit list.
2. Select the [Deploy] tab and enter "IDL:ODsample/anytest:1.0" for the Implementation Repository ID and "simple_s" for the Program EXE File.
Leave the "Restart WorkUnit after deployment" check box selected.
3. Click [Show Details [Show]] to display CORBA Application and Interface
4. Click [CORBA Application [Show]] to display the detailed settings of the CORBA application. Select "SYNC_END" for the operation mode.
5. Click [Interface [Show]] to display the interface definition items. Click the Add Interface button to display the interface definition addition screen.
6. Enter "IDL:ODsample/anytest:1.0" for the Interface Repository ID and "ODsample::anytest" for the Naming Service Name.
7. Click the Add button to add the interface definition.
8. After the interface definition is complete, click the Deploy button to deploy the application.

5) Starting the CORBA WorkUnit

Start the CORBA WorkUnit that was set in the WorkUnit definition.

Enabling "Restart WorkUnit after deployment" during deployment setting automatically starts the WorkUnit when deployment is finished. This function is enabled by default.

The WorkUnit is also started automatically when Interstage starts. Normally, automatic starting is also enabled by default.

1. Select the [Status] tab for the WorkUnit to be started.
2. Click the Start button.

6) Running the Client Application

Create a client application according to the procedure in the Distributed Application Development Guide (CORBA Service Edition). In this example, the sample application provided at /opt/FSUNod/src/samples/complex/samplelist.C/data/any is used.

```
# cd /opt/FSUNod/src/samples/complex/samplelist.C/data/any
# make (Create an application.)
```

Run the client application.

Execute the following at the client application storage destination:

```
# simple_c
TC_ODsample_sample1:para1 = [300] para2 = [test]
TC_ODsample_sample2:para1 = [x] para2 = [0.001000]
TC_ODsample_sample3:para1 = [y] para2 = [0.000100]
```

7) Stopping the CORBA WorkUnit

Stop the WorkUnit that is running.

1. Select the [Status] tab for the WorkUnit to be stopped.
2. Click the Stop button.
3. "WorkUnit Stop Selection" page is displayed. Select "Normal Stop" and click the Stop button.

8) Undeploying the CORBA Application

Undeploy the CORBA application.

1. Select the [Undeploy] tab.
2. Select the check box to the left of the Implementation Repository ID to be undeployed, and click Undeploy.

9) Deleting the CORBA WorkUnit

Delete the CORBA WorkUnit.

1. Select the [Status] tab and click the Delete button.
2. The dialog message "WorkUnit will be deleted. Do you wish to continue?" is displayed. Click OK.

E.1.2 Procedure for Operation Using a Sample Java Application

This section explains the procedure for creating an environment in which a CORBA WorkUnit (CORBA application WorkUnit) can be operated using the Interstage Management Console.

If a CORBA application is to be run in a WorkUnit, an application execution environment must be created according to the following procedure. Using a sample Java application, this section explains how to create a WorkUnit, operate the application, and then delete the WorkUnit.

- Creating a CORBA application
- Starting the Interstage Management Console
- Creating a CORBA WorkUnit
- Deploying the CORBA application
- Starting the CORBA WorkUnit
- Running the Client Application
- Stopping the CORBA WorkUnit
- Undeploying the CORBA application
- Deleting the CORBA WorkUnit

This section explains the procedure for starting the sample application located at `/opt/FSUNod/src/samples/complex/samplelist.Java/data/any`.

The procedure for Java version "JDK 6" (the JDK installation path `"/opt/FJSVawjbc/jdk6"`) is shown below.

1) Creating a CORBA Application

Create a server application according to the execution procedure provided in the Distributed Application Development Guide (CORBA Service Edition). In this example, the sample application provided under `/opt/FSUNod/src/samples/complex/samplelist.Java/data/any` is used.

```
# PATH=/usr/bin:/opt/FJSVawjbc/jdk6/jre/bin:/opt/FJSVawjbc/jdk6/bin:/opt/
FSUNod/bin:$PATH
# export PATH
# OD_HOME=/opt/FSUNod
# export OD_HOME (Set OD_HOME in the environment variable.)
# CLASSPATH=.:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH
# export CLASSPATH (Set the classpath in the environment variable.)
# LD_LIBRARY_PATH=$OD_HOME/lib:$LD_LIBRARY_PATH
# export LD_LIBRARY_PATH (Set the library path in the environment variable.)
```

```
# cd /opt/FSUNod/src/samples/complex/samplelist.Java/data/any
# make (Create an application.)
```

2) Starting the Interstage Management Console

Start the Interstage Management Console by specifying "https://(host-name):12000/IsAdmin" from the Web browser.

3) Creating a CORBA WorkUnit

Create a WorkUnit definition to enable the CORBA application to run in a WorkUnit.

1. Select [System], [WorkUnits], and then the [Create a new WorkUnit] tab.
2. Select "CORBA" as the WorkUnit type.

The default "WU001" is entered for the WorkUnit name.

3. Click the [WorkUnit Settings [Show]] link to display detailed settings and enter /opt/FJSVawjbc/jdk6/bin for the Application Folder; /opt/FSUNod/src/samples/complex/samplelist.Java/data/any for the Application Working Directory; /opt/FSUNod/lib for the Library Path; and OD_IMPLID=IDL:ODsample/anytest:1.0 for the Environment variables. Click the Create button.

4) Deploying the CORBA Application

Deploy the CORBA application so that it can run in the WorkUnit.

To run more than one application on one WorkUnit, repeat the deployment.

1. Select the [View WorkUnit Status] tab and click the newly created WorkUnit (WU001) in the WorkUnit list.
2. Select the [Deploy] tab, and enter 'IDL:ODsample/anytest:1.0' for the Implementation Repository ID and "java" for the Program EXE File.

Leave the "Restart WorkUnit after deployment" check box selected.

3. Click [CORBA Application [Show]] to display the detailed settings of the CORBA application.
Enter "/opt/FSUNod/src/samples/complex/samplelist.Java/data/any" and "/opt/FSUNod/etc/class/ODjava4.jar" on separate lines, for the Classpath; enter class name "simple_s" for the Startup parameters, and select "SYNC_END" for the operation mode.
4. Click [Interface [Show]] to display the interface definition items. Click the Add Interface button to display the interface definition addition screen.
5. Enter "IDL:ODsample/anytest:1.0" for the Interface Repository ID and "ODsample::anytest" for the Naming Service Name.
6. Click the Add button to add the interface definition.
7. After the interface definition is complete, click the Deploy button to deploy the application.

5) Starting the CORBA WorkUnit

Start the CORBA WorkUnit that was set in the WorkUnit definition.

Enabling "Restart WorkUnit after deployment" during deployment setting automatically starts the WorkUnit when deployment is finished. This function is enabled by default.

The WorkUnit is also started automatically when Interstage starts. Normally, automatic starting is also enabled by default.

1. Select the [Status] tab for the WorkUnit to be started.
2. Click the Start button.

6) Running the Client Application

Create a client application according to the procedure in the Distributed Application Development Guide (CORBA Service Edition). In this example, the sample application provided at /opt/FSUNod/src/samples/complex/samplelist.Java/data/any is used.

```
# PATH=/usr/bin:/opt/FJSVawjbc/jdk6/jre/bin:/opt/FJSVawjbc/jdk6/bin:/opt/
FSUNod/bin:$PATH
# export PATH
```

```
# CLASSPATH=.: /opt/FSUNod/etc/class/ODjava4.jar:$CLASSPATH
# export CLASSPATH
# LD_LIBRARY_PATH=/opt/FSUNod/lib:$LD_LIBRARY_PATH
# export LD_LIBRARY_PATH
# cd /opt/FSUNod/src/samples/complex/samplelist.Java/data/any
```

Run the client application.

Execute the following at the client application storage destination:

```
# exec-CL
rtn.param1 = 300
rtn.param2 =
out.param1 = x
out.param2 = 0.0010
inout.param1 = y
inout.param2 = 1.0E-4
```

7) Stopping the CORBA WorkUnit

Stop the WorkUnit that is running.

1. Select the [Status] tab for the WorkUnit to be stopped.
2. Click the Stop button.
3. "WorkUnit Stop Selection" page is displayed. Select "Normal Stop" and click the Stop button.

8) Undeploying the CORBA Application

Undeploy the CORBA application.

1. Select the [Undeploy] tab.
2. Select the check box to the left of the Implementation Repository ID to be undeployed, and click Undeploy.

9) Deleting the CORBA WorkUnit

Delete the CORBA WorkUnit.

1. Select the [Status] tab and click the Delete button.
2. The dialog message "WorkUnit will be deleted. Do you wish to continue?" is displayed. Click OK.

E.1.3 Procedure for Operation Using a Sample C++ Application

This section explains the procedure for creating an environment in which a CORBA WorkUnit (CORBA application WorkUnit) can be operated using the Interstage Management Console.

If a CORBA application is to be run in a WorkUnit, an application execution environment must be created according to the following procedure. Using a sample C++ application, this section explains how to create a WorkUnit, operate the application, and then delete the WorkUnit.

- Creating a CORBA application
- Starting the Interstage Management Console
- Creating a CORBA WorkUnit
- Deploying the CORBA application
- Starting the CORBA WorkUnit
- Running the Client Application
- Stopping the CORBA WorkUnit
- Undeploying the CORBA application

- Deleting the CORBA WorkUnit

This section explains the process for starting the sample application located at `/opt/FSUNod/src/samples/complex/samplelist.C++/data/any`.

1) Creating a CORBA Application

Create a server application according to the execution procedure provided in the Distributed Application Development Guide (CORBA Service Edition). Here, the sample application provided under `/opt/FSUNod/src/samples/complex/samplelist.C++/data/any` is used.

```
# OD_HOME=/opt/FSUNod
# export OD_HOME (Set OD_HOME in the environment variable.)
# LD_LIBRARY_PATH=$OD_HOME/lib:$LD_LIBRARY_PATH
# export LD_LIBRARY_PATH (Set the library path in the environment variable.)
# cd /opt/FSUNod/src/samples/complex/samplelist.C++/data/any
# make (Create an application.)
```

2) Starting the Interstage Management Console

Start the Interstage Management Console by specifying `"https://(host-name):12000/IsAdmin"` from the Web browser.

3) Creating a CORBA WorkUnit

Create a WorkUnit definition to enable the CORBA application to run in a WorkUnit.

1. Select [System], [WorkUnits], and then the [Create a new WorkUnit] tab.
2. Select "CORBA" as the WorkUnit type.

The default "WU001" is entered for the WorkUnit name.

3. Click the [WorkUnit Settings [Show]] link to display detailed settings and enter `/opt/FSUNod/src/samples/complex/samplelist.C++/data/any` for the Application Folder and enter `/opt/FSUNod/src/samples/complex/samplelist.C++/data/any` for the Application Working Directory. Click the Create button.

4) Deploying the CORBA Application

Deploy the CORBA application so that it can run in the WorkUnit.

1. Select the [View WorkUnit Status] tab and click on the newly created WorkUnit (WU001) in the WorkUnit list.
2. Select the [Deploy] tab, and enter "IDL:ODsample/anytest:1.0" for the Implementation Repository ID and "simple_s" for the Program EXE File.
Leave the "Restart WorkUnit after deployment" check box selected.
3. Click [Show Details [Show]] to display CORBA Application and Interface.
4. Click [CORBA Application [Show]] to display the detailed settings of the CORBA application. Select "SYNC_END" for the operation mode.
5. Click [Interface [Show]] to display the interface definition items. Click the Add Interface button to display the interface definition addition screen.
6. Enter "IDL:ODsample/anytest:1.0" for the Interface Repository ID and "ODsample::anytest" for the Naming Service Name.
7. Click the Add button to add the interface definition.
8. After the interface definition is complete, click the Deploy button to deploy the application.

5) Starting the CORBA WorkUnit

Start the CORBA WorkUnit that was set in the WorkUnit definition.

Enabling "Restart WorkUnit after deployment" during deployment setting automatically starts the WorkUnit when deployment is finished. This function is enabled by default.

The WorkUnit is also started automatically when Interstage starts. Normally, automatic starting is also enabled by default.

1. Select the [Status] tab for the WorkUnit to be started.
2. Click the Start button.

6) Running the Client Application

Create a client application according to the procedure in the Distributed Application Development Guide (CORBA Service Edition). In this example, the sample application provided at /opt/FSUNod/src/samples/complex/samplelist.C++/data/any is used.

```
# cd /opt/FSUNod/src/samples/complex/samplelist.C++/data/any
# make (Create an application.)
```

Run the client application.

Execute the following at the client application storage destination:

```
# simple_c
smp1->para1 = 100
smp1->para2 = OUT
smp2->para1 = x
smp2->para2 = 0.01
smp3->para1 = z
smp3->para2 = 0.001
```

7) Stopping the CORBA WorkUnit

Stop the WorkUnit that is running.

1. Select the [Status] tab for the WorkUnit to be stopped.
2. Click the Stop button.
3. "WorkUnit Stop Selection" page is displayed. Select "Normal Stop" and click the Stop button.

8) Undeploying the CORBA Application

Undeploy the CORBA application.

1. Select the [Undeploy] tab.
2. Select the check box to the left of the Implementation Repository ID to be undeployed, and click Undeploy.

9) Deleting the CORBA WorkUnit

Delete the CORBA WorkUnit.

1. Select the [Status] tab and click the Delete button.
2. The dialog message "WorkUnit will be deleted. Do you wish to continue?" is displayed. Click OK.

E.1.4 Procedure for Operation Using a Sample COBOL Application

This section explains the procedure for creating an environment in which a CORBA WorkUnit (CORBA application WorkUnit) can be operated using the Interstage Management Console.

If a CORBA application is to be run in a WorkUnit, an application execution environment must be created according to the following procedure. Using a sample COBOL application, this section explains how to create a WorkUnit, operate the application, and then delete the WorkUnit.

- Creating a CORBA application
- Starting the Interstage Management Console
- Creating a CORBA WorkUnit
- Deploying the CORBA application
- Starting the CORBA WorkUnit

- Running the Client Application
- Stopping the CORBA WorkUnit
- Undeploying the CORBA application
- Deleting the CORBA WorkUnit

This section explains the process for starting the sample application located at `/opt/FSUNod/src/samples/complex/samplelist.COBOL/data/any`.

1) Creating a CORBA Application

Create a server application according to the execution procedure provided in the Distributed Application Development Guide (CORBA Service Edition). In this example, the sample application provided under `/opt/FSUNod/src/samples/complex/samplelist.COBOL/data/any` is used.

```
# OD_HOME=/opt/FSUNod
# export OD_HOME (Set OD_HOME in the environment variable.)
# LD_LIBRARY_PATH=$OD_HOME/lib:$LD_LIBRARY_PATH
# export LD_LIBRARY_PATH (Set the library path in the environment variable.)
# cd /opt/FSUNod/src/samples/complex/samplelist.COBOL/data/any
# make (Create an application.)
```

2) Starting the Interstage Management Console

Start the Interstage Management Console by specifying `"https://(host-name):12000/IsAdmin"` from the Web browser.

3) Creating a CORBA WorkUnit

Create a WorkUnit definition to enable the CORBA application to run in a WorkUnit.

1. Select [System], [WorkUnits], and then the [Create a new WorkUnit] tab.
2. Select "CORBA" as the WorkUnit type.

The default "WU001" is entered for the WorkUnit name.

3. Click the [WorkUnit Settings [Show]] link to display detailed settings and enter `/opt/FSUNod/src/samples/complex/samplelist.COBOL/data/any` for the Application Folder and `/opt/FSUNod/src/samples/complex/samplelist.COBOL/data/any` for the Application Working Directory.

Also enter `/opt/FJSVcbl/lib`, `/opt/FSUNod/lib/nt`, `/opt/FSUNod/src/samples/complex/samplelist.COBOL/data/any` for the library path, and `NLSPATH=/opt/FJSVcbl/lib/nls/%L/%N.cat:/opt/FJSVcbl/lib/nls/C/%N.cat:/usr/dt/lib/nls/msg/%L/%N.cat` for the Environment variables. Click the Create button.

4) Deploying the CORBA Application

Deploy the CORBA application so that it can run in the WorkUnit.

1. Select the [View WorkUnit Status] tab and click the newly created WorkUnit (WU001) in the WorkUnit list.
2. Select the [Deploy] tab, and enter "IDL:INTF_A:1.0" for the Implementation Repository ID and "024_s" for the Program EXE File. Leave the "Restart WorkUnit after deployment" check box selected.
3. Click [Show Details [Show]] to display Show CORBA Application and Show Interface Definition.
4. Click [Interface [Show]] to display the interface definition items. Click the Add Interface button to display the interface definition addition screen.
5. Enter "IDL:INTF_A:1.0" for the Interface Repository ID and "INTF_A" for the Naming Service Name. Also enter `/opt/FSUNod/src/samples/complex/samplelist.COBOL/data/any/libINTF-A.so` for the Library Path.
6. Click the Add button to add the interface definition.
7. After the interface definition is complete, click the Deploy button to deploy the application.

5) Starting the CORBA WorkUnit

Start the CORBA WorkUnit that was set in the WorkUnit definition.

Enabling "Restart WorkUnit after deployment" during deployment setting automatically starts the WorkUnit when deployment is finished. This function is enabled by default.

The WorkUnit is also started automatically when Interstage starts. Normally, automatic starting is also enabled by default.

1. Select the [Status] tab for the WorkUnit to be started.
2. Click the Start button.

6) Running the Client Application

Create a client application according to the procedure in the Distributed Application Development Guide (CORBA Service Edition). In this example, the sample application provided at /opt/FSUNod/src/samples/complex/samplelist.COBOL/data/any is used.

```
# LD_LIBRARY_PATH=/opt/FSUNod/src/samples/complex/samplelist.COBOL/data/any:/  
opt/FSUNod/lib/nt:/opt/FJSVcbl/lib:$LD_LIBRARY_PATH  
# export LD_LIBRARY_PATH (Set the library path in the environment variable.)  
# cd /opt/FSUNod/src/samples/complex/samplelist.COBOL/data/any  
# make (Create an application.)
```

Run the client application.

Execute the following at the client application storage destination:

```
# 024_c  
CLIENT START!!  
IN-PARAM VALUE: 001743234  
IO-PARAM VALUE: 8888  
INTF-A-OP RETURN!!  
RESULT VALUE : .89333000E 02  
A-OUT-P VALUE : .555555555000000000E 02  
A-IO-P VALUE : +000012345
```

7) Stopping the CORBA WorkUnit

Stop the WorkUnit that is running.

1. Select the [Status] tab for the WorkUnit to be stopped.
2. Click the Stop button.
3. "WorkUnit Stop Selection" page is displayed. Select "Normal Stop" and click the Stop button.

8) Undeploying the CORBA Application

Undeploy the CORBA application.

1. Select the [Undeploy] tab.
2. Select the check box to the left of the Implementation Repository ID to be undeployed, and click Undeploy.

9) Deleting the CORBA WorkUnit

Delete the CORBA WorkUnit.

1. Select the [Status] tab and click the Delete button.
2. The dialog message "WorkUnit will be deleted. Do you wish to continue?" is displayed. Click OK.

E.2 For Windows(R)

- [Procedure for Operation Using a Sample C Application](#)

- [Procedure for Operation Using a Sample Java Application](#)
- [Procedure for Operation Using a Sample C++ Application](#)
- [Procedure for Operation Using a Sample COBOL Application](#)

E.2.1 Procedure for Operation Using a Sample C Application

This section explains the procedure for creating an environment in which a CORBA WorkUnit (CORBA application WorkUnit) can be operated using the Interstage Management Console.

If a CORBA application is to be run in a WorkUnit, an application execution environment must be created according to the following procedure. Using a sample C application, this section explains how to create a WorkUnit, operate the application, and then delete the WorkUnit.

- Creating a CORBA application
- Starting the Interstage Management Console
- Creating a CORBA WorkUnit
- Deploying the CORBA application
- Starting the CORBA WorkUnit
- Running the Client Application
- Stopping the CORBA WorkUnit
- Undeploying the CORBA application
- Deleting the CORBA WorkUnit

This section explains the procedure for starting the sample application located at C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.C\DATA\ANY_S.

1) Creating a CORBA Application

Create a server application according to the execution procedure provided in the Distributed Application Development Guide (CORBA Service Edition). Here, the sample application provided under C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.C\DATA\ANY_S is used.

1. Change the current folder to the following folder:

```
# C:
# 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. Build does ANY_S.exe

2) Starting the Interstage Management Console

Start the Interstage Management Console by specifying "https://(host-name):12000/IsAdmin" from the Web browser.

3) Creating a CORBA WorkUnit

Create a WorkUnit definition to enable the CORBA application to run in a WorkUnit.

1. Select [System], [WorkUnits], and then the [Create a new WorkUnit] tab.
2. Select "CORBA" as the WorkUnit type.

The default "WU001" is entered for the WorkUnit name.

3. **Windows32**

Click the [WorkUnit Settings [Show]] link to display detailed settings. Enter:

- C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.C\DATA\ANY_S\Release
for the Application Folder, Enter:
- C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.C\DATA\ANY_S\Release
for the Application Working Directory.

Windows64

Click the [WorkUnit Settings [Show]] link to display detailed settings. Enter:

- C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.C\DATA\ANY_S\Itanium\Release
for the Application Folder. Enter:
- C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.C\DATA\ANY_S\Itanium\Release
for the Application Working Directory.

4. Click the Create button.

4) Deploying the CORBA Application

Deploy the CORBA application so that it can run in the WorkUnit.

1. Select the [View WorkUnit Status] tab and click on the newly created WorkUnit (WU001) in the WorkUnit list.
2. Select the [Deploy] tab and enter "IDL:ODsample/anytest:1.0" for the Implementation Repository ID and " ANY_S.exe" for the Program EXE File.
Leave the "Restart WorkUnit after deployment" check box selected.
3. Click [Show Details [Show]] to display CORBA Application and Interface
4. Click [CORBA Application [Show]] to display the detailed settings of the CORBA application. Select "SYNC_END" for the operation mode.
5. Click [Interface [Show]] to display the interface definition items. Click the Add Interface button to display the interface definition addition screen.
6. Enter "IDL:ODsample/anytest:1.0" for the Interface Repository ID and "ODsample::anytest" for the Naming Service Name.
7. Click the **Add** button to add the interface definition.
8. After the interface definition is complete, click the Deploy button to deploy the application.

5) Starting the CORBA WorkUnit

Start the CORBA WorkUnit that was set in the WorkUnit definition.

Enabling "Restart WorkUnit after deployment" during deployment setting automatically starts the WorkUnit when deployment is finished. This function is enabled by default.

The WorkUnit is also started automatically when Interstage starts. Normally, automatic starting is also enabled by default.

1. Select the [Status] tab for the WorkUnit to be started.
2. Click the Start button.

6) Running the Client Application

Create a client application according to the procedure in the Distributed Application Development Guide (CORBA Service Edition). In this example, the sample application provided at C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.C\DATA\ANY_C is used.

1. Change the current folder to the following folder:

```
# C:  
# cd C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.C\DATA\ANY_C
```

2. Execute the IDL compiler from the command prompt to generate the skeletons.

```
# IDLc simple.idl
```

3. Build does ANY_C.exe

Run the client application.

Execute the following at the client application storage destination:

Windows32

```
# cd Release  
# ANY_C  
TC_ODsample_sample1:para1 = [300] para2 = [test]  
TC_ODsample_sample2:para1 = [x] para2 = [0.001000]  
TC_ODsample_sample3:para1 = [y] para2 = [0.000100]
```

Windows64

```
# cd Itanium\Release  
# ANY_C  
TC_ODsample_sample1:para1 = [300] para2 = [test]  
TC_ODsample_sample2:para1 = [x] para2 = [0.001000]  
TC_ODsample_sample3:para1 = [y] para2 = [0.000100]
```

7) Stopping the CORBA WorkUnit

Stop the WorkUnit that is running.

1. Select the [Status] tab for the WorkUnit to be stopped.
2. Click the Stop button.
3. "WorkUnit Stop Selection" page is displayed. Select "Normal Stop" and click the Stop button.

8) Undeploying the CORBA Application

Undeploy the CORBA application.

1. Select the [Undeploy] tab.
2. Select the check box to the left of the Implementation Repository ID to be undeployed, and click Undeploy.

9) Deleting the CORBA WorkUnit

Delete the CORBA WorkUnit.

1. Select the [Status] tab and click the Delete button.
2. The dialog message "WorkUnit will be deleted. Do you wish to continue?" is displayed. Click OK.

E.2.2 Procedure for Operation Using a Sample Java Application

This section explains the procedure for creating an environment in which a CORBA WorkUnit (CORBA application WorkUnit) can be operated using the Interstage Management Console.

If a CORBA application is to be run in a WorkUnit, an application execution environment must be created according to the following procedure. Using a sample Java application, this section explains how to create a WorkUnit, operate the application, and then delete the WorkUnit.

- Creating a CORBA application
- Starting the Interstage Management Console
- Creating a CORBA WorkUnit
- Deploying the CORBA application
- Starting the CORBA WorkUnit
- Running the Client Application
- Stopping the CORBA WorkUnit
- Undeploying the CORBA application
- Deleting the CORBA WorkUnit

This section explains the procedure for starting the sample application located at C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.Java\DATA\ANY.

The procedure for Java version "JDK 6" (the JDK installation path "C:\Interstage\JDK6\bin") is shown below.

1) Creating a CORBA Application

Create a server application according to the execution procedure provided in the Distributed Application Development Guide (CORBA Service Edition). In this example, the sample application provided under C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.Java\DATA\ANY is used.

```
# cd C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.Java\DATA\ANY
# make (Create an application.)
```

Press any key to continue.

2) Starting the Interstage Management Console

Start the Interstage Management Console by specifying "https://(host-name):12000/IsAdmin" from the Web browser.

3) Creating a CORBA WorkUnit

Create a WorkUnit definition to enable the CORBA application to run in a WorkUnit.

1. Select [System], [WorkUnits], and then the [Create a new WorkUnit] tab.
2. Select "CORBA" as the WorkUnit type.

The default "WU001" is entered for the WorkUnit name.

3. Click the [WorkUnit Settings [Show]] link to display detailed settings and enter C:\Interstage\JDK6\bin for the Application Folder; C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.Java\DATA\ANY for the Application Working Directory; OD_IMPLID=IDL:ODsample/anytest:1.0 for the Environment variables. Click the Create button.

4) Deploying the CORBA Application

Deploy the CORBA application so that it can run in the WorkUnit.

To run more than one application on one WorkUnit, repeat the deployment.

1. Select the [View WorkUnit Status] tab and click the newly created WorkUnit (WU001) in the WorkUnit list.
2. Select the [Deploy] tab, and enter "IDL:ODsample/anytest:1.0" for the Implementation Repository ID and "java.exe" for the Program EXE File.

Leave the "Restart WorkUnit after deployment" check box selected.

3. Click [CORBA Application [Show]] to display the detailed settings of the CORBA application.

Enter the following:

- Classpath: "C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.Java\DATA\ANY"

- Startup parameters: "-Xrs", and class name "simple_s" (the parameters are separated by a page break)
 - Operation mode: "SYNC_END"
4. Click [Interface [Show]] to display the interface definition items. Click the Add Interface button to display the interface definition addition screen.
 5. Enter "IDL:ODsample/anytest:1.0" for the Interface Repository ID and "ODsample::anytest" for the Naming Service Name.
 6. Click the Add button to add the interface definition.
 7. After the interface definition is complete, click the Deploy button to deploy the application.

5) Starting the CORBA WorkUnit

Start the CORBA WorkUnit that was set in the WorkUnit definition.

Enabling "Restart WorkUnit after deployment" during deployment setting automatically starts the WorkUnit when deployment is finished. This function is enabled by default.

The WorkUnit is also started automatically when Interstage starts. Normally, automatic starting is also enabled by default.

1. Select the [Status] tab for the WorkUnit to be started.
2. Click the Start button.

6) Running the Client Application

Create a client application according to the procedure in the Distributed Application Development Guide (CORBA Service Edition). In this example, the sample application provided at C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.Java\DATA\ANY is used.

```
# cd C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.Java\DATA\ANY
```

Run the client application.

Execute the following at the client application storage destination:

```
# exec-CL
rtn.param1 = 300
rtn.param2 =
out.param1 = x
out.param2 = 0.0010
inout.param1 = y
inout.param2 = 1.0E-4
```

Press any key to continue.

7) Stopping the CORBA WorkUnit

Stop the WorkUnit that is running.

1. Select the [Status] tab for the WorkUnit to be stopped.
2. Click the Stop button.
3. "WorkUnit Stop Selection" page is displayed. Select "Normal Stop" and click the Stop button.

8) Undeploying the CORBA Application

Undeploy the CORBA application.

1. Select the [Undeploy] tab.
2. Select the check box to the left of the Implementation Repository ID to be undeployed, and click Undeploy.

9) Deleting the CORBA WorkUnit

Delete the CORBA WorkUnit.

1. Select the [Status] tab and click the Delete button.
2. The dialog message "WorkUnit will be deleted. Do you wish to continue?" is displayed. Click OK.

E.2.3 Procedure for Operation Using a Sample C++ Application

This is not valid for Windows (64 bit).

This section explains the procedure for creating an environment in which a CORBA WorkUnit (CORBA application WorkUnit) can be operated using the Interstage Management Console.

If a CORBA application is to be run in a WorkUnit, an application execution environment must be created according to the following procedure. Using a sample C++ application, this section explains how to create a WorkUnit, operate the application, and then delete the WorkUnit.

- Creating a CORBA application
- Starting the Interstage Management Console
- Creating a CORBA WorkUnit
- Deploying the CORBA application
- Starting the CORBA WorkUnit
- Running the Client Application
- Stopping the CORBA WorkUnit
- Undeploying the CORBA application
- Deleting the CORBA WorkUnit

This section explains the process for starting the sample application located at C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.C++\DATA\ANY_S.

1) Creating a CORBA Application

Create a server application according to the execution procedure provided in the Distributed Application Development Guide (CORBA Service Edition). Here, the sample application provided under C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.C++\DATA\ANY_S is used.

1. Change the current folder to the following folder:

```
# C:
# 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 -vcpp simple.idl
```

3. Double-click the project workspace file (ANY_S.MDP) from Windows Explorer and invoke Visual C++.
4. Build does ANY_S.exe

2) Starting the Interstage Management Console

Start the Interstage Management Console by specifying "https://(host-name):12000/IsAdmin" from the Web browser.

3) Creating a CORBA WorkUnit

Create a WorkUnit definition to enable the CORBA application to run in a WorkUnit.

1. Select [System], [WorkUnits], and then the [Create a new WorkUnit] tab.
2. Select "CORBA" as the WorkUnit type.

The default "WU001" is entered for the WorkUnit name.

3. Click the [WorkUnit Settings [Show]] link to display detailed settings and enter C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.C++\DATA\ANY_S\Release for the Application Folder and enter C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.C++\DATA\ANY_S\Release for the Application Working Directory. Click the Create button.

4) Deploying the CORBA Application

Deploy the CORBA application so that it can run in the WorkUnit.

1. Select the [View WorkUnit Status] tab and click on the newly created WorkUnit (WU001) in the WorkUnit list.
2. Select the [Deploy] tab, and enter "IDL:ODsample/anytest:1.0" for the Implementation Repository ID and "ANY_S.exe" for the Program EXE File.
Leave the "Restart WorkUnit after deployment" check box selected.
3. Click [Show Details [Show]] to display CORBA Application and Interface.
4. Click [CORBA Application [Show]] to display the detailed settings of the CORBA application. Select "SYNC_END" for the operation mode.
5. Click [Interface [Show]] to display the interface definition items. Click the Add Interface button to display the interface definition addition screen.
6. Enter "IDL:ODsample/anytest:1.0" for the Interface Repository ID and "ODsample::anytest" for the Naming Service Name.
7. Click the Add button to add the interface definition.
8. After the interface definition is complete, click the Deploy button to deploy the application.

5) Starting the CORBA WorkUnit

Start the CORBA WorkUnit that was set in the WorkUnit definition.

Enabling "Restart WorkUnit after deployment" during deployment setting automatically starts the WorkUnit when deployment is finished. This function is enabled by default.

The WorkUnit is also started automatically when Interstage starts. Normally, automatic starting is also enabled by default.

1. Select the [Status] tab for the WorkUnit to be started.
2. Click the Start button.

6) Running the Client Application

Create a client application according to the procedure in the Distributed Application Development Guide (CORBA Service Edition). In this example, the sample application provided at C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.C++\DATA\ANY_C is used.

1. Change the current folder to the following folder:

```
# C:  
# cd C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.C++\DATA\ANY_C
```

2. Execute the IDL compiler from the command prompt to generate the skeletons.

```
# IDLc -vcpp simple.idl
```

3. Double-click the project workspace file (ANY_C.MDP) from Windows Explorer and invoke Visual C++.
4. Build does ANY_C.exe

Run the client application.

Execute the following at the client application storage destination:

```
# cd Release  
# ANY_C  
smp1->para1 = 100  
smp1->para2 = OUT
```

```
smp2->para1 = x
smp2->para2 = 0.01
smp3->para1 = z
smp3->para2 = 0.001
```

7) Stopping the CORBA WorkUnit

Stop the WorkUnit that is running.

1. Select the [Status] tab for the WorkUnit to be stopped.
2. Click the Stop button.
3. "WorkUnit Stop Selection" page is displayed. Select "Normal Stop" and click the Stop button.

8) Undeploying the CORBA Application

Undeploy the CORBA application.

1. Select the [Undeploy] tab.
2. Select the check box to the left of the Implementation Repository ID to be undeployed, and click Undeploy.

9) Deleting the CORBA WorkUnit

Delete the CORBA WorkUnit.

1. Select the [Status] tab and click the Delete button.
2. The dialog message "WorkUnit will be deleted. Do you wish to continue?" is displayed. Click OK.

E.2.4 Procedure for Operation Using a Sample COBOL Application

This is not valid for Windows (64 bit).

This section explains the procedure for creating an environment in which a CORBA WorkUnit (CORBA application WorkUnit) can be operated using the Interstage Management Console.

If a CORBA application is to be run in a WorkUnit, an application execution environment must be created according to the following procedure. Using a sample COBOL application, this section explains how to create a WorkUnit, operate the application, and then delete the WorkUnit.

- Creating a CORBA application
- Starting the Interstage Management Console
- Creating a CORBA WorkUnit
- Deploying the CORBA application
- Starting the CORBA WorkUnit
- Running the Client Application
- Stopping the CORBA WorkUnit
- Undeploying the CORBA application
- Deleting the CORBA WorkUnit

This section explains the process for starting the sample application located at /opt/FSUNod/src/samples/complex/samplelist.COBOL/data/any.

1) Creating a CORBA Application

Create a server application according to the execution procedure provided in the Distributed Application Development Guide (CORBA Service Edition). In this example, the sample application provided under C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.COBOL\DATA\ANY is used.

1. Change the current folder to the following folder:

```
# C:  
# cd C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.COBOL\DATA\ANY
```

2. Execute the IDL compiler from the command prompt to generate the skeletons.

```
# IDLc -cobol 024.idl
```

3. Build does SERVER-MAIN.EXE

2) Starting the Interstage Management Console

Start the Interstage Management Console by specifying "https://(host-name):12000/IsAdmin" from the Web browser.

3) Creating a CORBA WorkUnit

Create a WorkUnit definition to enable the CORBA application to run in a WorkUnit.

1. Select [System], [WorkUnits], and then the [Create a new WorkUnit] tab.
2. Select "CORBA" as the WorkUnit type.

The default "WU001" is entered for the WorkUnit name.

3. Click the [WorkUnit Settings [Show]] link to display detailed settings and enter C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.COBOL\DATA\ANY for the Application Folder and C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.COBOL\DATA\ANY for the Application Working Directory.
4. Click the Create button.

4) Deploying the CORBA Application

Deploy the CORBA application so that it can run in the WorkUnit.

1. Select the [View WorkUnit Status] tab and click the newly created WorkUnit (WU001) in the WorkUnit list.
2. Select the [Deploy] tab, and enter "IDL:INTF_A:1.0" for the Implementation Repository ID and "SERVER-MAIN.EXE" for the Program EXE File.
Leave the "Restart WorkUnit after deployment" check box selected.
3. Click [Show Details [Show]] to display Show CORBA Application and Show Interface Definition.
4. Click [Interface [Show]] to display the interface definition items. Click the Add Interface button to display the interface definition addition screen.
5. Enter "IDL:INTF_A:1.0" for the Interface Repository ID and "INTF_A" for the Naming Service Name.
6. Click the Add button to add the interface definition.
7. After the interface definition is complete, click the Deploy button to deploy the application.

5) Starting the CORBA WorkUnit

Start the CORBA WorkUnit that was set in the WorkUnit definition.

Enabling "Restart WorkUnit after deployment" during deployment setting automatically starts the WorkUnit when deployment is finished. This function is enabled by default.

The WorkUnit is also started automatically when Interstage starts. Normally, automatic starting is also enabled by default.

1. Select the [Status] tab for the WorkUnit to be started.
2. Click the Start button.

6) Running the Client Application

Create a client application according to the procedure in the Distributed Application Development Guide (CORBA Service Edition). In this example, the sample application provided at C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.COBOL\DATA\ANY is used.

1. Change the current folder to the following folder:

```
# C:  
# cd C:\Interstage\ODWIN\Src\SAMPLE\COMPLEX\samplelist.COBOL\DATA\ANY
```

2. Execute the IDL compiler from the command prompt to generate the skeletons.

```
# IDLc -cobol 024.idl
```

3. Build does CLIENT-MAIN.EXE

Run the client application.

Execute the following at the client application storage destination:

```
# CLIENT-MAIN  
CLIENT START!!  
IN-PARAM VALUE: 001743234  
IO-PARAM VALUE: 8888  
INTF-A-OP RETURN!!  
RESULT VALUE : .89333000E 02  
A-OUT-P VALUE : .55555555500000000E 02  
A-IO-P VALUE : +000012345
```

7) Stopping the CORBA WorkUnit

Stop the WorkUnit that is running.

1. Select the [Status] tab for the WorkUnit to be stopped.
2. Click the Stop button.
3. "WorkUnit Stop Selection" page is displayed. Select "Normal Stop" and click the Stop button.

8) Undeploying the CORBA Application

Undeploy the CORBA application.

1. Select the [Undeploy] tab.
2. Select the check box to the left of the Implementation Repository ID to be undeployed, and click Undeploy.

9) Deleting the CORBA WorkUnit

Delete the CORBA WorkUnit.

1. Select the [Status] tab and click the Delete button.
2. The dialog message "WorkUnit will be deleted. Do you wish to continue?" is displayed. Click OK.

E.3 For Linux

- [Procedure for Operation Using a Sample C Application](#)
- [Procedure for Operation Using a Sample Java Application](#)
- [Procedure for Operation Using a Sample C++ Application](#)
- [Procedure for Operation Using a Sample COBOL Application](#)

E.3.1 Procedure for Operation Using a Sample C Application

This section explains the procedure for creating an environment in which a CORBA WorkUnit (CORBA application WorkUnit) can be operated using the Interstage Management Console.

If a CORBA application is to be run in a WorkUnit, an application execution environment must be created according to the following procedure. Using a sample C application, this section explains how to create a WorkUnit, operate the application, and then delete the WorkUnit.

- Creating a CORBA application
- Starting the Interstage Management Console
- Creating a CORBA WorkUnit
- Deploying the CORBA application
- Starting the CORBA WorkUnit
- Running the Client Application
- Stopping the CORBA WorkUnit
- Undeploying the CORBA application
- Deleting the CORBA WorkUnit

This section explains the procedure for starting the sample application located at `/opt/FJSVod/src/samples/complex/samplelist.C/data/any`.

1) Creating a CORBA Application

Create a server application according to the execution procedure provided in the Distributed Application Development Guide (CORBA Service Edition). Here, the sample application provided under `/opt/FJSVod/src/samples/complex/samplelist.C/data/any` is used.

```
# OD_HOME=/opt/FJSVod
# export OD_HOME (Set OD_HOME in the environment variable.)
# LD_LIBRARY_PATH=$OD_HOME/lib:$LD_LIBRARY_PATH
# export LD_LIBRARY_PATH (Set the library path in the environment variable.)
# cd /opt/FJSVod/src/samples/complex/samplelist.C/data/any
# make (Create an application.)
```

2) Starting the Interstage Management Console

Start the Interstage Management Console by specifying "https://(host-name):12000/IsAdmin" from the Web browser.

3) Creating a CORBA WorkUnit

Create a WorkUnit definition to enable the CORBA application to run in a WorkUnit.

1. Select [System], [WorkUnits], and then the [Create a new WorkUnit] tab.
2. Select "CORBA" as the WorkUnit type.

The default "WU001" is entered for the WorkUnit name.

3. Click the [WorkUnit Settings [Show]] link to display detailed settings and enter `/opt/FJSVod/src/samples/complex/samplelist.C/data/any` for the Application Folder and enter `/opt/FJSVod/src/samples/complex/samplelist.C/data/any` for the Application Working Directory. Click the Create button.

4) Deploying the CORBA Application

Deploy the CORBA application so that it can run in the WorkUnit.

1. Select the [View WorkUnit Status] tab and click on the newly created WorkUnit (WU001) in the WorkUnit list.
2. Select the [Deploy] tab and enter "IDL:ODsample/anytest:1.0" for the Implementation Repository ID and "simple_s" for the Program EXE File.

Leave the "Restart WorkUnit after deployment" check box selected.

3. Click [Show Details [Show]] to display CORBA Application and Interface
4. Click [CORBA Application [Show]] to display the detailed settings of the CORBA application. Select "SYNC_END" for the operation mode.
5. Click [Interface [Show]] to display the interface definition items. Click the Add Interface button to display the interface definition addition screen.
6. Enter "IDL:ODsample/anytest:1.0" for the Interface Repository ID and "ODsample::anytest" for the Naming Service Name.
7. Click the **Add** button to add the interface definition.
8. After the interface definition is complete, click the Deploy button to deploy the application.

5) Starting the CORBA WorkUnit

Start the CORBA WorkUnit that was set in the WorkUnit definition.

Enabling "Restart WorkUnit after deployment" during deployment setting automatically starts the WorkUnit when deployment is finished. This function is enabled by default.

The WorkUnit is also started automatically when Interstage starts. Normally, automatic starting is also enabled by default.

1. Select the [Status] tab for the WorkUnit to be started.
2. Click the Start button.

6) Running the Client Application

Create a client application according to the procedure in the Distributed Application Development Guide (CORBA Service Edition). In this example, the sample application provided at /opt/FJSVod/src/samples/complex/samplelist.C/data/any is used.

```
# cd /opt/FJSVod/src/samples/complex/samplelist.C/data/any
# make (Create an application.)
```

Run the client application.

Execute the following at the client application storage destination:

```
# simple_c
TC_ODsample_sample1:para1 = [300] para2 = [test]
TC_ODsample_sample2:para1 = [x] para2 = [0.001000]
TC_ODsample_sample3:para1 = [y] para2 = [0.000100]
```

7) Stopping the CORBA WorkUnit

Stop the WorkUnit that is running.

1. Select the [Status] tab for the WorkUnit to be stopped.
2. Click the Stop button.
3. "WorkUnit Stop Selection" page is displayed. Select "Normal Stop" and click the Stop button.

8) Undeploying the CORBA Application

Undeploy the CORBA application.

1. Select the [Undeploy] tab.
2. Select the check box to the left of the Implementation Repository ID to be undeployed, and click Undeploy.

9) Deleting the CORBA WorkUnit

Delete the CORBA WorkUnit.

1. Select the [Status] tab and click the Delete button.
2. The dialog message "WorkUnit will be deleted. Do you wish to continue?" is displayed. Click OK.

E.3.2 Procedure for Operation Using a Sample Java Application

This section explains the procedure for creating an environment in which a CORBA WorkUnit (CORBA application WorkUnit) can be operated using the Interstage Management Console.

If a CORBA application is to be run in a WorkUnit, an application execution environment must be created according to the following procedure. Using a sample Java application, this section explains how to create a WorkUnit, operate the application, and then delete the WorkUnit.

- Creating a CORBA application
- Starting the Interstage Management Console
- Creating a CORBA WorkUnit
- Deploying the CORBA application
- Starting the CORBA WorkUnit
- Running the Client Application
- Stopping the CORBA WorkUnit
- Undeploying the CORBA application
- Deleting the CORBA WorkUnit

This section explains the procedure for starting the sample application located at `/opt/FJSVod/src/samples/complex/samplelist.Java/data/any`.

The procedure for Java version "JDK 6" (the JDK installation path `/opt/FJSVawjbc/jdk6`) is shown below.

1) Creating a CORBA Application

Create a server application according to the execution procedure provided in the Distributed Application Development Guide (CORBA Service Edition). In this example, the sample application provided under `/opt/FJSVod/src/samples/complex/samplelist.Java/data/any` is used.

```
# PATH=/usr/bin:/opt/FJSVawjbc/jdk6/jre/bin:/opt/FJSVawjbc/jdk6/bin:/opt/  
FJSVod/bin:$PATH  
# export PATH  
# OD_HOME=/opt/FJSVod  
# export OD_HOME (Set OD_HOME in the environment variable.)  
# CLASSPATH=.:$OD_HOME/etc/class/ODjava4.jar:$CLASSPATH  
# export CLASSPATH (Set the classpath in the environment variable.)  
# LD_LIBRARY_PATH=$OD_HOME/lib:$LD_LIBRARY_PATH  
# export LD_LIBRARY_PATH (Set the library path in the environment variable.)  
# cd /opt/FJSVod/src/samples/complex/samplelist.Java/data/any  
# make (Create an application.)
```

2) Starting the Interstage Management Console

Start the Interstage Management Console by specifying `"https://(host-name):12000/IsAdmin"` from the Web browser.

3) Creating a CORBA WorkUnit

Create a WorkUnit definition to enable the CORBA application to run in a WorkUnit.

1. Select [System], [WorkUnits], and then the [Create a new WorkUnit] tab.
2. Select "CORBA" as the WorkUnit type.

The default "WU001" is entered for the WorkUnit name.

3. Click the [WorkUnit Settings [Show]] link to display detailed settings and enter `/opt/FJSVawjbc/jdk6/bin` for the Application Folder; `/opt/FJSVod/src/samples/complex/samplelist.Java/data/any` for the Application Working Directory; `/opt/FJSVod/lib` for the Library Path; and `OD_IMPLID=IDL:ODsample/anytest:1.0` for the Environment variables. Click the Create button.

4) Deploying the CORBA Application

Deploy the CORBA application so that it can run in the WorkUnit.

1. Select the [View WorkUnit Status] tab and click the newly created WorkUnit (WU001) in the WorkUnit list.
2. Select the [Deploy] tab, and enter "IDL:ODsample/anytest:1.0" for the Implementation Repository ID and "java" for the Program EXE File.
Leave the "Restart WorkUnit after deployment" check box selected.
3. Click [CORBA Application [Show]] to display the detailed settings of the CORBA application.
Enter "/opt/FJSVod/src/samples/complex/samplelist.Java/data/any", "/opt/FJSVod/etc/class/ODjava4.jar" for the Classpath; enter class name "simple_s" for the Startup parameters, and select 'SYNC_END' for the operation mode.
4. Click [Interface [Show]] to display the interface definition items. Click the Add Interface button to display the interface definition addition screen.
5. Enter 'IDL:ODsample/anytest:1.0' for the Interface Repository ID and 'ODsample::anytest' for the Naming Service Name.
6. Click the Add button to add the interface definition.
7. After the interface definition is complete, click the Deploy button to deploy the application.

5) Starting the CORBA WorkUnit

Start the CORBA WorkUnit that was set in the WorkUnit definition.

Enabling 'Restart WorkUnit after deployment' during deployment setting automatically starts the WorkUnit when deployment is finished. This function is enabled by default.

The WorkUnit is also started automatically when Interstage starts. Normally, automatic starting is also enabled by default.

1. Select the [Status] tab for the WorkUnit to be started.
2. Click the Start button.

6) Running the Client Application

Create a client application according to the procedure in the Distributed Application Development Guide (CORBA Service Edition). In this example, the sample application provided at /opt/FJSVod/src/samples/complex/samplelist.Java/data/any is used.

```
# PATH=/usr/bin:/opt/FJSVawjbc/jdk6/jre/bin:/opt/FJSVawjbc/jdk6/bin:/opt/  
FJSVod/bin:$PATH  
# export PATH  
# CLASSPATH=./opt/FJSVod/etc/class/ODjava4.jar:$CLASSPATH  
# export CLASSPATH  
#LD_LIBRARY_PATH=/opt/FJSVod/lib:$LD_LIBRARY_PATH  
# export LD_LIBRARY_PATH  
# cd /opt/FJSVod/src/samples/complex/samplelist.Java/data/any
```

Run the client application.

Execute the following at the client application storage destination:

```
# exec-CL  
in.param1 = a  
in.param2 = 1.0E-5  
inout.param1 = c  
inout.param2 = 1.0E-4  
rtn.param1 = 300  
rtn.param2 = test  
out.param1 = x  
out.param2 = 0.0010  
inout.param1 = y  
inout.param2 = 1.0E-4
```

7) Stopping the CORBA WorkUnit

Stop the WorkUnit that is running.

1. Select the [Status] tab for the WorkUnit to be stopped.
2. Click the Stop button.
3. "WorkUnit Stop Selection" page is displayed. Select "Normal Stop" and click the Stop button.

8) Undeploying the CORBA Application

Undeploy the CORBA application.

1. Select the [Undeploy] tab.
2. Select the check box to the left of the Implementation Repository ID to be undeployed, and click Undeploy.

9) Deleting the CORBA WorkUnit

Delete the CORBA WorkUnit.

1. Select the [Status] tab and click the Delete button.
2. The dialog message "WorkUnit will be deleted. Do you wish to continue?" is displayed. Click OK.

E.3.3 Procedure for Operation Using a Sample C++ Application

This section explains the procedure for creating an environment in which a CORBA WorkUnit (CORBA application WorkUnit) can be operated using the Interstage Management Console.

If a CORBA application is to be run in a WorkUnit, an application execution environment must be created according to the following procedure. Using a sample C++ application, this section explains how to create a WorkUnit, operate the application, and then delete the WorkUnit.

- Creating a CORBA application
- Starting the Interstage Management Console
- Creating a CORBA WorkUnit
- Deploying the CORBA application
- Starting the CORBA WorkUnit
- Running the Client Application
- Stopping the CORBA WorkUnit
- Undeploying the CORBA application
- Deleting the CORBA WorkUnit

This section explains the process for starting the sample application located at /opt/FJSVod/src/samples/complex/samplelist.C++/data/any.

1) Creating a CORBA Application

Create a server application according to the execution procedure provided in the Distributed Application Development Guide (CORBA Service Edition). Here, the sample application provided under /opt/FJSVod/src/samples/complex/samplelist.C++/data/any is used.

```
# OD_HOME=/opt/FJSVod
# export OD_HOME (Set OD_HOME in the environment variable.)
# LD_LIBRARY_PATH=$OD_HOME/lib:$LD_LIBRARY_PATH
# export LD_LIBRARY_PATH (Set the library path in the environment variable.)
# cd /opt/FJSVod/src/samples/complex/samplelist.C++/data/any
# make (Create an application.)
```

2) Starting the Interstage Management Console

Start the Interstage Management Console by specifying "https://(host-name):12000/IsAdmin" from the Web browser.

3) Creating a CORBA WorkUnit

Create a WorkUnit definition to enable the CORBA application to run in a WorkUnit.

1. Select [System], [WorkUnits], and then the [Create a new WorkUnit] tab.
2. Select "CORBA" as the WorkUnit type.

The default "WU001" is entered for the WorkUnit name.

3. Click the [WorkUnit Settings [Show]] link to display detailed settings and enter /opt/FJSVod/src/samples/complex/samplelist.C++/data/any for the Application Folder and enter /opt/FJSVod/src/samples/complex/samplelist.C++/data/any for the Application Working Directory. Click the Create button.

4) Deploying the CORBA Application

Deploy the CORBA application so that it can run in the WorkUnit.

1. Select the [View WorkUnit Status] tab and click on the newly created WorkUnit (WU001) in the WorkUnit list.
2. Select the [Deploy] tab, and enter "IDL:ODsample/anytest:1.0" for the Implementation Repository ID and "simple_s" for the Program EXE File.
Leave the "Restart WorkUnit after deployment" check box selected.
3. Click [Show Details [Show]] to display CORBA Application and Interface.
4. Click [CORBA Application [Show]] to display the detailed settings of the CORBA application. Select "SYNC_END" for the operation mode.
5. Click [Interface [Show]] to display the interface definition items. Click the Add Interface button to display the interface definition addition screen.
6. Enter "IDL:ODsample/anytest:1.0" for the Interface Repository ID and "ODsample::anytest" for the Naming Service Name.
7. Click the Add button to add the interface definition.
8. After the interface definition is complete, click the Deploy button to deploy the application.

5) Starting the CORBA WorkUnit

Start the CORBA WorkUnit that was set in the WorkUnit definition.

Enabling "Restart WorkUnit after deployment" during deployment setting automatically starts the WorkUnit when deployment is finished. This function is enabled by default.

The WorkUnit is also started automatically when Interstage starts. Normally, automatic starting is also enabled by default.

1. Select the [Status] tab for the WorkUnit to be started.
2. Click the Start button.

6) Running the Client Application

Create a client application according to the procedure in the Distributed Application Development Guide (CORBA Service Edition). In this example, the sample application provided at /opt/FJSVod/src/samples/complex/samplelist.C++/data/any is used.

```
# cd /opt/FJSVod/src/samples/complex/samplelist.C++/data/any
# make (Create an application.)
```

Run the client application.

Execute the following at the client application storage destination:

```
# simple_c
smpl->para1 = 100
```



```
smp1->para2 = OUT
smp2->para1 = x
smp2->para2 = 0.01
smp3->para1 = z
smp3->para2 = 0.001
```

7) Stopping the CORBA WorkUnit

Stop the WorkUnit that is running.

1. Select the [Status] tab for the WorkUnit to be stopped.
2. Click the Stop button.
3. "WorkUnit Stop Selection" page is displayed. Select "Normal Stop" and click the Stop button.

8) Undeploying the CORBA Application

Undeploy the CORBA application.

1. Select the [Undeploy] tab.
2. Select the check box to the left of the Implementation Repository ID to be undeployed, and click Undeploy.

9) Deleting the CORBA WorkUnit

Delete the CORBA WorkUnit.

1. Select the [Status] tab and click the Delete button.
2. The dialog message "WorkUnit will be deleted. Do you wish to continue?" is displayed. Click OK.

E.3.4 Procedure for Operation Using a Sample COBOL Application

This section explains the procedure for creating an environment in which a CORBA WorkUnit (CORBA application WorkUnit) can be operated using the Interstage Management Console.

If a CORBA application is to be run in a WorkUnit, an application execution environment must be created according to the following procedure. Using a sample COBOL application, this section explains how to create a WorkUnit, operate the application, and then delete the WorkUnit.

- Creating a CORBA application
- Starting the Interstage Management Console
- Creating a CORBA WorkUnit
- Deploying the CORBA application
- Starting the CORBA WorkUnit
- Running the Client Application
- Stopping the CORBA WorkUnit
- Undeploying the CORBA application
- Deleting the CORBA WorkUnit

This section explains the process for starting the sample application located at `/opt/FJSVod/src/samples/complex/samplelist.COBOL/data/any`.

1) Creating a CORBA Application

Create a server application according to the execution procedure provided in the Distributed Application Development Guide (CORBA Service Edition). In this example, the sample application provided under `/opt/FJSVod/src/samples/complex/samplelist.COBOL/data/any` is used.

```
# OD_HOME=/opt/FJSVod
# export OD_HOME (Set OD_HOME in the environment variable.)
# LD_LIBRARY_PATH=$OD_HOME/lib:$LD_LIBRARY_PATH
# export LD_LIBRARY_PATH (Set the library path in the environment variable.)
# cd /opt/FJSVod/src/samples/complex/samplelist.COBOL/data/any
# make (Create an application.)
```

2) Starting the Interstage Management Console

Start the Interstage Management Console by specifying "https://(host-name):12000/IsAdmin" from the Web browser.

3) Creating a CORBA WorkUnit

Create a WorkUnit definition to enable the CORBA application to run in a WorkUnit.

1. Select [System], [WorkUnits], and then the [Create a new WorkUnit] tab.
2. Select "CORBA" as the WorkUnit type.

The default "WU001" is entered for the WorkUnit name.

3. Click the [WorkUnit Settings [Show]] link to display detailed settings and enter /opt/FJSVod/src/samples/complex/samplelist.COBOL/data/any for the Application Folder and /opt/FJSVod/src/samples/complex/samplelist.COBOL/data/any for the Application Working Directory.

Also enter /opt/FJSVod/lib, /opt/FJSVod/lib/nt, /opt/FJSVod/src/samples/complex/samplelist.COBOL/data/any for the library path, and NLSPATH=/opt/FJSVod/lib/nls/%L/%N.cat:/opt/FJSVod/lib/nls/C/%N.cat:/usr/dt/lib/nls/msg/%L/%N.cat for the Environment variables. Click the Create button.

4) Deploying the CORBA Application

Deploy the CORBA application so that it can run in the WorkUnit.

1. Select the [View WorkUnit Status] tab and click the newly created WorkUnit (WU001) in the WorkUnit list.
2. Select the [Deploy] tab, and enter "IDL:INTF_A:1.0" for the Implementation Repository ID and "024_s" for the Program EXE File.
Leave the "Restart WorkUnit after deployment" check box selected.
3. Click [Show Details [Show]] to display Show CORBA Application and Show Interface Definition.
4. Click [Interface [Show]] to display the interface definition items. Click the Add Interface button to display the interface definition addition screen.
5. Enter "IDL:INTF_A:1.0" for the Interface Repository ID and "INTF_A" for the Naming Service Name.
Also enter "/opt/FJSVod/src/samples/complex/samplelist.COBOL/data/any/libINTF-A.so" for the Library Path.
6. Click the Add button to add the interface definition.
7. After the interface definition is complete, click the Deploy button to deploy the application.

5) Starting the CORBA WorkUnit

Start the CORBA WorkUnit that was set in the WorkUnit definition.

Enabling "Restart WorkUnit after deployment" during deployment setting automatically starts the WorkUnit when deployment is finished. This function is enabled by default.

The WorkUnit is also started automatically when Interstage starts. Normally, automatic starting is also enabled by default.

1. Select the [Status] tab for the WorkUnit to be started.
2. Click the Start button.

6) Running the Client Application

Create a client application according to the procedure in the Distributed Application Development Guide (CORBA Service Edition). In this example, the sample application provided at /opt/FJSVod/src/samples/complex/samplelist.COBOL/data/any is used.

```
# LD_LIBRARY_PATH=/opt/FJSVod/src/samples/complex/samplelist.COBOL/data/any:/  
opt/FJSVod/lib/nt:/opt/FJSVcbl/lib:$LD_LIBRARY_PATH  
# export LD_LIBRARY_PATH (Set the library path in the environment variable.)  
# cd /opt/FJSVod/src/samples/complex/samplelist.COBOL/data/any  
# make (Create an application.)
```

Run the client application.

Execute the following at the client application storage destination:

```
# 024_c  
CLIENT START!!  
IN-PARAM VALUE: 001743234  
IO-PARAM VALUE: 8888  
INTF-A-OP RETURN!!  
RESULT VALUE : .89333000E 02  
A-OUT-P VALUE : .555555555000000000E 02  
A-IO-P VALUE : +000012345
```

7) Stopping the CORBA WorkUnit

Stop the WorkUnit that is running.

1. Select the [Status] tab for the WorkUnit to be stopped.
2. Click the Stop button.
3. "WorkUnit Stop Selection" page is displayed. Select "Normal Stop" and click the Stop button.

8) Undeploying the CORBA Application

Undeploy the CORBA application.

1. Select the [Undeploy] tab.
2. Select the check box to the left of the Implementation Repository ID to be undeployed, and click Undeploy.

9) Deleting the CORBA WorkUnit

Delete the CORBA WorkUnit.

1. Select the [Status] tab and click the Delete button.
2. The dialog message "WorkUnit will be deleted. Do you wish to continue?" is displayed. Click OK.

Appendix F CORBA WorkUnit Activation Change

Solaris

This appendix explains the CORBA WorkUnit activation change function.

If this function is used, it is possible to switch application program modules and change the WorkUnit operating environment (WorkUnit configuration information) without stopping the WorkUnit itself. This is called "activation change".

During activation change of a CORBA WorkUnit, a new process is started, and the new process and the process that is currently running are switched.

It is also possible to start the new process and put it on standby beforehand. This means that it is possible to switch the new and old processes at a particular point. The old process can also be put on standby temporarily after the switch, so that it can be switched back if an error occurs.

CORBA WorkUnit activation change operations are described below.

Note that the CORBA WorkUnit activation change function can only be used in the Enterprise Edition.

Note

In this function, commands are used to execute several procedures before the actual activation change occurs. For this reason, a thorough knowledge of the activation change function is required, and procedures must be followed and executed correctly.

If it is possible to stop the WorkUnit as part of application program maintenance, it is recommended that the activation change function not be used. Stop the WorkUnit, switch the application program, and then restart the WorkUnit instead.

F.1 Procedure for CORBA WorkUnit Activation Change

This section explains the procedure for CORBA WorkUnit activation change.

This function is typically used in the following situations:

- Correction of application problems in online business (WorkUnit)
- Periodic changing of the application operating environment in a WorkUnit configuration (environment variables, and so on)

These operations are executed when applications are not running, or when the number of accesses is low, such as at night. Execution of these operations is not recommended when there are large numbers of requests, because application regression problems or an operating environment defect may occur after the change.

The four phases of CORBA WorkUnit activation change are as follows. To execute the activation change, execute the command for each phase for the WorkUnit that is running.

1. [WorkUnit Configuration Change Phase](#)
2. [Preparation \("prepare"\) Phase](#)
3. [Switch to New Environment \("change"\) Phase](#)
4. [Old Environment Deletion or Restoration \(Commit or Rollback\) Phase](#)

The current activation change phase can be checked using the "*ischeckwustat*" command.

F.1.1 WorkUnit Configuration Change Phase

In the WorkUnit configuration change phase, register the corrected WorkUnit configuration used for activation change in Interstage.

If the activation change function is used to correct an application program, the modules that are currently running cannot be overwritten directly. For this reason, store the correction target application module in a different directory to the one in which the modules that are currently running are stored. Next, change "Application Path" or "Application Library Path" (in Java, this is "Application Class Path") of the WorkUnit configuration to the directory in which the corrected module is stored, and then register the WorkUnit configuration change.

Execute the following procedure.

1. Create the application storage directory.

Example

If the application module that is currently running is stored in `"/apl/WU001/svapl"`, create `"/apl/WU001/svapl_UPDATE1"`.

2. Create a backup of the WorkUnit configuration file.

A backup of the WorkUnit configuration file that is currently running must be created before the WorkUnit configuration is changed.

3. Change the WorkUnit configuration file.

Change "Application Path (PATH)", Application Library Path (Library for Application)", or "Application Class Path (CLASSPATH for Application)" of the WorkUnit configuration to the path of the directory in which the corrected module is stored.

Example

PATH: `/apl/WU001/svapl_UPDATE1`

Change the environment variables in the application operating environment at the same time.

4. Register the WorkUnit configuration file after the change in Interstage.

Use the `"ischangewudef"` command to change the WorkUnit configuration. This registers the WorkUnit configuration used by the new environment after the activation change in Interstage.

At this stage, two statuses are registered: the WorkUnit configuration that is currently running, and the WorkUnit configuration that is used after the activation change.

The `ischangewudef` command can only be executed if the corresponding WorkUnit is running ("Execute" is displayed for the WorkUnit when the `islistwu` command is run). If this command is executed, the status changes from normal operation status to "activation being changed" status, and the phase changes to the "WorkUnit configuration change phase".

Note

- When the corrected application module is stored in the executable environment, make sure the application module that is currently running is not overwritten. If it is overwritten, an error may occur in the WorkUnit that is running. In this case, enter revision information in the directory, and store it in a different directory.
- It is also possible to prevent overwriting by entering revision information in the executable module (file name) after the correction, and changing the module name. To change the module name, set the name of the module after the correction in "Executable File (Executable File name)" of the WorkUnit configuration, and then execute the activation change.
- In the "WorkUnit configuration change phase", `ischangewudef -o` can be used to overwrite the configuration used for activation change. However, in phases from the "Preparation ("prepare") phase" onwards, this command cannot be executed until the activation change is completed. This is because it prevents the execution of the `isaddwudef` normal configuration registration command while the activation is being changed.
- If the CORBA application is in library format, only the library name is specified for the solib defined in the CORBA application definition intfid. You must also specify the library storage destination directory in 'Library for Application' of the WorkUnit definition.

Use the `"F.4.2 isinfcchangewudef"` command to refer to information about the WorkUnit configuration registered for use in activation change.

F.1.2 Preparation ("prepare") Phase

In the preparation ("prepare") phase, use the `ispreparewu` command to create a new WorkUnit environment based on the WorkUnit configuration registered for use in activation change, and start a new process. The new process starts up with standby status, meaning that requests from the client are not processed when the "prepare" phase is completed.

- When the preparation ("prepare") phase is completed, requests from the client for old processes are processed. Requests from the client for new processes are not processed. At this point, the application process number in the corresponding WorkUnit is twice the number for normal operation. If the process concurrency is changed, the value is the total of the old environment process concurrency and the new environment process concurrency.
- Set a value for `proc_conc_max` (in the CORBA server application configuration information registered in the implementation repository) that is twice the value for normal operation (the total value for the old environment process concurrency and the new environment process concurrency if the process concurrency is changed).

- The "*ispreparewu*" command can only be executed for a WorkUnit configuration change phase WorkUnit.

F.1.3 Switch to New Environment ("change") Phase

In the switch to new environment ("change") phase, use the "*ischangewu*" command to temporarily stop the processing of old process requests (after the processing of requests in progress is completed, processing stops temporarily). Processing requests that have accumulated in the current queue, and requests after that, are processed when the processing of new processes starts.

To abort an activation change, execute the "*isrollbackwu*" command. This deletes the new process environment, and operations continue in the old environment.

Note

- If an error occurs in an application in the new environment at the point when the switch to the new environment ("change") phase is completed, the old process is on standby with a "temporarily stopped" status so that it can be switched back. At this point, the application process number in the corresponding WorkUnit is still twice the number for normal operation (the total value for the old environment process concurrency and the new environment process concurrency if the process concurrency is changed).
- The "*ischangewu*" command can only be executed for a preparation ("prepare") phase WorkUnit.

F.1.4 Old Environment Deletion or Restoration (Commit or Rollback) Phase

In the old environment deletion or restoration (commit or rollback) phase, delete the old environment or execute a switch back to the old environment.

After the "change" phase, check the operating conditions for the application in the new environment. If there are no problems, delete the old environment using the "*iscommitwu*" command to execute the deletion request.

If there is an error in the application in the new environment, use the "*isrollbackwu*" command to restore the old environment and return the status before activation change.

Note

- If the "*iscommitwu*" command is used to delete the old environment, and the "*isrollbackwu*" command is used to delete the new environment that was restored to the old environment, it might mean that the application is running on a process that has been stopped. These commands therefore provide options for stopping processes synchronously (after application processing finishes) and for shutting down processes.
- The "*iscommitwu*" command can only be executed for a switch to new environment ("change") phase WorkUnit.
- If the "*isrollbackwu*" command is used to restore the old environment, the status before activation change is returned. For this reason, the WorkUnit configuration registered for use in activation change is deleted from the Interstage system. To re-execute activation change, start again from "1) WorkUnit configuration change phase".

The "*isrollbackwu*" command can be executed in each of the following phases: WorkUnit configuration change phase, Preparation ("prepare") phase, Switch to new environment ("change") phase. Regardless of the phase, the status before activation change is returned after the "*isrollbackwu*" command is executed.

F.2 WorkUnit Configurations that can be Changed during Activation Change

WorkUnit configuration information that can be changed during activation change is shown below.

The "Application Program" section cannot be added/deleted. To add/delete the "Application Program" section, stop the WorkUnit and then re-register the WorkUnit configuration.

CORBA server application configuration information registered in the implementation repository cannot be changed. To change this information, stop the WorkUnit and then re-register the WorkUnit configuration.

Section	Configuration option	Contents	Changeable?
WORK UNIT	Name	Set the WorkUnit name. The WorkUnit name cannot be changed.	No

Section	Configuration option	Contents	Changeable?
	Kind	Set "CORBA" for a CORBA application. The WorkUnit type cannot be changed.	No
Control Option	Path	Set the path of the directory in which the executable file is stored. To switch the application, specify the path of the directory in which the corrected module is stored. (Note) Store the corrected module in a different directory to the one in which the modules that are currently running are stored.	Yes
	Current Directory	Set the path for creating the application current directory.	Yes
	Application Retry Count	Set the number of continuous shut downs until automatic restart is no longer possible for the application.	Yes
	Path for Application	Set the path (the PATH environment variable) that is used when the application is running.	Yes
	Library for Application	Set the path for the application library that is used when the application is running (the LD_LIBRARY_PATH environment variable). If the application subprogram is stored in a different directory to the application executable file, specify the directory path for storing the corrected module. (Note) Store the corrected module in a different directory to the one in which the modules that are currently running are stored.	Yes
	Environment Variable	Set the environment variable that is used when the application is running.	Yes
	Maximum Processing Time for Exit Program	Set the maximum processing time monitoring value for the exit program.	Yes
	WorkUnit Exit Program	Set the name of the exit program that is issued once when the WorkUnit starts and once when it finishes. (Note) The changed WorkUnit exit program is valid after the switch to the new environment ("change") phase of the activation change. If the WorkUnit is stopped in the activation change preparation ("prepare") phase, the WorkUnit exit program before the change is valid.	Yes
Executable File of Exit Program for Salvage	Set the name of the executable file for the WorkUnit exit program and the process salvage exit program. (Note) When salvaging a process in an old environment, the value before the change is valid. When	Yes	

Section	Configuration option	Contents	Changeable?
		salvaging a process in a new environment, the value after the change is valid.	
	Request Assignment Mode	Request message assignment mode (Note) Queue-related configuration information cannot be changed.	No
	Traffic Director Monitor Mode	Set whether or not to make this WorkUnit a target of crash monitoring during IPCOM linkage. WorkUnits that are targets of monitoring cannot be changed while they are running.	No
	Startup Time	Specify the monitoring time until startup of the WorkUnit is completed. (Note) Monitor the process startup time in the activation change preparation ("prepare") phase.	Yes
	Shutdown Time	Set the monitoring time (in seconds) until the WorkUnit is stopped completely. (Note) Monitor the process stop time in the old environment deletion or restoration (commit or rollback) phase of the activation change.	Yes
	Process Degeneracy	Specify whether or not to allow the WorkUnit to continue running when automatic restart of the application fails.	Yes
	Number of Revision Directories	Specify the number of revision directories in the current directory of the WorkUnit. (Note) If the current directory is changed by the activation change, this is valid if the WorkUnit directory exists in the new directory.	Yes
Application Program	Impl ID	Specify the implementation repository ID of the CORBA application that starts up in the WorkUnit. (Note) The implementation repository ID cannot be changed.	No
	Executable File	Specify the executable module for the CORBA application. If the application is switched, change the value specified in "Executable File" to enter and manage revision information in the executable file.	Yes
	Concurrency	Set the process concurrency for the application.	Yes
	Maximum Processing Time	Set the maximum processing time monitoring value for the application (in seconds).	Yes
	Terminate Process for Time out	Specify whether or not to shut down the process that is running in the corresponding application	Yes

Section	Configuration option	Contents	Changeable?
		when the maximum processing time for the application is exceeded.	
	Maximum Processing Time for Exit Program	Set the maximum processing time monitoring value for the exit program (in seconds).	Yes
	Maximum Queuing Message	Set the maximum queue number for messages that can accumulate in the queue. (Note) Queue-related configuration information cannot be changed.	No
	Queuing Message To Notify Alarm	Set the accumulated queue number for notifying the alarm. The alarm is notified when the accumulated queue number reaches this value. (Note) Queue-related configuration information cannot be changed.	No
	Queuing Message To Notify Resumption	Set the accumulated queue number for resuming monitoring of the alarm notice. If the accumulated queue number exceeds the monitored queuing number, monitoring of the accumulated queue number is resumed when it is the same as this value. (Note) Queue-related configuration information cannot be changed.	No
	CLASSPATH for Application	Set the classpath that is used when the application is running. To switch the Java application, specify the classpath in which the corrected module is stored. (Note) Store the corrected module in a different directory to the one in which the modules that are currently running are stored.	Yes
	Environment Variable	Set the environment variable that is used when the application is running.	Yes
	Exit Program for Process Salvage	Set the process salvage exit program. Only C programs can be specified. (Note) When salvaging a process in an old environment, the value before the change is valid. When salvaging a process in a new environment, the value after the change is valid.	Yes
	Executable File of Exit Program for Salvage	Set the name of the executable file for the WorkUnit exit program and the process salvage exit program. (Note)	Yes

Section	Configuration option	Contents	Changeable?
		When salvaging a process in an old environment, the value before the change is valid. When salvaging a process in a new environment, the value after the change is valid.	
	Param for Executable File	Set the parameter that is passed when the application starts up. If more than one parameter is set, set this option more than once. (Note) This is valid after the process starts up in the new environment in the activation change preparation ("prepare") phase.	Yes
	Request Assignment Mode	Request message assignment mode (Note) Queue-related configuration information cannot be changed.	No
	Buffer Number	Set the buffer number for the queue. (Note) Queue-related configuration information cannot be changed.	No
	Buffer Size	Set the data length for executing the queue operation for one request. (Note) Queue-related configuration information cannot be changed.	No
	Path	Set the path of the directory in which the executable file for the application program is stored. To switch the application, specify the path of the directory in which the corrected module is stored. (Note) Store the corrected module in a different directory to the one in which the modules that are currently running are stored.	Yes

Note

In CORBA server application configuration information registered in the implementation repository, if settings for retaining instance data have been implemented in each client application (iswitch=ON), the new process does not inherit the instance data from the old process. For this reason, a problem may occur in applications in which the retaining of instance data is required.

When executing activation change, the possibility that instance data will not be retained in the created application must be considered.

Also be sure prior to activation change to set a value for the maximum process concurrency (proc_conc_max) that is twice the value for normal operation (the total value for the old environment process concurrency and the new environment process concurrency if the process concurrency is changed).

When activation change is executed, the process concurrency is temporarily twice the value for normal operation. If the value for proc_conc_max is low, the preparation ("prepare") phase fails.

F.3 Notes on Executing Activation Change

This section contains notes about executing activation change.

F.3.1 Relationship between Activation Change and Existing Functions

Updating the WorkUnit Configuration (Update of the Configuration using the "isaddwundef -o" Command)

The "*isaddwundef -o*" command can be used to update the WorkUnit configuration even if the WorkUnit is running (the changes are valid after the next startup). However, updating of the WorkUnit configuration is prevented if the activation is being changed (from the WorkUnit configuration change phase onwards). Changes during the WorkUnit configuration change phase can only be made using the *ischangewundef* command. *ischangewundef* cannot be used during or after the preparation ("prepare") phase.

After the activation change is finished, the "*isaddwundef -o*" command can be used to update the WorkUnit configuration.

Changing the Process Concurrency Dynamically

Changing of the process concurrency dynamically is prevented while the activation is being changed (from the WorkUnit configuration change phase onwards). To change the process concurrency dynamically, wait until the activation change is finished.

Activation change is also prevented during dynamic changing of the process concurrency, but is possible once dynamic changing of the process concurrency is completed.

Degeneracy Operation when Automatic Restart of the Application Fails

Activation change is possible during normal operation, even if the degeneracy operation is valid when automatic restart of the application fails. Activation change is also possible with "Running degeneracy" status. If there is an error in the application program or operating environment, the correct status can be restored by implementing the activation change. In this case, the status of the WorkUnit after the activation change is "Running normally (execute)".

Restore processing for a degeneracy operation status WorkUnit cannot be executed while the activation is being changed in the following phases: WorkUnit configuration change phase, Preparation ("prepare") phase, Switch to new environment ("change" phase)).

If the process number is set to "0" while the activation is being changed, the WorkUnit crashes.

Restart of the Process while the Activation is being Changed

In the switch to new environment ("change") phase of activation change, if the application in progress in the old environment crashes the process is restarted in the old environment and put on standby status. If the restart of the process fails, the WorkUnit crashes (if the settings for degeneracy operation when automatic restart of the application fails are valid, the status changes to degeneracy operation status).

F.3.2 WorkUnit Environment after Restart

If the WorkUnit stops during the activation change, the environments for restarting the WorkUnit in each phase are as shown in the following table.

Regardless of whether the WorkUnit for which the activation is being changed shuts down independently or because of an Interstage crash, the procedure for restarting after the crash is the same.

Status before crash	Environment after restart	Remarks
WorkUnit configuration change phase	Old environment	After the restart, the status is the same as before the activation change. (*1)
Preparation ("prepare") phase	Old environment	After the restart, the status is the same as before the activation change. (*1)
Switch to new environment ("change") phase	New environment	After the restart, the status is the same as on completion of the activation change. (*2)
Status where the old environment is deleted, and the activation change is completed	New environment	After the restart, the status is the same as on completion of the activation change.

Status before crash	Environment after restart	Remarks
Status where the old environment is restored, and the status before the activation change is returned	Old environment	After the restart, the status is the same as before the activation change.

*1 If a crash occurs in the WorkUnit configuration change phase and the activation change preparation ("prepare") phase, the start status after the restart is the same as before the activation change was executed.

*2 To restart the WorkUnit in the old environment after the crash, re-register the WorkUnit configuration for the old environment using the "*isaddwudel*" command, and then restart the WorkUnit.

F.3.3 Stopping a WorkUnit while the Activation is being Changed

While the activation is being changed, a WorkUnit can only be stopped by shutting it down in the "prepare" phase or the "change" phase.

Activation change status	Can the WorkUnit be stopped?	Environment after restart	Remarks
WorkUnit configuration change phase	Normal: Yes Synchronous: Yes Shutdown: Yes	Old environment	After the restart, the status is the same as before the activation change. (*1)
Preparation ("prepare") phase	Normal: No Synchronous: No Shutdown: Yes	Old environment	After the restart, the status is the same as before the activation change. (*1)
Switch to new environment ("change") phase	Normal: No Synchronous: No Shutdown: Yes	New environment	After the restart, the status is the same as on completion of the activation change. (*2)
Status where the old environment is deleted, and the activation change is completed	Normal: Yes Synchronous: Yes Shutdown: Yes	New environment	After the restart, the status is the same as on completion of the activation change.
Status where the old environment is restored, and the status before the activation change is returned	Normal: Yes Synchronous: Yes Shutdown: Yes	Old environment	After the restart, the status is the same as before the activation change.

*1 If the WorkUnit is stopped using the "*stop*" command in the WorkUnit configuration change phase and the activation change preparation ("prepare") phase, it is assumed that the activation change has been canceled, and the start status after the restart is the same as before the activation change was executed.

*2 If the WorkUnit is shut down using the "*stop*" command in the switch to new environment ("change") phase, it is assumed that the activation change has been completed, and the start status after the restart is the same as when the activation change is completed. To start the WorkUnit with the status before the activation change, re-register the WorkUnit configuration for the old environment using the "*isaddwudel*" command, and then restart the WorkUnit.

F.3.4 Current Directory

The application process work directory (refer Note below) is created using an application process Process ID Name. If the activation change is implemented, new environment work directories are created, with previous environment work directories left as they are.

Previous environment work directories are not deleted once the activation change is complete.

For this reason, the number of work directories may increase, compressing the file system, if an activation change is repeated.

Additionally, if an activation change is repeated, you must change the current directory specified in the WorkUnit definition before and after the activation change. If this directory is no longer necessary, delete it once the activation change is complete.

Note

The work directory is made up of the following structure:

```
Current directory specified in the WorkUnit definition/WorkUnit name/Application execution process ID
```

F.4 Activation Change Command References

This section explains the commands that are used to execute activation changes.

F.4.1 ischangewudef

Name

ischangewudef

WorkUnit configuration change for the activation change

Format

```
ischangewudef [-M system] [-o] WorkUnit configuration file name
```

Description

The "*ischangewudef*" command executes the WorkUnit configuration change for the activation change. This command has the following options and arguments:

-M system 

Specify the name of the target system.

If this option is omitted, the default system is specified.

-o

If a WorkUnit configuration with the same name already exists, it is overwritten. If a WorkUnit configuration does not exist, a new one is registered. If this option is omitted, a new WorkUnit configuration is not registered if there is already a WorkUnit configuration with the same name.

WorkUnit configuration file name

Specify the file name of the text file described in the WorkUnit configuration. If there are any spaces in the file name, put the file name in single or double quotation marks.

Notes

- Incomplete configurations cannot be registered.
- The "*ischangewudef*" command can only be used by the user or superuser that started the WorkUnit.
- The "*ischangewudef*" command can only be used for "CORBA" WorkUnit types.
- This command cannot be used on a Managed Server.

Example

Registering this command in the "WU001.wu" file in which the WorkUnit configuration is described:

```
ischangewudef WU001.wu
```

Registering this command in the "WU001.wu" file in which the WorkUnit configuration is described by overwriting the existing contents:

```
ischangewudef -o WU001.wu
```

F.4.2 isinfchangewudef

Name

isinfchangewudef

Displays the contents of the WorkUnit configuration in the configuration change phase of the activation change

Format

```
isinfchangewudef [-M system] wuname
```

Description

The "*isinfchangewudef*" command outputs the contents of the WorkUnit configuration registered for activation change in the configuration change phase of the activation change in standard output.

This command can be used to refer to the contents of the WorkUnit configuration registered for activation change when the CORBA WorkUnit has started up, and after the "*ischangewudef*" command is executed from the configuration change phase of the activation change up until the switch to new environment ("change") phase.

This command has the following argument:

-M system 

Specify the name of the target system.

If this option is omitted, the default system is specified.

wuname

Specify the name of the WorkUnit to be displayed.

Example

```
isinfchangewudef WU001
[WORK UNIT]
Name: WU001
Kind: CORBA

[Control Option]
Application Retry Count: 0
Maximum Processing Time for Exit Program: 300
Request Assignment Mode: LIFO
Output of Stack Trace: NO
Startup Time: 180
Shutdown Time: 180
Traffic Director Monitor Mode: NO
Process Degeneracy: NO
Start Log: NO
Number of Revision Directories: 1
[Application Program]
Executable File: simple_s
Concurrency: 1
Maximum Processing Time: 0
Terminate Process for Time out: NO
Maximum Processing Time for Exit Program: 300
Maximum Queuing Message: 0
Request Assignment Mode: LIFO
Impl ID: IDL: test1/intf1: 1.0
```

F.4.3 ispreparewu

Name

ispreparewu

WorkUnit activation change preparation

Format

```
ispreparewu [-M system] wuname
```

Description

The "*ispreparewu*" command executes WorkUnit activation change preparation.

This command has the following argument:

-M system

Specify the name of the target system.

If this option is omitted, the default system is specified.

wuname

Specify the name of the WorkUnit for activation change preparation.

Notes

- Take a backup of the WorkUnit configuration file before the activation change.
- If settings for retaining instance data have been implemented in each client application (iswitch=ON), the new process does not inherit the instance data from the old process.
- Set the combined process concurrency for the old and new processes for the started application in the "proc_conc_max" server application information (maximum process concurrency) value registered in the implementation repository. "proc_conc_max" should be set before the activation change.
- The "*ispreparewu*" command can only be used by the user or superuser that started the WorkUnit.
- The "*ispreparewu*" command can only be used for "CORBA" WorkUnit types.

Example

Executing activation change preparation for WorkUnit "WU001":

```
ispreparewu WU001
```

F.4.4 ischangewu

Name

ischangewu

Executes activation change for the WorkUnit

Format

```
ischangewu [-M system] wuname
```

Description

The "*ischangewu*" command executes activation change for a WorkUnit for which activation change preparation is completed (the switch to the new environment).

This command has the following argument:

-M system Solaris

Specify the name of the target system.

If this option is omitted, the default system is specified.

wuname

Specify the name of the WorkUnit for which activation change is to be executed.

Notes

- If settings for retaining instance data have been implemented in each client application (iswitch=ON), the new process does not inherit the instance data from the old process.
- The "*ischangewu*" command can only be used by the user or superuser that started the WorkUnit.
- The "*ischangewu*" command can only be used for "CORBA" WorkUnit types.

Example

Executing the activation change for WorkUnit "WU001":

```
ischangewu WU001
```

F.4.5 iscommitwu

Name

iscommitwu

Deletes the old environment in the activation change for the WorkUnit

Format

```
iscommitwu [-M system] [-c] wuname
```

Description

The "*iscommitwu*" command deletes the old environment in the activation change for the WorkUnit, completes the activation change, and continues operations in the new environment.

This command has the following arguments:

-M system Solaris

Specify the name of the target system.

If this option is omitted, the default system is specified.

-c

Shuts down the old environment process after the "change" phase.

If this option is omitted, processes in the old environment are stopped synchronously (after the processing of requests in progress is completed, processing stops).

wuname

Specify the name of the WorkUnit for which activation change is to be executed.

Notes

- If "SYNC_END" is specified in the running mode after activation of the server application, finish the process explicitly after the return from the activation method (such as the issue of the "exit" function).
- The "*iscommitwu*" command can only be used by the user or superuser that started the WorkUnit.
- The "*iscommitwu*" command can only be used for "CORBA" WorkUnit types.

Example

Deleting the old environment for WorkUnit "WU001".

```
iscommitwu WU001
```

F.4.6 isrollbackwu

Name

isrollbackwu

Restores the old environment in the activation change for the WorkUnit

Format

```
isrollbackwu [-M system] [-c] wuname
```

Description

The "*isrollbackwu*" command returns the status to the status before the execution of activation change when the WorkUnit activation is being changed or when the WorkUnit was stopped during activation change.

This command has the following arguments:

-M system 

Specify the name of the target system.

If this option is omitted, the default system is specified.

-c

Shuts down the new environment process after the "change" phase is completed.

If this option is omitted, processes in the new environment are stopped synchronously (after the processing of requests in progress is executed, processing stops).

wuname

Specify the name of the WorkUnit for which activation change is to be executed.

Notes

- If "SYNC_END" is specified in the running mode after activation of the server application, finish the process explicitly after the return from the activation method (such as the issue of the "exit" function).
- The "*isrollbackwu*" command can only be used by the user or superuser that started the WorkUnit.
- The "*isrollbackwu*" command can only be used for "CORBA" WorkUnit types.

Example

Restoring the old environment for WorkUnit "WU001".

```
isrollbackwu WU001
```

F.4.7 ischeckwustat

Name

ischeckwustat

Displays the status of the activation change for the WorkUnit

Format

```
ischeckwustat [-M system] wuname | -a
```

Description

The "*ischeckwustat*" command displays the current activation change phase for the specified WorkUnit, or the activation change status for all WorkUnits.

The following information is displayed:

- WorkUnit name
- WorkUnit type
- activation change status

The following status is displayed:

- execute : Running normally
- registered : The configuration change phase
- prepare : The "prepare" phase
- change : The "change" phase

This command has the following arguments:

-M system Solaris

Specify the name of the target system.

If this option is omitted, the default system is specified.

wuname

To check the activation change status of a specific WorkUnit, specify the name of the WorkUnit.

-a

Specify this option to check the activation change status for all WorkUnits.

Note

- The "*ischeckwustat*" command can only be used for "CORBA" WorkUnit types.

Example

```
>ischeckwustat WU001
  wuname      kind      status
  WU001      CORBA      registered

>ischeckwustat -a
  wuname      kind      status
  WU001      CORBA      execute
  WU002      CORBA      registered
  WU003      CORBA      prepare
  WU004      CORBA      change
  WU005      CORBA      execute
```